1. 드라이브 마운트, 모듈 설치, 한글폰트 설정

구글코랩 런타임 연결 끊김 방지
https://teddylee777.github.io/colab/google-colab-%EB%9F%B0%ED%83%80%EC%9E%84-%EC%97%B0%EA%B2%B0%EB%81%8A%EA%B9%80%EB%B0%A9%EC%A7%80

```
## 구글 드라이브 마운트
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
## 건우

import os
os.chdir('/content/drive/MyDrive/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_데이터_210803')

from google.colab import output

# # !cp 파일1 파일2 # 파일1을 파일2로 복사 붙여넣기
# !cp "./RAW_DATA.zip" "data.zip"

# # data.zip을 현재 디렉터리에 압축해제
# !unzip "data.zip"
```

```
from google.colab import output

# !cp 파일1 파일2 # 파일1을 파일2로 복사 붙여넣기
!cp "/content/drive/MyDrive/TAVE_빅콘테스트/제출데이터/RAW_DATA.zip" "data.zip"

# data.zip을 현재 디렉터리에 압축해제
!unzip "data.zip"
```

```
Archive:  data.zip
외부데이터/USD_CAD 내역.csv:  mismatching "local" filename (ь Ш┐ы╣A ы H▒ь Э┤ э Д▒/USD_CA
         continuing with "central" filename version
  inflating: 외부데이터/USD_CAD 내역.csv
외부데이터/USD_CLP 내역.csv:  mismatching "local" filename (ь Ш┐ы╣A ы H▒ь Э┤ э Д▒/USD_CL
         continuing with "central" filename version
  inflating: 외부데이터/USD_CLP 내역.csv
외부데이터/USD_CNH 내역.csv:  mismatching "local" filename (ь Ш┐ы╣A ы H▒ь Э┤ э Д▒/USD_CN
         continuing with "central" filename version
  inflating: 외부데이터/USD_CNH 내역.csv
외부데이터/USD_GBP 내역.csv:  mismatching "local" filename (ь Ш┐ы╣A ы H▒ь Э┤ э Д▒/USD_GB
         continuing with "central" filename version
  inflating: 외부데이터/USD_GBP 내역.csv
외부데이터/USD_KRW 내역.csv:  mismatching "local" filename (ь Ш┐ы╣A ы H▒ь Э┤ э Д▒/USD_KR
         continuing with "central" filename version
  inflating: 외부데이터/USD_KRW 내역.csv
외부데이터/USD_MYR 내역.csv:  mismatching "local" filename (ь Ш┐ы╣A ы H▒ь Э┤ э Д▒/USD_MY
         continuing with "central" filename version
```

```
  inflating: 외부데이터/USD_MYR 내역.csv
외부데이터/USD_NOK 내역.csv:  mismatching "local" filename (ь Ш┐ы╡А ы Н▒ь Э┤ э Д▒/USD_N
        continuing with "central" filename version
  inflating: 외부데이터/USD_NOK 내역.csv
외부데이터/USD_PEN 내역.csv:  mismatching "local" filename (ь Ш┐ы╡А ы Н▒ь Э┤ э Д▒/USD_P
        continuing with "central" filename version
  inflating: 외부데이터/USD_PEN 내역.csv
외부데이터/USD_THB 내역.csv:  mismatching "local" filename (ь Ш┐ы╡А ы Н▒ь Э┤ э Д▒/USD_T
        continuing with "central" filename version
  inflating: 외부데이터/USD_THB 내역.csv
외부데이터/USD_VND 내역.csv:  mismatching "local" filename (ь Ш┐ы╡А ы Н▒ь Э┤ э Д▒/USD_V
        continuing with "central" filename version
  inflating: 외부데이터/USD_VND 내역.csv
외부데이터/국제유가2021-09-09.csv:  mismatching "local" filename (ь Ш┐ы╡А ы Н▒ь Э┤ э Д▒
        continuing with "central" filename version
  inflating: 외부데이터/국제유가2021-09-09.csv
제공데이터/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_문제데이터.xlsx:  mismatching "
        continuing with "central" filename version
  inflating: 제공데이터/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_문제데이터.xlsx
제공데이터/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_자율평가데이터.xlsx:  mismatchi
        continuing with "central" filename version
  inflating: 제공데이터/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_자율평가데이터.xls
제공데이터/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_평가데이터_update_210831.xlsx:
