온라인 채널 제품 판매량 예측 AI 온라인 해커톤

Team KyleYeo

여훈기

최재무

정찬영

신건우

Contents

Setting

Import

```python
import random
import os
import multiprocessing
from tqdm.auto import tqdm
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import spatial

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
```

Google Drive Mount

```python
from google.colab import drive

drive.mount('/content/drive')
```

Hyperparameter Setting

```python
CFG = {
    'TRAIN_WINDOW_SIZE':60, # 60일치로 학습
    'PREDICT_SIZE':21, # 21일치 예측
    'EPOCHS':20,
    'LEARNING_RATE':0.000075,
    'BATCH_SIZE':1024,
    'SEED':777
}
```

Colab python version : 3.10.12

```python
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
```

Versions

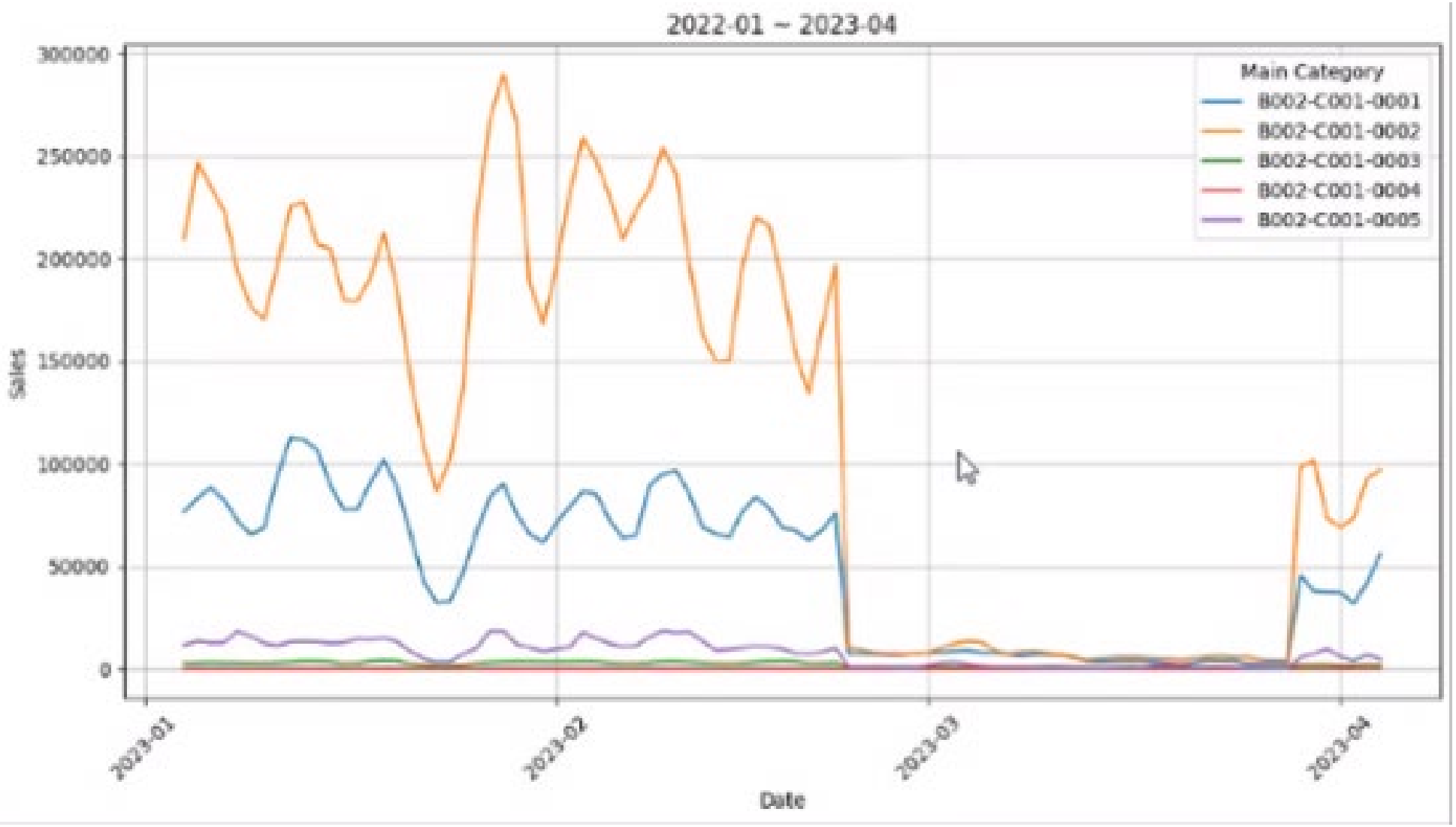
- pandas - 1.5.3
- numpy - 1.23.5
- matplotlib - 3.7.1
- tqdm - 4.66.1
- sklearn - 1.2.2
- scipy - 1.10.1
- torch - 2.0.1+cu118 ver.

