M5 Forecasting - Accuracy

Estimate the unit sales of Walmart retail goods

CONTENTS

01

Introduction

01 Introduction

Description

Estimate the unit sales of Walmart retail goods

The objective of the M5 forecasting competition is to advance the theory and practice of forecasting by identifying the method(s) that provide the most accurate point forecasts for each of the 42,840 ti me series of the competition.

01 Introduction

Imported Libraries

Basic Libraries & Initial Setting

In [1]:
```
import numpy as np
import pandas as pd
pd.set_option('display.max_rows', 100, 'display.max_columns', 30)
from  datetime import datetime, timedelta
from tqdm.auto import tqdm
import warnings
warnings.filterwarnings(action='ignore')
import random
import pickle
from typing import Union
```

Libraries for Visualization & EDA

In [2]:
```
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.offline as pyo
pyo.init_notebook_mode()

import missingno

import pyecharts
pyecharts.online()
from pyecharts import Line, Bar, Pie, Overlap, Gauge
from IPython.display import display, Image

import bokeh
from bokeh.io import show
from bokeh.palettes import Spectral9
from bokeh.plotting import figure, show, output_notebook, reset_output
output_notebook()

import altair as alt

import calplot

import colorcet as cc
rgb_light_palette = [(0, 122, 255), # Blue
    (255, 149, 0), # Orange
    (52, 199, 89), # Green
    (255, 59, 48), # Red
    (175, 82, 222),# Purple
    (255, 45, 85), # Pink
    (88, 86, 214), # Indigo
    (90, 200, 250),# Teal
    (255, 204, 0)  # Yellow
]
light_palette = np.array(rgb_light_palette)/255
```

BokehJS 2.2.3 successfully loaded.

Libraries for Modeling

In [114]:
```
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
from fbprophet import Prophet
import lightgbm as lgb
from lightgbm import LGBMClassifier, plot_importance, plot_metric, plot_tree
import xgboost as xgb
from xgboost import XGBRegressor
import optuna
from optuna import Trial
from optuna.samplers import TPESampler
from optuna.visualization import plot_contour, plot_optimization_history
from optuna.visualization import plot_parallel_coordinate, plot_slice, plot_param_importances
```

Basic

numpy, pandas, datetime, tqdm, warnings, r
andom, pickle, Union

Visualization & EDA

matplotlib, matplotlib, seaborn, sklearn, plotl
y, missingno, pyecharts, bokeh, altair, calplot,
colorcet

Modeling

sklearn, statsmodels, fbprophet, lightgbm, xg
boost, optuna

01 Introduction

WRMSSE

The accuracy of the point forecasts will be evaluated using the Root Mean Squared Scaled Error (RMSSE),
Yt : the actual future value of the examined time series at point t,
Yt^ : the generated forecast,
n: the length of the training sample (number of historical observations),
h: the forecasting horizon.

$$\boldsymbol{RMSSE} = \sqrt{\frac{1}{h}\frac{\sum_{t=n+1}^{n+h}\left(Y_t - \widehat{Y}_t\right)^2}{\frac{1}{n-1}\sum_{t=2}^{n}(Y_t - Y_{t-1})^2}},$$

After estimating the RMSSE for all the 42,840 time series of the competition, the participating methods will be ranked using the Weighted RMSSE (WRMSSE), as described latter in this Guide, using the following formula:

$$\boldsymbol{WRMSSE} = \sum_{i=1}^{42{,}840} w_i * RMSSE,$$

where wi is the weight of the ith series of the competition. **A lower WRMSSE score is better**

01 Introduction

WRMSSE

Code making class for Creating WRMSSE

```
class WRMSSEEvaluator(object):

  def __init__(self, train_df: pd.DataFrame, valid_df: pd.DataFrame,
               calendar: pd.DataFrame, prices: pd.DataFrame):
    train_y = train_df.loc[:, train_df.columns.str.startswith('d_')]
    train_target_columns = train_y.columns.tolist()
    weight_columns = train_y.iloc[:, -28:].columns.tolist()

    train_df['all_id'] = 'all'  # for lv1 aggregation

    id_columns = train_df.loc[:, ~train_df.columns.str.startswith('d_')]\
                 .columns.tolist()
    valid_target_columns = valid_df.loc[:, valid_df.columns.str.startswith('d_')]\
                           .columns.tolist()

    if not all([c in valid_df.columns for c in id_columns]):
      valid_df = pd.concat([train_df[id_columns], valid_df],
                           axis=1, sort=False)

    self.train_df = train_df
    self.valid_df = valid_df
    self.calendar = calendar
    self.prices = prices

    self.weight_columns = weight_columns
    self.id_columns = id_columns
    self.valid_target_columns = valid_target_columns

    weight_df = self.get_weight_df()

    self.group_ids = (
      'all_id',
      'state_id',
      'store_id',
      'cat_id',
      'dept_id',
      ['state_id', 'cat_id'],
      ['state_id', 'dept_id'],
      ['store_id', 'cat_id'],
      ['store_id', 'dept_id'],
      'item_id',
      ['item_id', 'state_id'],
      ['item_id', 'store_id']
    )

    for i, group_id in enumerate(tqdm(self.group_ids)):
      train_y = train_df.groupby(group_id)[train_target_columns].sum()
      scale = []
      for _, row in train_y.iterrows():
        series = row.values[np.argmax(row.values != 0):]
        scale.append(((series[1:] - series[:-1]) ** 2).mean())
      setattr(self, f'lv{i + 1}_scale', np.array(scale))
      setattr(self, f'lv{i + 1}_train_df', train_y)
      setattr(self, f'lv{i + 1}_valid_df', valid_df.groupby(group_id)\
              [valid_target_columns].sum())

      lv_weight = weight_df.groupby(group_id)[weight_columns].sum().sum(axis=1)
      setattr(self, f'lv{i + 1}_weight', lv_weight / lv_weight.sum())

  def get_weight_df(self) -> pd.DataFrame:
    day_to_week = self.calendar.set_index('d')['wm_yr_wk'].to_dict()
    weight_df = self.train_df[['item_id', 'store_id'] + self.weight_columns]\
                .set_index(['item_id', 'store_id'])
    weight_df = weight_df.stack().reset_index()\
                .rename(columns={'level_2': 'd', 0: 'value'})
    weight_df['wm_yr_wk'] = weight_df['d'].map(day_to_week)

    weight_df = weight_df.merge(self.prices, how='left',
                                on=['item_id', 'store_id', 'wm_yr_wk'])
    weight_df['value'] = weight_df['value'] * weight_df['sell_price']
    weight_df = weight_df.set_index(['item_id', 'store_id', 'd'])\
                .unstack(level=2)['value']\
                .loc[zip(self.train_df.item_id, self.train_df.store_id), :]\
                .reset_index(drop=True)
    weight_df = pd.concat([self.train_df[self.id_columns],
                           weight_df], axis=1, sort=False)
    return weight_df

  def rmsse(self, valid_preds: pd.DataFrame, lv: int) -> pd.Series:
    valid_y = getattr(self, f'lv{lv}_valid_df')
    score = ((valid_y - valid_preds) ** 2).mean(axis=1)
    scale = getattr(self, f'lv{lv}_scale')
    return (score / scale).map(np.sqrt)

  def score(self, valid_preds: Union[pd.DataFrame,
                                     np.ndarray]) -> float:
    assert self.valid_df[self.valid_target_columns].shape \
           == valid_preds.shape

    if isinstance(valid_preds, np.ndarray):
      valid_preds = pd.DataFrame(valid_preds,
                                 columns=self.valid_target_columns)

    valid_preds = pd.concat([self.valid_df[self.id_columns],
                             valid_preds], axis=1, sort=False)

    all_scores = []
    for i, group_id in enumerate(self.group_ids):

      valid_preds_grp =
valid_preds.groupby(group_id)[self.valid_target_columns].sum()
      setattr(self, f'lv{i + 1}_valid_preds', valid_preds_grp)

      lv_scores = self.rmsse(valid_preds_grp, i + 1)
      setattr(self, f'lv{i + 1}_scores', lv_scores)

      weight = getattr(self, f'lv{i + 1}_weight')
      lv_scores = pd.concat([weight, lv_scores], axis=1,
                            sort=False).prod(axis=1)

      all_scores.append(lv_scores.sum())

    self.all_scores = all_scores

    return np.mean(all_scores)

wrmsse_calendar = pd.read_csv('../../m5-data/calendar.csv')
wrmsse_prices = pd.read_csv('../../m5-data/sell_prices.csv')
wrmsse_sales = pd.read_csv('../../m5-data/sales_train_evaluation.csv')
wrmsse_train_df = wrmsse_sales.iloc[:, :-28]
wrmsse_valid_df = wrmsse_sales.iloc[:, -28:]

evaluator = WRMSSEEvaluator(wrmsse_train_df, wrmsse_valid_df,
                            wrmsse_calendar, wrmsse_prices)
```

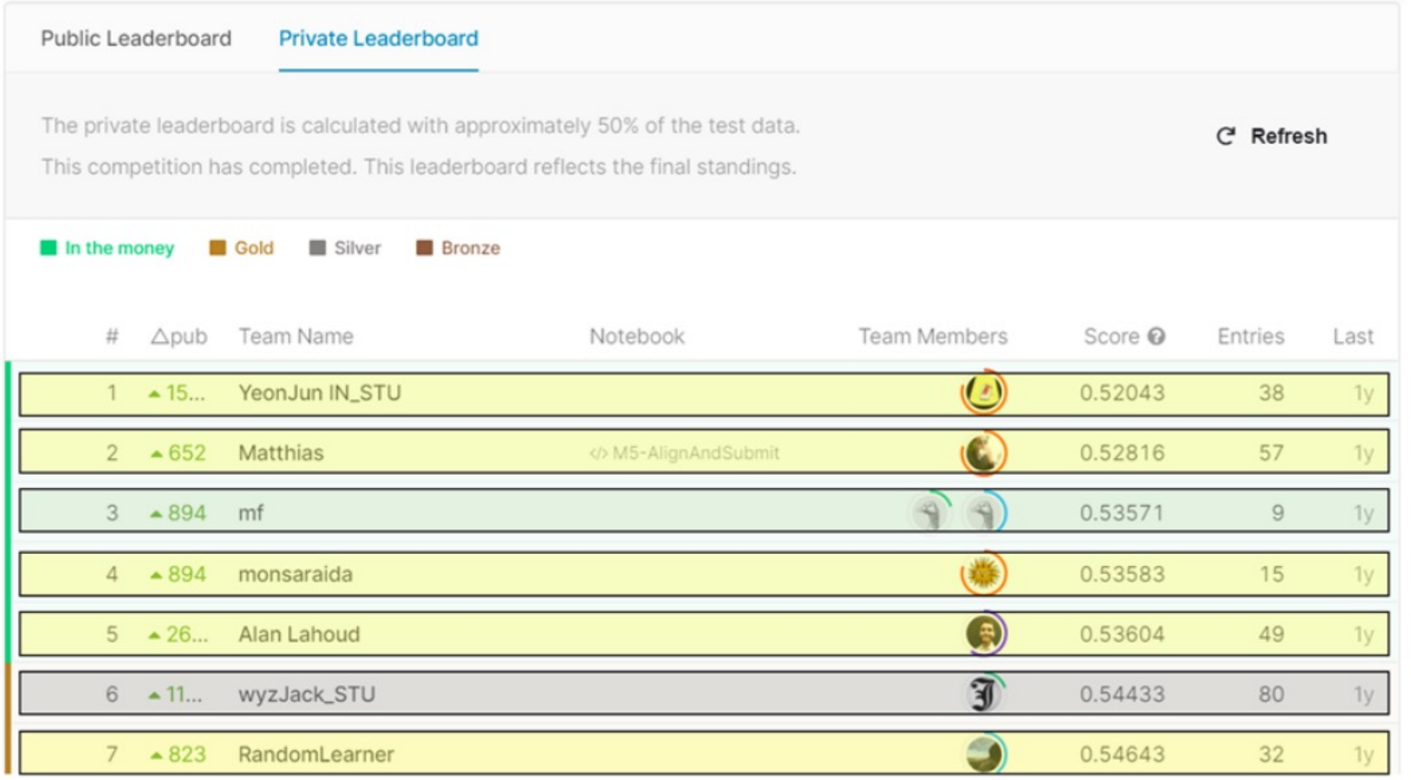
Code for Visualizing based on WRMSSE

```
def create_viz_df(df,lv):

    df = df.T.reset_index()
    if lv in [6,7,8,9,11,12]:
        df.columns = [i[0] + '_' + i[1] if i != ('index','') \
                      else i[0] for i in df.columns]
    df = df.merge(calendar.loc[:, ['d','date']], how='left',
                  left_on='index', right_on='d')
    df['date'] = pd.to_datetime(df.date)
    df = df.set_index('date')
    df = df.drop(['index', 'd'], axis=1)

    return df


def create_dashboard(evaluator):

    wrmsses = [np.mean(evaluator.all_scores)] + evaluator.all_scores
    labels = ['Overall'] + [f'Level {i}' for i in range(1, 13)]

    ## WRMSSE by Level
    plt.figure(figsize=(12,5))
    ax = sns.barplot(x=labels, y=wrmsses)
    ax.set(xlabel='', ylabel='WRMSSE')
    plt.title('WRMSSE by Level', fontsize=20, fontweight='bold')
    for index, val in enumerate(wrmsses):
        ax.text(index*1, val+.01, round(val,4), color='black',
                ha="center")

    # configuration array for the charts
    n_rows = [1, 1, 4, 1, 3, 3, 3, 3, 3, 3, 3]
    n_cols = [1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
    width = [7, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14]
    height = [4, 3, 12, 3, 9, 9, 9, 9, 9, 9, 9]

    for i in range(1,13):

        scores = getattr(evaluator, f'lv{i}_scores')
        weights = getattr(evaluator, f'lv{i}_weight')

        if i > 1 and i < 9:
            if i < 7:
                fig, axs = plt.subplots(1, 2, figsize=(12, 3))
            else:
                fig, axs = plt.subplots(2, 1, figsize=(12, 8))

            ## RMSSE plot
            scores.plot.bar(width=.8, ax=axs[0], color='g')
            axs[0].set_title(f"RMSSE", size=14)
            axs[0].set(xlabel='', ylabel='RMSSE')
            if i >= 4:
                axs[0].tick_params(labelsize=8)
            for index, val in enumerate(scores):
                axs[0].text(index*1, val+.01, round(val,4), color='black',
                        ha="center", fontsize=10 if i == 2 else 8)

            ## Weight plot
            weights.plot.bar(width=.8, ax=axs[1])
            axs[1].set_title(f"Weight", size=14)
            axs[1].set(xlabel='', ylabel='Weight')
            if i >= 4:
                axs[1].tick_params(labelsize=8)
            for index, val in enumerate(weights):
                axs[1].text(index*1, val+.01, round(val,2), color='black',
                        ha="center", fontsize=10 if i == 2 else 8)

            fig.suptitle(f'Level {i}: {evaluator.group_ids[i-1]}', size=24 ,
                         y=1.1, fontweight='bold')
            plt.tight_layout()
            plt.show()

        trn = create_viz_df(getattr(evaluator, f'lv{i}_train_df')\
                            .iloc[:, -28*3:], i)
        val = create_viz_df(getattr(evaluator, f'lv{i}_valid_df'), i)
        pred = create_viz_df(getattr(evaluator, f'lv{i}_valid_preds'), i)

        n_cate = trn.shape[1] if i < 7 else 9

        fig, axs = plt.subplots(n_rows[i-1], n_cols[i-1],
                                figsize=(width[i-1],height[i-1]))
        if i > 1:
            axs = axs.flatten()

        ## Time series plot
        for k in range(0, n_cate):

            ax = axs[k] if i > 1 else axs

            trn.iloc[:, k].plot(ax=ax, label='train')
            val.iloc[:, k].plot(ax=ax, label='valid')
            pred.iloc[:, k].plot(ax=ax, label='pred')
            ax.set_title(f"{trn.columns[k]}  RMSSE:{scores[k]:.4f}", size=14)
            ax.set(xlabel='', ylabel='sales')
            ax.tick_params(labelsize=8)
            ax.legend(loc='upper left', prop={'size': 10})

        if i == 1 or i >= 9:
            fig.suptitle(f'Level {i}: {evaluator.group_ids[i-1]}', size=24 ,
                         y=1.1, fontweight='bold')
        plt.tight_layout()
        plt.show()
```

01 Introduction

Leaderboard

From M5 leaderboard, we can see that LightGBM is good choice



Public Leaderboard | Private Leaderboard

The private leaderboard is calculated with approximately 50% of the test data.
This competition has completed. This leaderboard reflects the final standings.

Refresh

In the money | Gold | Silver | Bronze

#	△pub	Team Name	Notebook	Team Members	Score	Entries	Last
1	▲ 15...	YeonJun IN_STU			0.52043	38	1y
2	▲ 652	Matthias	</> M5-AlignAndSubmit		0.52816	57	1y
3	▲ 894	mf			0.53571	9	1y
4	▲ 894	monsaraida			0.53583	15	1y
5	▲ 26...	Alan Lahoud			0.53604	49	1y
6	▲ 11...	wyzJack_STU			0.54433	80	1y
7	▲ 823	RandomLearner			0.54643	32	1y

LightGBM

DeepAR

No Solution

02

The Datasets

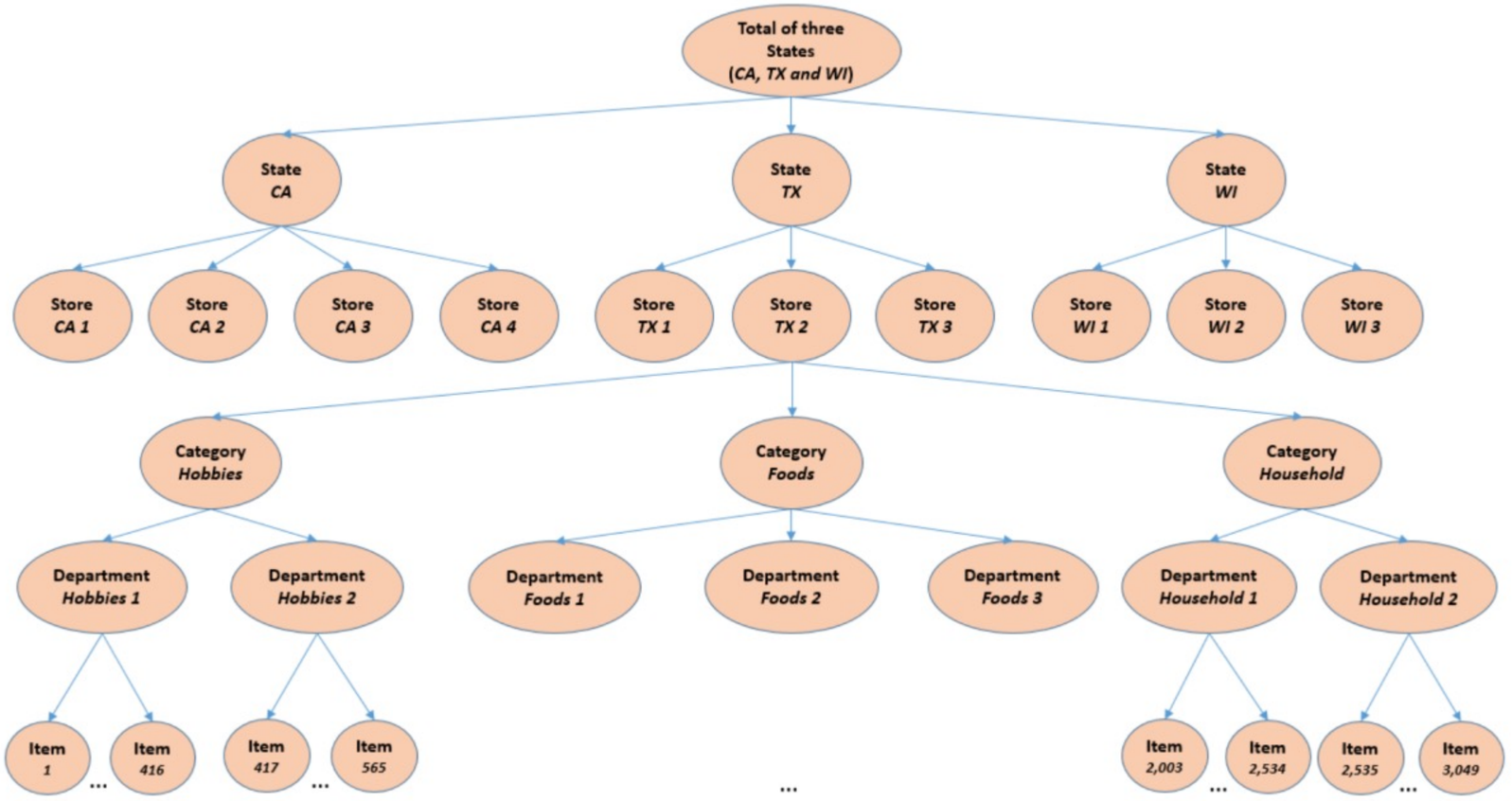
01. Overview
02. Data Files

02 The Datasets

01 Overview

02 Data Files

Data Hierarchy



02 The Datasets

01. Overview

02 Data Files

calendar.csv

- date - The date in a “y-m-d” format.
- wm_yr_wk - The id of the week the date belongs to.
- weekday - The type of the day (Saturday, Sunday, ..., Friday).
- wday - The id of the weekday, starting from Saturday.
- month - The month of the date.
- year - The year of the date.
- event_name_1 - If the date includes an event, the name of this event.
- event_type_1 - If the date includes an event, the type of this event.
- event_name_2 - If the date includes a second event, the name of this event.
- event_type_2 - If the date includes a second event, the type of this evenr.
- snap_CA, snap_TX, and snap_WI - A binary variable (0 or 1) indicating whether the stores of CA, TX or WI allow SNAP purchases on the examined date.

	date	wm_yr_wk	weekday	wday	month	year	d	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA	snap_TX	snap_WI
0	2011-01-29	11101	Saturday	1	1	2011	d_1	NaN	NaN	NaN	NaN	0	0	0
1	2011-01-30	11101	Sunday	2	1	2011	d_2	NaN	NaN	NaN	NaN	0	0	0
2	2011-01-31	11101	Monday	3	1	2011	d_3	NaN	NaN	NaN	NaN	0	0	0
3	2011-02-01	11101	Tuesday	4	2	2011	d_4	NaN	NaN	NaN	NaN	1	1	0
4	2011-02-02	11101	Wednesday	5	2	2011	d_5	NaN	NaN	NaN	NaN	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1964	2016-06-15	11620	Wednesday	5	6	2016	d_1965	NaN	NaN	NaN	NaN	0	1	1
1965	2016-06-16	11620	Thursday	6	6	2016	d_1966	NaN	NaN	NaN	NaN	0	0	0
1966	2016-06-17	11620	Friday	7	6	2016	d_1967	NaN	NaN	NaN	NaN	0	0	0
1967	2016-06-18	11621	Saturday	1	6	2016	d_1968	NaN	NaN	NaN	NaN	0	0	0
1968	2016-06-19	11621	Sunday	2	6	2016	d_1969	NBAFinalsEnd	Sporting	Father's day	Cultural	0	0	0

02 The Datasets

01. Overview

02 Data Files

sell_prices.csv

- store_id - The id of the store where the product is sold.
- item_id - The id of the product.
- wm_yr_wk - The id of the week.
- sell_price - The price of the product for the given week/store. The price is provided per week (average across seven days). If not available, this means that the product was not sold during the examined week.
  Note that although prices are constant at weekly basis, they may change through time (both training and test set).

	store_id	item_id	wm_yr_wk	sell_price
0	CA_1	HOBBIES_1_001	11325	9.58
1	CA_1	HOBBIES_1_001	11326	9.58
2	CA_1	HOBBIES_1_001	11327	8.26
3	CA_1	HOBBIES_1_001	11328	8.26
4	CA_1	HOBBIES_1_001	11329	8.26
...	...	...	...	...
6841116	WI_3	FOODS_3_827	11617	1.00
6841117	WI_3	FOODS_3_827	11618	1.00
6841118	WI_3	FOODS_3_827	11619	1.00
6841119	WI_3	FOODS_3_827	11620	1.00
6841120	WI_3	FOODS_3_827	11621	1.00

6841121 rows × 4 columns

02 The Datasets

01. Overview

02 Data Files

sales_train_validation.csv

- item_id - The id of the product.
- dept_id - The id of the department the product belongs to.
- cat_id - The id of the category the product belongs to.
- store_id - The id of the store where the product is sold.
- state_id - The State where the store is located.
- d_1, d_2, …, d_i, … d_1913 - The number of units sold at day i, starting from 2011-01-29.

	id	item_id	dept_id	cat_id	store_id	state_id	d_1	d_2	d_3	d_4	d_5	...	d_1908	d_1909	d_1910	d_1911	d_1912	d_1913
0	HOBBIES_1_001_CA_1_validation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	...	1	1	3	0	1	1
1	HOBBIES_1_002_CA_1_validation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	...	0	1	0	0	0	0
2	HOBBIES_1_003_CA_1_validation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	...	1	1	0	1	1	1
3	HOBBIES_1_004_CA_1_validation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	...	1	0	1	3	7	2
4	HOBBIES_1_005_CA_1_validation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	...	1	1	2	2	2	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
30485	FOODS_3_823_WI_3_validation	FOODS_3_823	FOODS_3	FOODS	WI_3	WI	0	0	2	2	0	...	0	0	1	0	0	1
30486	FOODS_3_824_WI_3_validation	FOODS_3_824	FOODS_3	FOODS	WI_3	WI	0	0	0	0	0	...	0	0	0	0	1	0
30487	FOODS_3_825_WI_3_validation	FOODS_3_825	FOODS_3	FOODS	WI_3	WI	0	6	0	2	2	...	0	1	0	0	1	0
30488	FOODS_3_826_WI_3_validation	FOODS_3_826	FOODS_3	FOODS	WI_3	WI	0	0	0	0	0	...	0	1	0	3	1	3
30489	FOODS_3_827_WI_3_validation	FOODS_3_827	FOODS_3	FOODS	WI_3	WI	0	0	0	0	0	...	0	0	0	0	0	0

02 The Datasets

01. Overview

02 Data Files

sales_train_evaluation.csv

- item_id - 제품의 ID.
- dept_id - 제품이 속한 부서의 ID (cat_id의 하위 카테고리).
- cat_id - 제품이 속한 카테고리의 ID.
- store_id - 제품이 판매되는 매장의 ID
- state_id - 매장이 속해있는 주(state)
- d_1, d_2, …, d_i, … d_1941 - The number of units sold at day i, starting from 2011-01-29. i일에 판매된 단위 수(2011-01-29일부터 시작).