        continuing with "central" filename version
  inflating: 제공데이터/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_평가데이터_update_
```

```
## 한글폰트 깨짐현상 해결
!sudo apt-get install -y fonts-nanum
!sudo fc-cache -fv
!rm ~/.cache/matplotlib -rf

!pip install sktime ##시계열 train, test 분리할때 사용
```

```
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package fonts-nanum.
(Reading database ... 148492 files and directories currently installed.)
Preparing to unpack .../fonts-nanum_20170925-1_all.deb ...
Unpacking fonts-nanum (20170925-1) ...
Setting up fonts-nanum (20170925-1) ...
Processing triggers for fontconfig (2.12.6-0ubuntu2) ...
/usr/share/fonts: caching, new cache contents: 0 fonts, 1 dirs
/usr/share/fonts/truetype: caching, new cache contents: 0 fonts, 3 dirs

/usr/share/fonts/truetype/humor-sans: caching, new cache contents: 1 fonts, 0 dirs
/usr/share/fonts/truetype/liberation: caching, new cache contents: 16 fonts, 0 dirs
/usr/share/fonts/truetype/nanum: caching, new cache contents: 10 fonts, 0 dirs
/usr/local/share/fonts: caching, new cache contents: 0 fonts, 0 dirs
/root/.local/share/fonts: skipping, no such directory
/root/.fonts: skipping, no such directory
/var/cache/fontconfig: cleaning cache directory
/root/.cache/fontconfig: not cleaning non-existent cache directory
/root/.fontconfig: not cleaning non-existent cache directory
fc-cache: succeeded
Collecting sktime
```

```
  Downloading sktime-0.7.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.8
     |████████████████████████████████| 5.8 MB 5.1 MB/s
Requirement already satisfied: pandas>=1.1.0 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: wheel in /usr/local/lib/python3.7/dist-packages (from sktim
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.7/dist-packages (fr
Collecting scikit-learn>=0.24.0
  Downloading scikit_learn-0.24.2-cp37-cp37m-manylinux2010_x86_64.whl (22.3 MB)
     |████████████████████████████████| 22.3 MB 1.5 MB/s
Requirement already satisfied: numba>=0.50 in /usr/local/lib/python3.7/dist-packages (from
Collecting statsmodels>=0.12.1
  Downloading statsmodels-0.12.2-cp37-cp37m-manylinux1_x86_64.whl (9.5 MB)
     |████████████████████████████████| 9.5 MB 54.5 MB/s
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: llvmlite<0.35,>=0.34.0.dev0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from py
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages (fr
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-2.2.0-py3-none-any.whl (12 kB)
Requirement already satisfied: patsy>=0.5 in /usr/local/lib/python3.7/dist-packages (from
Installing collected packages: threadpoolctl, statsmodels, scikit-learn, sktime
  Attempting uninstall: statsmodels
    Found existing installation: statsmodels 0.10.2
    Uninstalling statsmodels-0.10.2:
      Successfully uninstalled statsmodels-0.10.2
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 0.22.2.post1
    Uninstalling scikit-learn-0.22.2.post1:
      Successfully uninstalled scikit-learn-0.22.2.post1
Successfully installed scikit-learn-0.24.2 sktime-0.7.0 statsmodels-0.12.2 threadpoolctl-2
```