OS

- Google Colab Pro+
- GPU - V100
- 고용량 RAM 사용

Data Interpolation

데이터 보간 전 그래프


2022-01 ~ 2023-04
Main Category
B002-C001-0001
B002-C001-0002
B002-C001-0003
B002-C001-0004
B002-C001-0005
Sales
300000
250000
200000
150000
100000
50000
0
2023-01
2023-02
2023-03
2023-04
Date

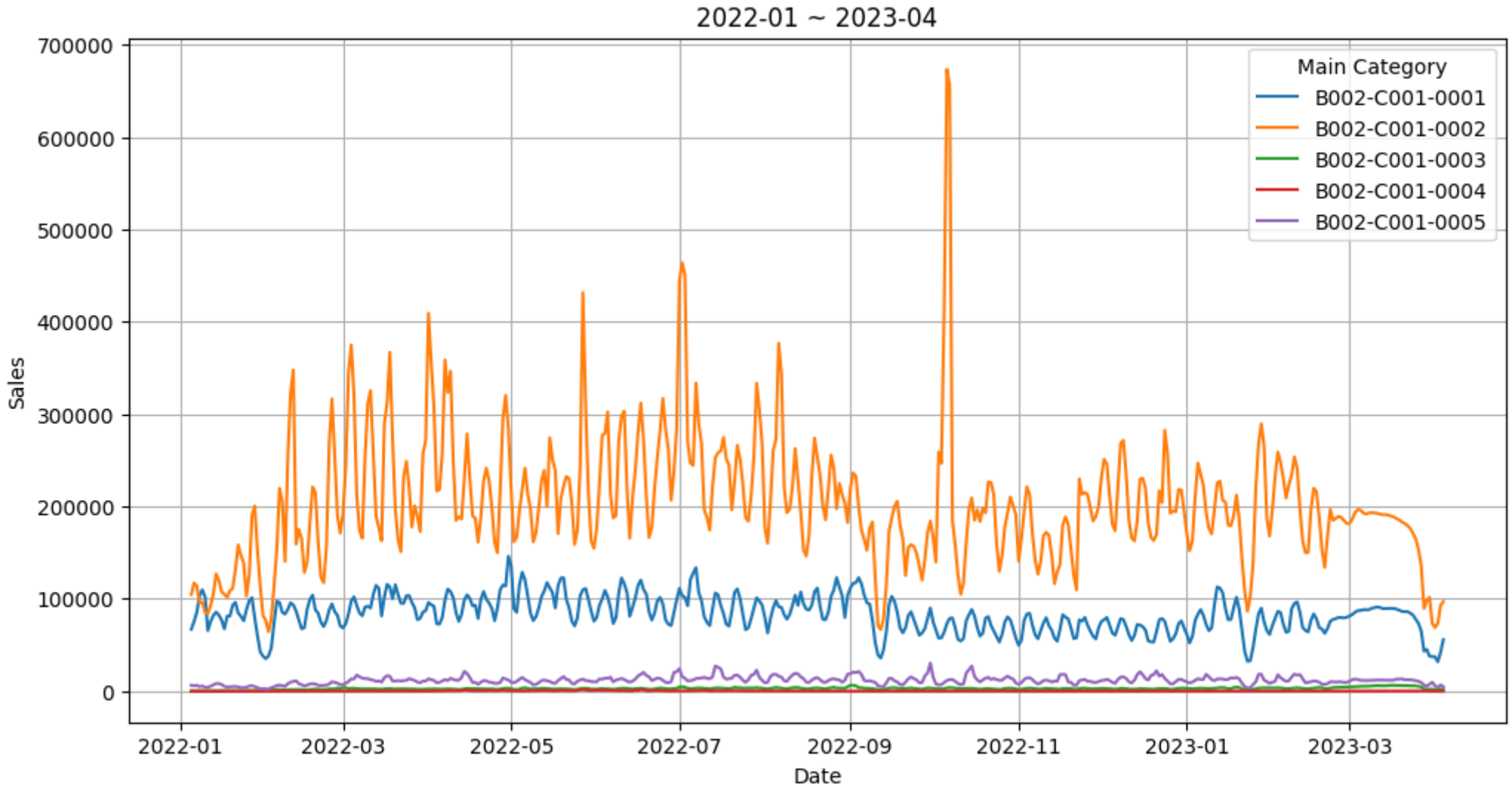
Data Interpolation

EDA를 진행하는 도중 결측치로 의심되는 구간을 발견했다. 내부 토의로 보간하기로 결정하고,
여러 방식으로 보간을 시도한 결과, Moving Average + previous & next values를 이용하여
Simplicity하게 보간된 데이터가 가장 좋은 성능을 보였다.

interpolation

```
for i in range(0, 15890):
  a, b = 414, 424 # 결측 구간 이전 10일 구간
  ee = 34 # 결측 일수

  z1 = train_df.iloc[i, -7:].sum() # 결측 구간 이후 7일동안의 값 더함
  z1 = round(z1 / 7) # 7일동안의 값 평균 계산

  for j in range(1, 35): # 결측 구간 보간 진행
    z = train_df.iloc[i, a:b].sum() # 결측 구간 이전 10일동안의 값 더함.
    z = round(z / 10) # 10일동안의 값 평균 계산
    sum = round(z - ((z - z1) / ee)) # 결측 이전 시점의 평균값과 결측 이후 시점의 평균값을 뺀 상태에서 남은 보간 일수만큼 나누어줌. 이후 결측 이전 시점의 평균값에 빼줌
    if sum <= 0: # 0일 경우 넘어감.
      continue
    else:
      train_df.iloc[i, b] += sum # 결측 구간에 계산된 값 삽입
    a += 1
    b += 1
    ee -= 1

train_df.tail(20)
```

데이터 보간 후 그래프


2022-01 ~ 2023-04
Main Category
B002-C001-0001
B002-C001-0002
B002-C001-0003
B002-C001-0004
B002-C001-0005
Sales
700000
600000
500000
400000
300000
200000
100000
0
2022-01
2022-03
2022-05
2022-07
2022-09
2022-11
2023-01
2023-03
Date

Feature Engineering

Feature Engineering

추가한 feature는 다음과 같다.

brand

brand_keyword_cnt 데이터 활용

마지막 3주 간의
관련 키워드 언급 횟수를
정규화하여 평균

Popularity1,2

Popularity3,4

월별 판매량 기반으로 브랜드 충성도를 정의

Approach1:
월별 평균 판매량이 높을수록
브랜드의 충성도는 높을 것이다.