> 📌 경기가 종료되기 한 달 전에 validation행의 값들이 제공되었습니다. 이로 미루어 볼 때 sales_train_validation.csv은 처음부터 공개되었고, 경기종료 한 달 전부터 sales_train_evaluation.csv를 공개했을것으로 추측됩니다.

	id	item_id	dept_id	cat_id	store_id	state_id	d_1	d_2	d_3	d_4	d_5	...	d_1936	d_1937	d_1938	d_1939	d_1940	d_1941
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	...	0	0	3	3	0	1
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	...	1	0	0	0	0	0
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	...	0	0	2	3	0	1
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	...	0	1	3	0	2	6
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	...	1	0	0	2	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
30485	FOODS_3_823_WI_3_evaluation	FOODS_3_823	FOODS_3	FOODS	WI_3	WI	0	0	2	2	0	...	1	1	0	0	1	1
30486	FOODS_3_824_WI_3_evaluation	FOODS_3_824	FOODS_3	FOODS	WI_3	WI	0	0	0	0	0	...	0	0	1	0	1	0
30487	FOODS_3_825_WI_3_evaluation	FOODS_3_825	FOODS_3	FOODS	WI_3	WI	0	6	0	2	2	...	0	1	0	1	0	2
30488	FOODS_3_826_WI_3_evaluation	FOODS_3_826	FOODS_3	FOODS	WI_3	WI	0	0	0	0	0	...	6	0	1	1	1	0
30489	FOODS_3_827_WI_3_evaluation	FOODS_3_827	FOODS_3	FOODS	WI_3	WI	0	0	0	0	0	...	4	0	2	2	5	1

02 The Datasets

01. Overview

02 Data Files

sample_submission.csv

- Predict F1-F28. 각 행에 대해 판매되는 품목의 예측일수 28일(F1-F28)을 예측
- validation행의 경우 d_1914 - d_1941에 해당, evaluation(test)행의 경우 d_1942 - d_1969에 해당

	id	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24	F25	F26	F27	F28
0	HOBBIES_1_001_CA_1_validation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	HOBBIES_1_002_CA_1_validation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	HOBBIES_1_003_CA_1_validation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	HOBBIES_1_004_CA_1_validation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	HOBBIES_1_005_CA_1_validation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
60975	FOODS_3_823_WI_3_evaluation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
60976	FOODS_3_824_WI_3_evaluation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
60977	FOODS_3_825_WI_3_evaluation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
60978	FOODS_3_826_WI_3_evaluation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
60979	FOODS_3_827_WI_3_evaluation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

03

Exploratory Data Analysis

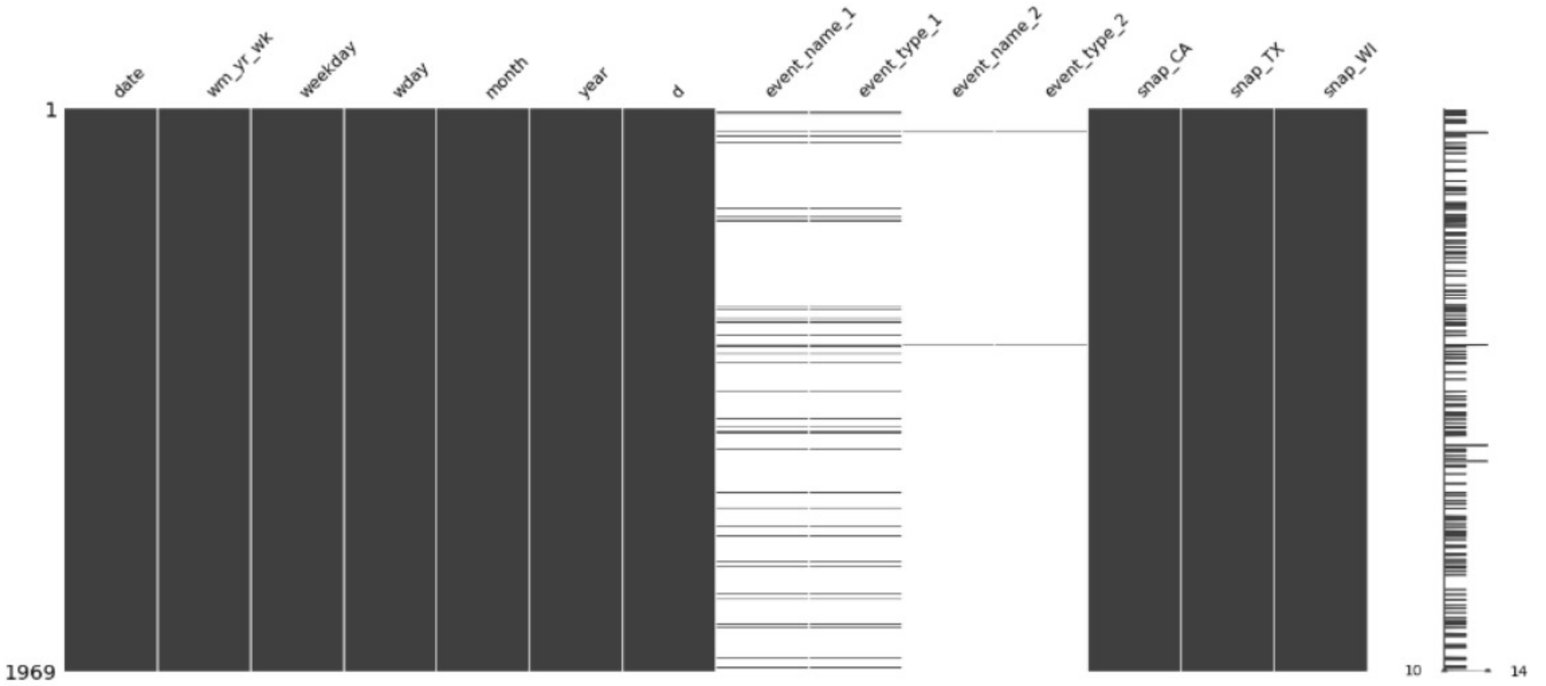
01. NaN Check
02. Special Days
03. Sales Analysis
04. Sample Sales Data

03 EDA

Visualize NaN

Using `isnull( )` -> In three csv files, there is no NaN. But there are plenty of `NaN` in `calendar.csv`

```
missingno.matrix(calendar);
```



03 EDA

01 NaN Check

02 Special Days

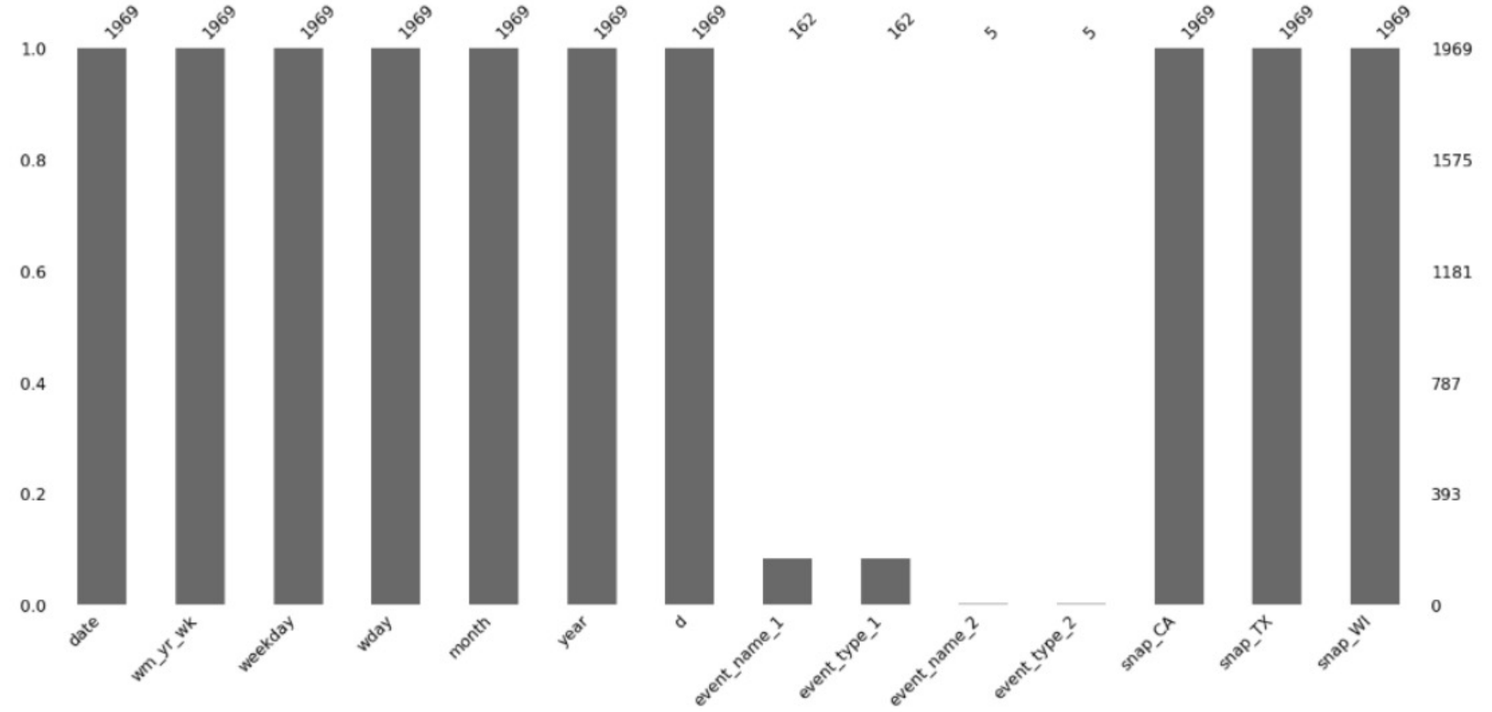
03. Sales Analysis

04. Sample Sales Data

Visualize NaN

Using `isnull( )` -> In three csv files, there is no NaN. But there are plenty of `NaN` in `calendar.csv`

```
missingno.bar(calendar);
```



03 EDA

01 NaN Check

02 Special Days

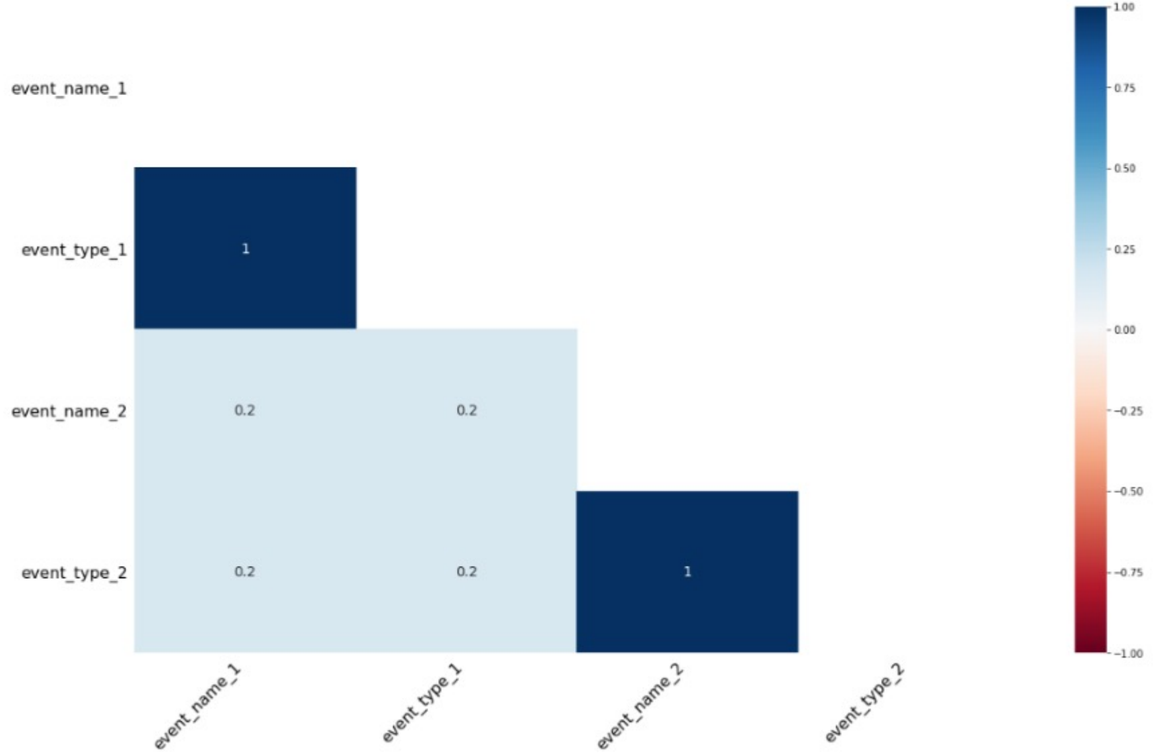
03. Sales Analysis

04. Sample Sales Data

Visualize NaN

Using `isnull( )` -> In three csv files, there is no NaN. But there are plenty of `NaN` in `calendar.csv`

```
missingno.heatmap(calendar);
```



03 EDA

01. NaN Check

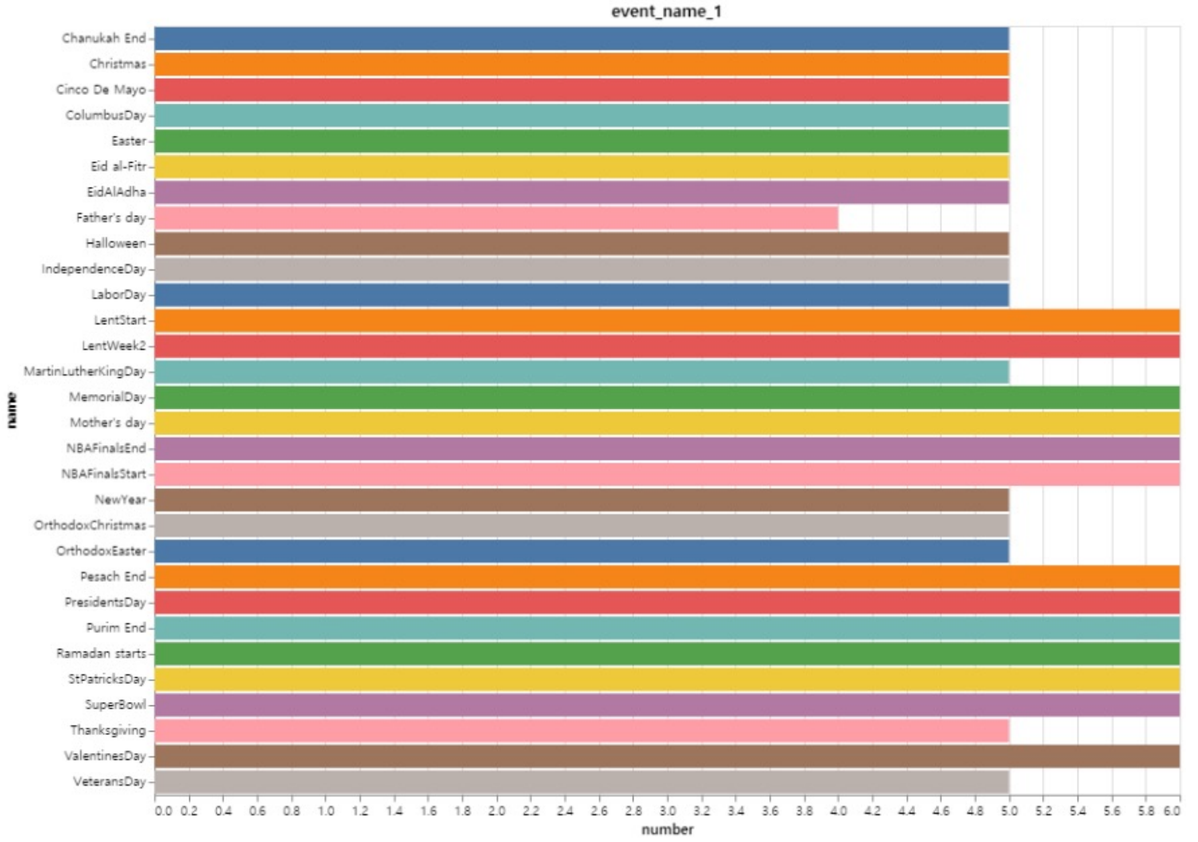
02 Special Days

03. Sales Analysis

04. Sample Sales Data

Event Days

Check the number and name of each event by visualizing 'event_name_1'



03 EDA

01. NaN Check

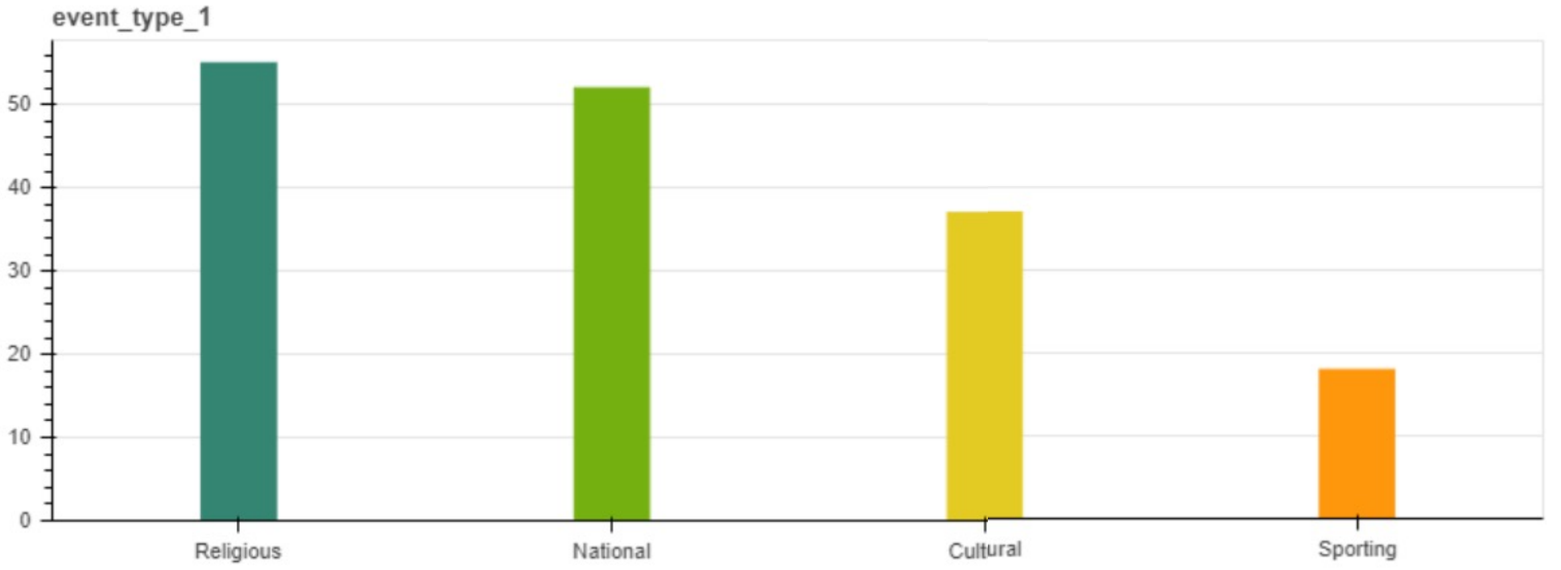
02 Special Days

03. Sales Analysis

04. Sample Sales Data

Event Days

Visualize 'event_type_1' - This shows that there are relatively many religious days in the United States



03 EDA

01. NaN Check

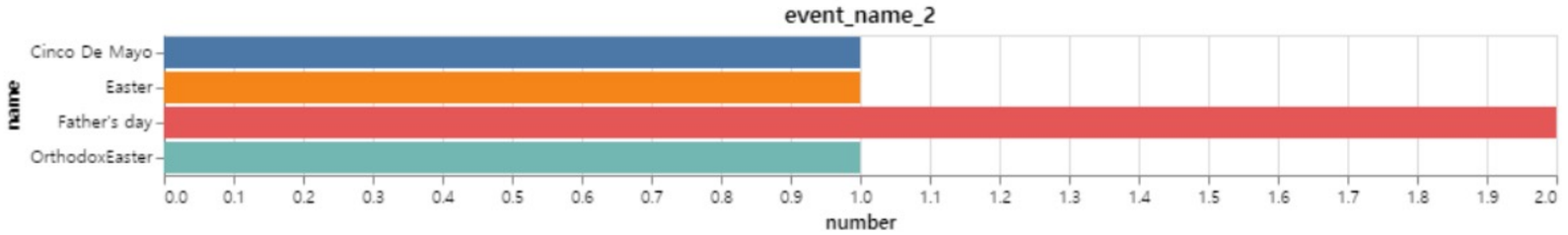
02 Special Days

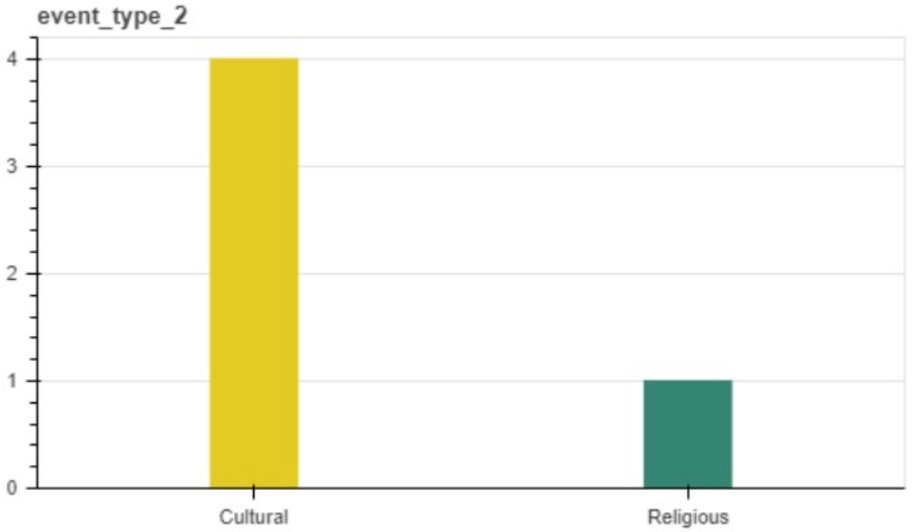
03. Sales Analysis

04. Sample Sales Data

Event Days

Visualize 'event_name_2' and 'event_type_2'





03 EDA

01. NaN Check

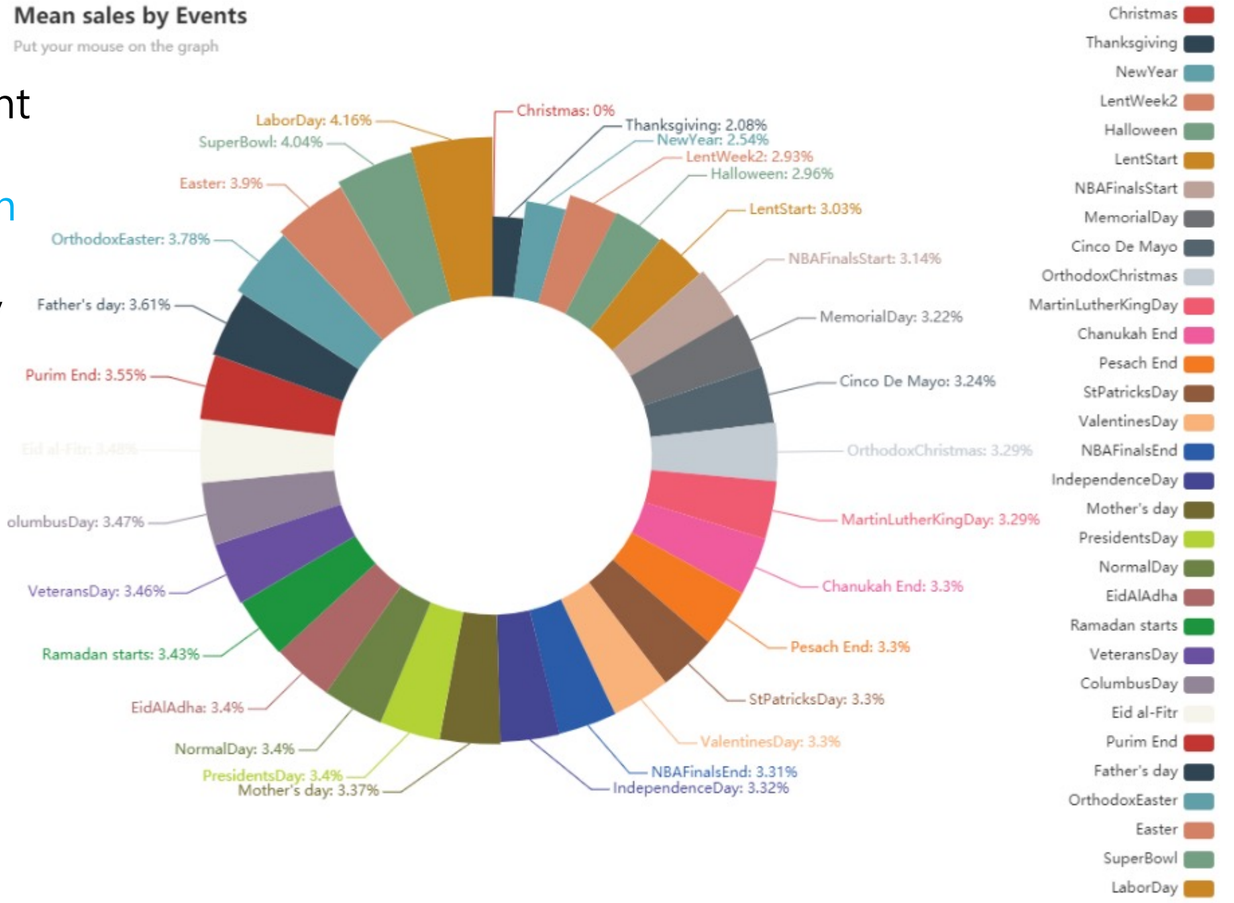
02 Special Days

03. Sales Analysis

04. Sample Sales Data

Event Days

Analyze the sales by event by averaging the total sales of all marts for each event. And compare the average sales of ordinary days.