런타임 재시작하기

2. 모듈 및 데이터불러오기

```
## 기본적인 모듈
import pandas as pd
import warnings ; warnings.filterwarnings(action='ignore')
import numpy as np
from datetime import datetime
from tqdm.notebook import tqdm ## 진행상황 표시

## 시각화 및 폰트 지정할때 사용
import matplotlib.pyplot as plt ; plt.rc('font', family='NanumBarunGothic',size=20) ## 기본 시각
import seaborn as sns ## 기본 시각화
from sklearn.metrics import silhouette_samples, silhouette_score  # 실루엣 계수 시각화
from sklearn.preprocessing import LabelEncoder ## 클러스터링 할때 사용
from sklearn.cluster import KMeans ## 클러스터링

## 모델링 할때 사용
```

```
from sktime.forecasting.model_selection import temporal_train_test_split ## 시계열 train, test
import xgboost as xgb ## 클러스터링 예측할때 사용
from xgboost import XGBRegressor ## 가격 예측할때 사용
from sklearn.metrics import mean_squared_error ## mse 계산할때 사용
```

```
## 데이터 경로 지정
TRAIN_PATH_2015_2019 = './제공데이터/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_문제데이
TRAIN_PATH_2020 = './제공데이터/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_자율평가데이터
TARGET_PATH_2021 = './제공데이터/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_평가데이터_u

## 2015년 ~ 2020년 학습 데이터 불러오기
train_full_data=pd.DataFrame()
train_data_2015_2019=pd.read_excel(TRAIN_PATH_2015_2019)
train_data_2020=pd.read_excel(TRAIN_PATH_2020)

## 데이터 연결
train_full_data=pd.concat([train_data_2015_2019,train_data_2020])
train_full_data.reset_index(drop=True,inplace=True)
```