Approach2:
월별 평균 판매량이 높고 일정할수록
브랜드의 충성도는 높을 것이다.

→ 베이스라인 train 기간인
90일 기준으로 평균을 계산

**상관관계 분석을 통해,
Popularity1,4를 최종 선정**

‘brand’ feature

brand_keyword_cnt(브랜드 키워드 언급량) 데이터와 당일~몇일 후의 판매량과 상관관계가 일부 있음을 확인했다.
여러 Test를 진행하였고, 가장 최근의 Trend에 비중을 두는 것이 정확도 향상에 도움이 될 것이라 판단하여
예측 기간 직전 3주간의 평균을 feature로 사용했다.

```python
# brand_keyword_cnt.csv 마지막 3주 값만 가지는 Dataframe으로 변환

# 원하는 날짜 범위
start_date = '2023-03-15'
end_date = '2023-04-04'

# DataFrame 변환
df = pd.DataFrame(brand_df)
date_columns = pd.date_range(start_date, end_date)
df_selected = df[['브랜드'] + date_columns.strftime('%Y-%m-%d').tolist()]

# DataFrame 생성
df = pd.DataFrame(df_selected)

# '2023-03-05'부터 마지막 컬럼까지의 데이터 선택
date_columns = df.columns[1:]

# 각 행별로 해당 날짜 컬럼들의 값을 정규화한 뒤 평균 계산
normalized_values = df[date_columns].apply(lambda row: MinMaxScaler().fit_transform(row.values.reshape(-1, 1)).mean(), axis=1)

# 정규화된 평균 값을 새로운 컬럼에 추가하되, NaN 값을 0으로 변경
df['brand'] = normalized_values.fillna(0)

df['brand']
```

‘brand’ feature

Train dataframe에 추출된 ‘brand’ feature를 추가한 모습

	ID	제품	대분류	중분류	소분류	브랜드	brand
0	0	B002-00001-00001	B002-C001-0002	B002-C002-0007	B002-C003-0038	B002-00001	0.562504
1	1	B002-00002-00001	B002-C001-0003	B002-C002-0008	B002-C003-0044	B002-00002	0.301587
2	2	B002-00002-00002	B002-C001-0003	B002-C002-0008	B002-C003-0044	B002-00002	0.301587
3	3	B002-00002-00003	B002-C001-0003	B002-C002-0008	B002-C003-0044	B002-00002	0.301587
4	4	B002-00003-00001	B002-C001-0001	B002-C002-0001	B002-C003-0003	B002-00003	0.316015

‘popularity1,4’ feature

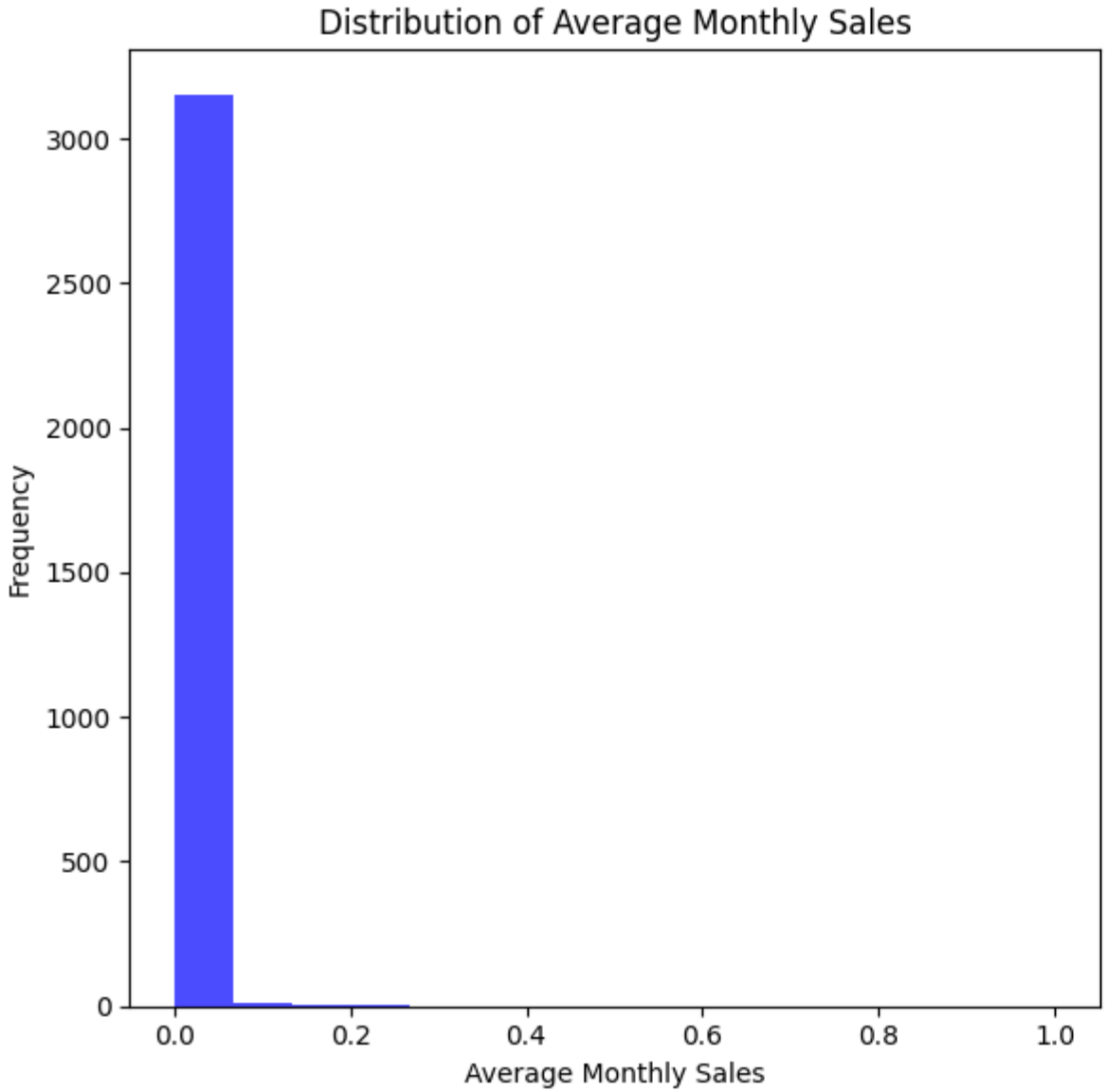
```python
# 브랜드 충성도 feature engineering

# - Approach1: 월별 평균 판매량이 높을수록 브랜드의 충성도는 높을 것이다.
#     - 충성도 측정 방법: 브랜드별 월별 판매량 및 평균 판매량 계산 후 정규화 -> 값이 높을수록 충성도가 높은 브랜드임을 가정

# - Approach2: 월별 평균 판매량이 높고 일정할수록 브랜드의 충성도는 높을 것이다.
#     - 충성도 측정 방법: 평균 월별 판매량과 CV (Coefficient of Variation) 기반으로 각각 가중치 적용 후 weighted mean 계산 -> 값이 높을수록 충성도가 높은 브랜드임을 가정

class brand_popularity:
    def __init__(self, df: pd.DataFrame, w1, w2, target_df: pd.DataFrame):
        '''
        df: 브랜드별 월별 판매량이 산출된 df,
        w1: 월별 판매량에 적용할 가중치
        w2: 월별 판매량의 일정함(표준편차)에 적용할 가중치
        target_df: 계산된 feature를 mapping할 df
        '''
        self.df =df
        self.w1 = w1
        self.w2 = w2
        self.target_df = target_df

    def sales_based_popularity(self):
        '''
        브랜드별 월별 평균 판매량을 활용한 충성도 점수
        '''
        average_monthly_sales = self.df.mean()
        normalized_popularity = average_monthly_sales / average_monthly_sales.max()
        return normalized_popularity
```

