시계열 데이터를 이용한

Kaggle Sticker 판매 예측 프로젝트

YB 3조 | 강혜준, 고은서, 김지원, 박소윤

주제 및 개요

시계열 데이터를 이용한 캐글 스티커 판매 예측 💡

- 다양한 국가의 가상 매장에서 나온 다양한 캐글 브랜드 스티커 판매 매출을 예측하는 Kaggle Playground Series

- 평가: MAPE(Mean Absolute Percentage Error)
- ID, Date, Country, Store, Product, NumSold

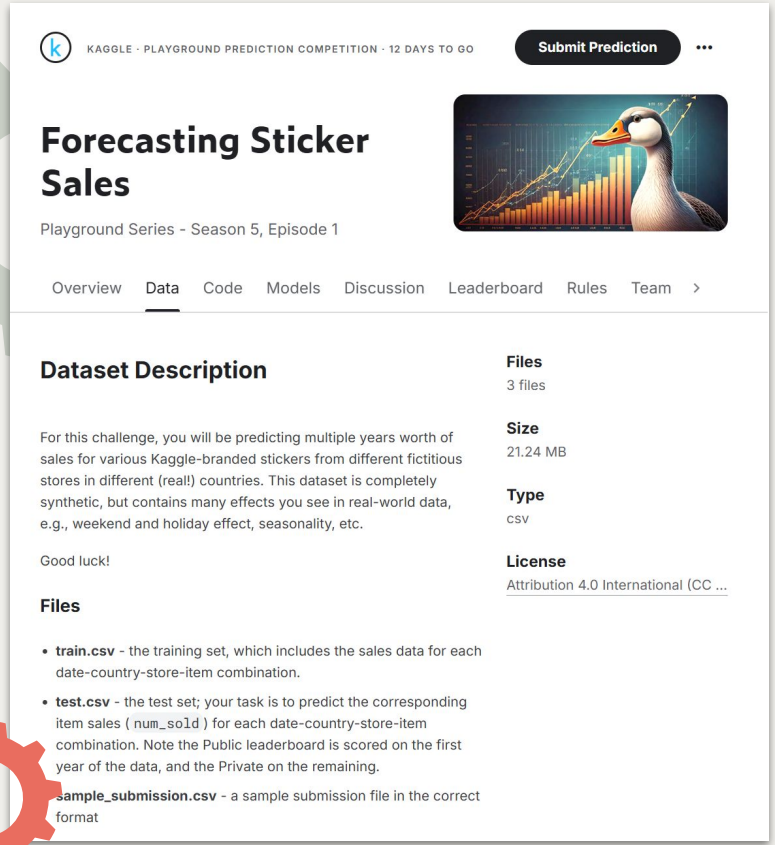


Table of contents

01

Data Engineering

데이터

기본 데이터[1] + 추가 데이터[2]

1 기본 데이터 (캐글 제공)
: date, country, store, product별 num_sold(판매량)

date	2010.01-2016.12 / 2017.01-2019.12
country	Canada, Finland, Italy, Kenya, Norway, Singapore
Store	Discount Stickers, Premium Sticker Mart, Stickers for Less
product	Holographic Goose, Kaggle, Kaggle Tiers, Kerneler, Kerneler Dark Mode

	id	date	country	store	product	num_sold
0	0	2010-01-01	Canada	Discount Stickers	Holographic Goose	NaN
1	1	2010-01-01	Canada	Discount Stickers	Kaggle	973.0
2	2	2010-01-01	Canada	Discount Stickers	Kaggle Tiers	906.0

2 추가 데이터 (조사하여 입력한 자료)

거시경제 지표 : GDP(nominal), GDP per Capita, Growth Rate

국가별 판매량 차이에 경제 규모가 영향을 미칠 것

	country	year	nominal gdp	nominal gdp per capita	growth rate
0	Canada	2010	1620000000000	47600	3.09
1	Canada	2011	1790000000000	52200	3.14
2	Canada	2012	1830000000000	52700	1.76

공휴일 정보 : 국가별 공휴일 날짜 정보

공휴일 여부에 따라 판매량에 차이가 있을 것

	country	date
0	Canada	2010-01-01
1	Canada	2010-04-02
2	Canada	2010-04-05

결측치 처리

타겟에만 결측치. 시계열 데이터이기 때문에 해당 칼럼 삭제 불가
-> 결측치 패턴 분석 후 적절한 imputation 필요

결측치 패턴 발견 : Canada와 Kenya의 일부 product에서만 나타남

	num_sold
id	0
date	0
country	0
store	0
product	0
num_sold	8871
year	0
nominal gdp	0
nominal gdp per capita	0
growth rate	0
holiday	0

country	num_sold
Canada	4246
Finland	0
Italy	0
Kenya	4625
Norway	0
Singapore	0

store	product	num_sold
Discount Stickers	Holographic Goose	2557
	Kaggle	0
	Kaggle Tiers	0
	Kerneler	1
	Kerneler Dark Mode	0
Premium Sticker Mart	Holographic Goose	380
	Kaggle	0
	Kaggle Tiers	0
	Kerneler	0
	Kerneler Dark Mode	0
Stickers for Less	Holographic Goose	1308
	Kaggle	0
	Kaggle Tiers	0
	Kerneler	0
	Kerneler Dark Mode	0

store	product	num_sold
Discount Stickers	Holographic Goose	2557
	Kaggle	0
	Kaggle Tiers	0
	Kerneler	63
	Kerneler Dark Mode	1
Premium Sticker Mart	Holographic Goose	646
	Kaggle	0
	Kaggle Tiers	0
	Kerneler	0
	Kerneler Dark Mode	0
Stickers for Less	Holographic Goose	1358
	Kaggle	0
	Kaggle Tiers	0
	Kerneler	0
	Kerneler Dark Mode	0

Discount Stickers - Holographic Goose는 전체 NaN
-> 해당 가게에서 해당 제품을 팔지 않는 것으로 추정
-> 제거 후 모델 피팅한 뒤, 후처리에서 다루기로 함

이외의 결측치는 보간법으로 처리

*보간법: 앞뒤 데이터로 추정하여 채우는 방법

02

Statistical Model

ARIMA 모델

대표적인 시계열* 모델 ARIMA(p, d, q)

AR

Autoregressive(자기회귀)
이전 값 -> 현재 값 선형추정

p: 자기 회귀 차수
몇개의 과거 값을 사용할 것인가

I

Integrated(차분*)
차분을 통해 데이터 정상화*

d: 차분 횟수

MA

Moving Average(이동평균)
과거의 예측 오차를 활용해 미래 값 예측

q: 이동평균 차수
몇개의 과거 오차를 사용할 것인가

파라미터 결정을 위해 정상성 검정/ 시계열 분해, ACF, PACF plot 분석

*시계열 데이터: 시간을 인덱스로 하여 정렬된 y값
*차분: 현재 값에서 이전 값을 빼는 것
*정상성: 시간에 따라 평균, 분산, 공분산이 일정한 상태

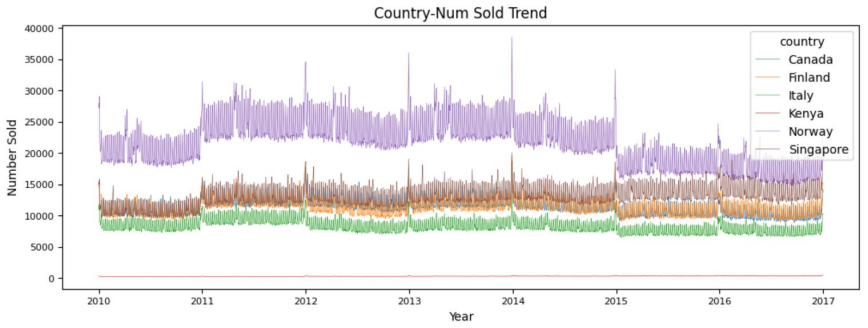
시계열 분해 | ACF | PCAF

Country 별 데이터 추세

a. 추세선 그래프

```
country_sum = df.groupby(['date','country'])['num_sold'].sum().reset_index()
# country_sum
```

```
# 각 나라 별 추세 그래프

import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(12,4))
ax = sns.lineplot(data=country_sum, x='date', y='num_sold', hue='country', linewidth=0.4)
ax.set_title('Country-Num Sold Trend')
ax.set_xlabel("Year", fontsize=10)
ax.set_ylabel("Number Sold", fontsize=10)
ax.tick_params(axis="both", labelsize=8) #레이블 글자 크기 작게
plt.show()
```