03 EDA

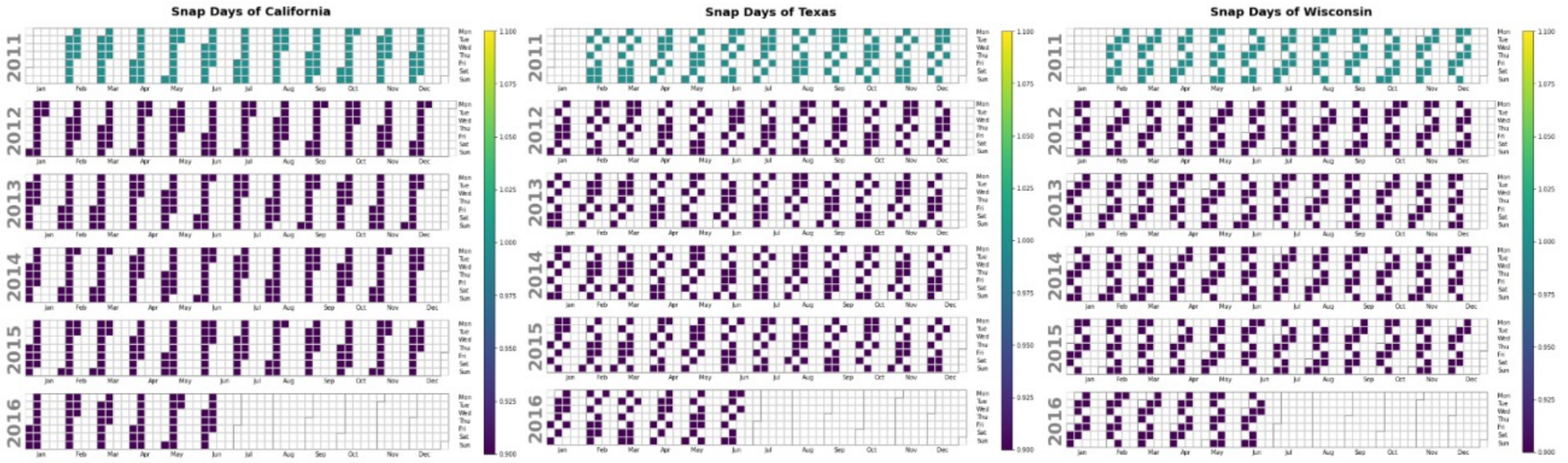
01. NaN Check

02 Special Days

03. Sales Analysis

04. Sample Sales Data

Snap Days



We can see that there are no fixed dates for SNAP every month, SNAP days occur on different dates every month. I also noticed that SnapDay had different dates for each week.

03 EDA

Snap Day Sales Ratio for each categories

California

Hobbies Household Food

In snap days, there's different ratio for each category.
Food category has high sales ratio.

03 EDA

01. NaN Check

02 Special Days

03. Sales Analysis

04. Sample Sales Data

Snap Day Sales Ratio for each categories

Texas

Hobbies

Household

Food

There's slight difference ratio in each states.

03 EDA

01. NaN Check

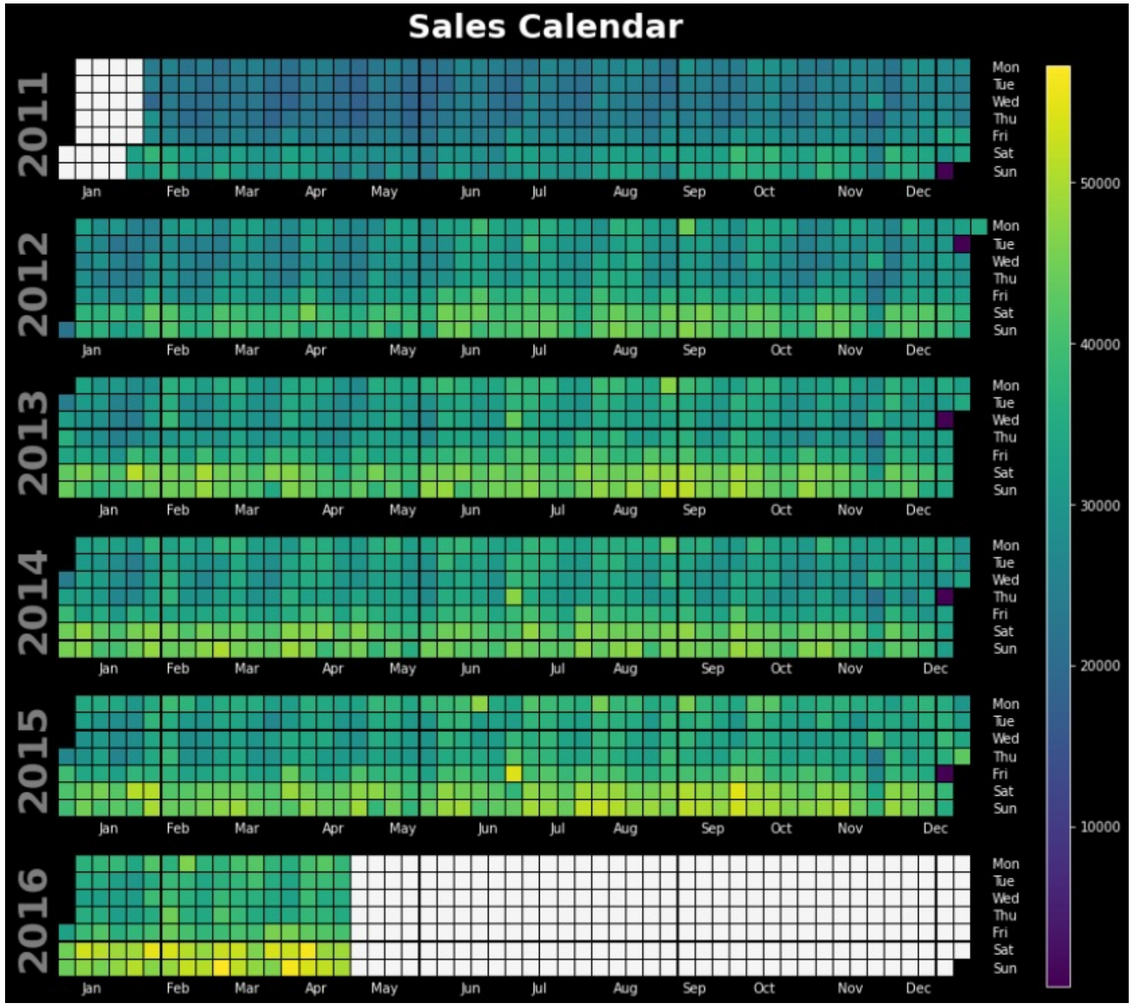
02 Special Days

03 Sales Analysis

04. Sample Sales Data

Calendar Plot

Walmart is apparently closed in Christmas. Overall sales are relatively **high on weekends** in comparison weekdays.



03 EDA

01. NaN Check

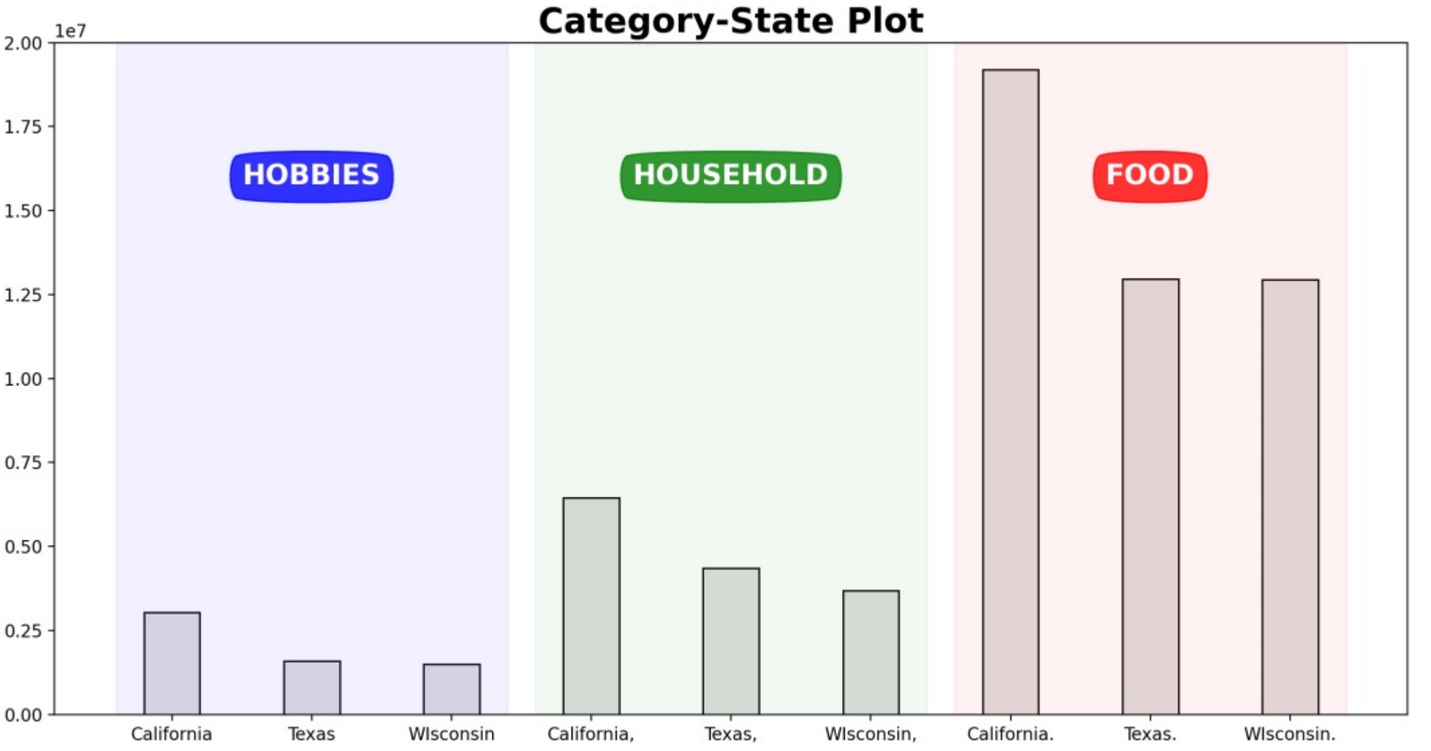
02 Special Days

03 Sales Analysis

04. Sample Sales Data

Category Plot

Visualized it to see how many products in each category are sold every three states. There was no noticeable difference in Texas except for a slightly higher percentage of home-related merchandise sales



03 EDA

01. NaN Check

02 Special Days

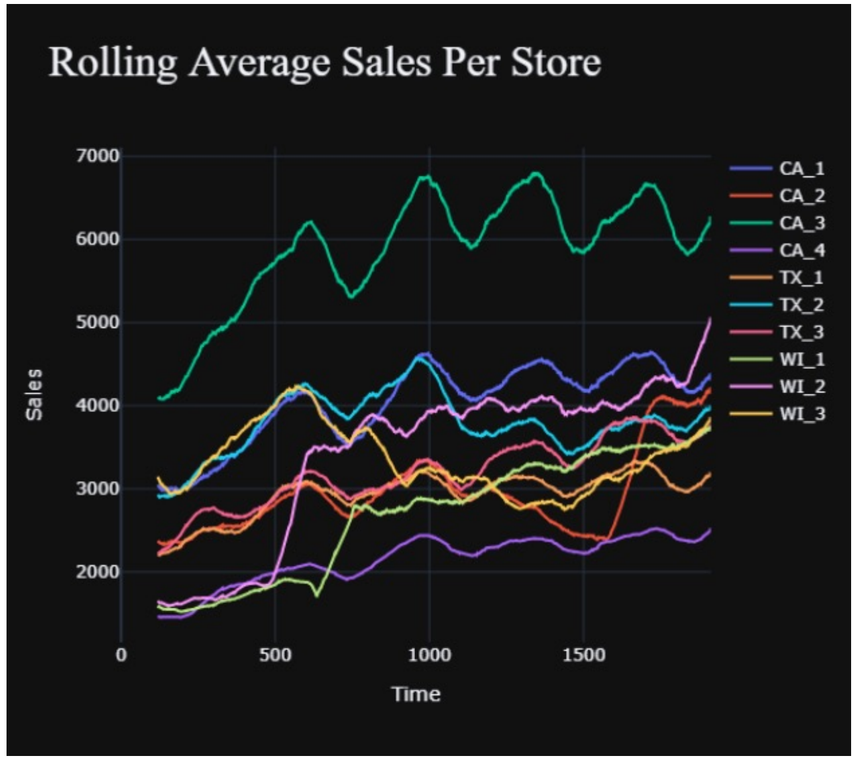
03 Sales Analysis

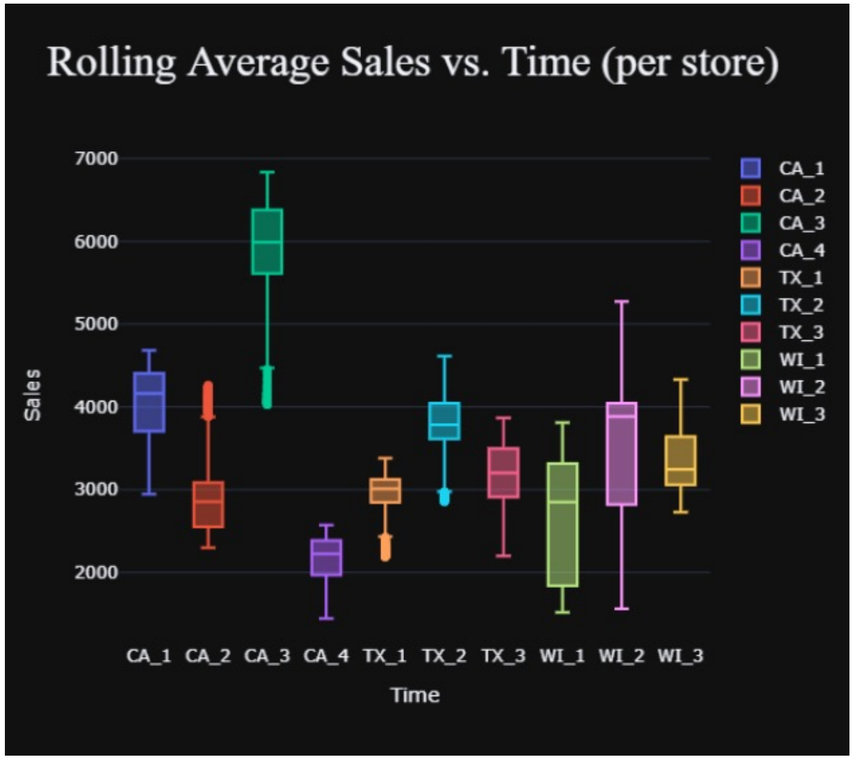
04. Sample Sales Data

Sales Plot

Compared the rolling sales for each store in our dataset. California's 3rd store seems to have the highest mean sales. Wisconsin's first store grew significantly faster from d_500 to d_600 and second store as well from d_600 to d_700. I guess there was big promotion events sequentially in Wisconsin.

Compared the sales distribution for each store in the dataset. CA_1 and CA_4 data are left-skewed. CA_2, WI_3 data are right-skewed. Except the first store of Wisconsin, every store's sales data are not concentrated





03 EDA

01. NaN Check

02 Special Days

03. Sales Analysis

04 Sample Sales Data

Sampling and Visualizing to check sales

Below plots are very fluctuating, due to the fact that plenty of factors affect the sales on a given day. Even some range of days, the sales quantity is zero. I guess there's out of stock or may not be available due to some reasons

04

Feature Engineering

01. Feature Generation

02. Data Memory Reduction

04 Feature Engineering

01 Feature Generation

02 Data Memory Reduction

Road Files

Load csv files for Feature Engineering. This time, to save memory,
I have loaded it according to the certain type.

	id	item_id	dept_id	cat_id	store_id	state_id	d_1	d_2	d_3	d_4	...	d_1938	d_1939	d_1940	d_1941
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	3	3	0	1
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	0	0	0	0
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	2	3	0	1
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	3	0	2	6
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	0	2	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
30485	FOODS_3_823_WI_3_evaluation	FOODS_3_823	FOODS_3	FOODS	WI_3	WI	0	0	2	2	...	0	0	1	1
30486	FOODS_3_824_WI_3_evaluation	FOODS_3_824	FOODS_3	FOODS	WI_3	WI	0	0	0	0	...	1	0	1	0
30487	FOODS_3_825_WI_3_evaluation	FOODS_3_825	FOODS_3	FOODS	WI_3	WI	0	6	0	2	...	0	1	0	2
30488	FOODS_3_826_WI_3_evaluation	FOODS_3_826	FOODS_3	FOODS	WI_3	WI	0	0	0	0	...	1	1	1	0
30489	FOODS_3_827_WI_3_evaluation	FOODS_3_827	FOODS_3	FOODS	WI_3	WI	0	0	0	0	...	2	2	5	1

04 Feature Engineering

01 Feature Generation

02 Data Memory Reduction

Melting

Melted the date column into one column

ID	d_1	d_2	....	d_1940	d_1941
HOBBIES_1_001	5		....		
HOBBIES_1_002	4		....		
HOBBIES_1_003	2		....		
HOBBIES_1_004	6		....		2
HOBBIES_1_005			....		17
HOBBIES_1_006			....		12
HOBBIES_1_007			....		16

ID	d	sales
HOBBIES_1_001	d_1	5
HOBBIES_1_002	d_1	4
HOBBIES_1_003	d_1	2
HOBBIES_1_004	d_1	6
HOBBIES_1_005	d_1	
....	.....	....
HOBBIES_1_003	d_1941	
HOBBIES_1_004	d_1941	2
HOBBIES_1_005	d_1941	17
HOBBIES_1_006	d_1941	12
HOBBIES_1_007	d_1941	16

	id	item_id	dept_id	store_id	cat_id	state_id	d	sales
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	CA_1	HOBBIES	CA	d_1	0
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	CA_1	HOBBIES	CA	d_1	0
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	CA_1	HOBBIES	CA	d_1	0
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	CA_1	HOBBIES	CA	d_1	0
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	CA_1	HOBBIES	CA	d_1	0
...	...	...	...	...	...	...	...	...
59181085	FOODS_3_823_WI_3_evaluation	FOODS_3_823	FOODS_3	WI_3	FOODS	WI	d_1941	1
59181086	FOODS_3_824_WI_3_evaluation	FOODS_3_824	FOODS_3	WI_3	FOODS	WI	d_1941	0
59181087	FOODS_3_825_WI_3_evaluation	FOODS_3_825	FOODS_3	WI_3	FOODS	WI	d_1941	2
59181088	FOODS_3_826_WI_3_evaluation	FOODS_3_826	FOODS_3	WI_3	FOODS	WI	d_1941	0
59181089	FOODS_3_827_WI_3_evaluation	FOODS_3_827	FOODS_3	WI_3	FOODS	WI	d_1941	1

59181090 rows × 8 columns

04 Feature Engineering

01 Feature Generation

02 Data Memory Reduction

Calendar

Created one feature by averaging the sales of each event day.

	date	wm_yr_wk	weekday	wday	month	year	d	snap_CA	snap_TX	snap_WI	event
0	2011-01-29	11101	Saturday	1	1	2011	d_1	0	0	0	34609
1	2011-01-30	11101	Sunday	2	1	2011	d_2	0	0	0	34609
2	2011-01-31	11101	Monday	3	1	2011	d_3	0	0	0	34609
3	2011-02-01	11101	Tuesday	4	2	2011	d_4	1	1	0	34609
4	2011-02-02	11101	Wednesday	5	2	2011	d_5	1	0	1	34609
...	...	...	...	...	...	...	...	...	...	...	...
1964	2016-06-15	11620	Wednesday	5	6	2016	d_1965	0	1	1	34609
1965	2016-06-16	11620	Thursday	6	6	2016	d_1966	0	0	0	34609
1966	2016-06-17	11620	Friday	7	6	2016	d_1967	0	0	0	34609
1967	2016-06-18	11621	Saturday	1	6	2016	d_1968	0	0	0	34609
1968	2016-06-19	11621	Sunday	2	6	2016	d_1969	0	0	0	35063

⇦

```
'Christmas': 15.6,
'Thanksgiving': 21130.0,
'NewYear': 25768.4,
'LentWeek2': 29747.0,
'Halloween': 30059.4,
'LentStart': 30727.0,
'NBAFinalsStart': 31874.4,
'MemorialDay': 32699.6,
'OrthodoxChristmas': 33344.2,
'MartinLutherKingDay': 33378.6,
'Chanukah End': 33426.8,
'StPatricksDay': 33459.666666666664,
'ValentinesDay': 33495.666666666664,
'NBAFinalsEnd': 33564.2,
'IndependenceDay': 33710.8,
'Cinco De Mayo': 34411.4,
'PresidentsDay': 34446.333333333336,
'EidAlAdha': 34504.0,
'NormalDay': 34609.33651149747,
'Ramadan starts': 34779.6,
'VeteransDay': 35118.8,
'ColumbusDay': 35164.8,
'Eid al-Fitr': 35267.6,
'Purim End': 35957.0,
"Mother's day": 36111.0,
'Pesach End': 36280.0,
"Father's day": 36562.0,
'Easter': 39517.6,
'SuperBowl': 40924.0,
'OrthodoxEaster': 41228.0,
'LaborDay': 42154.6}
```

04 Feature Engineering

01 Feature Generation

02 Data Memory Reduction

Calendar

To save model training time by reducing the number of features, we reduced the snap day columns to one column

snap_CA	snap_TX	snap_WI
0	0	0
0	0	0
0	0	0
1	1	0
1	0	1
...	...	...
0	1	1
0	0	0
0	0	0
0	0	0
0	0	0

⇨

snap
0
0
0
0
0
...
0
0
0
0
0

	id	item_id	dept_id	store_id	cat_id	state_id	d	sales	date	wm_yr_wk	weekday	wday	month	year	event	snap
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	CA_1	HOBBIES	CA	d_1	0	2011-01-29	11101	Saturday	1	1	2011	34609	0
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	CA_1	HOBBIES	CA	d_1	0	2011-01-29	11101	Saturday	1	1	2011	34609	0
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	CA_1	HOBBIES	CA	d_1	0	2011-01-29	11101	Saturday	1	1	2011	34609	0
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	CA_1	HOBBIES	CA	d_1	0	2011-01-29	11101	Saturday	1	1	2011	34609	0
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	CA_1	HOBBIES	CA	d_1	0	2011-01-29	11101	Saturday	1	1	2011	34609	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59181085	FOODS_3_823_WI_3_evaluation	FOODS_3_823	FOODS_3	WI_3	FOODS	WI	d_1941	1	2016-05-22	11617	Sunday	2	5	2016	34609	0
59181086	FOODS_3_824_WI_3_evaluation	FOODS_3_824	FOODS_3	WI_3	FOODS	WI	d_1941	0	2016-05-22	11617	Sunday	2	5	2016	34609	0
59181087	FOODS_3_825_WI_3_evaluation	FOODS_3_825	FOODS_3	WI_3	FOODS	WI	d_1941	2	2016-05-22	11617	Sunday	2	5	2016	34609	0
59181088	FOODS_3_826_WI_3_evaluation	FOODS_3_826	FOODS_3	WI_3	FOODS	WI	d_1941	0	2016-05-22	11617	Sunday	2	5	2016	34609	0
59181089	FOODS_3_827_WI_3_evaluation	FOODS_3_827	FOODS_3	WI_3	FOODS	WI	d_1941	1	2016-05-22	11617	Sunday	2	5	2016	34609	0

59181090 rows × 16 columns

04 Feature Engineering

01 Feature Generation

02 Data Memory Reduction

Sell Prices

Created a new column by adjusting the moving average of the price differences of the product by window size.

```
for win in [1, 2, 4, 8]:
    prices['rm_diff_price_{}'.format(win)] = prices[["store_id","item_id","sell_price"]].groupby(
        ["store_id","item_id"])["sell_price"].transform(lambda x : x.rolling(win).mean())
    prices['rm_diff_price_{}'.format(win)] = ((prices['sell_price'] - prices['rm_diff_price_{}'.format(win)]
                                              )/prices['sell_price']).round(3)
prices
```

	store_id	item_id	wm_yr_wk	sell_price	rm_diff_price_1	rm_diff_price_2	rm_diff_price_4	rm_diff_price_8
0	CA_1	HOBBIES_1_001	11325	9.58	0.0	NaN	NaN	NaN
1	CA_1	HOBBIES_1_001	11326	9.58	0.0	0.00	NaN	NaN
2	CA_1	HOBBIES_1_001	11327	8.26	0.0	-0.08	NaN	NaN
3	CA_1	HOBBIES_1_001	11328	8.26	0.0	0.00	-0.08	NaN
4	CA_1	HOBBIES_1_001	11329	8.26	0.0	0.00	-0.04	NaN
...	...	...	...	...	...	...	...	...
6841116	WI_3	FOODS_3_827	11617	1.00	0.0	0.00	0.00	0.0
6841117	WI_3	FOODS_3_827	11618	1.00	0.0	0.00	0.00	0.0
6841118	WI_3	FOODS_3_827	11619	1.00	0.0	0.00	0.00	0.0
6841119	WI_3	FOODS_3_827	11620	1.00	0.0	0.00	0.00	0.0
6841120	WI_3	FOODS_3_827	11621	1.00	0.0	0.00	0.00	0.0

6841121 rows × 8 columns

04 Feature Engineering

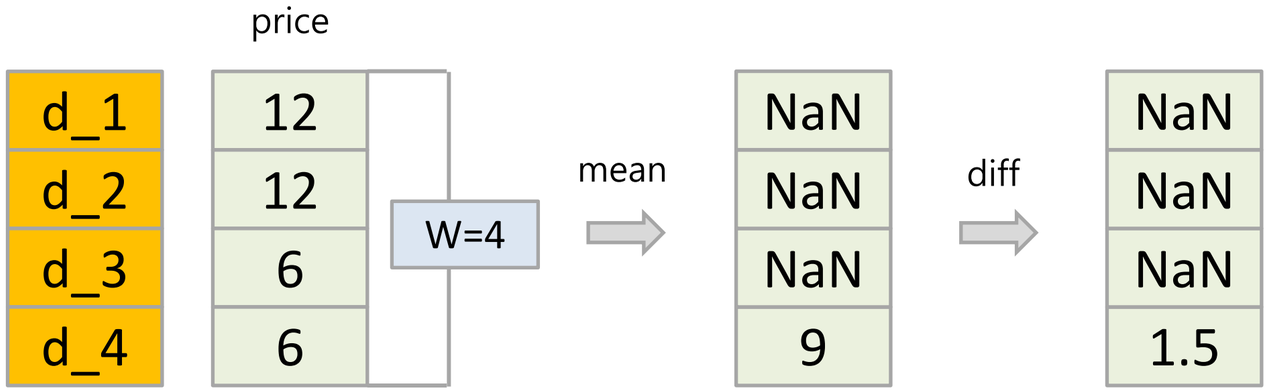
01 Feature Generation

02 Data Memory Reduction

02 Test Set Generation

03. Data Memory Reducing

Sell Prices



04 Feature Engineering

01 Feature Generation

02 Data Memory Reduction

Sell Prices

Created a new column by adjusting the moving average of the price differences of the product by window size.