‘popularity1,4’ feature

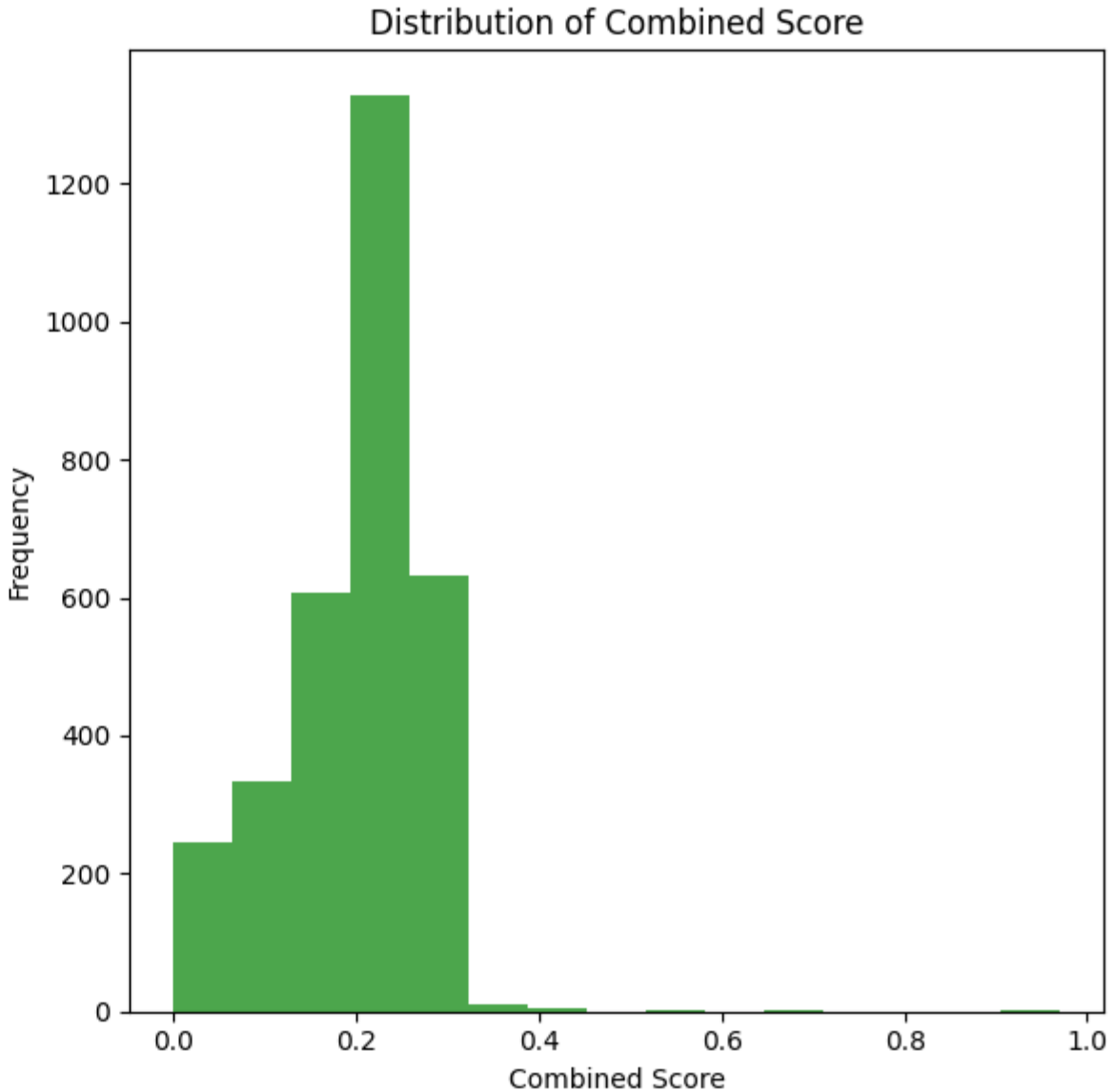
```python
def mixed_based_popularity(self):
    '''
    월별 판매량과 판매량의 표준편차 (cv)를 활용한 충성도 점수
    '''
    average_monthly_sales = self.df.mean()
    normalized_popularity = average_monthly_sales / average_monthly_sales.max()
    cv = self.df.std() / self.df.mean()
    normalized_cv = cv / cv.max()

    combined_score = (self.w1 * normalized_popularity) + (self.w2 * (1 - normalized_cv))
    return combined_score
```