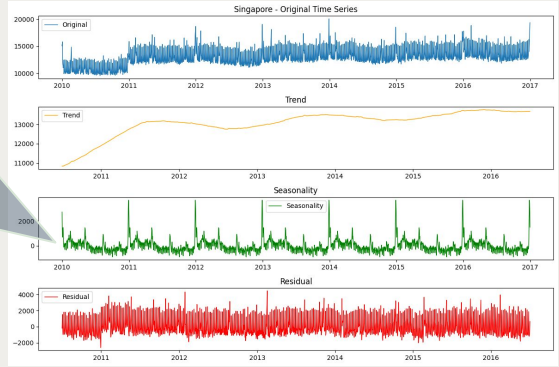


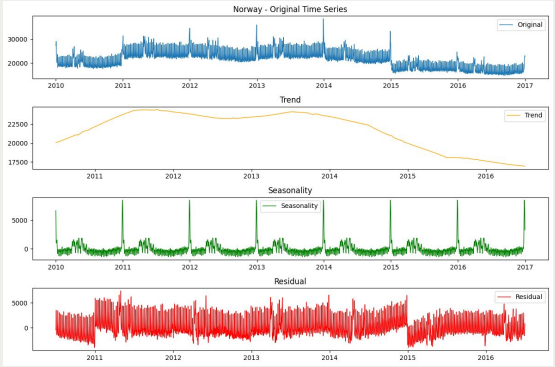
b. 시계열 분해

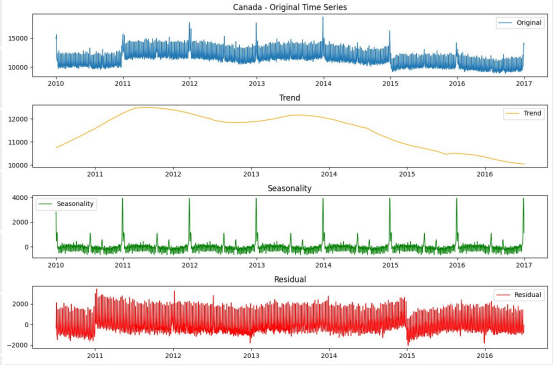
```
# 시계열 분해
countries = country_sum['country'].unique()

for country in countries:
    plt.figure(figsize=(12, 8))
    country_data = country_sum[country_sum['country'] == country]

    # 인덱스를 date로 해야함
    ts_data = country_data.set_index('date')['num_sold']
    decomposition = seasonal_decompose(ts_data, model='additive', period=365)  # 일별 데

    # 추세, 계절성, 잔차 그래프 # 4행 1열로 나열
    plt.subplot(411)
    plt.plot(ts_data, label='Original', linewidth=1)
    plt.title(f'{country} - Original Time Series')
    plt.legend()

    plt.subplot(412)
    plt.plot(decomposition.trend, label='Trend', linewidth=1, color='orange')
    plt.title('Trend')
    plt.legend()

    plt.subplot(413)
    plt.plot(decomposition.seasonal, label='Seasonality', linewidth=1, color='green')
    plt.title('Seasonality')
    plt.legend()

    plt.subplot(414)
    plt.plot(decomposition.resid, label='Residual', linewidth=1, color='red')
    plt.title('Residual')
    plt.legend()

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

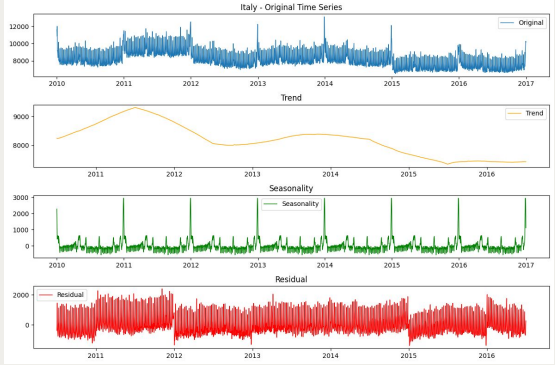
시계열 분해 | ACF | PCAF

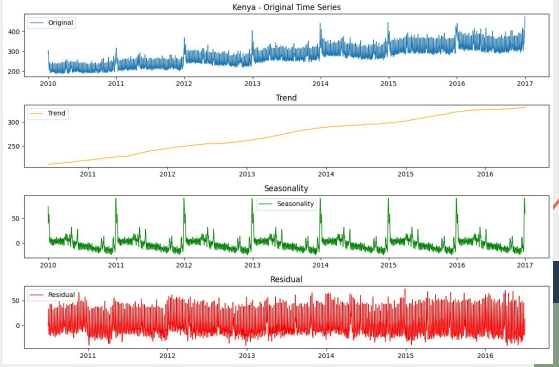
- **추세**: 증가 혹은 감소하는 추세가 보이진 않음. 선형 회귀 사용 불가능
- **계절성**: 연초에 판매량이 높음
> 가중치를 두어 활용











시계열 분해 | ACF | PCAF

Country 별 데이터 추세

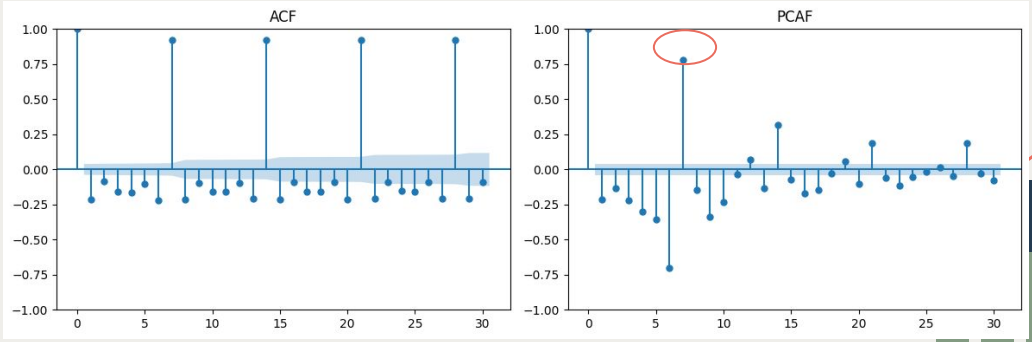
c. ACF, PACF

- ADF 테스트: 시계열이 정상성인지 확인하는 테스트. p-value가 0.05 이상이므로 차분을 진행함
- **분석 결과:**
  Norway데이터만 분석한 결과 (period=365인 일별 데이터), **7일 주기로 확인함.**

```python
# ADF 테스트
country_sum = df.groupby(['date','country'])['num_sold'].sum().reset_index()
country_data = country_sum[country_sum['country'] == 'Norway']
from statsmodels.tsa.stattools import adfuller
result = adfuller(country_data['num_sold'])
print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

```
ADF Statistic: -1.7729903886863916
p-value: 0.3939512433153839
```

```python
# 비정상으로 판단(추세가 있음)-> 차분 진행후 PCAF 그래프를 함께 그리기
country_data['diff'] = country_data['num_sold'].diff()
country_data.dropna(inplace=True)

fig, axes = plt.subplots(1,2, figsize=(12,4))
plot_acf(country_data['diff'], lags=30, ax=axes[0])
axes[0].set_title('ACF')
plot_pacf(country_data['diff'], lags=30, ax=axes[1])
axes[1].set_title('PCAF')
plt.tight_layout()
plt.show()
```