	id	item_id	dept_id	wday	month	year	event	snap	rm_diff_price_1	rm_diff_price_2	rm_diff_price_4	rm_diff_price_8
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	1	1	2011	34609	0	0.0	NaN	NaN	NaN
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	1	1	2011	34609	0	0.0	NaN	NaN	NaN
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	1	1	2011	34609	0	0.0	NaN	NaN	NaN
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	1	1	2011	34609	0	0.0	NaN	NaN	NaN
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	1	1	2011	34609	0	0.0	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
59181085	FOODS_3_823_WI_3_evaluation	FOODS_3_823	FOODS_3	2	5	2016	34609	0	0.0	0.0	0.0	0.0
59181086	FOODS_3_824_WI_3_evaluation	FOODS_3_824	FOODS_3	2	5	2016	34609	0	0.0	0.0	0.0	0.0
59181087	FOODS_3_825_WI_3_evaluation	FOODS_3_825	FOODS_3	2	5	2016	34609	0	0.0	0.0	0.0	0.0
59181088	FOODS_3_826_WI_3_evaluation	FOODS_3_826	FOODS_3	2	5	2016	34609	0	0.0	0.0	0.0	0.0
59181089	FOODS_3_827_WI_3_evaluation	FOODS_3_827	FOODS_3	2	5	2016	34609	0	0.0	0.0	0.0	0.0

46881677 rows × 20 columns

Sales

Created a new column by adjusting the moving average of the sales of the product by window size.

```python
def lags_wins(df):
    lags = [7, 14, 28]
    lag_cols = [f"lag_{lag}" for lag in lags]
    for lag, lag_col in zip(lags, lag_cols):
        df[lag_col] = df[["id","sales"]].groupby("id")["sales"].shift(lag)

    wins = [7, 14, 28]
    for win in wins :
        for lag,lag_col in zip(lags, lag_cols):
            df[f"rmean_{lag}_{win}"] = df[["id", lag_col]].groupby("id")[lag_col].transform(lambda x : x.rolling(win).mean())
    return df
```

...	rm_diff_price_2	rm_diff_price_4	rm_diff_price_8	lag_7	lag_14	lag_28	rmean_7_7	rmean_14_7	rmean_28_7	rmean_7_14	rmean_14_14	rmean_28_14	rmean_7_28	rmean_14_28	rmean_28_28
...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	0.0	0.0	0.0	1.0	1.0	0.0	1.142857	0.428571	0.571429	0.785714	0.571429	0.785714	0.714286	0.678571	0.892857
...	0.0	0.0	0.0	1.0	2.0	1.0	1.000000	1.142857	0.857143	1.071429	1.357143	0.857143	1.142857	1.107143	0.964286
...	0.0	0.0	0.0	1.0	1.0	3.0	1.000000	1.285714	1.142857	1.142857	1.214286	0.928571	1.142857	1.071429	0.928571
...	0.0	0.0	0.0	2.0	2.0	0.0	0.714286	1.428571	0.000000	1.071429	0.785714	0.000000	0.571429	0.392857	1.250000
...	0.0	0.0	0.0	0.0	1.0	0.0	0.571429	1.428571	0.000000	1.000000	0.857143	0.000000	0.571429	0.428571	1.071429

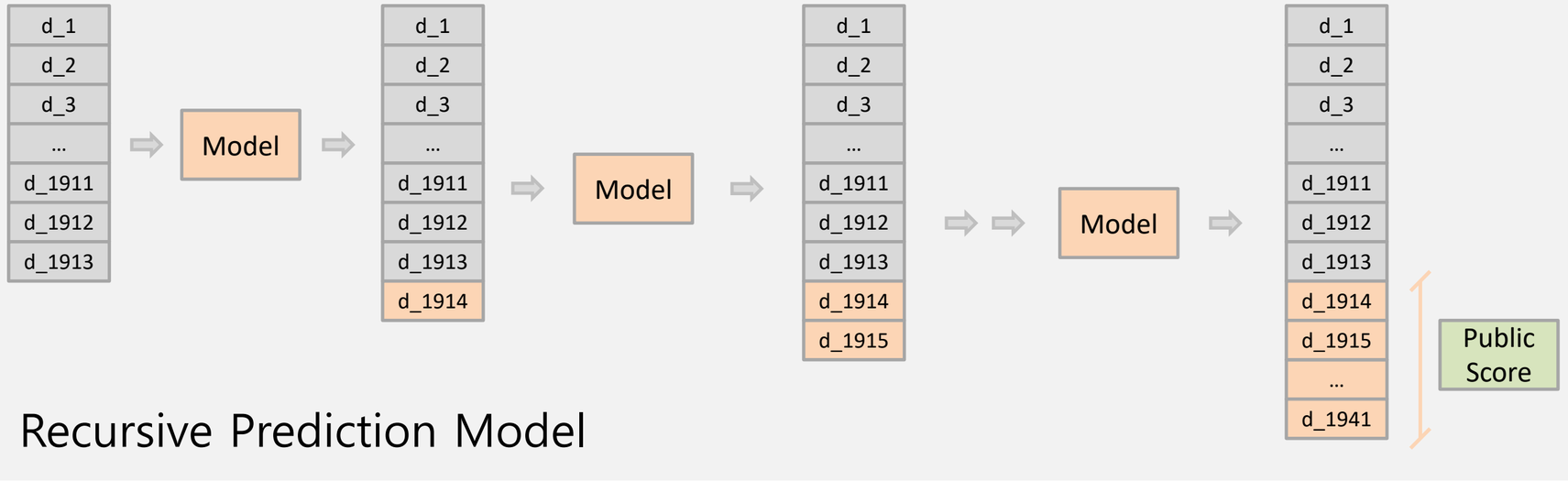
04 Feature Engineering

01 Feature Generation

02 Data Memory Reduction

Valid set generation

	id	item_id	dept_id	store_id	cat_id	state_id	d	sales	date	wm_yr_wk	weekday	wday	month	year	event	snap	sell_price	rm_diff_price_1	rm_diff_price_2	rm_diff_price_4	rm_diff_price_8
0	HOBBIES_1_008_CA_1_evaluation	HOBBIES_1_008	HOBBIES_1	CA_1	HOBBIES	CA	d_1	12.0	2011-01-29	11101	Saturday	1	1	2011	34609	0	0.46	0.0	NaN	NaN	NaN
1	HOBBIES_1_008_CA_1_evaluation	HOBBIES_1_008	HOBBIES_1	CA_1	HOBBIES	CA	d_2	15.0	2011-01-30	11101	Sunday	2	1	2011	34609	0	0.46	0.0	NaN	NaN	NaN
2	HOBBIES_1_008_CA_1_evaluation	HOBBIES_1_008	HOBBIES_1	CA_1	HOBBIES	CA	d_3	0.0	2011-01-31	11101	Monday	3	1	2011	34609	0	0.46	0.0	NaN	NaN	NaN
3	HOBBIES_1_008_CA_1_evaluation	HOBBIES_1_008	HOBBIES_1	CA_1	HOBBIES	CA	d_4	0.0	2011-02-01	11101	Tuesday	4	2	2011	34609	1	0.46	0.0	NaN	NaN	NaN
4	HOBBIES_1_008_CA_1_evaluation	HOBBIES_1_008	HOBBIES_1	CA_1	HOBBIES	CA	d_5	0.0	2011-02-02	11101	Wednesday	5	2	2011	34609	1	0.46	0.0	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
47735392	FOODS_3_825_WI_3_evaluation	FOODS_3_825	FOODS_3	WI_3	FOODS	WI	d_1969	NaN	2016-06-19	11621	Sunday	2	6	2016	35063	0	3.98	0.0	0.0	0.0	0.0
47735393	FOODS_3_826_WI_3_evaluation	FOODS_3_826	FOODS_3	WI_3	FOODS	WI	d_1968	NaN	2016-06-18	11621	Saturday	1	6	2016	34609	0	1.28	0.0	0.0	0.0	0.0
47735394	FOODS_3_826_WI_3_evaluation	FOODS_3_826	FOODS_3	WI_3	FOODS	WI	d_1969	NaN	2016-06-19	11621	Sunday	2	6	2016	35063	0	1.28	0.0	0.0	0.0	0.0
47735395	FOODS_3_827_WI_3_evaluation	FOODS_3_827	FOODS_3	WI_3	FOODS	WI	d_1968	NaN	2016-06-18	11621	Saturday	1	6	2016	34609	0	1.00	0.0	0.0	0.0	0.0
47735396	FOODS_3_827_WI_3_evaluation	FOODS_3_827	FOODS_3	WI_3	FOODS	WI	d_1969	NaN	2016-06-19	11621	Sunday	2	6	2016	35063	0	1.00	0.0	0.0	0.0	0.0



Recursive Prediction Model

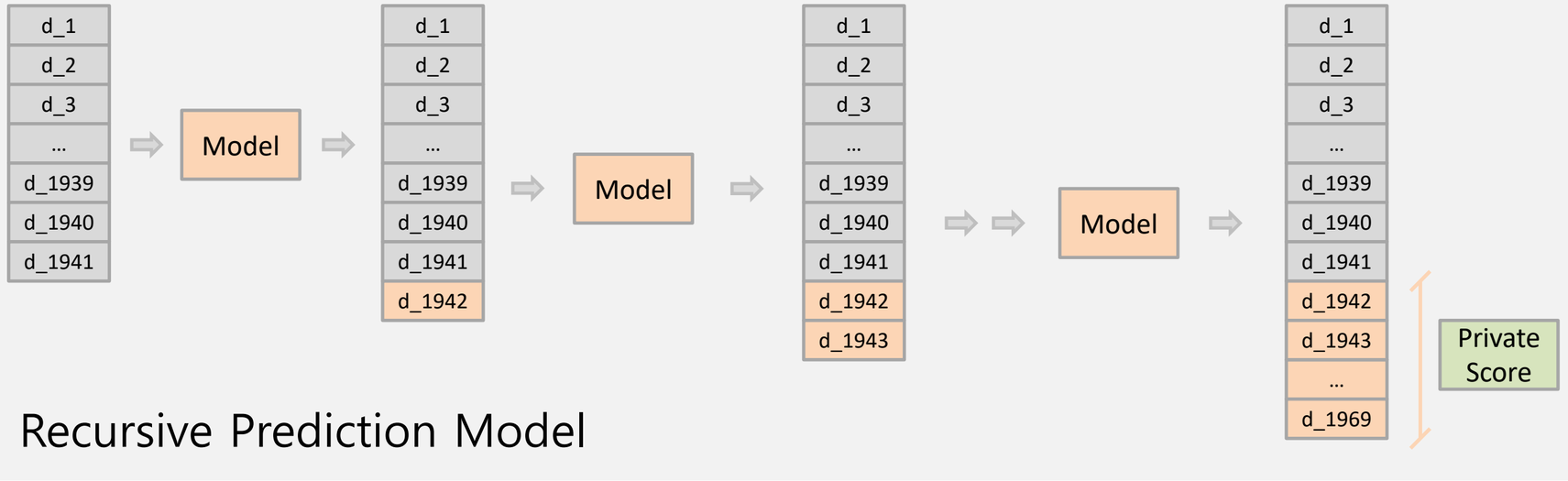
04 Feature Engineering

01 Feature Generation

02 Data Memory Reduction

Test set generation

	id	item_id	dept_id	store_id	cat_id	state_id	d	sales	date	wm_yr_wk	weekday	wday	month	year	event	snap	sell_price	rm_diff_price_1	rm_diff_price_2	rm_diff_price_4	rm_diff_price_8
0	HOBBIES_1_008_CA_1_evaluation	HOBBIES_1_008	HOBBIES_1	CA_1	HOBBIES	CA	d_1	12.0	2011-01-29	11101	Saturday	1	1	2011	34609	0	0.46	0.0	NaN	NaN	NaN
1	HOBBIES_1_008_CA_1_evaluation	HOBBIES_1_008	HOBBIES_1	CA_1	HOBBIES	CA	d_2	15.0	2011-01-30	11101	Sunday	2	1	2011	34609	0	0.46	0.0	NaN	NaN	NaN
2	HOBBIES_1_008_CA_1_evaluation	HOBBIES_1_008	HOBBIES_1	CA_1	HOBBIES	CA	d_3	0.0	2011-01-31	11101	Monday	3	1	2011	34609	0	0.46	0.0	NaN	NaN	NaN
3	HOBBIES_1_008_CA_1_evaluation	HOBBIES_1_008	HOBBIES_1	CA_1	HOBBIES	CA	d_4	0.0	2011-02-01	11101	Tuesday	4	2	2011	34609	1	0.46	0.0	NaN	NaN	NaN
4	HOBBIES_1_008_CA_1_evaluation	HOBBIES_1_008	HOBBIES_1	CA_1	HOBBIES	CA	d_5	0.0	2011-02-02	11101	Wednesday	5	2	2011	34609	1	0.46	0.0	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
47735392	FOODS_3_825_WI_3_evaluation	FOODS_3_825	FOODS_3	WI_3	FOODS	WI	d_1969	NaN	2016-06-19	11621	Sunday	2	6	2016	35063	0	3.98	0.0	0.0	0.0	0.0
47735393	FOODS_3_826_WI_3_evaluation	FOODS_3_826	FOODS_3	WI_3	FOODS	WI	d_1968	NaN	2016-06-18	11621	Saturday	1	6	2016	34609	0	1.28	0.0	0.0	0.0	0.0
47735394	FOODS_3_826_WI_3_evaluation	FOODS_3_826	FOODS_3	WI_3	FOODS	WI	d_1969	NaN	2016-06-19	11621	Sunday	2	6	2016	35063	0	1.28	0.0	0.0	0.0	0.0
47735395	FOODS_3_827_WI_3_evaluation	FOODS_3_827	FOODS_3	WI_3	FOODS	WI	d_1968	NaN	2016-06-18	11621	Saturday	1	6	2016	34609	0	1.00	0.0	0.0	0.0	0.0
47735396	FOODS_3_827_WI_3_evaluation	FOODS_3_827	FOODS_3	WI_3	FOODS	WI	d_1969	NaN	2016-06-19	11621	Sunday	2	6	2016	35063	0	1.00	0.0	0.0	0.0	0.0



Recursive Prediction Model

04 Feature Engineering

01 Feature Generation

02 Data Memory Reduction

Memory reducing

By changing the type to use fewer bytes through the astype() of numpy, I can save memory. The less bytes I use, the less range I can express, so I reduced the memory by checking the minimum and maximum values of the data I are dealing with.

```
for dtype in ['float16', 'float32', 'float64']:
    print(np.finfo(dtype))
```

```
Machine parameters for float16
---------------------------------------------------------------
precision =   3   resolution = 1.00040e-03
machep =    -10   eps =        9.76562e-04
negep =     -11   epsneg =     4.88281e-04
minexp =    -14   tiny =       6.10352e-05
maxexp =     16   max =        6.55040e+04
nexp =        5   min =        -max
---------------------------------------------------------------

Machine parameters for float32
---------------------------------------------------------------
precision =   6   resolution = 1.0000000e-06
machep =    -23   eps =        1.1920929e-07
negep =     -24   epsneg =     5.9604645e-08
minexp =   -126   tiny =       1.1754944e-38
maxexp =    128   max =        3.4028235e+38
nexp =        8   min =        -max
---------------------------------------------------------------

Machine parameters for float64
---------------------------------------------------------------
precision =  15   resolution = 1.0000000000000001e-15
machep =    -52   eps =        2.2204460492503131e-16
negep =     -53   epsneg =     1.1102230246251565e-16
minexp =  -1022   tiny =       2.2250738585072014e-308
maxexp =   1024   max =        1.7976931348623157e+308
nexp =       11   min =        -max
---------------------------------------------------------------
```

```
for dtype in ['int8', 'int16', 'int32', 'int64']:
    print(np.iinfo(dtype))
```

```
Machine parameters for int8
---------------------------------------------------------------
min = -128
max = 127
---------------------------------------------------------------

Machine parameters for int16
---------------------------------------------------------------
min = -32768
max = 32767
---------------------------------------------------------------

Machine parameters for int32
---------------------------------------------------------------
min = -2147483648
max = 2147483647
---------------------------------------------------------------

Machine parameters for int64
---------------------------------------------------------------
min = -9223372036854775808
max = 9223372036854775807
---------------------------------------------------------------
```

04 Feature Engineering

01 Feature Generation

02 Data Memory Reduction

Memory reducing

In [22]:
```
def reduce_mem_usage(data):
    numerics = ['int8', 'int16', 'int32', 'int64', 'float32', 'float64']
    start_memory = data.memory_usage().sum() / 1024**2
    for col in data.columns:
        col_type = data[col].dtypes
        if col_type in numerics:
            c_min = data[col].min()
            c_max = data[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    data[col] = data[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    data[col] = data[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    data[col] = data[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    data[col] = data[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    data[col] = data[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    data[col] = data[col].astype(np.float32)
                else:
                    data[col] = data[col].astype(np.float64)
    end_memory = data.memory_usage().sum() / 1024**2
    print('Memory optimization from {:5.2f}MB to {:5.2f}MB ({:.1f}% reduction)'
          .format(start_memory, end_memory, 100 * (start_memory - end_memory) / start_memory))
    return data
```

In [24]:
```
calendar = reduce_mem_usage(calendar)
sell_prices = reduce_mem_usage(sell_prices)
train = reduce_mem_usage(train)
valid = reduce_mem_usage(valid)
test = reduce_mem_usage(test)
```

```
Memory optimization from  0.15MB to  0.06MB (60.8% reduction)
Memory optimization from 58.82MB to 45.77MB (22.2% reduction)
Memory optimization from 8541.17MB to 3488.95MB (59.2% reduction)
Memory optimization from 4696.12MB to 2505.34MB (46.7% reduction)
Memory optimization from 4781.61MB to 2550.93MB (46.7% reduction)
```

05

Modeling

01. Time Series Modeling
02. Tree Modeling

05 Modeling

01 Time Series Modeling

02 Tree Modeling

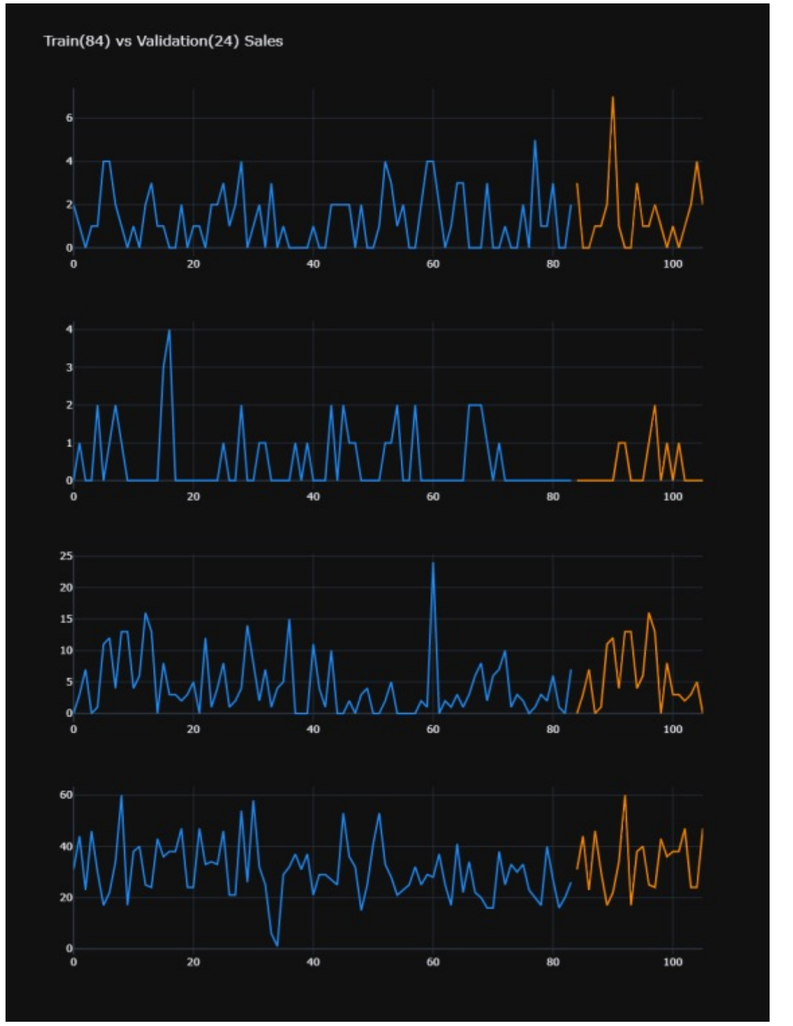
Train & Validate

Make simple training and validation s
ets to train and validate our models

train_dataset : 84개

val_dataset : 28개

5000th, 10000th, 15000th, 25000th samples are selected for Visualization



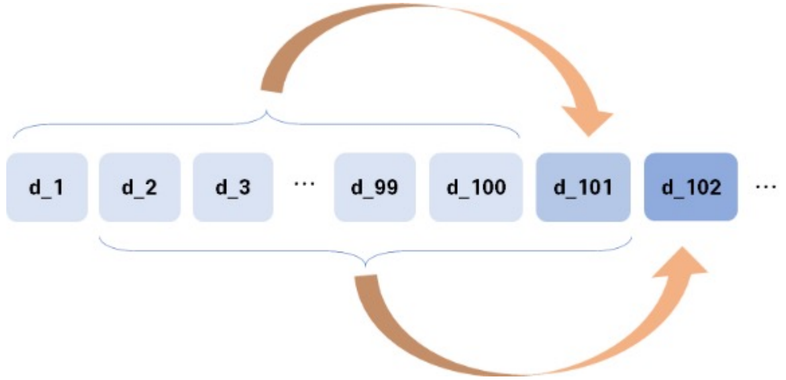
05 Modeling

01 Time Series Modeling

02 Tree Modeling

Simple Moving Average

Method




```
period = 30
prediction_sma = train_dataset.iloc[:, -period:].copy()
for i in range(len(val_dataset.loc[0])):
    prediction_sma['F'+str(i+1)] = prediction_sma.iloc[:, -period:].mean(axis=1)
prediction_sma = prediction_sma[['F'+str(i+1) for i in range(len(val_dataset.loc[0]))]]
wrmsse_sma = evaluator.score(prediction_sma.values)
print('wrmsse:', wrmsse_sma)
```

```
wrmsse: 1.0923862144779022
```

Expression

$$\hat{y}_{t+1} = \frac{1}{30} \cdot \sum_{t-30}^{t} y_n$$

05 Modeling

01 Time Series Modeling

02 Tree Modeling

Simple Exponential Smoothing

Method

Attach larger weights to more recent observations than to observations from the distant pasr. Forecasts are cal culated using weighted averages.

```
prediction_ses = []
for row in tqdm(train_dataset.values):
    fit2 = SimpleExpSmoothing(row, initialization_method='estimated').fit(
        smoothing_level=0.4, optimized=True)
    prediction_ses.append(fit2.forecast(28))
wrmsse_ses = evaluator.score(np.array(prediction_ses))
print('wrmsse:', wrmsse_ses)
```

```
100% 30490/30490 [03:30<00:00, 144.87it/s]

wrmsse: 1.1969485850464783
```

Expression

$$\hat{y}_{t+1} = \alpha \cdot y_t + \alpha \cdot (1-\alpha) \cdot y_{t-1} + \alpha \cdot (1-\alpha)^2 \cdot y_{t-2} + \dots$$

$$\hat{y}_{t+1} = \alpha \cdot y_t + (1-\alpha) \cdot \hat{y}_t$$

05 Modeling

01 Time Series Modeling

02 Tree Modeling

Double Exponential Smoothing

Method

It improves the weakness of Single Smoothing by the in troduction of a second equation with a second constant β, which must be chosen in conjunction with α.

```
prediction_des = []
for row in tqdm(train_dataset.values):
    fit = ExponentialSmoothing(row, seasonal_periods=12,
                               initialization_method='estimated').fit(
        smoothing_level = 0.2, smoothing_trend = 0.15)
    prediction_des.append(fit.forecast(28))
wrmsse_des = evaluator.score(np.array(prediction_des))
print('wrmsse:', wrmsse_des)
```

```
100% 30490/30490 [02:55<00:00, 173.92it/s]

wrmsse: 1.111435528465374
```

Expression

$$\hat{y}_{t+h} = l_t + h \cdot b_t$$

$$Level Equation : l_t = a \cdot y_t + (1-\alpha) \cdot (l_{t-1} + b_{t-1})$$
$$Trand Equation : b_t = \beta \cdot (l_t - l_{t-1}) + (1-\beta) \cdot b_{t-1}$$

05 Modeling

01 Time Series Modeling

02 Tree Modeling

Prophet

$$y(t) = g(t) + s(t) + h(t) + \epsilon_i$$

- **Trend**
  $g(t)$: piecewise linear or logistic growth curve for modelling non-periodic changes in time series
  주기적이지 않은 변화인 트렌드를 나타냅니다
- **Seasonality**
  $s(t)$: periodic changes (e.g. weekly/yearly seasonality)
  weekly, yearly 등 주기적으로 나타나는 패턴들을 포함합니다.
- **Holiday**
  $h(t)$: effects of holidays (user provided) with irregular schedules
  휴일과 같이 불규칙한 이벤트들을 나타냅니다.
- **Error**
  ϵ_i : error term accounts for any unusual changes not accommodated by the model
  정규분포라고 가정한 오차입니다.