‘popularity1,4’ feature

popularity1

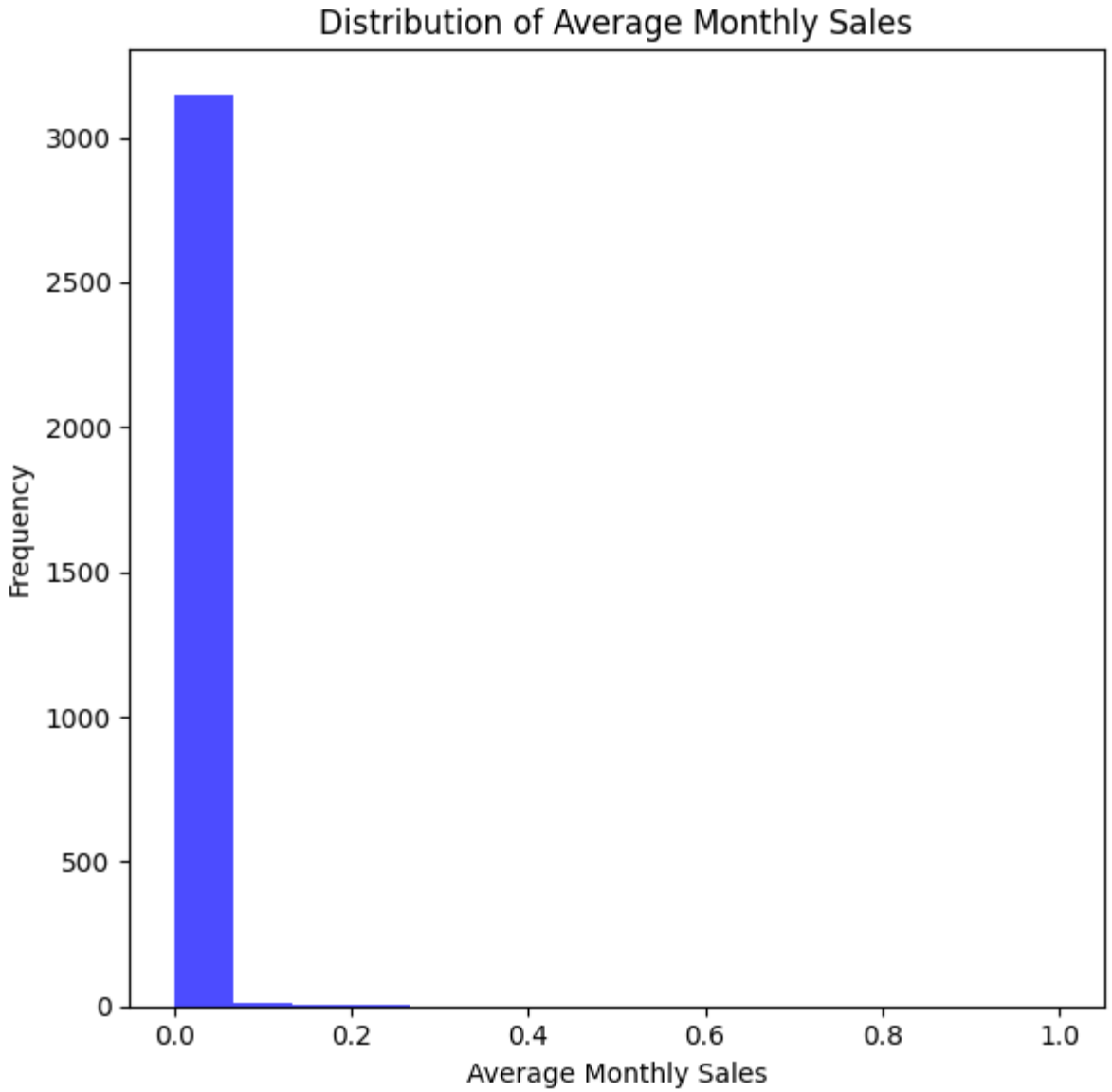


popularity2

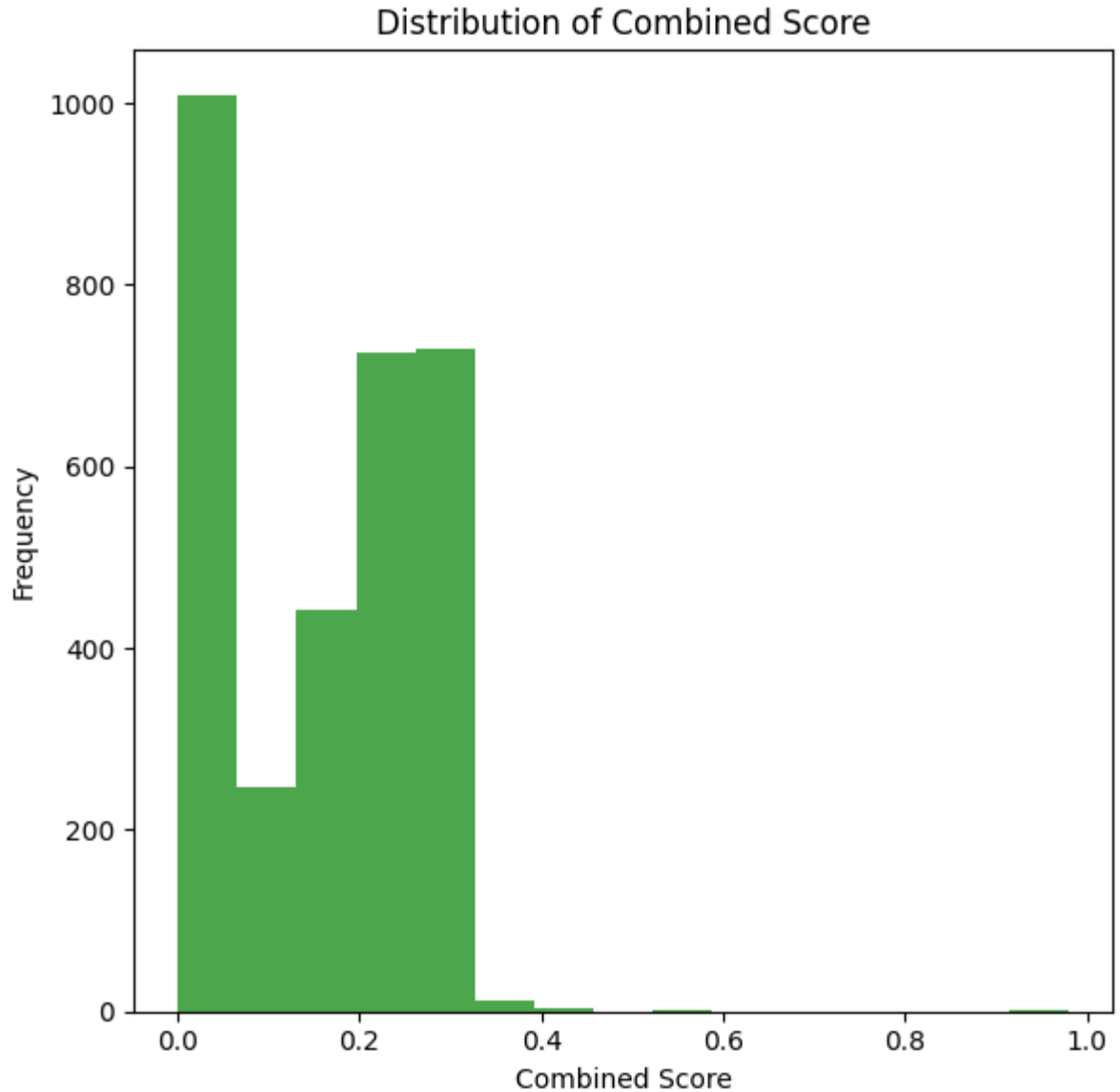


‘popularity1,4’ feature

popularity3



popularity4



‘popularity1,4’ feature

```
1 # Pearson correlation matrix
2 df_features.corr()
```

	popularity1	popularity2	popularity3	popularity4	total_sales	avg_sales
popularity1	1.000000	0.854671	0.975166	0.692979	0.243066	0.243066
popularity2	0.854671	1.000000	0.839259	0.858462	0.214701	0.214701
popularity3	0.975166	0.839259	1.000000	0.707194	0.204916	0.204916
popularity4	0.692979	0.858462	0.707194	1.000000	0.159251	0.159251
total_sales	0.243066	0.214701	0.204916	0.159251	1.000000	1.000000
avg_sales	0.243066	0.214701	0.204916	0.159251	1.000000	1.000000

```
1 # 비선형 관계를 파악하기 위해 각 feature와 브랜드별 총 판매량 사이의 Distance correlation coefficients 계산합니다
2 display(spatial.distance.correlation(df_features['popularity1'], df_features['total_sales']))
3 display(spatial.distance.correlation(df_features['popularity2'], df_features['total_sales']))
4 display(spatial.distance.correlation(df_features['popularity3'], df_features['total_sales']))
5 display(spatial.distance.correlation(df_features['popularity4'], df_features['total_sales']))
```

```
0.7569341821112185
0.7852989927187264
0.7950839016496316
0.8407490358050872
```

```
1 # 비선형 관계를 파악하기 위해 각 feature와 브랜드별 평균 판매량 사이의 Distance correlation coefficients 계산합니다
2 display(spatial.distance.correlation(df_features['popularity1'], df_features['avg_sales']))
3 display(spatial.distance.correlation(df_features['popularity2'], df_features['avg_sales']))
4 display(spatial.distance.correlation(df_features['popularity3'], df_features['avg_sales']))
5 display(spatial.distance.correlation(df_features['popularity4'], df_features['avg_sales']))
```

```
0.7569341821112185
0.7852989927187264
0.7950839016496316
0.8407490358050872
```

- Popularity1
  판매량과 그나마 가장 강한 선형적인 관계를 나타냄
  (0.24로 의미부여하기 어려운 상관계수이긴 하지만)
  또 강한 비선형 관계를 나타냄 (0.76)

- Populatiy4
  판매량과 가장 강한 비선형 관계를 나타냄 (0.84)