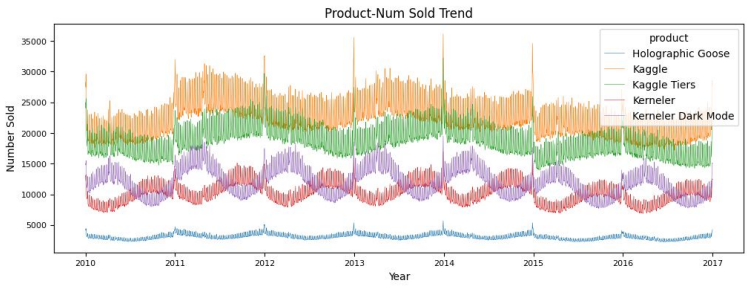
시계열 분해 | ACF | PCAF

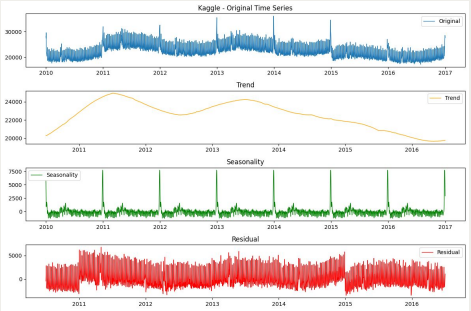
Product 별 데이터 추세

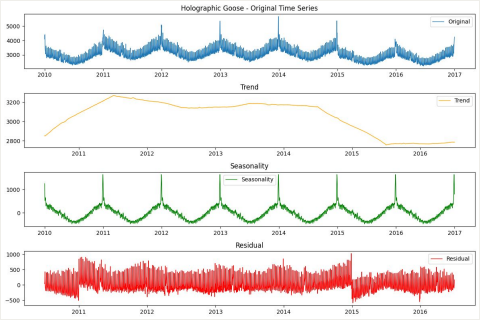
추세: **1년/2년** 주기
-> 모델링에 활용

1년의 계절성

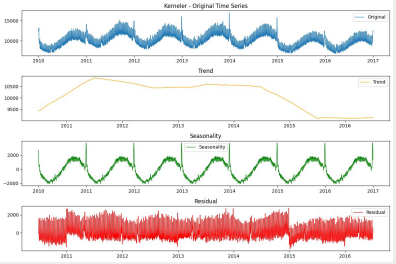
a. 추세선 그래프

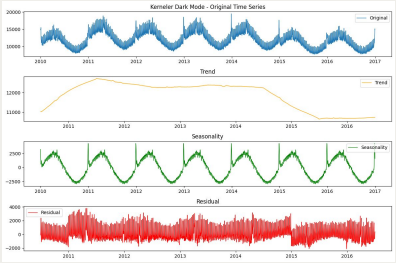






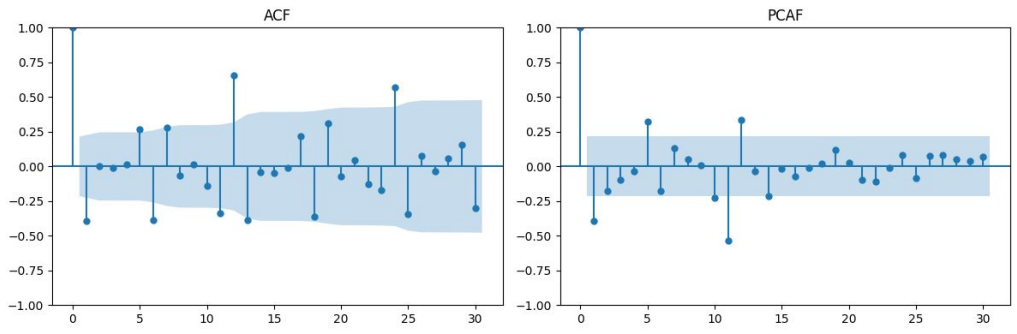
b. 시계열 분해





시계열 분해 | ACF | PCAF

(4) Product 별 데이터 추세

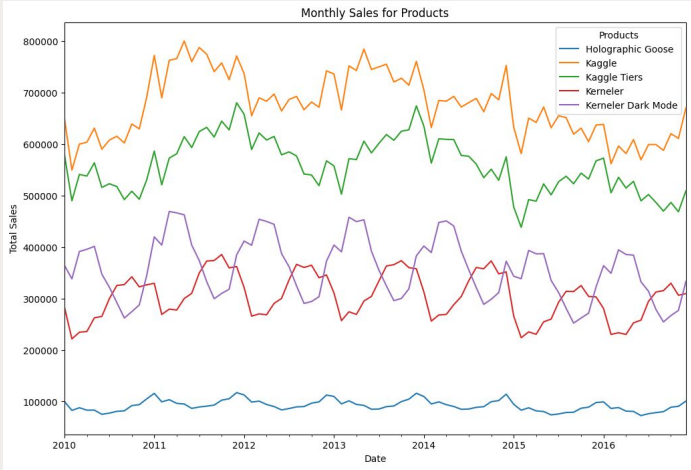


c. ACF, PACF

마찬가지로 **7일 주기**, 주말 판매량 높음
제품 별 월별 판매량, 년도 판매량 주기 등의 차이를 보임

d. 월/요일 별 판매량

- Holographic Goose: 12,1월 판매량 많음, 1년 주기(계절성)
- Kaggle: 5,6,7월 판매량 많음, 2년 주기
- Kaggle Tiers: 12,1월 판매량 많음, 2년 주기
- Kerneler: 8,9,10월 판매량 많음, 1년 주기
- Kerneler Dark Mode: 3,4,5월 판매량 많음, 1년 주기

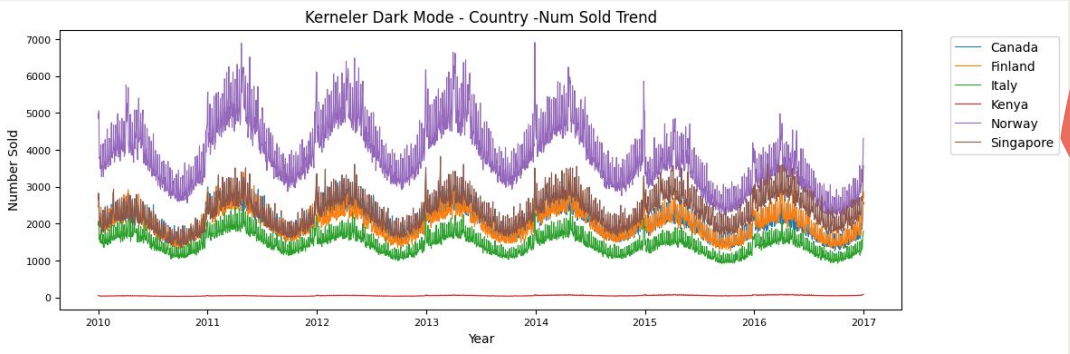


시계열 분해 | ACF | PCAF

결과: 주기성에 대한 인사이트

- 피처를 고정할수록 계절성이 뚜렷해짐
- country, store, product 별 데이터 모두 주기성을 7일로 가짐
- 요일 별 시각화 결과 주말에 판매량 높음

- Product 별로 판매량이 높은 달이 있음



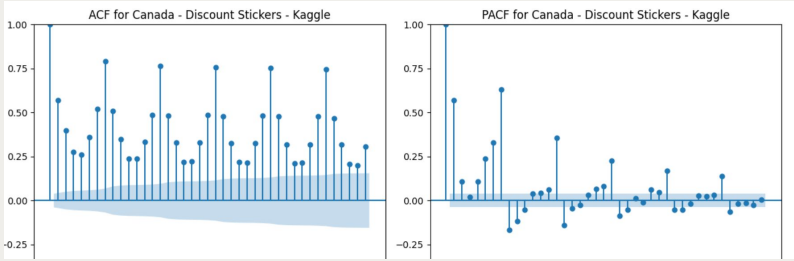
- Holographic Goose: 12,1월 판매량 높음. 1년 주기
- Kaggle: 5,6,7월 판매량 높음. 2년 주기
- Kaggle Tiers: 12,1월 판매량 높음, 2년 주기
- Kerneler: 8,9,10월 판매량 높음,1년 주기
- Kerneler Dark Mode: 3,4,5월 판매량 높음, 1년 주기