The main components of the Prophet model are
Trend, Seasonality, and Holiday

```
prophet_df = train_dataset.T
prophet_df.index = pd.date_range(start='20160104', periods=84)
prediction_pro = []
for i in tqdm(range(300)):
    df = prophet_df[i].reset_index()
    df.columns = ['ds', 'y']
    m = Prophet(growth='linear', uncertainty_samples=False,
                yearly_seasonality=False, daily_seasonality=True).fit(df)
    future = m.make_future_dataframe(periods=28)
    prediction_pro.append(m.predict(future).iloc[-28:, -1])
```

100% 300/300 [05:03<00:00, 1.01s/it]

05 Modeling

01 Time Series Modeling

02 Tree Modeling

ARIMA

- Auto-Regressive (AR) filter (long term):

  $y_t = c + \alpha_1 y_{t-1} + \dots \alpha_p y_{t-p} + \epsilon_t = c + \sum_{i=1}^{p} \alpha_i y_{t-i} + \epsilon_t$ → p

- Integration filter (stochastic trend)

  → d

- Moving Average (MA) filter (short term):

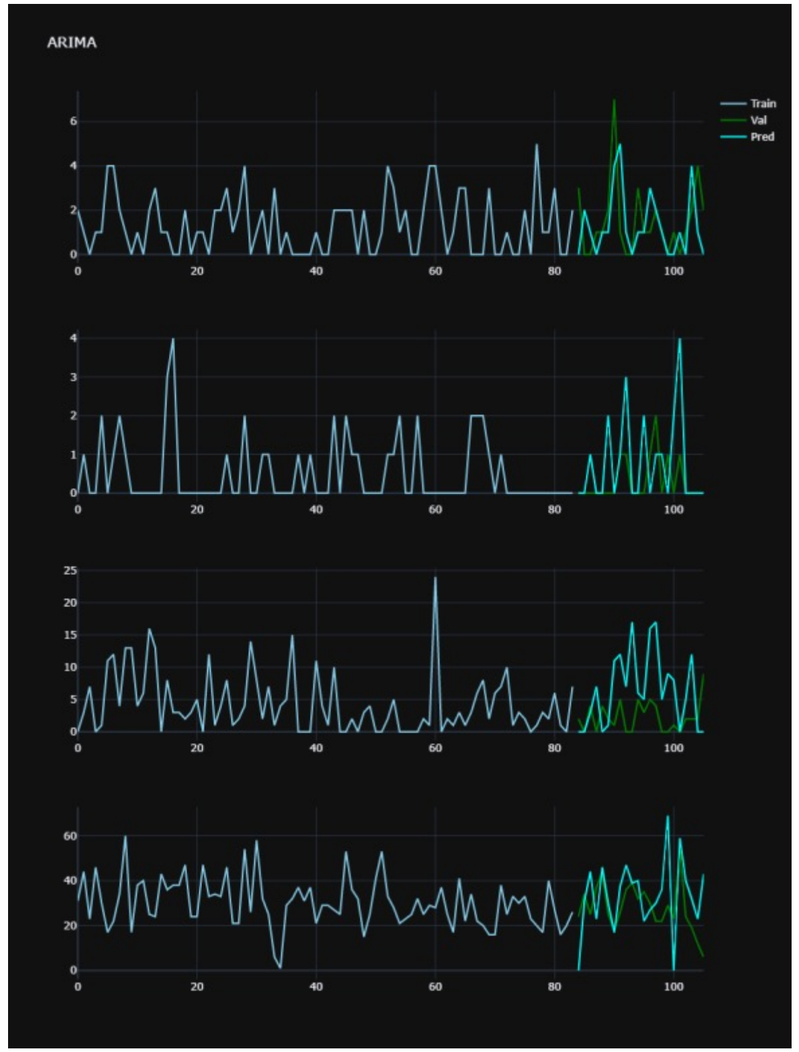
  $y_t = c + \epsilon_t + \beta_1 \epsilon_{t-1} + \cdots + \beta_q \epsilon_{t-q} = c + \epsilon_t + \sum_{i=1}^{q} \beta_i \epsilon_{t-i}$ → q

ARIMA: $y_t = c + \alpha_1 y_{t-1} + \cdots + \alpha_p y_{t-p} + \epsilon_t + \beta_1 \epsilon_{t-1} + \cdots + \beta_q \epsilon_{t-q}$

AutoRegressive Integrated Moving Average (ARIMA) is a combination of an autoregressive model (AR) and a moving averagemodel (MA) and a difference (I) to ensure the normality of data. For ARIMA (1,1,1), this means that AR(1) and I(1), MA(1) are combined models.

```
prediction_arima = []
warnings.filterwarnings(action='ignore')
for row in tqdm(train_dataset.values):
    arima = sm.tsa.statespace.SARIMAX(row, order=(1,1,1), seasonal_order=(0,1,0,7)).fit()
    prediction_arima.append(arima.predict(0, 27, typ='levels'))
prediction_arima = np.where(np.array(prediction_arima)<0, 0, prediction_arima)
wrmsse_arima = evaluator.score(prediction_arima)
print('wrmsse:', wrmsse_arima)
```

```
100% 30490/30490 [1:20:25<00:00, 6.32it/s]

wrmsse: 1.5358011636489257
```



05 Modeling

01 Time Series Modeling

02 Tree Modeling

Model Evaluation

05 Modeling

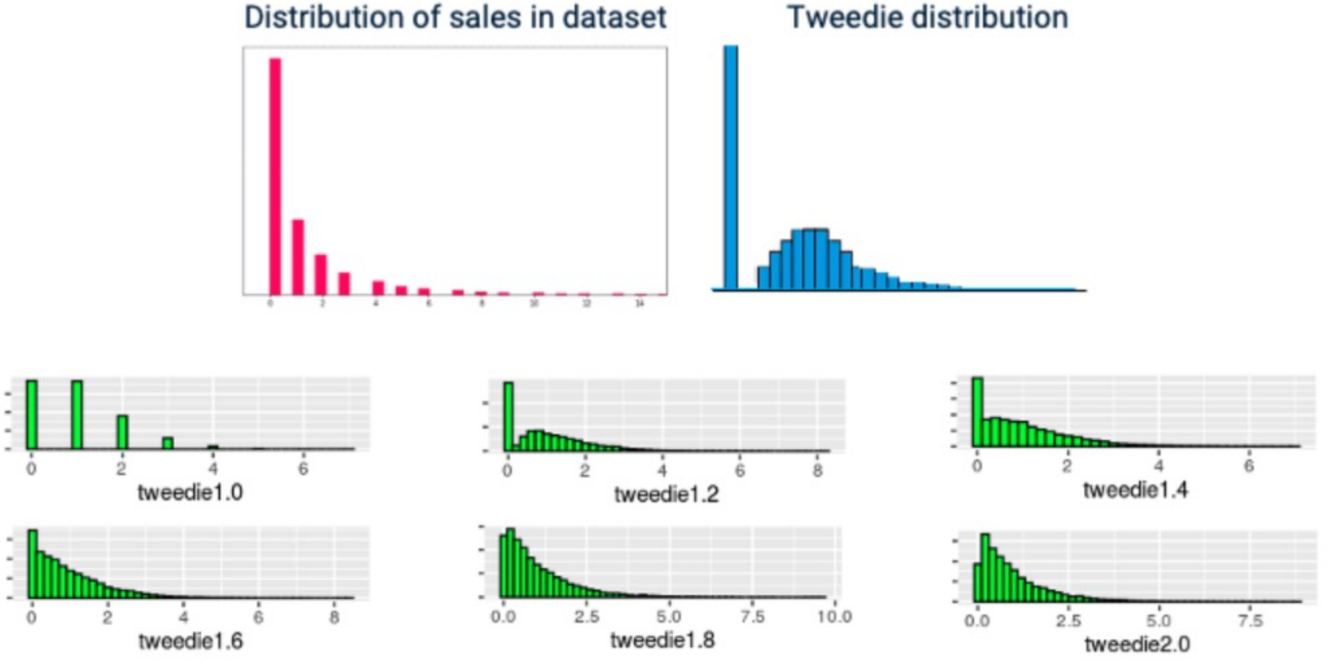
01. Time Series Modeling

02 Tree Modeling

Setting

Objective : Tweedie

Setting the tweedie loss as an objective function will basically force the model to maximize the likelihood of that distribution and thus predict the right amount of '0's



05 Modeling

01. Time Series Modeling

02 Tree Modeling

Setting

Loss Function : RMSE (root mean squared error)

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

By minimizing RMSE, we can get high score in LB (LeaderBoard)
We can set loss function by setting tree model's metric as 'rmse'

05 Modeling

01. Time Series Modeling

02 Tree Modeling

Setting

Early Stopping : Custom metric WRMSSE

```
def wrmsse_lgb(y_pred, dataset):

    wrmsse = evaluator.score(y_pred.reshape(28, -1).T)

    return 'wrmsse', wrmsse, False
```

```
def wrmsse_xgb(y_pred, dataset):

    wrmsse = evaluator.score(y_pred.reshape(28, -1).T)

    return 'wrmsse', wrmsse
```

```
lgbmodel = lgb.train(lgb_best_hyperparams, train_data, valid_sets=[valid_data],
                     early_stopping_rounds=200, feval= wrmsse_lgb , verbose_eval=100)
```

```
xgbmodel = xgb.train(xgb_best_hyperparams, dtrain, 8000, watchlist,
                     early_stopping_rounds=200, feval=wrmsse_xgb, verbose_eval=100)
```

For my tree modeling LightGBM, XGBosst, I made custom metric for early-stopping to avoid overfitting

05 Modeling

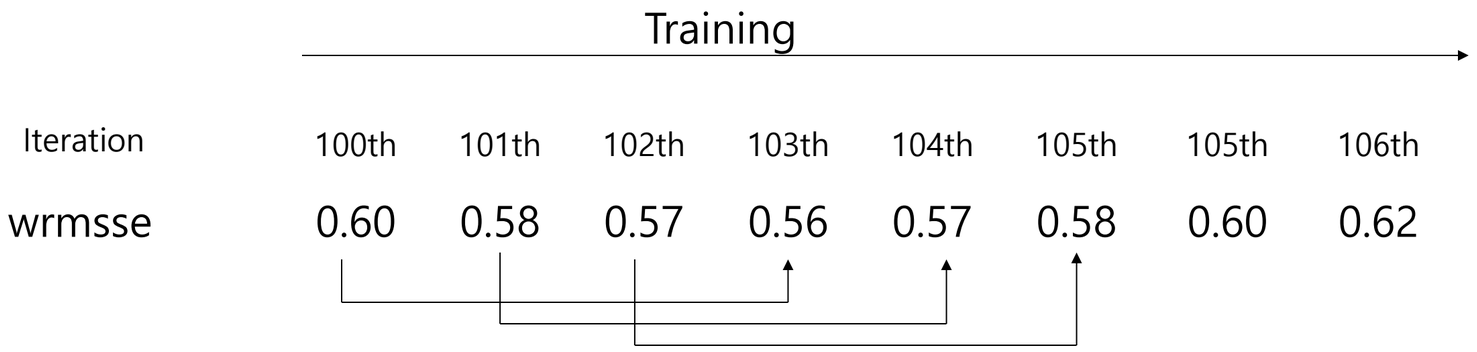
01. Time Series Modeling

02 Tree Modeling

Setting

Early Stopping : Custom metric WRMSSE

If early_stopping_rounds == 3:



For my tree modeling LightGBM, XGBosst, I made custom metric for early-stopping to avoid overfitting

05 Modeling

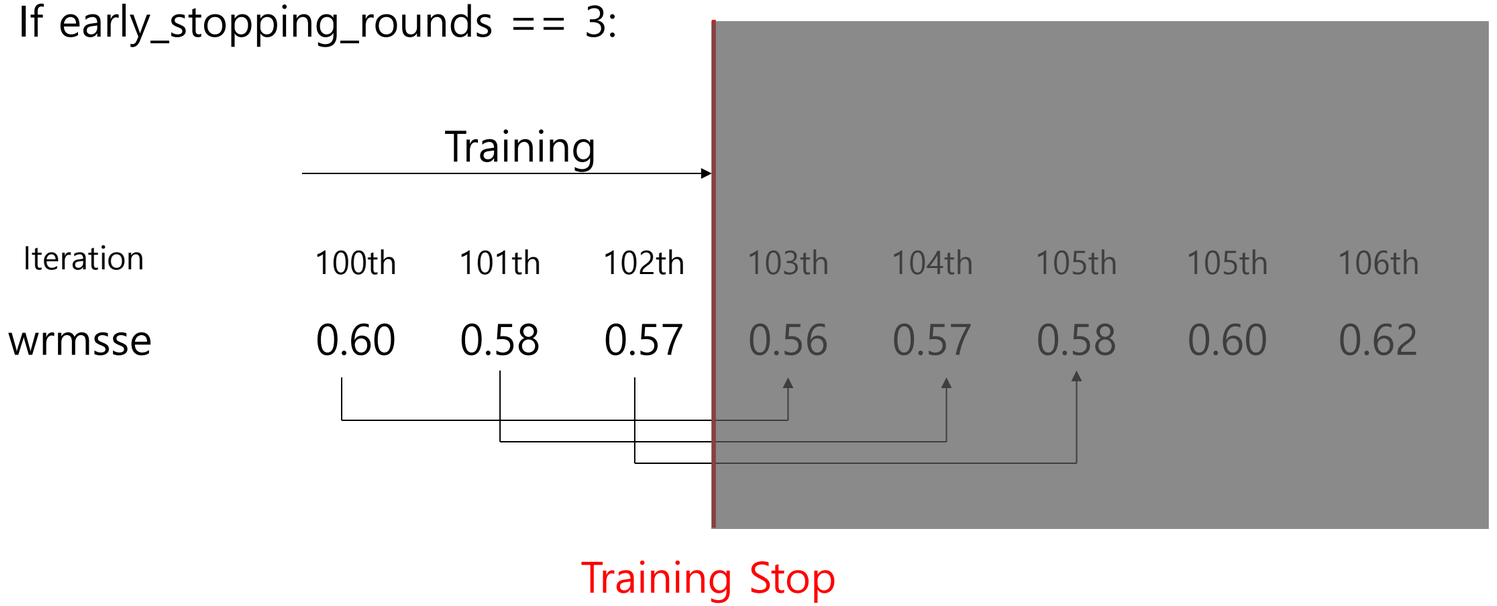
01. Time Series Modeling

02 Tree Modeling

Setting

Early Stopping : Custom metric WRMSSE

If early_stopping_rounds == 3:



For my tree modeling LightGBM, XGBosst, I made custom metric for early-stopping to avoid overfitting

05 Modeling

01. Time Series Modeling

02 Tree Modeling

LightGBM

Split Data

* d_398 ~ d_1913 (4 years + 56 days) train set
* d_1914 ~ d_1941 (28 days) validation set
* d_1942 ~ d_1619 (28 days) test set

```
category_cols = ['item_id','store_id', 'cat_id', 'state_id', 'wday', 'month', 'snap']
useless_cols = ["id", "dept_id", "date","d", "sales", "wm_yr_wk", "weekday", "sell_price"]
train_cols = train.columns[~train.columns.isin(useless_cols)]

df = train.copy()
days_train = ['d_'+str(c) for c in range(1914-365*4-28*2-1, 1914)]
days_val = ['d_'+str(c) for c in range(1914, 1942)]
df = df.replace([np.inf, -np.inf], 0)
#df.iloc[:, -9:] = df.iloc[:, -9:].fillna(0.0)
X_train = df[df['d'].isin(days_train)==True][train_cols]
y_train = df[df['d'].isin(days_train)==True]["sales"]
X_val_df = df[df['d'].isin(days_val)==True]
X_val_df["date"] = pd.to_datetime(X_val_df["date"])
X_val_dff = pd.DataFrame()
for delta in range(0, 28):
    day = datetime(2016, 4, 25) + timedelta(days=delta)
    vl = X_val_df.loc[X_val_df.date == day]
    X_val_dff = pd.concat([X_val_dff, vl])

X_val = X_val_dff[train_cols]
y_val = X_val_dff["sales"]
```

05 Modeling

01. Time Series Modeling

02 Tree Modeling

LightGBM

HP Tuning

```
def lgb_objective(trial: Trial) -> float:
    params_lgb = {

        #"learning_rate": trial.suggest_uniform("learning_rate", 0.08, 0.05),
        #"lambda_l1": trial.suggest_uniform("lambda_l1", 0.0, 1),
        #"lambda_l2": trial.suggest_uniform("lambda_l2", 0.0, 1),

        'boosting_type': 'gbdt',
        'objective': 'tweedie',
        'metric': 'rmse',
        'verbose': -1,
        'boost_from_average': False,
        'n_estimators': 5000,
        "learning_rate": trial.suggest_uniform("learning_rate", 0.015, 0.03),
        "tweedie_variance_power": trial.suggest_uniform("tweedie_variance_power", 1.05, 1.3),
        "num_leaves": trial.suggest_int("num_leaves", 1500, 2500),
        "min_data_in_leaf": trial.suggest_int("min_data_in_leaf", 3000, 5000),
        "max_bin": trial.suggest_int("max_bin", 200, 400),
        "feature_fraction": trial.suggest_uniform("feature_fraction", 0.3, 0.5),
    }

    train_data = lgb.Dataset(X_train, label = y_train, categorical_feature=category_cols)
    valid_data = lgb.Dataset(X_val, label = y_val, categorical_feature=category_cols)
                                                                                    # 진행상황 보
    lgbmodel = lgb.train(params_lgb, train_data, valid_sets=[valid_data], early_stopping_rounds=200,
                         feval=wrmsse_lgb, verbose_eval=None)
    wrmsse = evaluator.score(lgbmodel.predict(X_val).reshape(28, -1).T)

    return wrmsse
```

```
sampler = TPESampler(seed=42)
lgb_study = optuna.create_study(study_name="lgbm_parameter_opt", direction="minimize", sampler=sampler)
lgb_study.optimize(lgb_objective, n_trials=20)

lgb_best_hyperparams = lgb_study.best_trial.params
lgb_base_hyperparams = {'boosting_type': 'gbdt', 'objective': 'tweedie', 'metric': 'rmse', 'verbose': -1,
                        'boost_from_average': False, 'n_estimators': 5000}
lgb_best_hyperparams.update(lgb_base_hyperparams)
print("The best hyperparameters are:₩n", lgb_best_hyperparams)
```

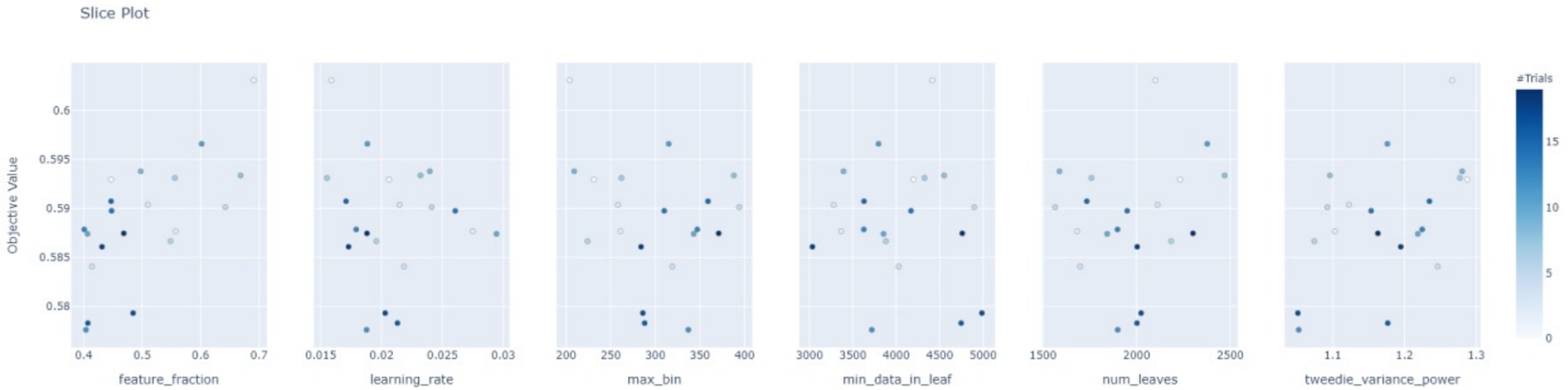
```
51282946}. Best is trial 4 with value: 0.5840686241966461.
[I 2021-05-31 23:40:57,958] Trial 9 finished with value: 0.5937750111137902 and parameters: {'learning_rate': 0.023968499682166276, 'tweedie_variance_power': 1.2804685587557794, 'num_leaves': 1588, 'min_data_in_leaf': 3392, 'max_bin': 209, 'feature_fraction': 0.4975990922897927}. Best is trial 4 with value: 0.5840686241966461.
[I 2021-06-01 00:20:45,825] Trial 10 finished with value: 0.587399221155524 and parameters: {'learning_rate': 0.029444342646692857, 'tweedie_variance_power': 1.2186992669704446, 'num_leaves': 1843, 'min_data_in_leaf': 3853, 'max_bin': 343, 'feature_fraction': 0.4064321224037979}. Best is trial 4 with value: 0.5840686241966461.
[I 2021-06-01 01:02:53,823] Trial 11 finished with value: 0.5965736709249124 and parameters: {'learning_rate': 0.018831651881477982, 'tweedie_variance_power': 1.176249417270692, 'num_leaves': 2377, 'min_data_in_leaf': 3795, 'max_bin': 315, 'feature_fraction': 0.6018301230871671}. Best is trial 4 with value: 0.5840686241966461.
[I 2021-06-01 01:46:08,624] Trial 12 finished with value: 0.5776277196987923 and parameters: {'learning_rate': 0.018770962054402065, 'tweedie_variance_power': 1.0527349836416215, 'num_leaves': 1899, 'min_data_in_leaf': 3718, 'max_bin': 337, 'feature_fraction': 0.4037791062361256}. Best is trial 12 with value: 0.5776277196987923.
[I 2021-06-01 02:32:18,202] Trial 13 finished with value: 0.5878530975982847 and parameters: {'learning_rate': 0.017915006622672335, 'tweedie_variance_power': 1.2248331819225307, 'num_leaves': 1898, 'min_data_in_leaf': 3626, 'max_bin': 347, 'feature_fraction': 0.4006855684663833}. Best is trial 12 with value: 0.5776277196987923.
[I 2021-06-01 03:08:19,824] Trial 14 finished with value: 0.5897495068133817 and parameters: {'learning_rate': 0.026059014746472115, 'tweedie_variance_power': 1.1536241420816025, 'num_leaves': 1950, 'min_data_in_leaf': 4169, 'max_bin': 310, 'feature_fraction': 0.4476466577826186}. Best is trial 12 with value: 0.5776277196987923.
[I 2021-06-01 04:04:24,645] Trial 15 finished with value: 0.5907256174140635 and parameters: {'learning_rate': 0.017082192563288814, 'tweedie_variance_power': 1.2348231801406788, 'num_leaves': 1734, 'min_data_in_leaf': 3628, 'max_bin': 359, 'feature_fraction': 0.4467718444008832}. Best is trial 12 with value: 0.5776277196987923.
[I 2021-06-01 05:20:45,904] Trial 16 finished with value: 0.5783102785891984 and parameters: {'learning_rate': 0.021307779991994 06, 'tweedie_variance_power': 1.1767786031598775, 'num_leaves': 2002, 'min_data_in_leaf': 4747, 'max_bin': 288, 'feature_fraction': 0.4070200948919725}. Best is trial 12 with value: 0.5776277196987923.
[I 2021-06-01 06:16:28,196] Trial 17 finished with value: 0.5793314222693554 and parameters: {'learning_rate': 0.020900843914065 36, 'tweedie_variance_power': 1.051544547765994, 'num_leaves': 2024, 'min_data_in_leaf': 4986, 'max_bin': 286, 'feature_fraction': 0.48447237392675013}. Best is trial 12 with value: 0.5776277196987923.
[I 2021-06-01 07:21:08,035] Trial 18 finished with value: 0.5860879663382954 and parameters: {'learning_rate': 0.017288816806593 15, 'tweedie_variance_power': 1.1949202119538775, 'num_leaves': 2003, 'min_data_in_leaf': 3034, 'max_bin': 284, 'feature_fraction': 0.4312886336056605}. Best is trial 12 with value: 0.5776277196987923.
[I 2021-06-01 08:14:34,105] Trial 19 finished with value: 0.5874549197986642 and parameters: {'learning_rate': 0.018801123575959 25, 'tweedie_variance_power': 1.163036927084835, 'num_leaves': 2301, 'min_data_in_leaf': 4760, 'max_bin': 371, 'feature_fraction': 0.4685139661007615 7}. Best is trial 12 with value: 0.5776277196987923.

The best hyperparameters are:
 {'learning_rate': 0.018770962054402065, 'tweedie_variance_power': 1.0527349836416215, 'num_leaves': 1899, 'min_data_in_leaf': 3718, 'max_bin': 337, 'feature_fraction': 0.4037791062361256, 'boosting_type': 'gbdt', 'objective': 'tweedie', 'metric': 'rmse', 'verbose': -1, 'boost_from_average': False, 'n_estimators': 5000}
```

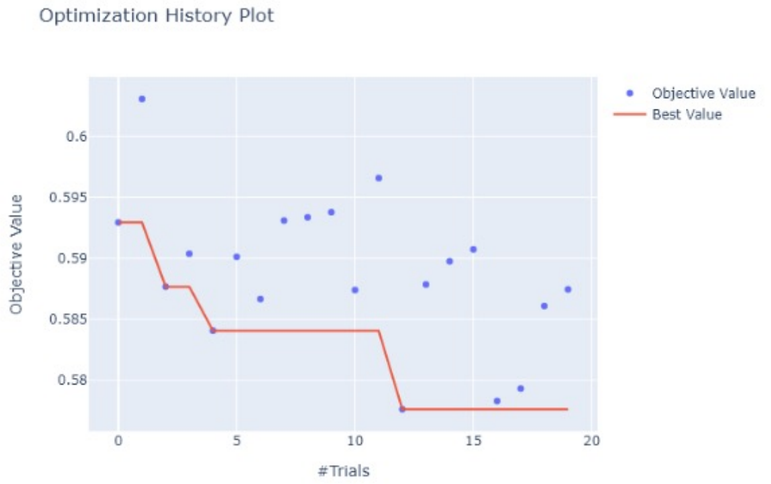
05 Modeling

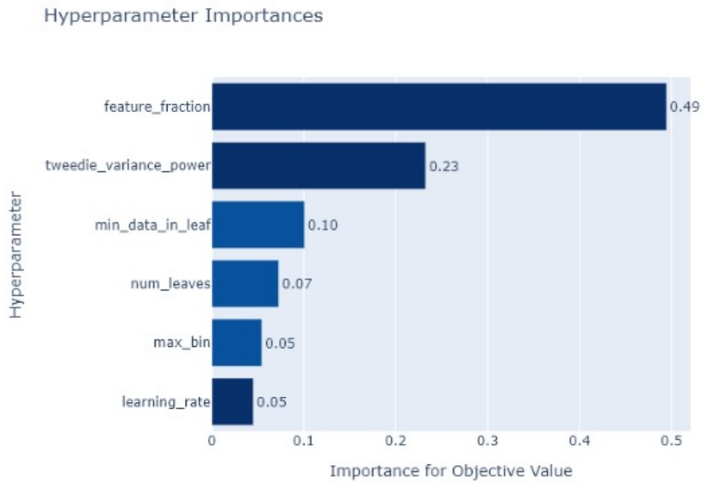
01. Time Series Modeling

02 Tree Modeling

LightGBM







05 Modeling

01. Time Series Modeling

02 Tree Modeling

LightGBM

Train

Best Hyperparameters

'boosting_type': 'gbdt',
'objective': 'tweedie',
'metric': 'rmse',
'verbose': -1,
'boost_from_average': False,
'n_estimators': 5000
'learning_rate': 0.018770362054402065,
'tweedie_variance_power': 1.0527349836416215,
'num_leaves': 1899,
'min_data_in_leaf': 3718,
'max_bin': 337,
'feature_fraction': 0.4037791062361256