▼

Popularity1,4를 최종 선정

Min-Max Scaling
Log Transformation

Min-Max Scaling

베이스라인의 Min-max scaling을 그대로 적용했다.

```
# 숫자형 변수들의 min-max scaling을 수행하는 코드입니다.
numeric_cols = train_df.columns[7:]
# 각 column의 min 및 max 계산
min_values = train_df[numeric_cols].min(axis=1)
max_values = train_df[numeric_cols].max(axis=1)
# 각 행의 범위(max-min)를 계산하고, 범위가 0인 경우 1로 대체
ranges = max_values - min_values
ranges[ranges == 0] = 1
# min-max scaling 수행
train_df[numeric_cols] = (train_df[numeric_cols].subtract(min_values, axis=0)).div(ranges, axis=0)
# max와 min 값을 dictionary 형태로 저장
scale_min_dict = min_values.to_dict()
scale_max_dict = max_values.to_dict()
```

Log transformation

학습 데이터의 분산 안정화를 위해 로그 변환을 적용했다.
변환된 학습 데이터로 인해 모델의 예측 성능이 좋아졌다.

```python
# Variance 안정화를 위해 로그 변환을 적용합니다.

def log_transformation(df: pd.DataFrame, start_date, offset = float):
    '''
    정상성 변환을 위해 로그 변환을 적용합니다.
    0이 많은 데이터 특성 때문에 일반적인 로그 변환보다는, 전체 데이터에 offset을 더하고 로그 변환을 진행합니다.
    '''
    df_log = df.copy()
    df_log = df_log.iloc[:, df.columns.get_loc(start_date):] + offset
    df_log = np.log(df_log)
    df.iloc[:, df.columns.get_loc('2022-01-01'):]  = df_log

    return df

train_data = log_transformation(train_df, '2022-01-01', 1)
```

학습/검정 데이터 생성

베이스라인의 학습데이터 생성 함수에서 step size를 2로 추가해 RAM을 절약했다.
결과 자체도 step size를 부여한 것이 성능이 더 좋았다.

```python
def make_train_data(data, train_size=CFG['TRAIN_WINDOW_SIZE'], predict_size=CFG['PREDICT_SIZE']):
    STEP_SIZE = 2

    num_rows = len(data)
    window_size = train_size + predict_size
    adjusted_size = (len(data.columns) - window_size + 1) // STEP_SIZE

    input_data = np.empty((num_rows * adjusted_size, train_size, len(data.iloc[0, :7]) + 1))
    target_data = np.empty((num_rows * adjusted_size, predict_size))

    for i in tqdm(range(num_rows)):
        encode_info = np.array(data.iloc[i, :7])
        sales_data = np.array(data.iloc[i, 7:])

        for j in range(0, len(sales_data) - window_size + 1, STEP_SIZE):
            window = sales_data[j: j + window_size]
            temp_data = np.column_stack((np.tile(encode_info, (train_size, 1)), window[:train_size]))
            input_data[i * adjusted_size + j // STEP_SIZE] = temp_data
            target_data[i * adjusted_size + j // STEP_SIZE] = window[train_size:]

    return input_data, target_data
```

학습/검정 데이터 생성

학습/검정 데이터 분할과 DataLoader 또한 베이스라인의 코드를 그대로 사용했다.
CV 또한 적용해보았지만, CV를 적용하지 않은 것이 Public score가 더 높았다.

```
# Train / Validation Split
data_len = len(train_input)
val_input = train_input[-int(data_len*0.2):]
val_target = train_target[-int(data_len*0.2):]
train_input = train_input[:-int(data_len*0.2)]
train_target = train_target[:-int(data_len*0.2)]
```

만들어진 데이터의 shape은 다음과 같다.

```
train_input.shape, train_target.shape, val_input.shape, val_target.shape, test_input.shape
```

```
((2453416, 60, 8),
 (2453416, 21),
 (613354, 60, 8),
 (613354, 21),
 (15890, 60, 8))
```