ARIMA - fitting

Step1. 전처리
ARIMA는 시계열 데이터를 필요로 하므로 country, store, product에 대해 groupby 처리하여 각 그룹이 시계열 데이터가 되도록 함

Step2. 파라미터 결정
각 그룹에 대해 정상성 검정, ACF, PACF plot을 확인하여 파라미터 결정

```
ADF Test for Canada - Discount Stickers - Kaggle:
ADF Statistic: -2.14388211765732
p-value: 0.22723919842545692
---
ADF Test for Canada - Discount Stickers - Kaggle Tiers:
ADF Statistic: -2.1262101058741676
p-value: 0.2341068079007131
---
ADF Test for Canada - Discount Stickers - Kerneler:
ADF Statistic: -2.289882688202473
p-value: 0.1752517712085747 7
```



대부분의 그룹의 p-value가 0.1 이상 (H0: 정상성 데이터)

그룹이 90개로 많아 auto_arima를 활용해 AIC를 최소로 하는 파라미터를 찾음
(라이브러리: `import pmdarima as pm`)

```
Country: Canada, Store: Discount Stickers, Product: Kaggle
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 2557
Model:               SARIMAX(5, 1, 2)   Log Likelihood              -13998.409
Date:                Mon, 27 Jan 2025   AIC                          28012.819
Time:                        12:57:52   BIC                          28059.588
Sample:                    01-01-2010   HQIC                         28029.779
                         - 12-31-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3110      0.016     19.288      0.000       0.279       0.343
```

ARIMA - fitting

Step3. ARIMA 모델 fitting

```
# Decoding function
country_map = {1: 'Canada', 2: 'Finland', 3: 'Italy', 4: 'Kenya', 5: 'Norway', 6: 'Singapore'}
store_map = {1: 'Discount Stickers', 2: 'Premium Sticker Mart', 3: 'Stickers for Less'}
product_map = {1: 'Holographic Goose', 2: 'Kaggle', 3: 'Kaggle Tiers', 4: 'Kerneler', 5: 'Kerneler Dark Mode'}

def decode(c, s, p):
    country = country_map.get(c, "Unknown")
    store = store_map.get(s, "Unknown")
    product = product_map.get(p, "Unknown")
    return (country, store, product)

def decode_group(group):
    return [decode(c, s, p) for c, s, p in group]
```

같은 파라미터를 쓰는 그룹끼리 묶음
편의상 각 level을 숫자로 매핑하여 작성하고, 디코딩하는 함수 생성

```
# ARIMA parameters for group1 to group13
arima_params = [(2, 1, 3), (2, 1, 4), (2, 1, 5), (3, 1, 2), (3, 1, 3), (3, 1, 5), (4, 1, 3), (4, 1, 4), (4, 1, 5),
(5, 1, 2), (5, 1, 3), (5, 1, 4), (5, 1, 5)]

# Define groups by parameters
group0 = [(1, 1, 1), (4, 1, 1)]
group1 = [(1, 3, 3), (2, 3, 2), (3, 3, 2), (5, 1, 2), (5, 2, 3), (6, 2, 3)]
group2 = [(1, 1, 5), (1, 2, 2), (2, 2, 1), (3, 3, 3), (5, 1, 1), (5, 1, 3), (6, 3, 1)]
...
```

ARIMA - fitting

Step3. ARIMA 모델 fitting

```
#group1
predictions = []

decoded_group1 = decode_group(group1)

for (country, store, product), group data  in grouped_df:
    if (country, store, product)  in decoded_group1:
        series = group data[ 'num sold']
        p, d, q = arima params[ 0]
        model = ARIMA(series, order=(p, d, q), freq=  'D')
        model_fit = model.fit()

        forecast = model_fit.forecast(steps=  1095)

        last date =  '2016-12-31'
        forecast_dates = pd.date_range(start=last_date, periods=  1095 + 1, freq='D')[1:]  # 예측할 날짜

        for i, date  in enumerate(forecast_dates):
            predictions.append({
                'date': date,
                'country': country,
                'store': store,
                'product': product,
                'predict': forecast[i]
            })

group1_predicted_df = pd.DataFrame(predictions)
```

그룹별로 피팅 -> 예측하여 데이터 프레임에 적용하는 작업을
모든 그룹에 대해 반복

ARIMA - result

Step4. 후처리 및 예측값 제출

모델로 예측하지 않은
Canada-Discount Stickers-Holographic Goose
Kenya-Discount Stickers-Holographic Goose를
0으로 예측하여 제출

MAPE: 0.234349%

	date	country	store	product	predict
0	2017-01-01	Finland	Stickers for Less	Kerneler Dark Mode	1050.524312
1	2017-01-02	Finland	Stickers for Less	Kerneler Dark Mode	985.731411
2	2017-01-03	Finland	Stickers for Less	Kerneler Dark Mode	953.285409
3	2017-01-04	Finland	Stickers for Less	Kerneler Dark Mode	941.775821
4	2017-01-05	Finland	Stickers for Less	Kerneler Dark Mode	964.790268
...	...	...	...	...	...
4375	2019-12-27	Norway	Premium Sticker Mart	Kerneler	1492.066031
4376	2019-12-28	Norway	Premium Sticker Mart	Kerneler	1499.372625
4377	2019-12-29	Norway	Premium Sticker Mart	Kerneler	1496.722879
4378	2019-12-30	Norway	Premium Sticker Mart	Kerneler	1486.146006
4379	2019-12-31	Norway	Premium Sticker Mart	Kerneler	1475.623923

ARIMAX - preprocessing

ARIMA 모델에 외생변수를 추가한 모델

ARIMA X

외생변수

```
test_df = test.copy()
test_df.shape
```

```
(98550, 5)
```

```
test_df['year'] = test_df['date'].dt.year.astype(int)
test_df.shape
```

```
(98550, 6)
```

```
test_df = pd.merge(test_df, gdp, on = ['country', 'year'], how = 'inner')
test_df.shape
```

```
(98550, 9)
```

```
holiday['holiday'] = 1

test_df = pd.merge(test_df, holiday[['country', 'date', 'holiday']], on = ['country', 'date'], how = 'left')
test_df['holiday'] = test_df['holiday'].fillna(0).astype(int)

test_df.shape
```

거시경제 지표 : GDP(nominal), GDP per Capita, Growth Rate
공휴일 정보 : 국가별 공휴일 날짜 정보 (binary)

기존 ARIMA 모델에 외생변수 피처를 추가하여 피팅
-> 이전 단계에서 찾은 파라미터를 그대로 사용

ARIMAX - fitting

ARIMAX 모델 fitting

```
#group1
predictions = []
decoded_group1 = decode_group(group1)

for (country, store, product), group_data  in grouped_df:
    if (country, store, product)  in decoded_group1:
        arima_y = group_data[ 'num_sold']
        arima_X = group_data[[ 'year', 'nominal gdp', 'nominal gdp per capita' , 'growth rate', 'holiday']]
        p, d, q = arima_params[ 0]
        model = SARIMAX(arima_y, order=(p, d, q), freq=  'D', exog = arima_X)
        model_fit = model.fit()
        test_X = test_df[(test_df[ 'country'] == country) &
                         (test_df[ 'store'] == store) &
                         (test_df[ 'product'] == product)]
        test_X = test_X[[ 'year', 'nominal gdp', 'nominal gdp per capita' , 'growth rate', 'holiday']]
        forecast = model_fit.get_forecast(steps=  1095, exog=test_X)
...
                'predict': forecast.predicted_mean[i]
            })

group1_predicted_df = pd.DataFrame(predictions)
```

ARIMA 모델에 X를 가져오는 코드를 추가하여 피팅

예측 값을 가져올 때 forecast.predicted_mean method로 가져와야함

ARIMAX - result

ARIMA와 같은 방식으로 후처리하여 피팅했으나 MAPE가 크게 증가하며 성능이 나빠짐

MAPE: 12.48877%

원인

1. 추가한 외생변수가 num_sold와 상관관계가 높지 않음

2. 다중공선성: 회귀 모델에서 다중공선성이 높으면 성능이 저하된다는 점을 간과함
   추가한 데이터 중 GDP 데이터에서 다중공선성 문제가 발생했을 수 있음
   Nominal GDP를 인구 수로 나눈 것이 GDP per Capita
   인구수는 몇년 사이에 큰 변화가 일어나지 않으므로 상수로 나눴다고 볼 수 있고, 이는 결국 같은 데이터를 의미함

   일반적으로 선진국은 높은 GDP와 낮은 성장률을, 개도국은 낮은 GDP와 높은 성장률을 가지므로 여기서도 강한 상관관계

Prophet이란?