Training with Best HP

```
train_data = lgb.Dataset(X_train, label = y_train, categorical_feature=category_cols)
valid_data = lgb.Dataset(X_val, label = y_val, categorical_feature=category_cols)

lgbmodel = lgb.train(lgb_best_hyperparams, train_data, valid_sets=[valid_data],
                     early_stopping_rounds=200, feval= wrmsse_lgb, verbose_eval=100)
```

```
Training until validation scores don't improve for 200 rounds
[100]   valid_0's rmse: 2.12437 valid_0's wrmsse: 0.918185
[200]   valid_0's rmse: 2.03291 valid_0's wrmsse: 0.652819
[300]   valid_0's rmse: 2.02241 valid_0's wrmsse: 0.599879
[400]   valid_0's rmse: 2.01915 valid_0's wrmsse: 0.585957
[500]   valid_0's rmse: 2.01796 valid_0's wrmsse: 0.578213
[600]   valid_0's rmse: 2.01811 valid_0's wrmsse: 0.576986
[700]   valid_0's rmse: 2.0184  valid_0's wrmsse: 0.576617
Early stopping, best iteration is:
[526]   valid_0's rmse: 2.01767 valid_0's wrmsse: 0.577628
```

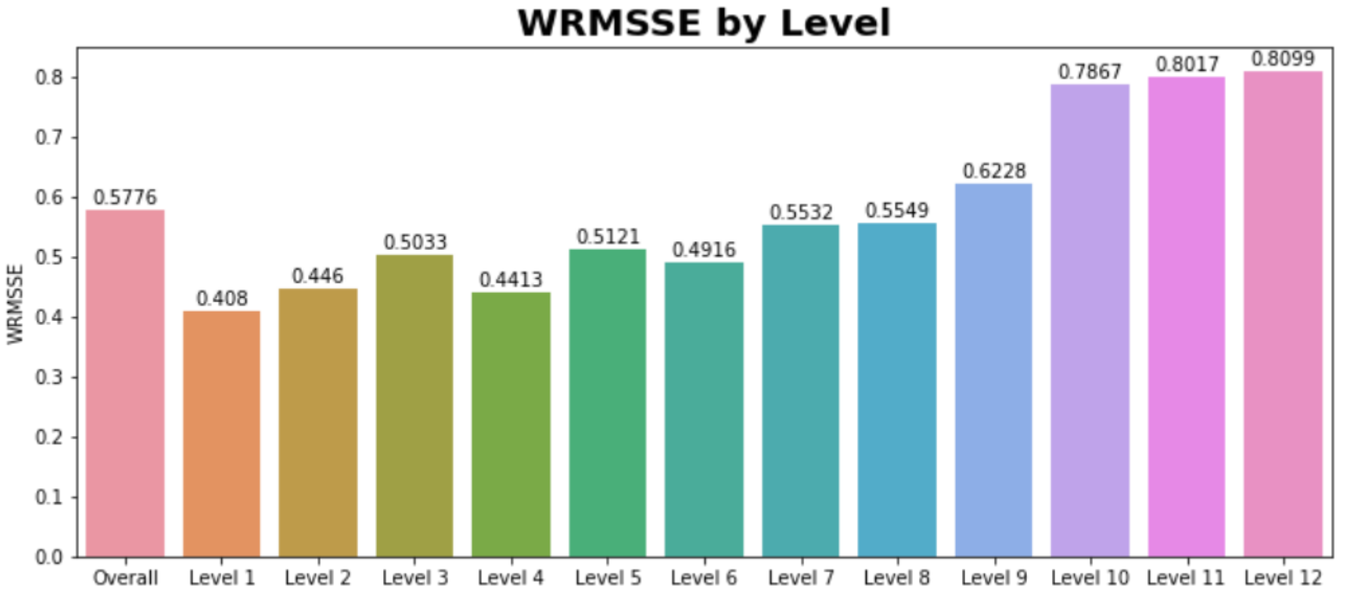
05 Modeling

01. Time Series Modeling

02 Tree Modeling

LightGBM

Evaluation



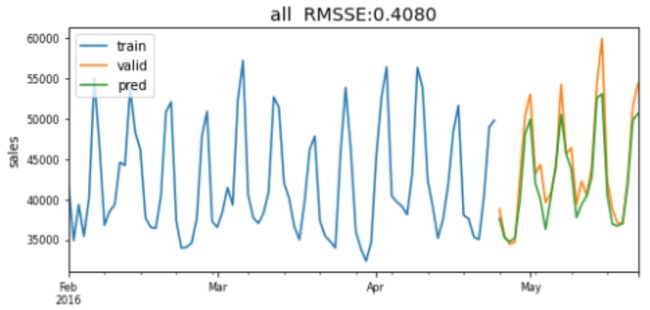
05 Modeling

01. Time Series Modeling

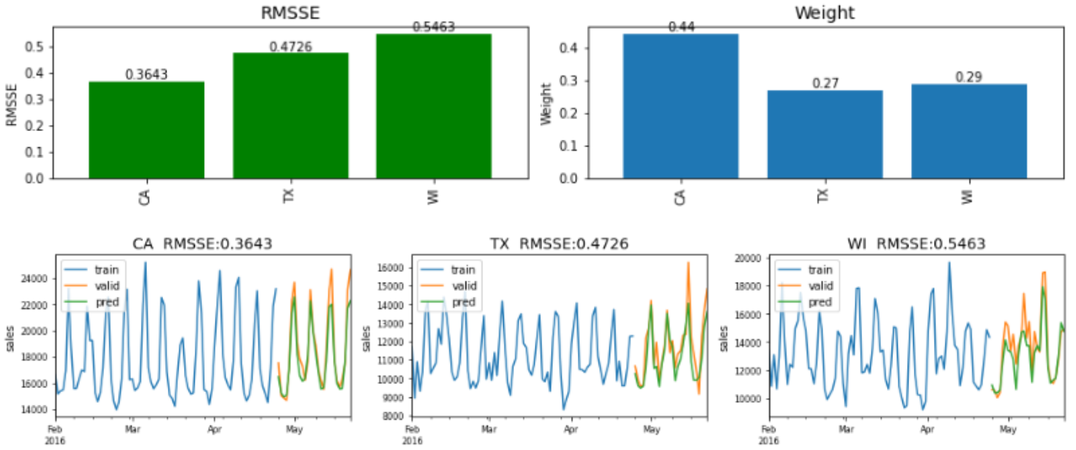
02 Tree Modeling

LightGBM

Level 1: all_id



Level 2: state_id



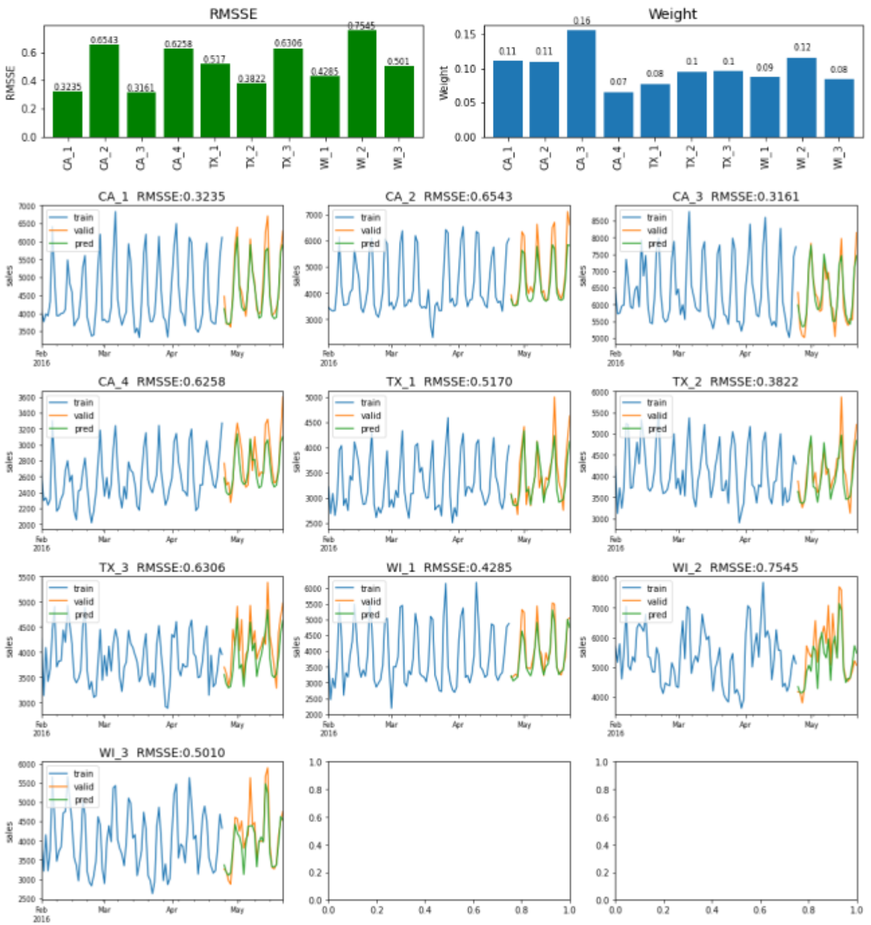
05 Modeling

01. Time Series Modeling

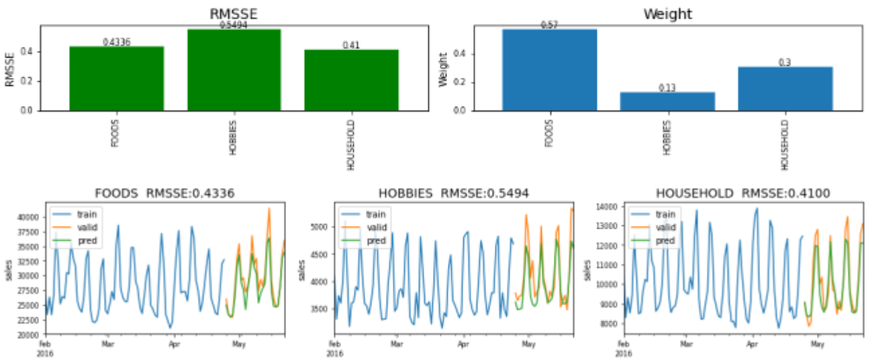
02 Tree Modeling

LightGBM

Level 3: store_id



Level 4: cat_id



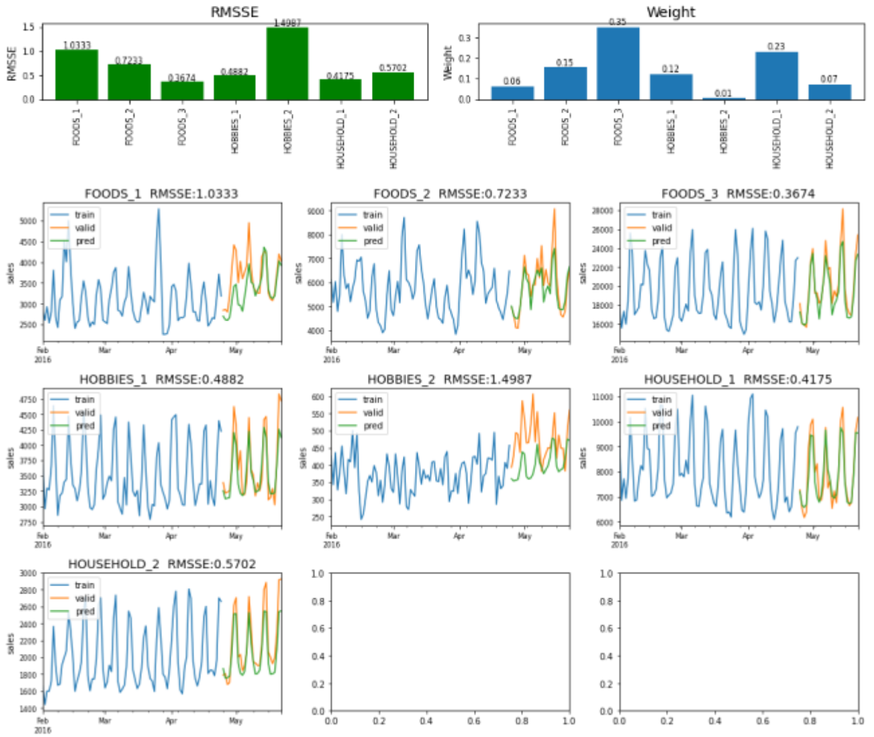
05 Modeling

01. Time Series Modeling

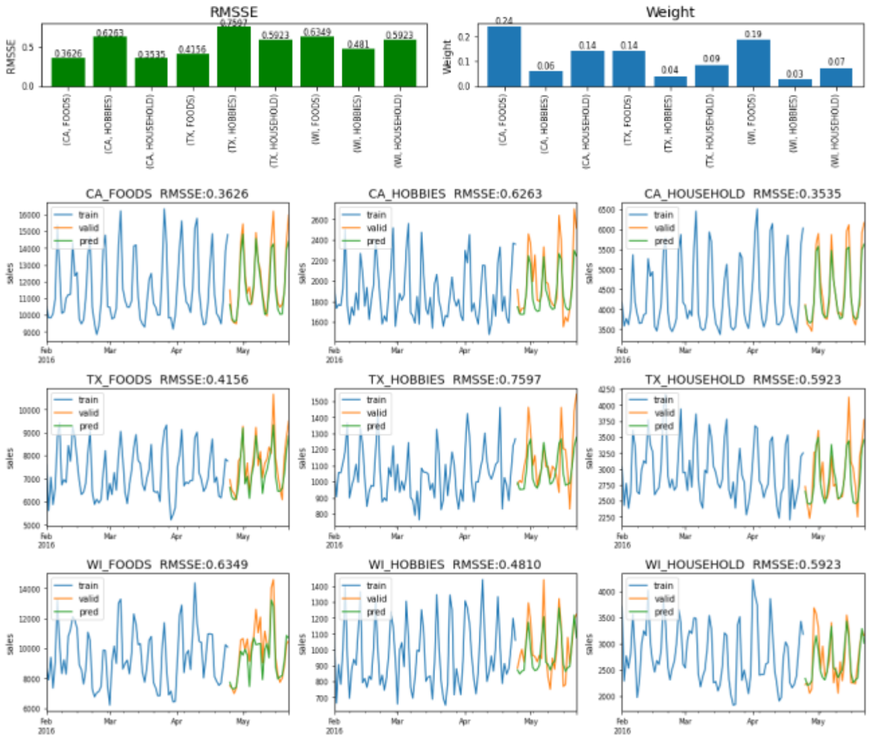
02 Tree Modeling

LightGBM

Level 5: dept_id



Level 6: ['state_id', 'cat_id']



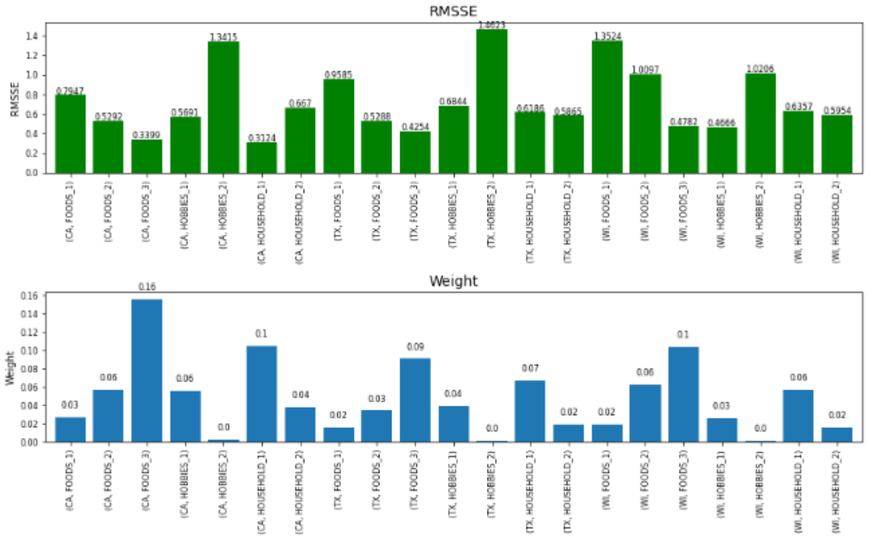
05 Modeling

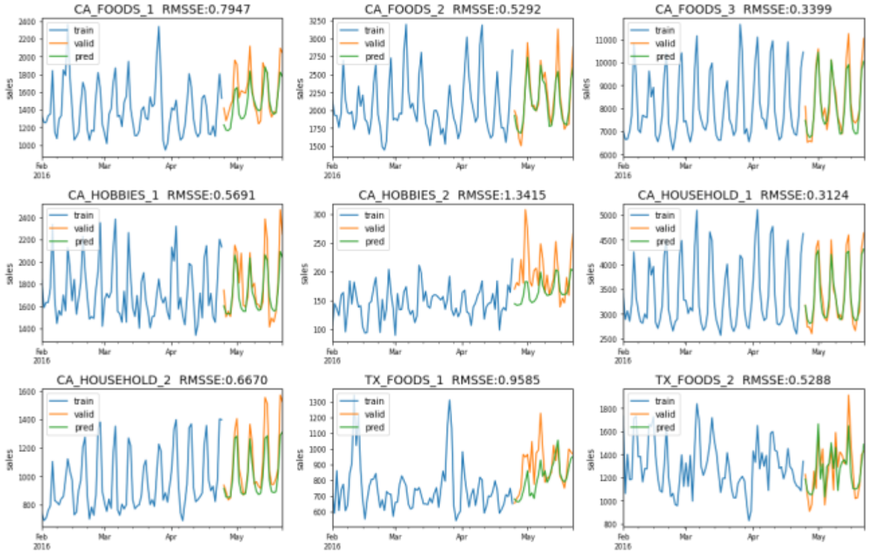
01. Time Series Modeling

02 Tree Modeling

LightGBM

Level 7: ['state_id', 'dept_id']





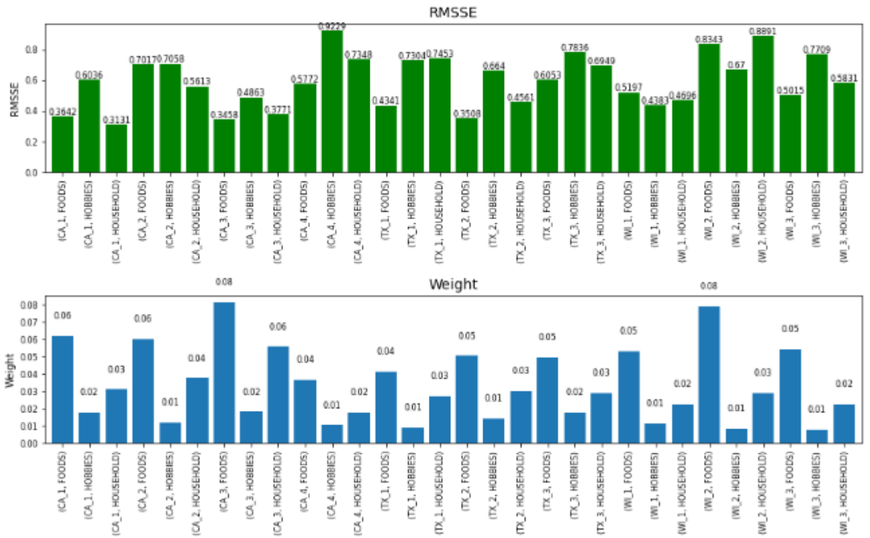
05 Modeling

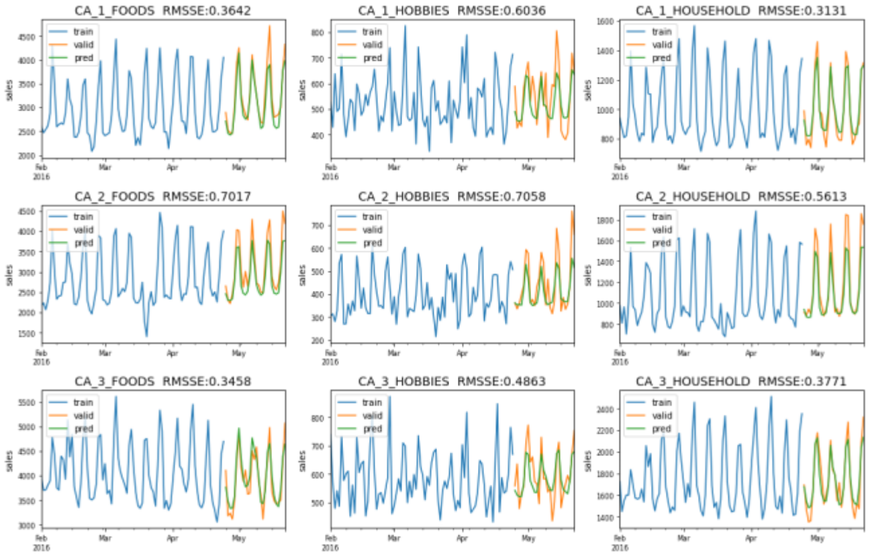
01. Time Series Modeling

02 Tree Modeling

LightGBM

Level 8: ['store_id', 'cat_id']





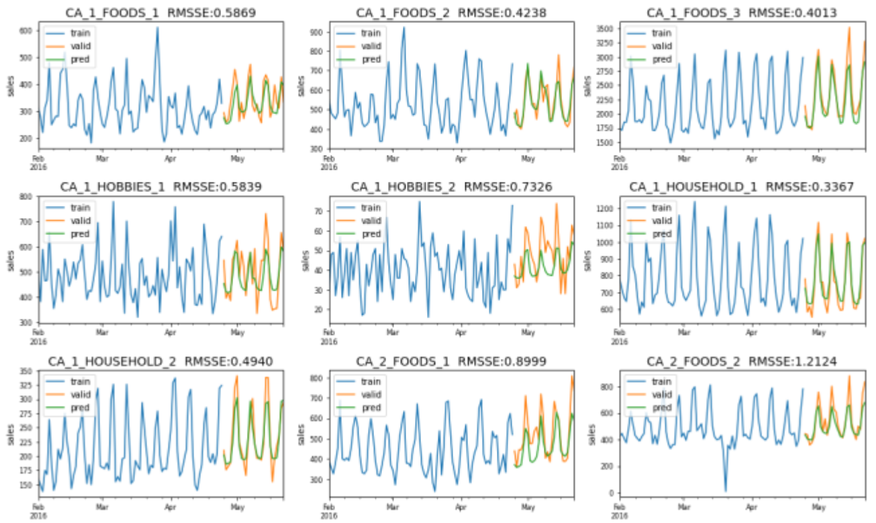
05 Modeling

01. Time Series Modeling

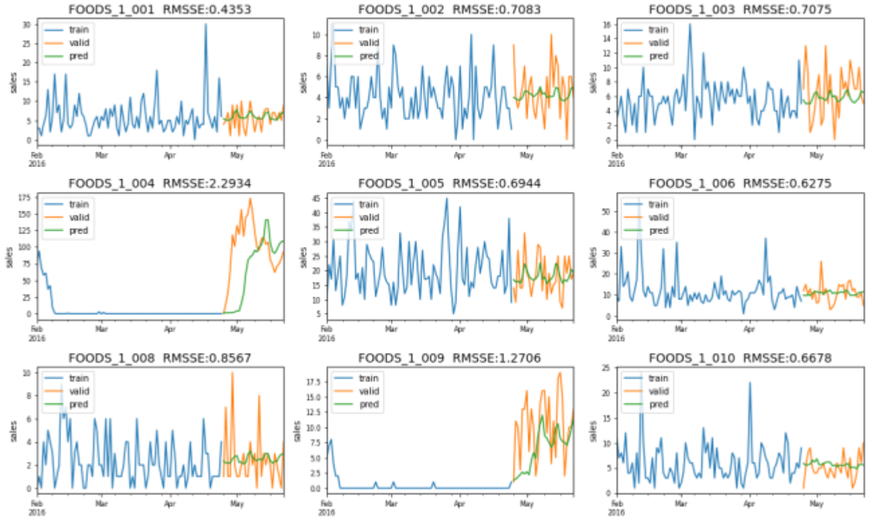
02 Tree Modeling

LightGBM

Level 9: ['store_id', 'dept_id']



Level 10: item_id



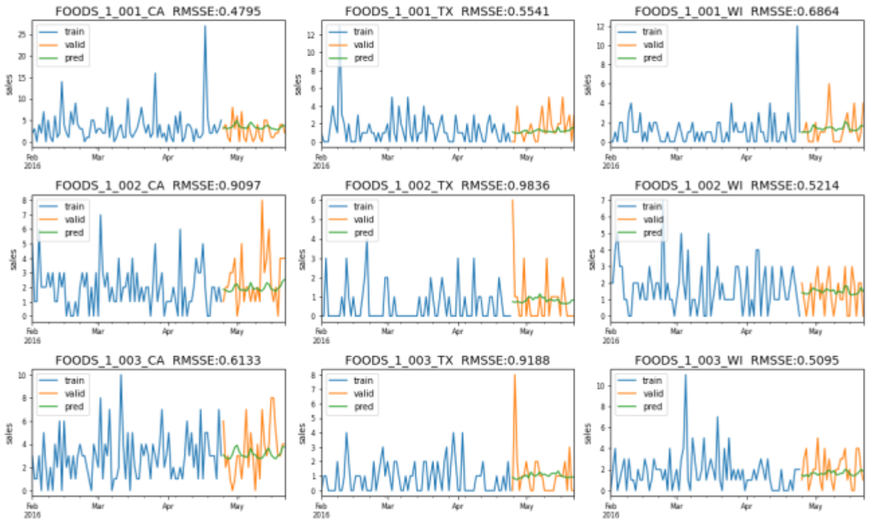
05 Modeling

01. Time Series Modeling

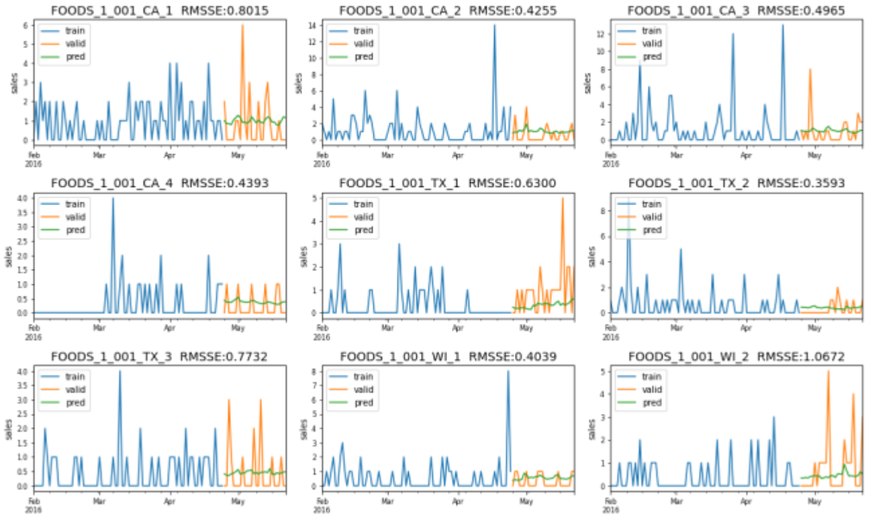
02 Tree Modeling

LightGBM

Level 11: ['item_id', 'state_id']