Model

모델 선언

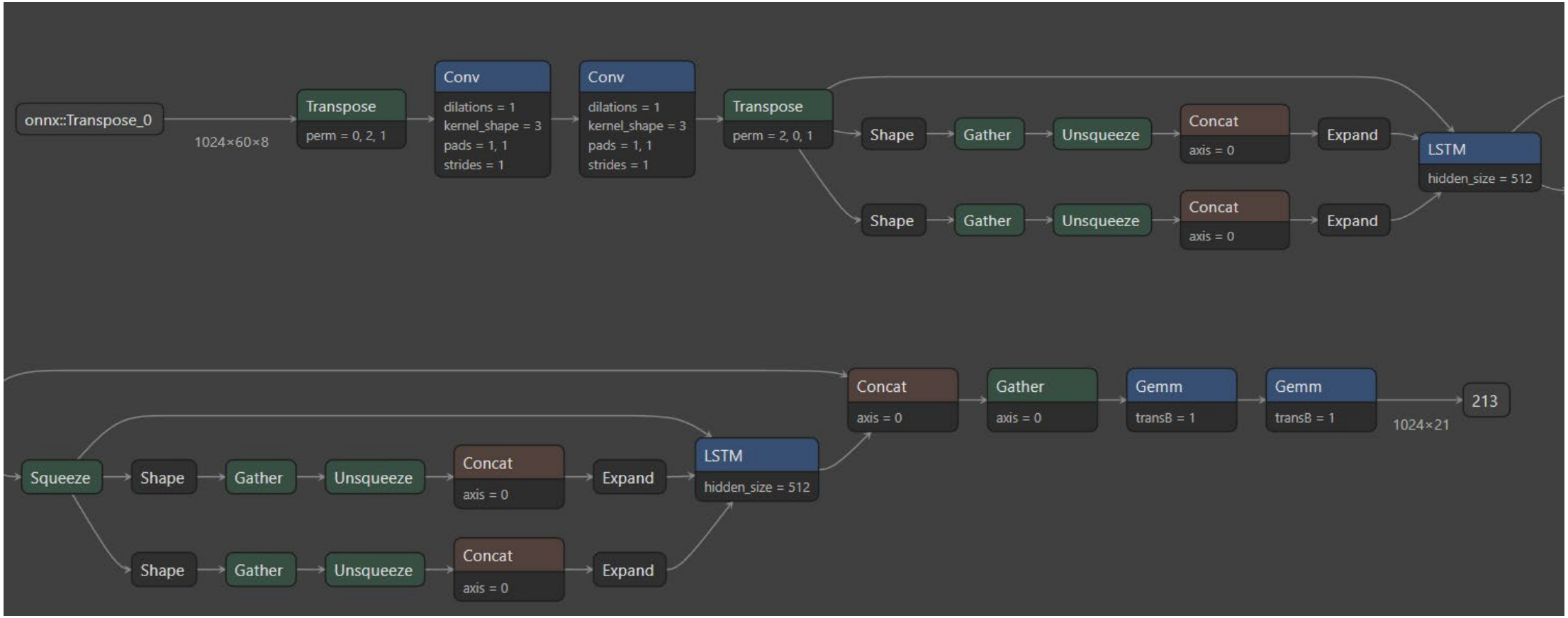
다음과 같이 Conv1d layer와 LSTM layer를 결합한 모델을 만들었다.

```
4  class Conv1dLSTMModel(nn.Module):
5      def __init__(self, input_size=8, hidden_size=512, output_size=CFG['PREDICT_SIZE']):
6          super(Conv1dLSTMModel, self).__init__()
7          self.conv1d_1 = nn.Conv1d(in_channels=input_size,
8                                    out_channels=16,
9                                    kernel_size=3,
10                                   stride=1,
11                                   padding=1)
12         self.conv1d_2 = nn.Conv1d(in_channels=16,
13                                   out_channels=32,
14                                   kernel_size=3,
15                                   stride=1,
16                                   padding=1)
17         self.lstm = nn.LSTM(input_size=32,
18                             hidden_size=hidden_size,
19                             num_layers=2,
20                             bias=True,
21                             bidirectional=False,
22                             batch_first=True)
23         self.dropout = nn.Dropout(0.1)
24
25         self.fc_layer1 = nn.Linear(512, 32)
26         self.fc_layer2 = nn.Linear(32, output_size)
```

```
28     def forward(self, x):
29         x = x.transpose(1, 2)
30
31         x = self.conv1d_1(x)
32
33         x = self.conv1d_2(x)
34
35         x = x.transpose(1, 2)
36
37         self.lstm.flatten_parameters()
38
39         _, (hidden, _) = self.lstm(x)
40
41         x = hidden[-1]
42
43         x = self.dropout(x)
44
45         x = self.fc_layer1(x)
46         x = self.fc_layer2(x)
47
48         return x
```

모델 선언

Netron으로 시각화한 모델 구조다.



모델 학습

```python
def train_with_early_stopping(model, optimizer, train_loader, val_loader, device, patience=3):
    model.to(device)
    criterion = nn.MSELoss().to(device)
    best_loss = float('inf')
    best_model = None
    no_improvement_count = 0  # 변수 초기화

    for epoch in range(1, CFG['EPOCHS']+1):
        model.train()
        train_loss = []

        for X, Y in tqdm(iter(train_loader)):
            X = X.to(device)
            Y = Y.to(device)

            optimizer.zero_grad()

            output = model(X)
            loss = criterion(output, Y)

            loss.backward()
            optimizer.step()

            train_loss.append(loss.item())

        val_loss = validation(model, val_loader, criterion, device)
        print(f'Epoch : [{epoch}] Train Loss : [{np.mean(train_loss):.5f}] Val Loss : [{val_loss:.5f}]')

        if val_loss < best_loss:  # 검증 손실이 감소한 경우
            best_loss = val_loss
            best_model = model
            no_improvement_count = 0  # 카운트 초기화
            print('Model Saved')
        else:
            no_improvement_count += 1

        if no_improvement_count >= patience:  # 일정 epoch 동안 검증 손실이 향상되지 않으면 종료
            print(f'Early stopping! No improvement for {patience} epochs.')
            print(f'TRAIN_WINDOW_SIZE : {CFG["TRAIN_WINDOW_SIZE"]}, PREDICT_SIZE : {CFG["PREDICT_SIZE"]},
            break

    return best_model
```

```python
def validation(model, val_loader, criterion, device):
    model.eval()
    val_loss = []

    with torch.no_grad():
        for X, Y in tqdm(iter(val_loader)):
            X = X.to(device)
            Y = Y.to(device)

            output = model(X)
            loss = criterion(output, Y)

            val_loss.append(loss.item())
    return np.mean(val_loss)
```

학습 시간 절약을 위해 early stop을 적용하였고,
Patience는 3으로 설정했다.

모델 예측, 역변환

베이스라인 코드를 사용하여 추론을 진행했다.
예측값을 역변환하고 반올림하여 후처리했다.

```
# inverse log transformation
offset = 1
pred = np.exp(pred) - offset

# 추론 결과를 inverse scaling
for idx in tqdm(range(len(pred))):
    pred.iloc[idx, :] = pred.iloc[idx, :] * (scale_max_dict[idx] - scale_min_dict[idx]) + scale_min_dict[idx]

# 결과 후처리
pred = np.round(pred, 0).astype(int)
```