Prophet이란? Facebook에서 개발한 시계열 예측 모델

자동으로 반영 및 학습하여 미래 값을 예측

- 추세 (Trend) : 데이터의 장기적인 방향성 (상승 또는 하강)
- 계절성 (Seasonality) : 주기적인 변동 (일별, 주별, 월별 등)
- 휴일효과 (Holidays) : 특정 날의 변동성 (예: 크리스마스, 블랙 프라이데이 등)

LSTM, GRU, Transformer, Temporal Fusion Transformer (TFT)

	Prophet	ARIMA / SARIMA
계절성	자동으로 처리	SARIMA로 수동 처리
트렌드	비선형 트렌드 및 변화점 자동 처리 (정상성을 가정하지 않음)	선형 트렌드만 처리 (정상성을 가정)
계산 복잡도	빠름, 대규모 데이터에 적합	비교적 느림

Prophet - Preprocessing

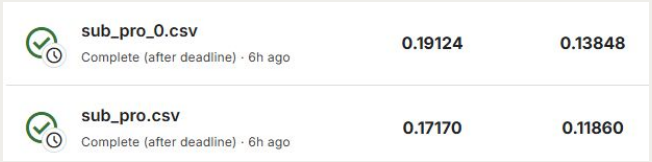
1. 날짜 형식 확인

- ds(날짜, datetime 형식)와 y(타깃 변수) 두가지 칼럼
- 한 날짜당 하나의 데이터만 존재

  (참고) 날짜 간격이 일정하지 않거나 특정 날짜에 데이터가 빠져도 문제없이 학습

2. 결측값 처리

[Canada ∩ Discount Stickers ∩ Holographic Goose]
[Kenya ∩ Discount Stickers ∩ Holographic Goose]



(sol 1) 해당 제품이 아예 판매되지 않는 것으로 가정. 추후 0으로 채움.

(sol 2) 다른 나라들의 같은 제품이 전체 판매량에서 차지하는 비율을 참고하여 결측값을 보간

Prophet - Preprocessing

```
[ ] #import origin data

    data = pd.read_csv('/content/drive/MyDrive/방학프로젝트/train.csv', parse_dates=['date'])
```

```
[ ] df2=data.copy()
```

```
[ ] import numpy as np

    countries = ['Finland', 'Italy', 'Singapore', 'Norway']
    goose_ratios = {}

    for country in countries:
        country_data = df2[(df2['country'] == country) & (df2['store'] == 'Discount Stickers')]
        total_sales_per_date = country_data.groupby('date')['num_sold'].sum()
        goose_sales_per_date = country_data[country_data['product'] == 'Holographic Goose'].groupby('date')['num_sold'].sum()
        goose_ratios[country] = goose_sales_per_date / total_sales_per_date

    for country in ['Canada', 'Kenya']:
        country_data = df2[(df2['country'] == country) & (df2['store'] == 'Discount Stickers')]

        for date in country_data['date'].unique():
            total_sales_country = country_data[country_data['date'] == date]['num_sold'].sum()

            if total_sales_country > 0:
                avg_goose_ratio = np.mean([goose_ratios[country].get(date, 0) for country in countries])
                goose_sales_prediction = avg_goose_ratio * total_sales_country
                df2.loc[(df2['country'] == country) &
                        (df2['store'] == 'Discount Stickers') &
                        (df2['date'] == date) &
                        (df2['product'] == 'Holographic Goose'), 'num_sold'] = goose_sales_prediction
```

```
[ ] #nan_country, nan_store, nan_product
    nan_country = ['Canada', 'Kenya']
    nan_store = ['Discount Stickers', 'Premium Sticker Mart', 'Stickers for Less']
    nan_product = ['Holographic Goose', 'Kerneler', 'Kerneler Dark Mode']

    #보간법으로 결측치 처리
    for country in nan_country:
        for store in nan_store:
            for product in nan_product:

                mask = (df2['country'] == country) & (df2['store'] == store) & (df2['product'] == product)
                subset = df2[mask]

                if subset['num_sold'].isnull().any():
                    df2.loc[mask, 'num_sold'] = subset['num_sold'].interpolate(method='linear')
```

일별로 각 나라별

[Finland, Discount Stickers, Holographic Goose] / [Finland, Discount Stickers, ALL]

계산하여 일별 평균내기

Prophet - modeling

❖ Prophet의 메소드

- 특정 국가의 휴일 정보 추가 → **add_country_holidays( )**
- 계절성을 수동으로 지정하여 추가 → **add_seasonality( )**
- 외부 변수 추가 → **add_regressor( )**

```
# 핀란드의 휴일 추가
model = Prophet( )
model. add_country_holidays(country_name='FI')

# 월별 계절성 추가
model.add_seasonality(name='monthly', period=30.5)

# 경제지표인 nominal gdp 추가
model.add_regressor('nominal gdp')
```

❖ Prophet의 하이퍼파라미터

- changepoint_prior_scale, seasonality_prior_scale, holidays_prior_scale : 반영 민감도 조절
- yearly_seasonality(True), weekly_seasonality(True), daily_seasonality(False) : 주기성 반영 여부
- seasonality_mode, growth : 계절성, 트렌드의 유형

Prophet - modeling

- Holographic Goose: 12,1월 판매량 높음. 1년 주기
- Kaggle: 5,6,7월 판매량 높음. 2년 주기
- Kaggle Tiers: 12,1월 판매량 높음, 2년 주기
- Kerneler: 8,9,10월 판매량 높음,1년 주기
- Kerneler Dark Mode: 3,4,5월 판매량 높음, 1년 주기

```python
import logging
from prophet import Prophet

# 로그 레벨을 WARNING으로 설정하여 INFO 및 DEBUG 메시지 출력하지 않도록 함
logging.basicConfig(level=logging.WARNING)
import pandas as pd
import numpy as np


for country in df2['country'].unique():
    for store in df2['store'].unique():
        for product in df2['product'].unique():
            cluster_data = df2[(df2['country'] == country) &
                               (df2['store'] == store) &
                               (df2['product'] == product)].copy()

            cluster_data = cluster_data.rename(columns={'date': 'ds', 'num_sold': 'y'})
            cluster_data = cluster_data[['ds', 'y']]

            model = Prophet()

            # 제품별로 맞는 월별 주기성을 추가
            if product == 'Holographic Goose':
                model.add_seasonality(name='monthly', period=30.5, fourier_order=8)
            elif product == 'Kaggle':
                model.add_seasonality(name='monthly', period=30.5, fourier_order=8)
            elif product == 'Kaggle Tiers':
                model.add_seasonality(name='monthly', period=30.5, fourier_order=8)
            elif product == 'Kerneler':
                model.add_seasonality(name='monthly', period=30.5, fourier_order=8)
            elif product == 'Kerneler Dark Mode':
                model.add_seasonality(name='monthly', period=30.5, fourier_order=8)

            # 조합에 해당하는 나라의 휴일 정보 추가
            if country == 'Finland':
                model.add_country_holidays(country_name='FI')
            elif country == 'Kenya':
                model.add_country_holidays(country_name='KE')
            elif country == 'Norway':
                model.add_country_holidays(country_name='NO')
            elif country == 'Singapore':
                model.add_country_holidays(country_name='SG')

            model.fit(cluster_data)

            test_data = test[(test['country'] == country) &
                             (test['store'] == store) &
                             (test['product'] == product)]

            # test_data의 date가 불연속적일 경우를 대비해 date를 인덱스로 하여 예측 실행
            if not test_data.empty:
                test_dates = test_data['date'].unique()
                future_dates = pd.DataFrame({'ds': test_dates})

                future = model.predict(future_dates)

                test.loc[(test['country'] == country) &
                         (test['store'] == store) &
                         (test['product'] == product) &
                         (test['date'].isin(future_dates['ds'])), 'num_sold_pred'] = future['yhat'].values
```

(country, store, product) 조합별 Prophet 모델을 하나씩 적용 → 6 * 3 * 5 = 90 가지를 따로 학습하여 따로 예측!