Level 12: ['item_id', 'store_id']



05 Modeling

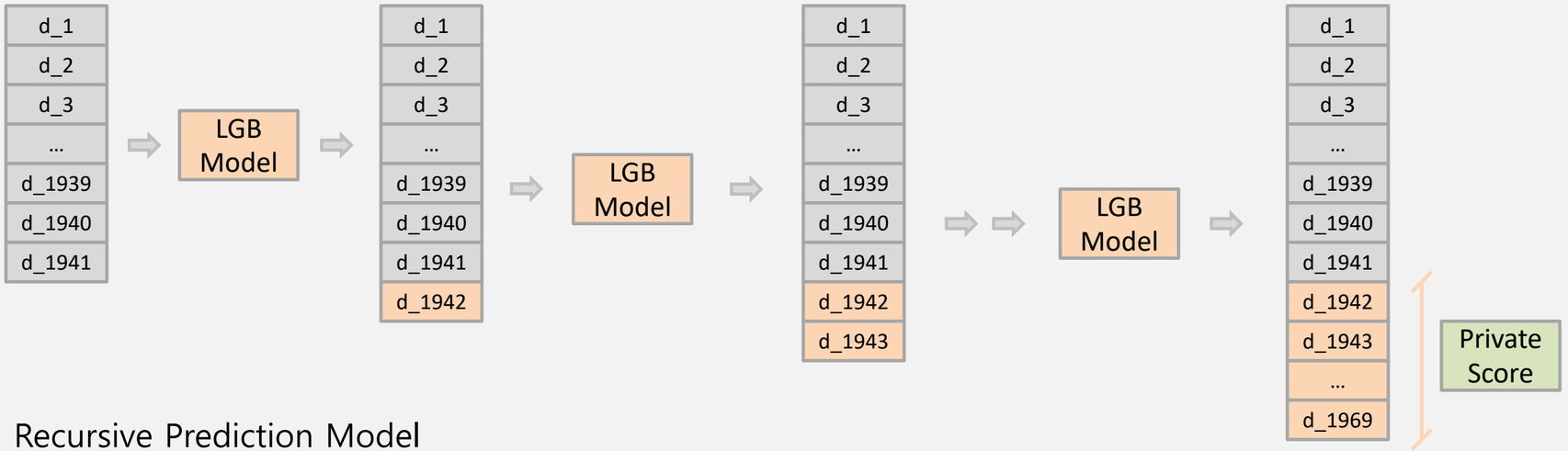
01. Time Series Modeling

02 Tree Modeling

LightGBM

Predict

```
cols = [f"F{i}" for i in range(1,29)]
test["date"] = pd.to_datetime(test["date"])
for time_delta in range(0, 28):
    day = datetime(2016, 5, 23) + timedelta(days=time_delta)
    print("Predict", day.date())
    tst = test[(test.date >= day - timedelta(days=60)) & (test.date <= day)].copy()
    lags_wins(tst)
    tst = tst.loc[tst.date == day , train_cols]
    test.loc[test.date == day, "sales"] = lgbmodel.predict(tst)
    del(tst)
```



Recursive Prediction Model

05 Modeling

01. Time Series Modeling

02 Tree Modeling

XGBoost

Split Data

* d_398 ~ d_1913 (4 years + 56 days) train set
* d_1914 ~ d_1941 (28 days) validation set
* d_1942 ~ d_1619 (28 days) test set

```
category_cols = ['item_id','store_id', 'cat_id', 'state_id', 'wday', 'month', 'snap']
useless_cols = ["id", "dept_id", "date","d", "sales", "wm_yr_wk", "weekday", "sell_price"]
train_cols = train.columns[~train.columns.isin(useless_cols)]

df = train.copy()
enc = LabelEncoder()
for col in df.columns:
    if df[col].dtype.name=='category':
        df[col] = enc.fit_transform(df[col])

df = df.replace([np.inf, -np.inf], 0)
days_train = ['d_'+str(c) for c in range(1914-365*4-28*2-1, 1914)]
days_val = ['d_'+str(c) for c in range(1914, 1942)]
#df.iloc[:, -9:] = df.iloc[:, -9:].fillna(0.0)
X_train = df[df['d'].isin(days_train)==True][train_cols]
y_train = df[df['d'].isin(days_train)==True]["sales"]
X_val_df = df[df['d'].isin(days_val)==True]
X_val_df["date"] = pd.to_datetime(X_val_df["date"])
X_val_dff = pd.DataFrame()
for delta in range(0, 28):
    day = datetime(2016, 4, 25) + timedelta(days=delta)
    vl = X_val_df.loc[X_val_df.date == day]
    X_val_dff = pd.concat([X_val_dff, vl])
X_val = X_val_dff[train_cols]
y_val = X_val_dff["sales"]

dtrain = xgb.DMatrix(X_train, label=y_train)
dvalid = xgb.DMatrix(X_val, label=y_val)
watchlist = [(dvalid, 'valid')]
```

05 Modeling

01. Time Series Modeling

02 Tree Modeling

XGBoost

HP Tuning

```
def xgb_objective(trial: Trial) -> float:
    params_xgb = {
        "random_state": 91373,
        "verbose": None,
        "objective": "reg:tweedie",
        "max_bin": 16,
        #"max_depth": 12,
        "tree_method": 'gpu_hist',
        "gpu_id": 0,
        #"learning_rate": 0.03,
        "tweedie_variance_power": trial.suggest_uniform("tweedie_variance_power", 1.05, 1.2),
        "learning_rate": trial.suggest_uniform("learning_rate", 0.05, 0.2),
        #"reg_alpha": trial.suggest_uniform("reg_alpha", 0.5, 1.0), # default=0
        #"reg_lambda": trial.suggest_uniform("reg_lambda", 0.5, 1.0), # default=1
        "max_depth": trial.suggest_int("max_depth", 9, 13),
        "colsample_bytree": trial.suggest_uniform("colsample_bytree", 0.4, 0.6), # default=0
        "colsample_bylevel": trial.suggest_uniform("colsample_bylevel", 0.1, 0.4),
        #"subsample": trial.suggest_uniform("subsample", 0.5, 0.8), # default=1
        "gamma": trial.suggest_uniform("gamma", 0.4, 0.7), # default=0
        #"min_child_weight": trial.suggest_uniform("min_child_weight", 5, 10), # default=1
    }


                                                                                    # 진행상황 보고싶을때 None을 100으로
    xgbmodel = xgb.train(params_xgb, dtrain, 8000, watchlist, early_stopping_rounds=200, feval=wrmsse_xgb, verbose_eval=None)

    wrmsse = evaluator.score(xgbmodel.predict(dvalid).reshape(28, -1).T)

    return wrmsse
```

```
sampler = TPESampler(seed=42)
xgb_study = optuna.create_study(study_name="xgbm_parameter_opt", direction="minimize", sampler=sampler)
xgb_study.optimize(xgb_objective, n_trials=30)

xgb_best_hyperparams = xgb_study.best_trial.params
xgb_base_hyperparams = {"random_state": 91373, "verbose": None, "objective": "reg:tweedie", "max_bin": 16,
                        "tree_method": 'gpu_hist', "gpu_id": 0}
xgb_best_hyperparams.update(xgb_base_hyperparams)
print("The best hyperparameters are:\n", xgb_best_hyperparams)
```

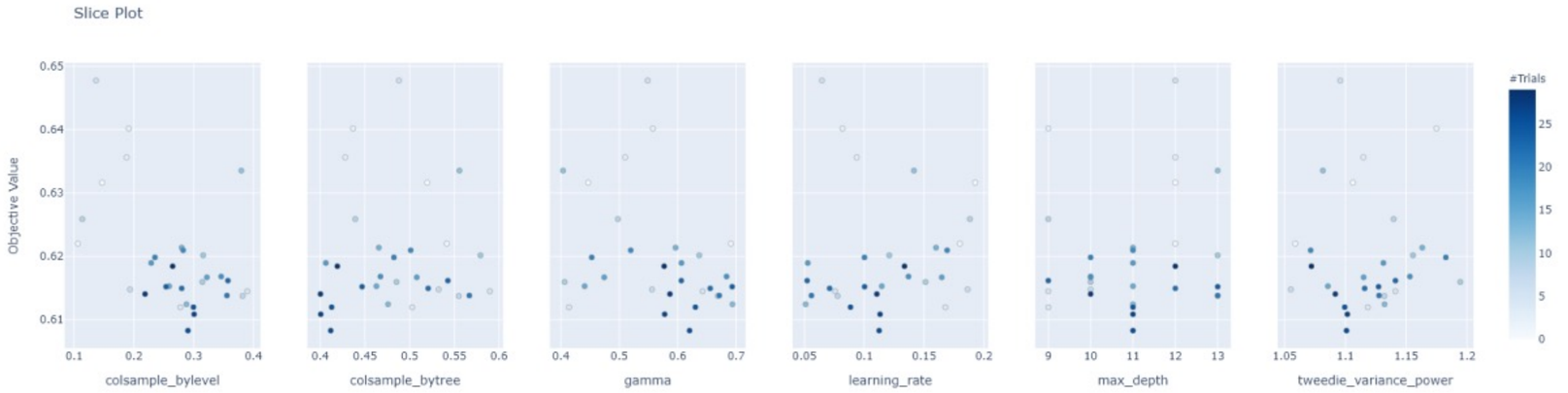
```
[I 2021-06-01 13:14:04,144] Trial 19 finished with value: 0.620695802752 39 and parameters: {'tweedie_variance_power': 1.07140890623293
34, 'learning_rate': 0.16923067624086902, 'max_depth': 11, 'colsample_bytree': 0.5012199340931082, 'colsample_bylevel': 0.2823834635412
114, 'gamma': 0.5195735870607295}. Best is trial 4 with value: 0.6119793337891344.
[I 2021-06-01 13:22:03,174] Trial 20 finished with value: 0.6198552583384508 and parameters: {'tweedie_variance_power': 1.1825546730536
205, 'learning_rate': 0.10018184178171756, 'max_depth': 10, 'colsample_bytree': 0.4824729180836622, 'colsample_bylevel': 0.235165363669
56386, 'gamma': 0.4528797534108698 3}. Best is trial 4 with value: 0.6119793337891344.
[I 2021-06-01 13:34:20,771] Trial 21 finished with value: 0.6138311770636872 and parameters: {'tweedie_variance_power': 1.1276073434057
31, 'learning_rate': 0.05587225107818225, 'max_depth': 13, 'colsample_bytree': 0.5667935971105463, 'colsample_bylevel': 0.3556429533165
585, 'gamma': 0.6710526705672427}. Best is trial 4 with value: 0.6119793337891344.
[I 2021-06-01 13:44:42,390] Trial 22 finished with value: 0.6149626780122763 and parameters: {'tweedie_variance_power': 1.1158114352666
018, 'learning_rate': 0.07105399041410723, 'max_depth': 12, 'colsample_bytree': 0.5207582180481959, 'colsample_bylevel': 0.279960581838
22253, 'gamma': 0.6556037471270 29}. Best is trial 4 with value: 0.6119793337891344.

[I 2021-06-01 13:53:09,056] Trial 23 finished with value: 0.6161825074509008 and parameters: {'tweedie_variance_power': 1.1411240350762
106, 'learning_rate': 0.05225931580965123, 'max_depth': 9, 'colsample_bytree': 0.5427740798951194, 'colsample_bylevel': 0.3572564604981
243, 'gamma': 0.6061469517914544}. Best is trial 4 with value: 0.6119793337891344.
[I 2021-06-01 14:03:04,551] Trial 24 finished with value: 0.6152230120396854 and parameters: {'tweedie_variance_power': 1.1272330687863
46, 'learning_rate': 0.10026443656636096, 'max_depth': 13, 'colsample_bytree': 0.4486901286190844, 'colsample_bylevel': 0.253760309981
00546, 'gamma': 0.6931690459321072}. Best is trial 4 with value: 0.6119793337891344.
[I 2021-06-01 14:11:41,495] Trial 25 finished with value: 0.6119977274399814 and parameters: {'tweedie_variance_power': 1.0994133370484
147, 'learning_rate': 0.06850131933243788, 'max_depth': 11, 'colsample_bytree': 0.4127820023001909, 'colsample_bylevel': 0.299637533481
30107, 'gamma': 0.6302606230953847}. Best is trial 4 with value: 0.6119793337891344.
[I 2021-06-01 14:20:00,127] Trial 26 finished with value: 0.6082970576678911 and parameters: {'tweedie_variance_power': 1.1009005809523
253, 'learning_rate': 0.1124357654715222, 'max_depth': 11, 'colsample_bytree': 0.4117415714476 8747, 'colsample_bylevel': 0.290398441597
4088, 'gamma': 0.6202372226587107}. Best is trial 26 with value: 0.6082970576678911.
[I 2021-06-01 14:28:44,555] Trial 27 finished with value: 0.6108758160197909 and parameters: {'tweedie_variance_power': 1.1015759299850
758, 'learning_rate': 0.11319882963936609, 'max_depth': 11, 'colsample_bytree': 0.4005951387143885, 'colsample_bylevel': 0.300717676841
51506, 'gamma': 0.5776513456832142}. Best is trial 26 with value: 0.6082970576678911.
[I 2021-06-01 14:36:27,057] Trial 28 finished with value: 0.6140707074997325 and parameters: {'tweedie_variance_power': 1.0916920453671
98, 'learning_rate': 0.11048295281966693, 'max_depth': 10, 'colsample_bytree': 0.40027137837009613, 'colsample_bylevel': 0.218641769956
33916, 'gamma': 0.5866129166746458}. Best is trial 26 with value: 0.6082970576678911.
[I 2021-06-01 14:45:04,674] Trial 29 finished with value: 0.6184582965003985 and parameters: {'tweedie_variance_power': 1.0720384531391
098, 'learning_rate': 0.13362058024330303, 'max_depth': 12, 'colsample_bytree': 0.41915196525097437, 'colsample_bylevel': 0.26477307044
53371, 'gamma': 0.5767545467959321}. Best is trial 26 with value: 0.6082970576678911.

The best hyperparameters are:
 {'tweedie_variance_power': 1.1009005809523253, 'learning_rate': 0.1124357654715222, 'max_depth': 11, 'colsample_bytree': 0.41174157144
768747, 'colsample_bylevel': 0.2903984415974068, 'gamma': 0.6202372226587107, 'random_state': 91373, 'verbose': None, 'objective': 're
g:tweedie', 'max_bin': 16, 'tree_method': 'gpu_hist', 'gpu_id': 0}
```

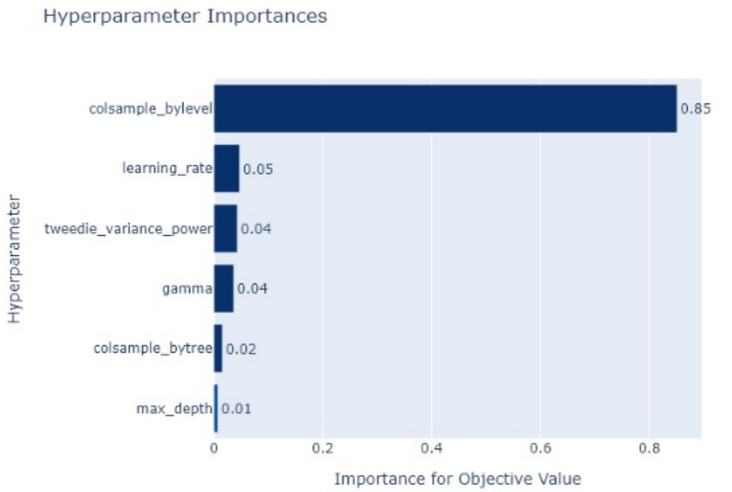
05 Modeling

01. Time Series Modeling

02 Tree Modeling

XGBoost





05 Modeling

01. Time Series Modeling

02 Tree Modeling

LightGBM

Train

Best Hyperparameters

'objective': 'reg:tweedie',
'max_bin': 16,
'tree_method': 'gpu_hist',
'gpu_id': 0
'tweedie_variance_power': 1.1009005809523253,
'learning_rate': 0.1124357654715222,
'max_depth': 11,
'colsample_bytree': 0.41174157144768747,
'colsample_bylevel': 0.2903984415974088,
'gamma': 0.6202372226587107

Training with Best HP

```
xgbmodel = xgb.train(xgb_best_hyperparams, dtrain, 8000, watchlist,
                     early_stopping_rounds=200, feval=wrmsse_xgb, verbose_eval=100)
```

```
[0]     valid-tweedie-nloglik@1.1009:15.38529    valid-wrmsse:7.71275
[100]   valid-tweedie-nloglik@1.1009:14.30073    valid-wrmsse:7.15402
[200]   valid-tweedie-nloglik@1.1009:14.29831    valid-wrmsse:7.16355
[211]   valid-tweedie-nloglik@1.1009:14.29814    valid-wrmsse:7.16379
```

```
evaluator.score(xgbmodel.predict(dvalid).reshape(28, -1).T)
```

```
0.6082970576678911
```