해당 나라의 휴일 정보 추가

각 제품에 대해 월별 주기성 추가

Leaderboard Score : 0.11860

03

Machine Learning Model

Feature engineering for ML

- Holographic Goose: 12,1월 판매량 높음. 1년 주기
- Kaggle: 5,6,7월 판매량 높음. 2년 주기
- Kaggle Tiers: 12,1월 판매량 높음, 2년 주기
- Kerneler: 8,9,10월 판매량 높음,1년 주기
- Kerneler Dark Mode: 3,4,5월 판매량 높음, 1년 주기

(1) 가중치 피처 생성

1. **월별 가중치 피처**
   각 제품 별 판매량이 많은 달:1.5, 그 외 달:1
2. **요일 별 가중치 피처**
   주말 판매량이 높은 것을 반영
3. **1년 또는 2년 주기성 피처**
   각 제품 별 주기성을 각각 반영하여 사인 코사인 변환.
   계절성 패턴 수치화
4. **주기성 + 요일**
   상호작용 변수 추가. 같이 작용하는 경우 고려

(2) 왜곡도 높은 피처 로그 변환

```
# 1. 월별 가중치 피처
df_cleaned_2['month_weight']=1
df_cleaned_2.loc[(df_cleaned_2['product']=='Holographic Goose')&(df_cleaned_2['month'].isin([12,1])),'month_weight']=1.5
df_cleaned_2.loc[(df_cleaned_2['product']=='Kaggle')&(df_cleaned_2['month'].isin([5,6,7])),'month_weight']=1.5
df_cleaned_2.loc[(df_cleaned_2['product']=='Kaggle Tiers')&(df_cleaned_2['month'].isin([12,1])),'month_weight']=1.5
df_cleaned_2.loc[(df_cleaned_2['product']=='Kerneler')&(df_cleaned_2['month'].isin([8,9,10])),'month_weight']=2
df_cleaned_2.loc[(df_cleaned_2['product']=='Kerneler Dark Mode')&(df_cleaned_2['month'].isin([3,4,5])),'month_weight']=2
```

```
<ipython-input-30-e39a0aaa7198>:3: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in
  df_cleaned_2.loc[(df_cleaned_2['product']=='Holographic Goose')&(df_cleaned_2['month'].isin([12,1])),'month_weight']=1.5
```

```
# 2. 요일별 가중치 피처 (평일:1, 주말:1.5)
weekday_weights = {0:1, 1:1, 2:1, 3:1, 4:1, 5:1.5, 6:1.5}
df_cleaned_2['weekday_weight'] = df_cleaned_2['weekday'].map(weekday_weights)
# df_cleaned_2.head()
```

```
# 2-2. 요일의 주기성 반영 피처(사인변환)
df_cleaned_2['sin_week'] = np.sin(2 * np.pi * df_cleaned_2['weekday'] / 7)
df_cleaned_2['cos_week'] = np.cos(2 * np.pi * df_cleaned_2['weekday'] / 7)
```

```
# 3. 1년/2년 주기성 피처

# 제품별 주기 매핑
product_cycles = {
    'Holographic Goose': 12,   # 1년 주기
    'Kaggle': 24,              # 2년 주기
    'Kaggle Tiers': 24,        # 2년 주기
    'Kerneler': 12,            # 1년 주기
    'Kerneler Dark Mode': 12  # 1년 주기
}
df_cleaned_2['cycle'] = df_cleaned_2['product'].map(product_cycles)

# 사인, 코사인 변환
df_cleaned_2['sin_cycle'] = np.sin(2 * np.pi * df_cleaned_2['month'] / df_cleaned_2['cycle'])
df_cleaned_2['cos_cycle'] = np.cos(2 * np.pi * df_cleaned_2['month'] / df_cleaned_2['cycle'])
```

```
# 4.주기성(년도) 피처: 피처*요일
df_cleaned_2['cycleandweek'] = df_cleaned_2['sin_cycle'] * df_cleaned_2['weekday']
```

```
# 원핫인코딩
df_cleaned_2 = pd.get_dummies(df_cleaned_2, columns=['country', 'store','product'])
```

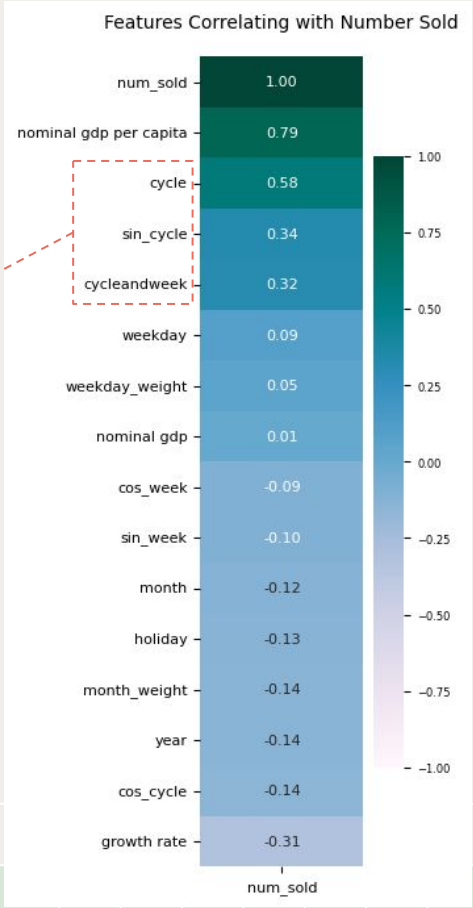
```
# 왜곡도가 1 이상인 피처 로그변환
df_cleaned_2['growth rate'] = np.log1p(df_cleaned_2['growth rate'])
df_cleaned_2['num_sold'] = np.log1p(df_cleaned_2['num_sold'])
```

Feature engineering for ML

- Holographic Goose: 12,1월 판매량 높음. 1년 주기
- Kaggle: 5,6,7월 판매량 높음. 2년 주기
- Kaggle Tiers: 12,1월 판매량 높음, 2년 주기
- Kerneler: 8,9,10월 판매량 높음,1년 주기
- Kerneler Dark Mode: 3,4,5월 판매량 높음, 1년 주기

(1) 가중치 피처 생성

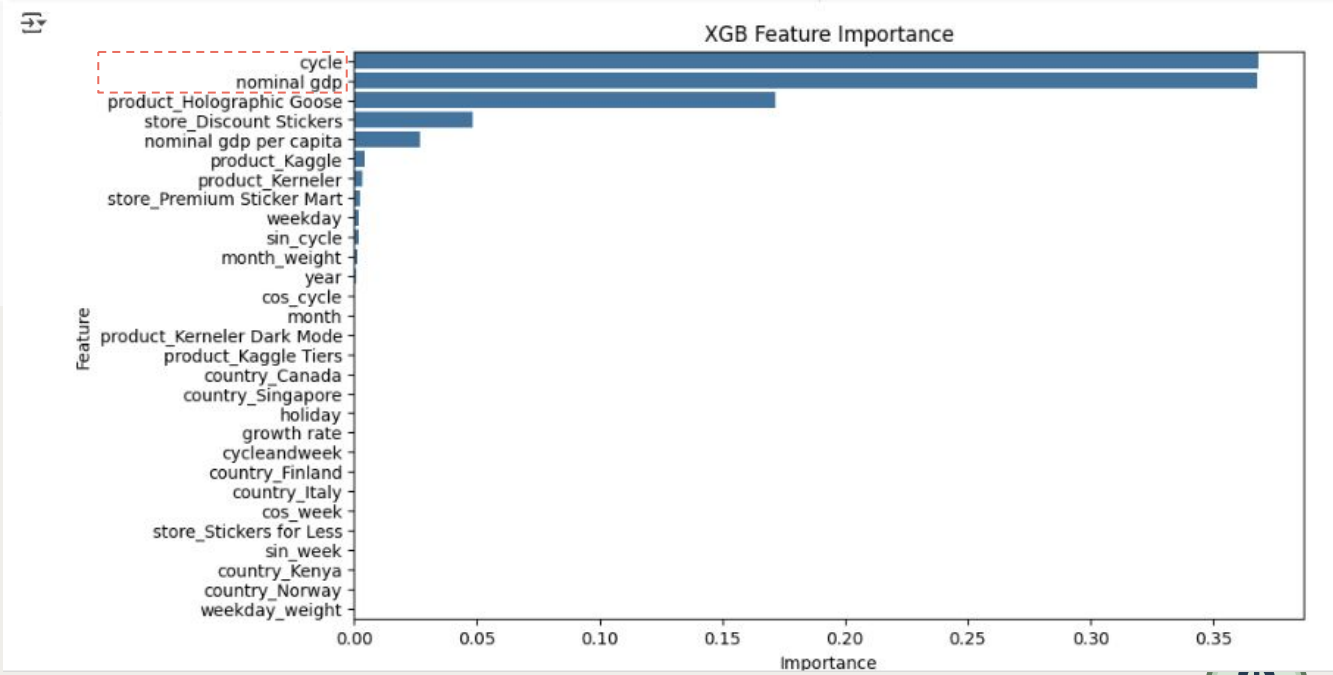
1. **월별 가중치 피처**
   각 제품 별 판매량이 많은 달:1.5, 그 외 달:1
2. **요일 별 가중치 피처**
   주말 판매량이 높은 것을 반영
3. **1년 또는 2년 주기성 피처**
   각 제품 별 주기성을 각각 반영하여 사인 코사인 변환.
   계절성 패턴 수치화
4. **주기성 + 요일**
   상호작용 변수 추가. 같이 작용하는 경우 고려



(2) 왜곡도 높은 피처 로그 변환

XGBoost - modeling

```
[ ]  # XGBoost 회귀 1차 모델링
     from xgboost import XGBRegressor

     xgb_reg = XGBRegressor(n_estimators=1000,
                            verbose=-1)

     xgb_reg.fit(X_train, y_train)
     xgb_pred = xgb_reg.predict(X_test)
     xgb_reg.fit(X_train, y_train)
     xgb_pred = xgb_reg.predict(X_test)
```



XGBoost – hyperparameter tuning

```python
import optuna
import xgboost as xgb
import numpy as np
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_absolute_percentage_error

X_train = X_train.values
y_train = y_train.values

def objective(trial):
    param = {
        "objective": "reg:squarederror",
        "tree_method": "auto",
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3, log=True),
        "max_depth": trial.suggest_int("max_depth", 3, 10),
        "min_child_weight": trial.suggest_float("min_child_weight", 1e-3, 10.0, log=True),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0),
        "reg_alpha": trial.suggest_float("reg_alpha", 1e-8, 1.0, log=True),
        "reg_lambda": trial.suggest_float("reg_lambda", 1e-8, 1.0, log=True),
        "n_estimators": trial.suggest_int("n_estimators", 100, 1000),
    }

    tscv = TimeSeriesSplit(n_splits=5)
    mape_scores = []
```

평가기준
mean_absolute_percentage_error

$$MAPE = \frac{1}{n}\sum\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100$$

```python
    for train_idx, test_idx in tscv.split(X_train):
        X_train_fold, X_test_fold = X_train[train_idx], X_train[test_idx]
        y_train_fold, y_test_fold = y_train[train_idx], y_train[test_idx]

        model = xgb.XGBRegressor(**param, early_stopping_rounds=10)
        model.fit(
            X_train_fold,
            y_train_fold,
            eval_set=[(X_test_fold, y_test_fold)],
            verbose=False
        )

        y_pred = np.expm1(model.predict(X_test_fold))
        y_true = np.expm1(y_test_fold)
        mape = mean_absolute_percentage_error(y_true, y_pred)
        mape_scores.append(mape)

    return np.mean(mape_scores)

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=50)
```

베이지안 최적화 기반의 Optuna 사용

TimeSeriesSplit로 훈련/테스트 분할 : 시간 순서를 유지하면서 검증

Split	Train Data	Test Data
1	[T1, T2, T3]	[T4]
2	[T1, T2, T3, T4]	[T5]
3	[T1, T2, T3, T4, T5]	[T6]

Final MAPE : 0.0949 %
Leaderboard Score : 0.12485

LGBM

(1) 모델링

- MAPE: 0.072

```
[ ]  1 # 다시 데이터 X,y 나누기 (검증용 test 데이터)
     2 X = df_cleaned_2.drop(columns=['num_sold'])
     3 y = df_cleaned_2[['num_sold', 'year']] #using 'year' as index
     4
     5 X_train = X[~(X['year'] == 2016)]
     6 y_train = y[~(y['year'] == 2016)]
     7 X_test = X[X['year'] == 2016]
     8 y_test = y[y['year'] == 2016]
     9
    10 #drop 'year
    11 y = y['num_sold']
    12 y_train = y_train['num_sold']
    13 y_test = y_test['num_sold']
    14
    15 print('X.shape: ', X.shape, 'y.shape: ', y.shape)
    16 print('X_train: ', X_train.shape, 'X_test: ', X_test.shape)
    17 print('y_train: ', y_train.shape, 'y_test: ', y_test.shape)

X.shape:  (225016, 29) y.shape:  (225016,)
X_train:  (192808, 29) X_test:  (32208, 29)
y_train:  (192808,) y_test:  (32208,)
```

```
[ ]  # LGBM 회귀 1차 모델링
     from lightgbm import LGBMRegressor
     lgbm_reg = LGBMRegressor(n_estimators=1000,
                              learning_rate=0.05,
                              num_leaves=100,
                              min_child_samples=100,
                             #  max_depth=4,
                              subsample=0.5,
                             #  colsample_bytree=0.4,
                             #  n_jobs=-1,
                              verbose=-1)
     lgbm_reg.fit(X_train, y_train)
     lgbm_pred = lgbm_reg.predict(X_test)

/usr/local/lib/python3.11/dist-packages/dask/dataframe/__init__.py:42: FutureWarning:
Dask dataframe query planning is disabled because dask-expr is not installed.

You can install it with `pip install dask[dataframe]` or `conda install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
```

```
[ ]  # MAPE 정의
     import numpy as np
     from sklearn.metrics import mean_absolute_percentage_error
     def mape(y_true, y_pred):
       return mean_absolute_percentage_error(y_true, y_pred)
```