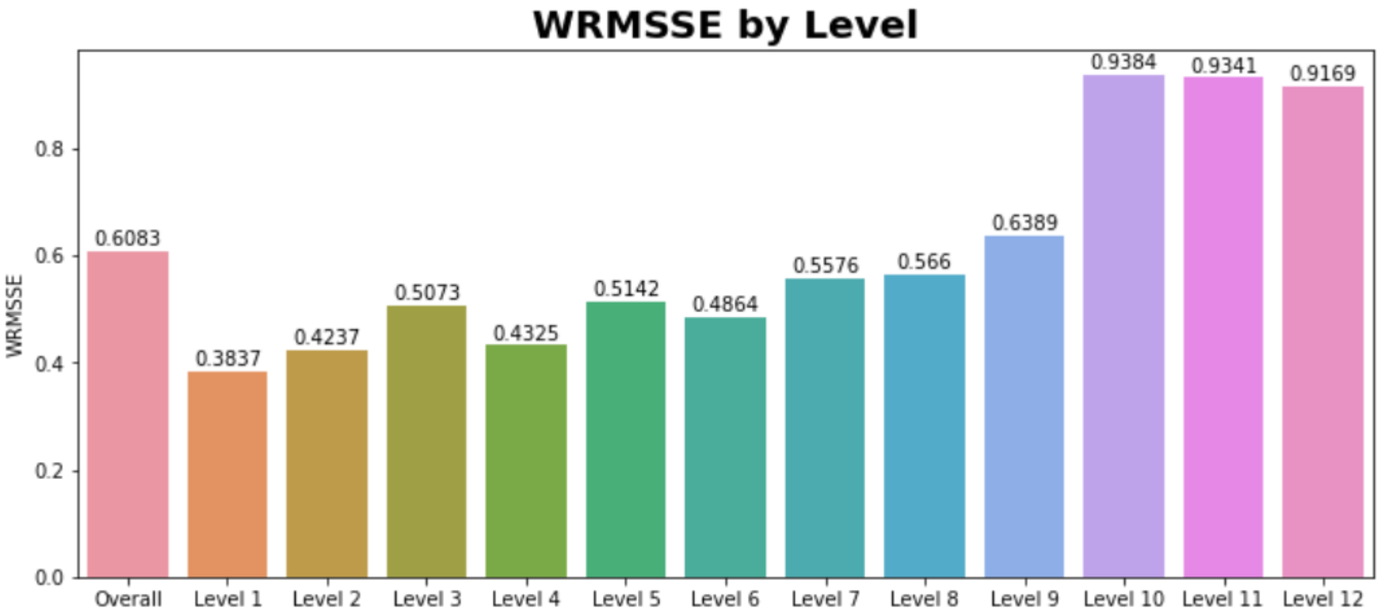
05 Modeling

01. Time Series Modeling

02 Tree Modeling

XGBoost

Evaluation



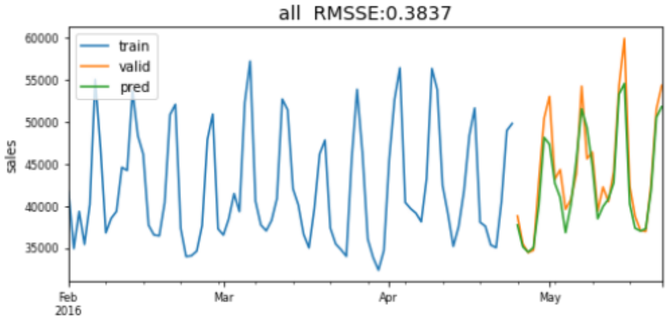
05 Modeling

01. Time Series Modeling

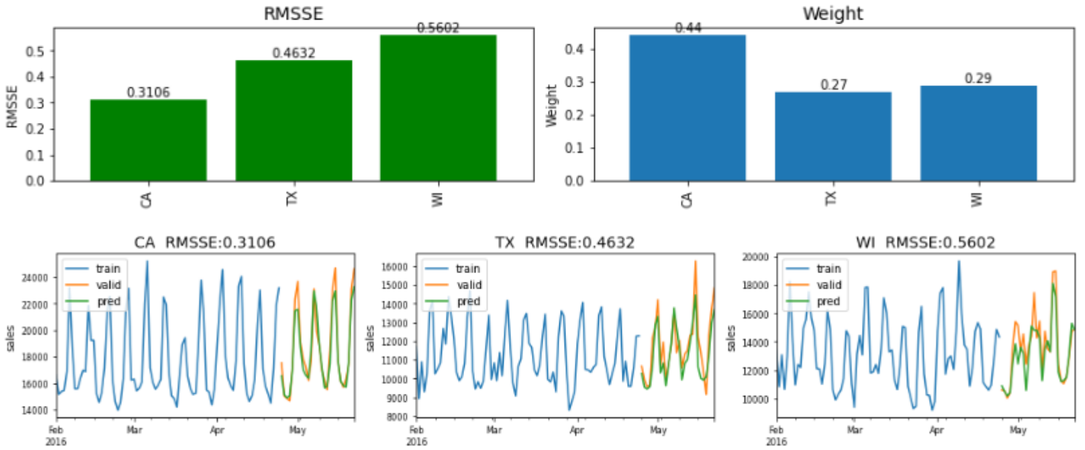
02 Tree Modeling

XGBoost

Level 1: all_id



Level 2: state_id



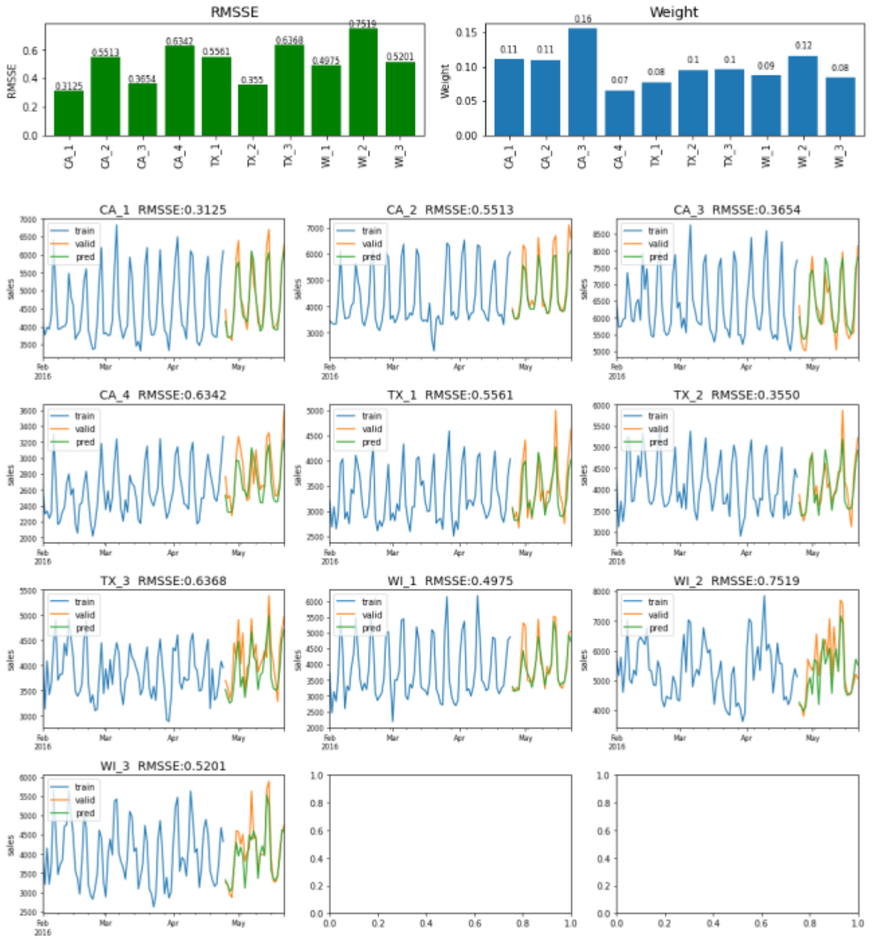
05 Modeling

01. Time Series Modeling

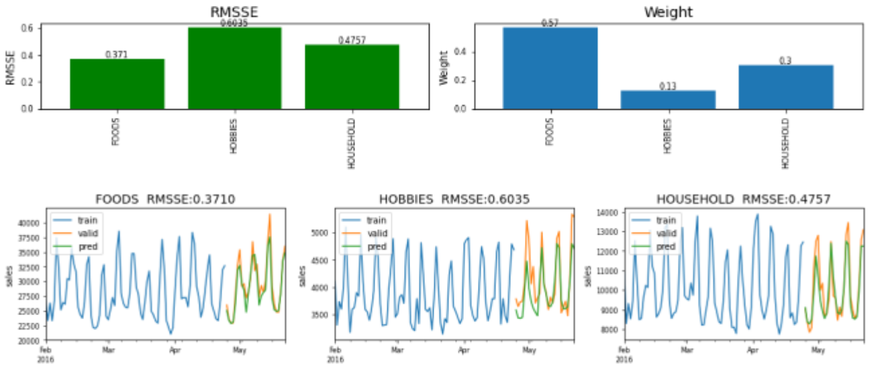
02 Tree Modeling

XGBoost

Level 3: store_id



Level 4: cat_id



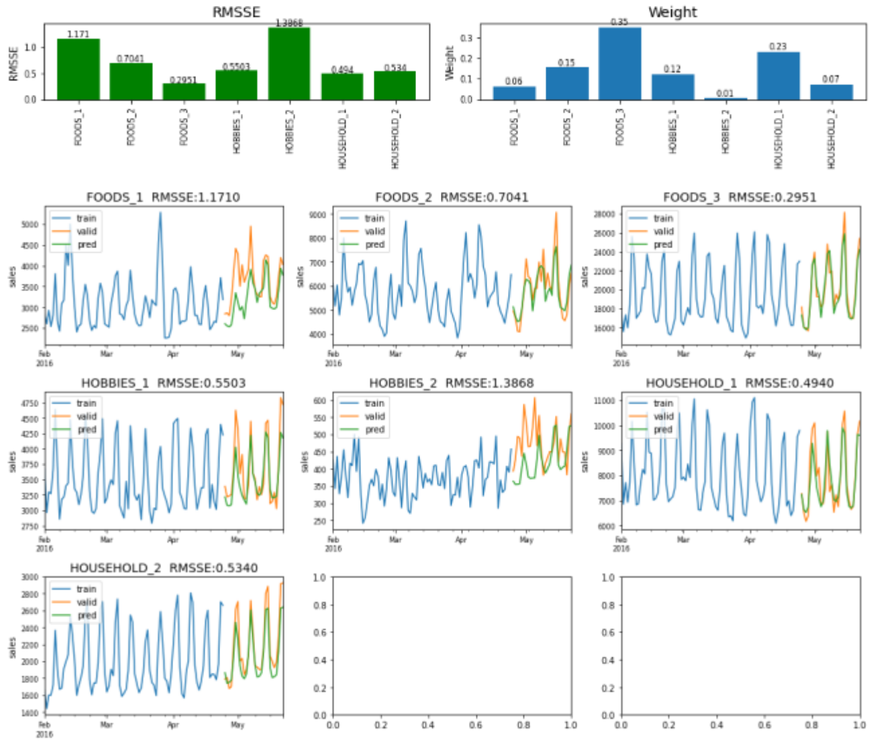
05 Modeling

01. Time Series Modeling

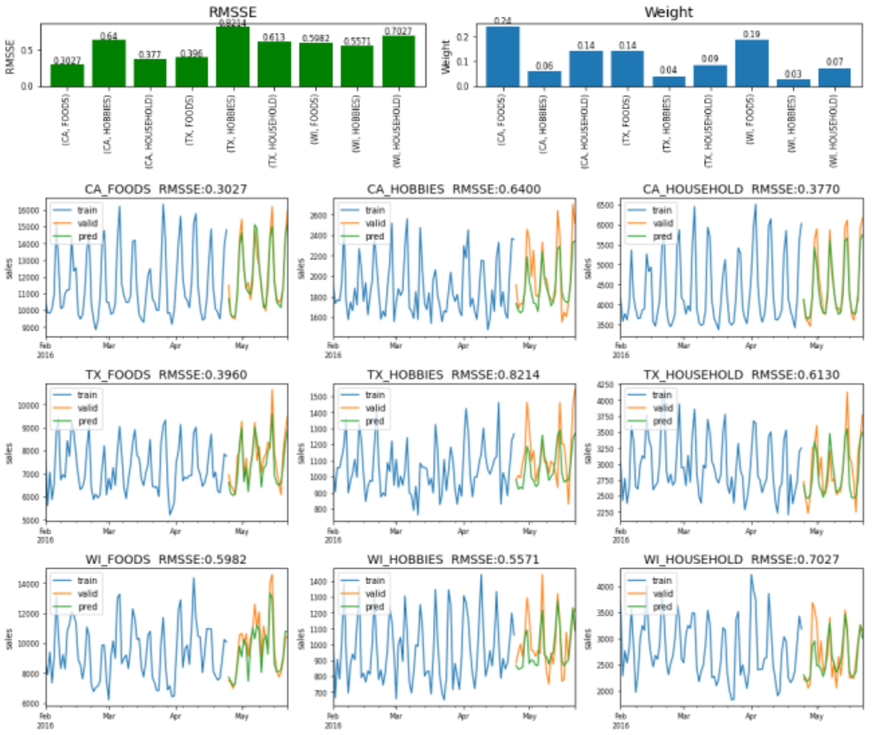
02 Tree Modeling

XGBoost

Level 5: dept_id



Level 6: ['state_id', 'cat_id']



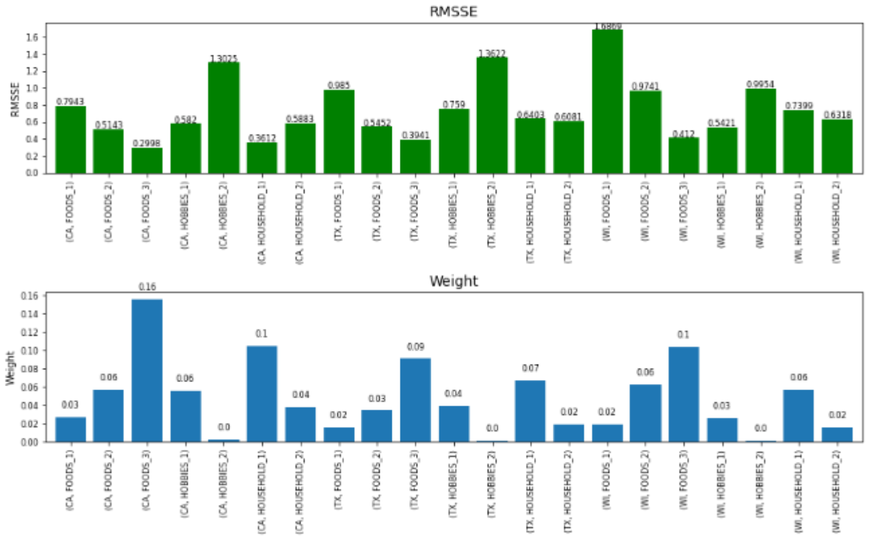
05 Modeling

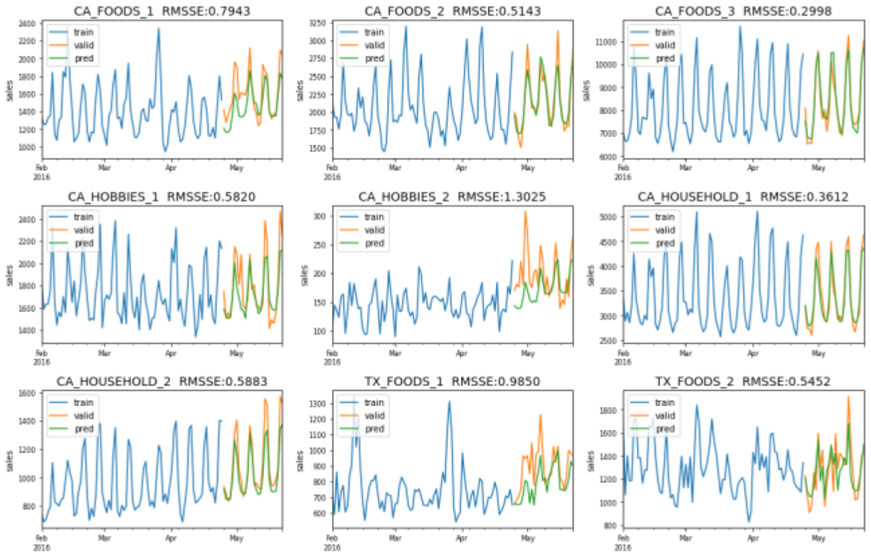
01. Time Series Modeling

02 Tree Modeling

XGBoost

Level 7: ['state_id', 'dept_id']





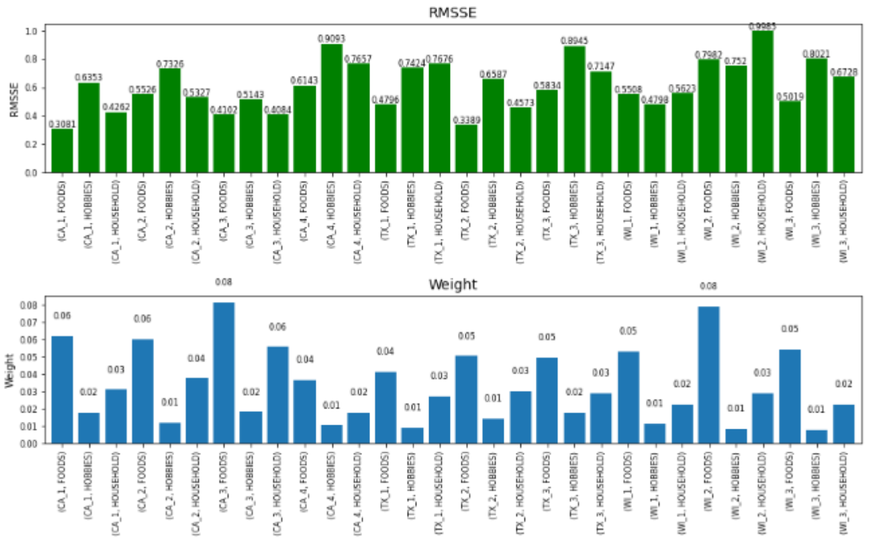
05 Modeling

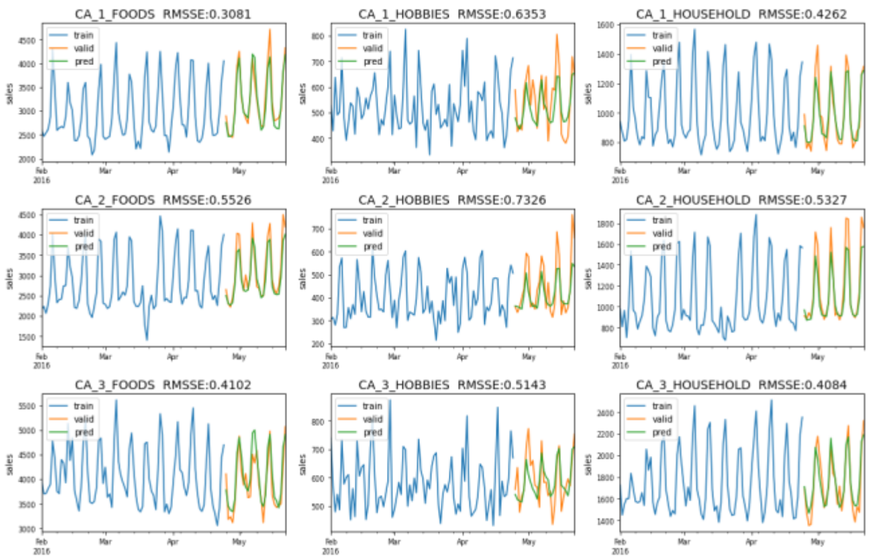
01. Time Series Modeling

02 Tree Modeling

XGBoost

Level 8: ['store_id', 'cat_id']





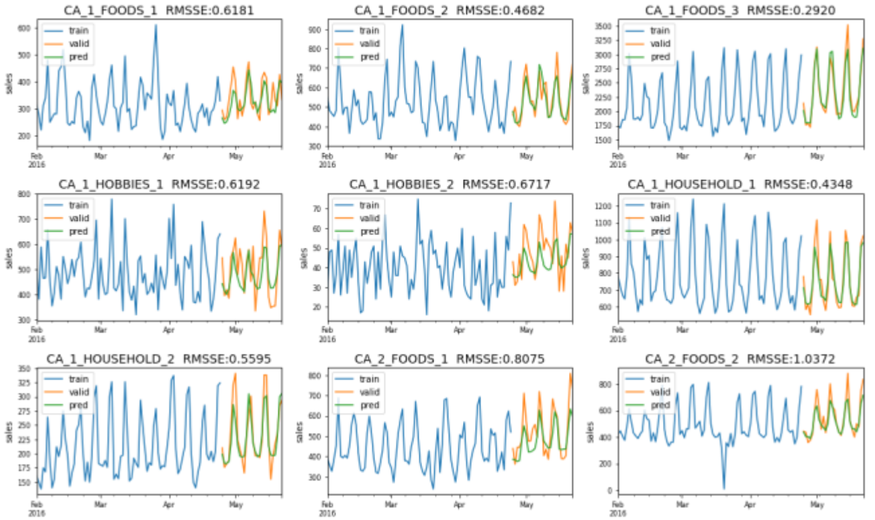
05 Modeling

01. Time Series Modeling

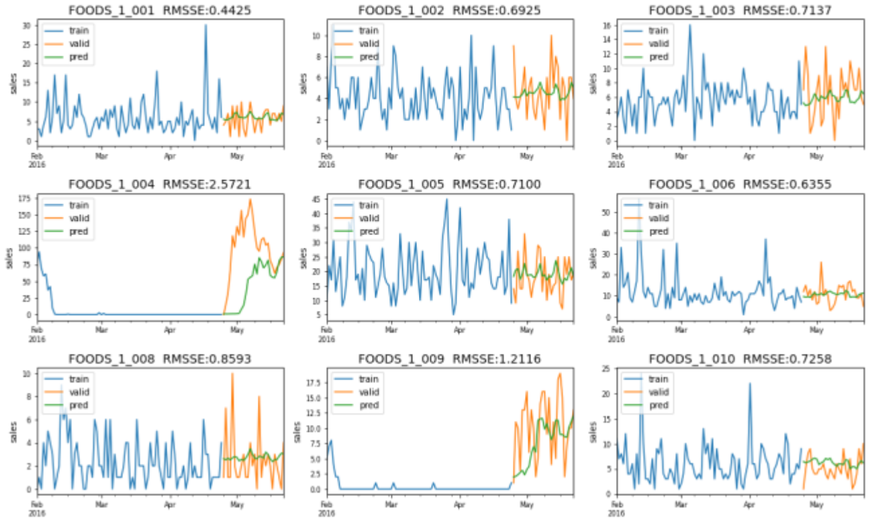
02 Tree Modeling

XGBoost

Level 9: ['store_id', 'dept_id']



Level 10: item_id



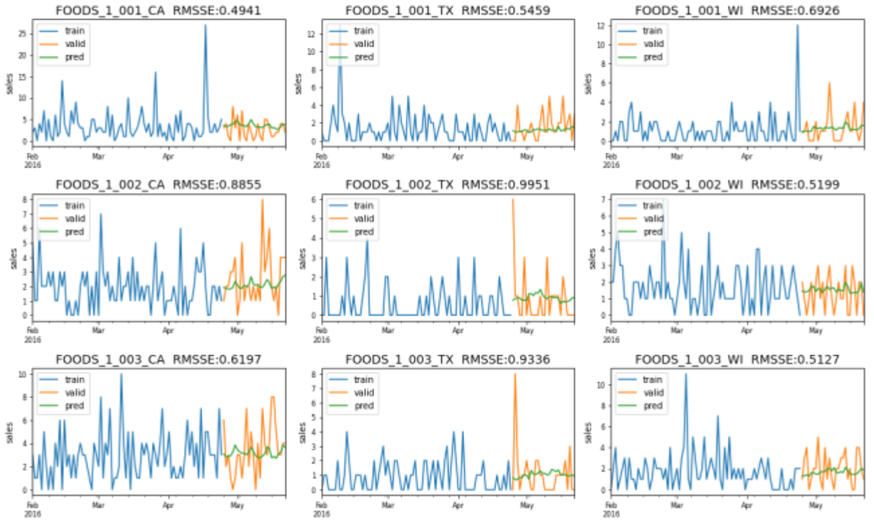
05 Modeling

01. Time Series Modeling

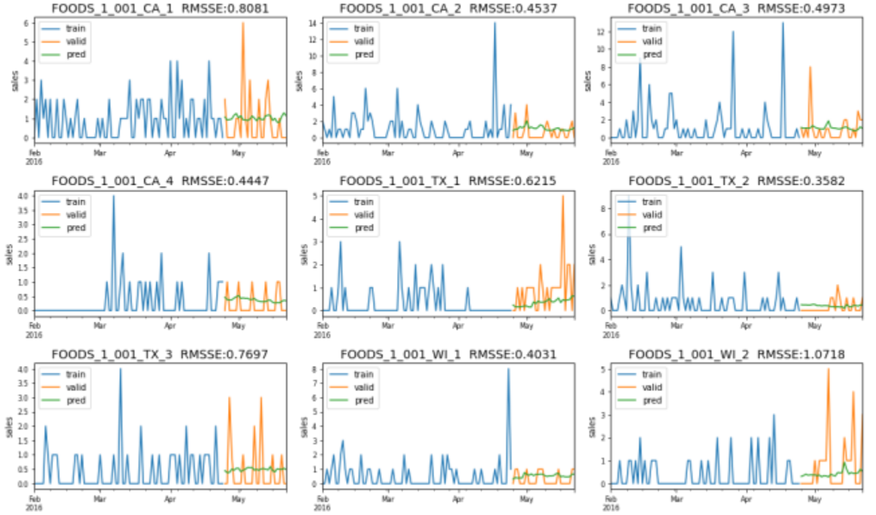
02 Tree Modeling

XGBoost

Level 11: ['item_id', 'state_id']



Level 12: ['item_id', 'store_id']



05 Modeling

01. Time Series Modeling

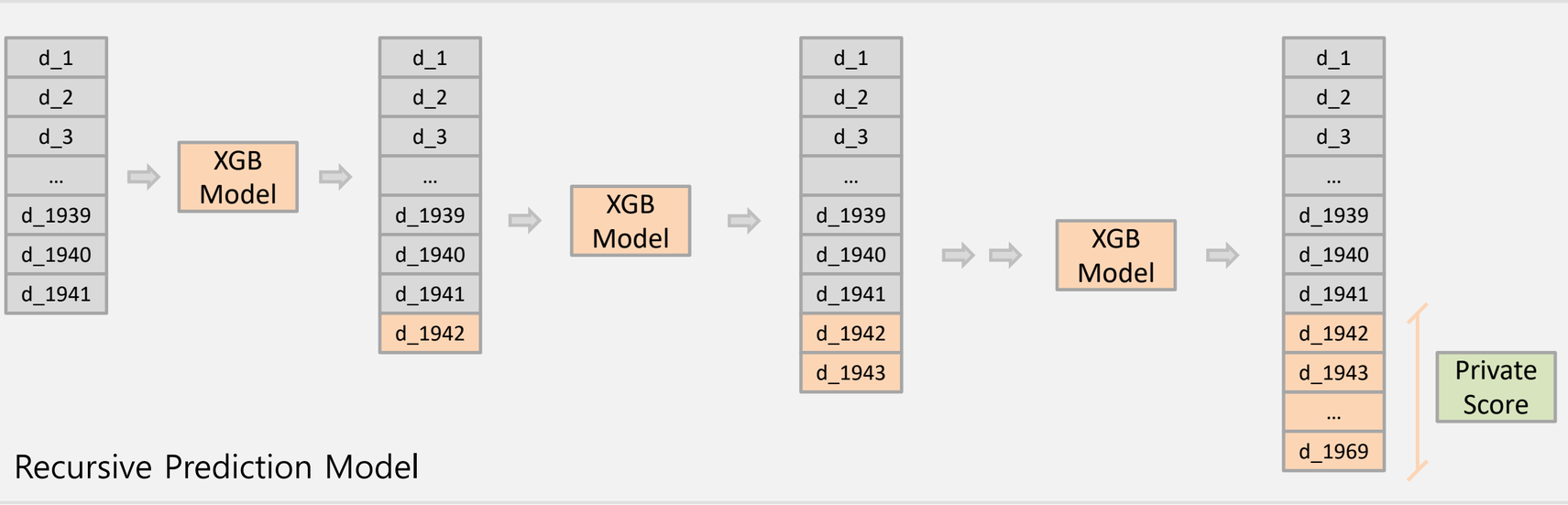
02 Tree Modeling

XGBoost

Predict

```
cols = [f"F{i}" for i in range(1,29)]
test["date"] = pd.to_datetime(test["date"])
df = train.copy()
df_enc = df[['item_id', 'store_id', 'cat_id', 'state_id']]
for col in df_enc.columns:
    df_enc[col] = enc.fit(df_enc[col])
    test[col] = enc.transform(test[col])
```

```
for time_delta in range(0, 28):
    day = datetime(2016, 5, 23) + timedelta(days=time_delta)
    print("Predict", day.date())
    tst = test[(test.date >= day - timedelta(days=60)) & (test.date <= day)].copy()
    lags_wins(tst)
    tst = tst.loc[tst.date == day , train_cols]
    test.loc[test.date == day, "sales"] = xgbmodel.predict(xgb.DMatrix(tst))
    del(tst)
```



Recursive Prediction Model

06

Ensemble

Blending

06 Ensemble

Blending

Blending

	Model	Public LB
TS Model	Simple Moving Average	1.09
	Simple Exponential Smoothing	1.20
	Double Exponential Smoothing	1.11
	Prophet	1.89
	ARIMA	1.54
DT Model	AutoLGB	0.662
	LightGBM	0.579
	XGBoost	0.634
NN Model	RNN	0.823
	LSTM	0.762
	Ensemble	0.531

06 Ensemble

Blending

Blending

To get the advantages of each model, we gave custom weight to models.

SMA	DES	LGB	XGB	LB(public)
0.2	0.2	0.3	0.3	0.973
0.15	0.15	0.4	0.3	0.896
0.15	0.15	0.5	0.2	0.864
0.1	0.1	0.6	0.2	0.744
0.1	0.1	0.8	0.0	0.582
0.1	0.1	0.7	0.1	0.531

```
evaluator.score(valid_pred_y_sma * 0.15 +
                valid_pred_y_des * 0.15 +
                valid_pred_y_lgb * 0.5 +
                valid_pred_y_xgb * 0.2)
```

0.8636235351355352

```
evaluator.score(valid_pred_y_sma * 0.1 +
                valid_pred_y_des * 0.1 +
                valid_pred_y_lgb * 0.6 +
                valid_pred_y_xgb * 0.2)
```

0.6112196129462661

```
evaluator.score(valid_pred_y_sma * 0.1 +
                valid_pred_y_des * 0.1 +
                valid_pred_y_lgb * 0.7 +
                valid_pred_y_xgb * 0.1)
```

0.5309244147868056

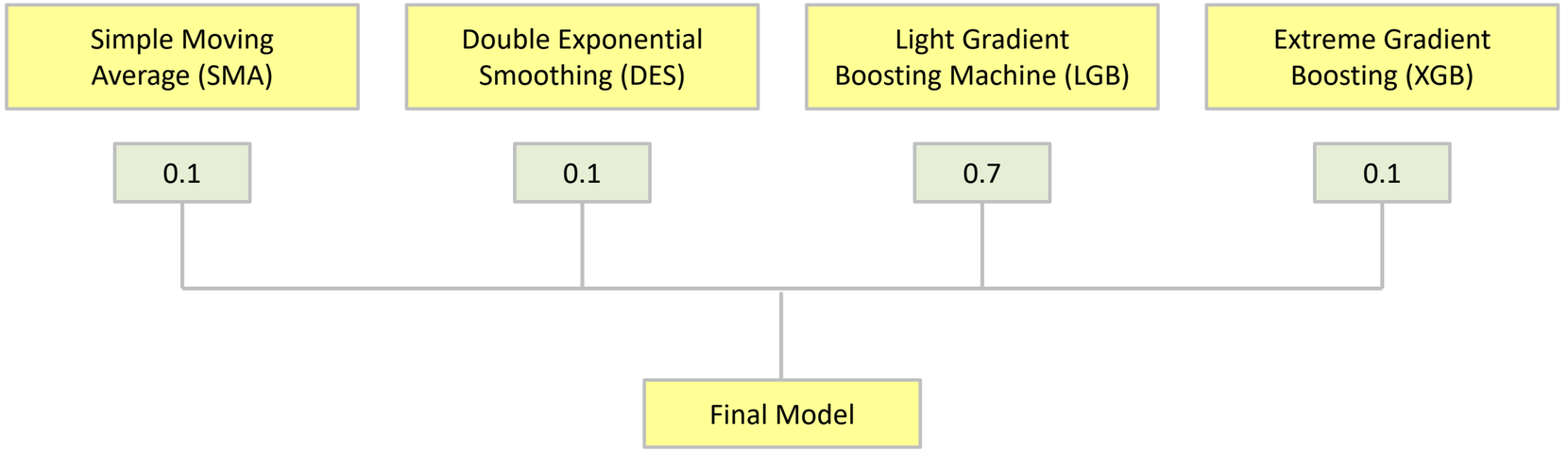
06 Ensemble

Blending

Blending

To get the advantages of each model, we gave custom weight to models.

```
test_pred_y_final = (
    test_pred_y_sma * 0.1 +
    test_pred_y_des * 0.1 +
    test_pred_y_lgb * 0.7 +
    test_pred_y_xgb * 0.1
                    )
```



07

Other Trials

07 Other Trials

Other Trials

Calendar Features

- Event day와 Snap day를 원 핫 인코딩, 레이블 인코딩하여 학습 시도
- 4주를 한 주 씩 나누어 새로운 feature 생성하여 학습 시도

Lag Features

- Lag를 (7, 14, 28)이 아닌 (1, 7, 14, 28)로 놓고 학습 -> no recursive에는 성능이 좋으나 recursive 예측에는 성능 저하
- window size (7, 14, 28)에 60을 추가하여 학습

Blending

시계열 모델 가중치는 0.2일 때, 트리 모델 가중치는 0.8일 때만 예측 정확도 상승

Tree Modeling

- objective를 tweedie가 아닌 rmse와 poisson으로 학습
- wrmsse custom metric 없이 early stopping 시도하거나 early stopping 없이 학습
- Train set을 4년+58일치 데이터가 아닌 단순 4년치 데이터, 5년치 데이터로 설정
- Optuna로 HP튜닝을 할 때 early stopping metric을 wrmsse가 아닌 rmse로 두고 튜닝

Neural Network Modeling

RNN, LSTM을 시도하였으나 학습시간 대비 성능이 좋지 않으므로 제거 (blending을 하여도 성능 저하)