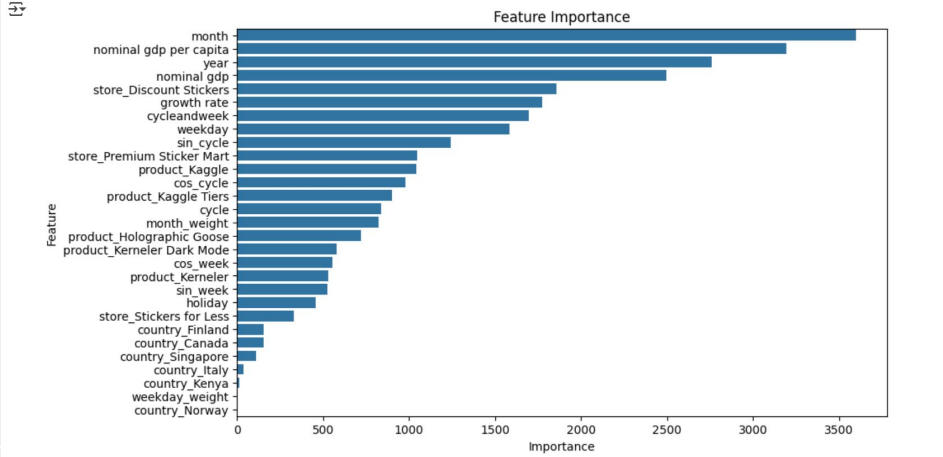
```
[ ]  # MAPE 성능계산
     y_test_original = np.expm1(y_test)
     lgbm_pred_original = np.expm1(lgbm_pred)
     mape(y_test_original, lgbm_pred_original)

0.0721959489371396
```

LGBM

(1) 모델링

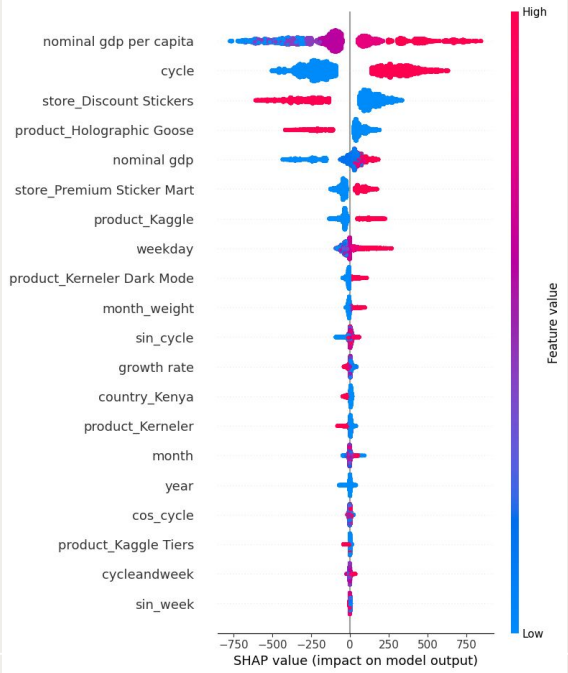
- MAPE: 0.072

```
[ ]  1 # Feature importance
     2 importance = lgbm_reg.feature_importances_
     3 feature_names = X_train.columns
     4 importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importance})
     5 importance_df = importance_df.sort_values(by='Importance', ascending=False)
     6 plt.figure(figsize=(10, 6))
     7 sns.barplot(x='Importance', y='Feature', data=importance_df)
     8 plt.title('Feature Importance')
     9 plt.show()
```



SHAP: 각 피처가 예측 값에 기여한 구체적인 크기와 방향(증가/감소)을 보여주는 그래프

```
1 import shap
2 explainer = shap.TreeExplainer(lgbm_reg)
3 shap_values = explainer.shap_values(X_test)
4 shap.summary_plot(shap_values, X_test)
```



LGBM

Optuna: 탐색할 하이퍼 파라미터의 값을 지정하면 범위 내에서 자동탐색을 진행

(2) 하이퍼 파라미터 튜닝 - Optuna와 Time Series Split를 이용

- !pip install optuna가 요구 됨

```
import optuna
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.model_selection import TimeSeriesSplit
import lightgbm as lgb
import numpy as np

def objective(trial, X_train, y_train):
    # 하이퍼파라미터 설정 (Optuna에서 하이퍼파라미터 탐색 범위 지정)
    param = {
        'objective': 'regression',
        'metric': 'mape',  # MAPE를 목표 지표로 설정
        'boosting_type': 'gbdt',
        'verbose': -1,
        'num_leaves': trial.suggest_int('num_leaves', 10, 100),
        'learning_rate': trial.suggest_loguniform('learning_rate', 1e-5, 1e-1),
        'n_estimators': trial.suggest_int('n_estimators', 50, 500),
        'max_depth': trial.suggest_int('max_depth', 3, 12),
        'min_data_in_leaf': trial.suggest_int('min_data_in_leaf', 10, 100),
        'subsample': trial.suggest_uniform('subsample', 0.5, 1.0),
        'colsample_bytree': trial.suggest_uniform('colsample_bytree', 0.5, 1.0)
    }
```

```
    # TimeSeriesSplit 교차 검증 설정
    tscv = TimeSeriesSplit(n_splits=5)  # 예: 5개의 시계열 분할

    mape_scores = []  # 각 fold에서의 MAPE 점수를 저장할 리스트

    # TimeSeriesSplit을 사용하여 교차 검증 진행
    for train_idx, val_idx in tscv.split(X_train):
        # Use .iloc to index by position
        X_train_fold, X_val_fold = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_train_fold, y_val_fold = y_train.iloc[train_idx], y_train.iloc[val_idx]

        model = lgb.LGBMRegressor(**param)
        model.fit(X_train_fold, y_train_fold)

        y_pred_fold = model.predict(X_val_fold)

        # MAPE 계산
        fold_mape = mape(y_val_fold, y_pred_fold)
        mape_scores.append(fold_mape)

    # 평균 MAPE를 반환
    return np.mean(mape_scores)
```

LGBM

(2) 하이퍼 파라미터 튜닝

- MAPE: 0.072 > 0.0580

```
# Optuna로 하이퍼파라미터 튜닝을 수행
def tune_lgbm_with_optuna(X_train, y_train):
    study = optuna.create_study(direction='minimize')
    study.optimize(lambda trial: objective(trial, X_train, y_train), n_trials=100)

    print("Best hyperparameters: ", study.best_params)
    print("Best MAPE: ", study.best_value)

    return study # Return the study object

# 훈련 데이터에 대해 하이퍼파라미터 튜닝
study = tune_lgbm_with_optuna(X, y)
```

```
# 최적의 하이퍼파라미터로 최종 모델 학습
best_params_lgbm = study.best_params
best_model_lgbm = LGBMRegressor(**best_params_lgbm)
best_model_lgbm.fit(X_test, y_ttest)

y_pred_fin = np.expm1(best_model_lgbm.predict(X_test))
y_true_fin = np.expm1(y_test)
mape_final = mean_absolute_percentage_error(y_true_fin, y_pred_fin)

print(f"Final MAPE: {mape_final:.4f}")
```

Final MAPE : 0.0580 %
Leaderboard Score : 0.11263

제출 결과

모델 종류	ARIMA	ARIMAX	Prophet	XGBoost	LightGBM
Leaderboard Score	0.23435	12.48877	0.11860	0.12485	0.11263

BEST!!

LGBM > Prophet > XGBoost > ARIMA라는 결과로 보아 이 데이터는 :

- XGBoost > ARIMA → 비선형적인 관계가 분명히 포함된다.
- Prophet > XGBoost → 시계열 패턴(추세+계절성)이 분명하다.
- LGBM > Prophet → 시간 외에도 외부요인이 중요한 영향을 미친다.

04

Conclusion

문제 해결 관점에서 결론 도출

1. LGBM 모델이 가장 성능이 좋음
2. 통계적 방법 < 빅데이터 방법론
3. 사전 예측이 가능한 머신러닝

sub_lgbm.csv Complete · 3d ago	0으로 채운 후	0.13786	0.13144	☐
sub_lgbm.csv Complete · 3d ago	0으로 채우기 전	0.11761	0.11263	☐

한계와 의의

1. **한계**
   a. 실제 데이터가 아님
   b. SARIMA를 구동하지 못함(RAM 이슈)

2. **의의**
   a. 복습
   b. 새로운 도전(시계열 데이터)

2024-2 YB 3조 방학 프로젝트

Q&A

Thank you