3. 제공 데이터 전처리

실루엣 계수 및 시각화를 이용해서 상세 어종별 최적 군집 선택

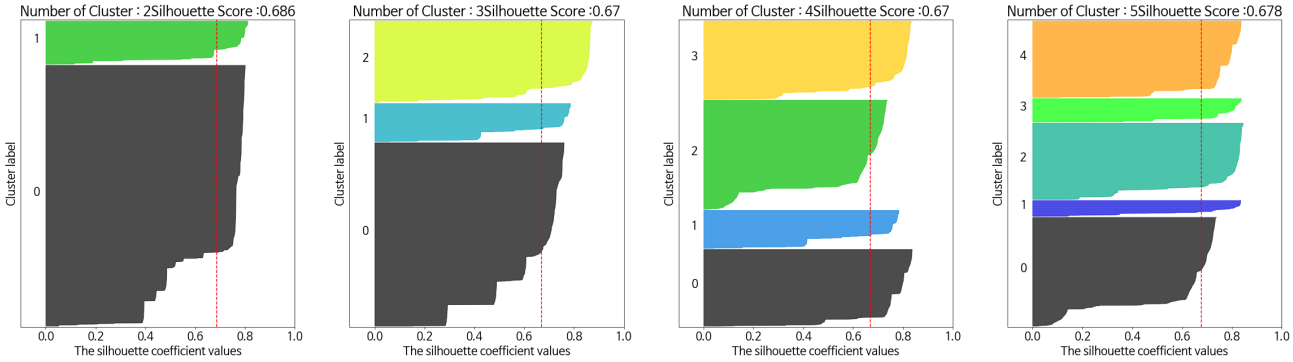
```
## 상세어종이 오징어, 연어, 흰다리 새우인 데이터만 보기
train_full_data=train_full_data[train_full_data['P_NAME'].isin(['오징어','연어','흰다리새우'])]


## 필요없는 변수들 제거
## P_TYPE : 모두 같아서 제거함
## CATEGORY_1, CATEGORY_2 : 원하는 제품의 경우 모든 내용이 같으므로 제거함
## CTRY_2 : 제조국과 판매국이 같은경우가 약 97%정도이므로 국가 정보변수 제거 해도 큰 영향 없을
train_full_data.drop(['P_TYPE','CTRY_2','CATEGORY_1','CATEGORY_2'],axis=1,inplace=True)


## 제품이 가격에 큰 영향을 줄것이라 판단하여 제품을 따로 분류하고, 모델링을 진행할 것이기 때문에
train_data_squid=train_full_data[train_full_data['P_NAME']=='오징어'] ## 오징어 제품
train_data_salmon=train_full_data[train_full_data['P_NAME']=='연어'] ## 연어 제품
train_data_white_shrimp=train_full_data[train_full_data['P_NAME']=='흰다리새우'] ## 흰다리새우


## 제품별 새로운 파생변수 생성 (제조국_수입용도_수입형태)
## 제조국, 수입용도, 수입형태 모든 조합별 파생변수를 만들기에는 데이터에 비해 변수가 많이 생길것
for input_p_name in ['train_data_squid','train_data_salmon','train_data_white_shrimp']:
    globals() [input_p_name]['ctry_purpose_import_type']=globals() [input_p_name].apply(lambda x
```

```
## 실루엣 계수 사용
## 실루엣 계수 - 실루엣 계수란 각 군집 간의 거리가 얼마나 효율적으로 분리돼 있는지를 나타내는 계
## 시각화 해석 방법 1 - 계수가 1에 가까울수록 다른 군집과 거리가 멀다는 것을 의미
## 시각화 해석 방법 2 - 평균값에 비해 계수가 떨어지는 군집이 존재하면 안됨

def visualize_silhouette(cluster_lists, X_features):
```

```

    from sklearn.datasets import make_blobs
    from sklearn.cluster import KMeans
    from sklearn.metrics import silhouette_samples, silhouette_score
    import matplotlib.pyplot as plt
    import matplotlib.cm as cm
    import math

    # 입력값으로 클러스터링 갯수들을 리스트로 받아서, 각 갯수별로 클러스터링을 적용하고 실루엣
    n_cols = len(cluster_lists)

    # plt.subplots()으로 리스트에 기재된 클러스터링 수만큼의 sub figures를 가지는 axs 생성 Wn",
    fig, axs = plt.subplots(figsize=(10*n_cols, 10), nrows=1, ncols=n_cols)


    # 리스트에 기재된 클러스터링 갯수들을 차례로 iteration 수행하면서 실루엣 개수 시각화Wn",
    for ind, n_cluster in enumerate(cluster_lists):

       # KMeans 클러스터링 수행하고, 실루엣 스코어와 개별 데이터의 실루엣 값 계산. Wn",
        clusterer = KMeans(n_clusters = n_cluster, max_iter=500, random_state=0)
        cluster_labels = clusterer.fit_predict(X_features)

        sil_avg = silhouette_score(X_features, cluster_labels)
        sil_values = silhouette_samples(X_features, cluster_labels)

        y_lower = 10
        axs[ind].set_title('Number of Cluster : '+ str(n_cluster)+
                           'Silhouette Score :' + str(round(sil_avg,3)) )
        axs[ind].set_xlabel("The silhouette coefficient values")
        axs[ind].set_ylabel("Cluster label")
        axs[ind].set_xlim([-0.1, 1])
        axs[ind].set_ylim([0, len(X_features) + (n_cluster + 1) * 10])
        axs[ind].set_yticks([])  # Clear the yaxis labels / ticksWn",
        axs[ind].set_xticks([0, 0.2, 0.4, 0.6, 0.8, 1])

# 클러스터링 갯수별로 fill_betweenx( )형태의 막대 그래프 표현.
        for i in range(n_cluster):
            ith_cluster_sil_values = sil_values[cluster_labels==i]
            ith_cluster_sil_values.sort()

            size_cluster_i = ith_cluster_sil_values.shape[0]
            y_upper = y_lower + size_cluster_i

            color = cm.nipy_spectral(float(i) / n_cluster)
            axs[ind].fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_sil_values,
                                   facecolor=color, edgecolor=color, alpha=0.7)
            axs[ind].text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
            y_lower = y_upper + 10

        axs[ind].axvline(x=sil_avg, color="red", linestyle="--")


## 실루엣 계수 사용을 위해 라벨인코딩
le = LabelEncoder()
train_data_squid['sil_squid'] = le.fit_transform(train_data_squid['ctry_purpose_import_type'].va

```

```
## 불필요한 열 제거
train_data_squid_sil = train_data_squid.drop(columns=['REG_DATE', 'CTRY_1', 'P_PURPOSE', 'P_NAME
```

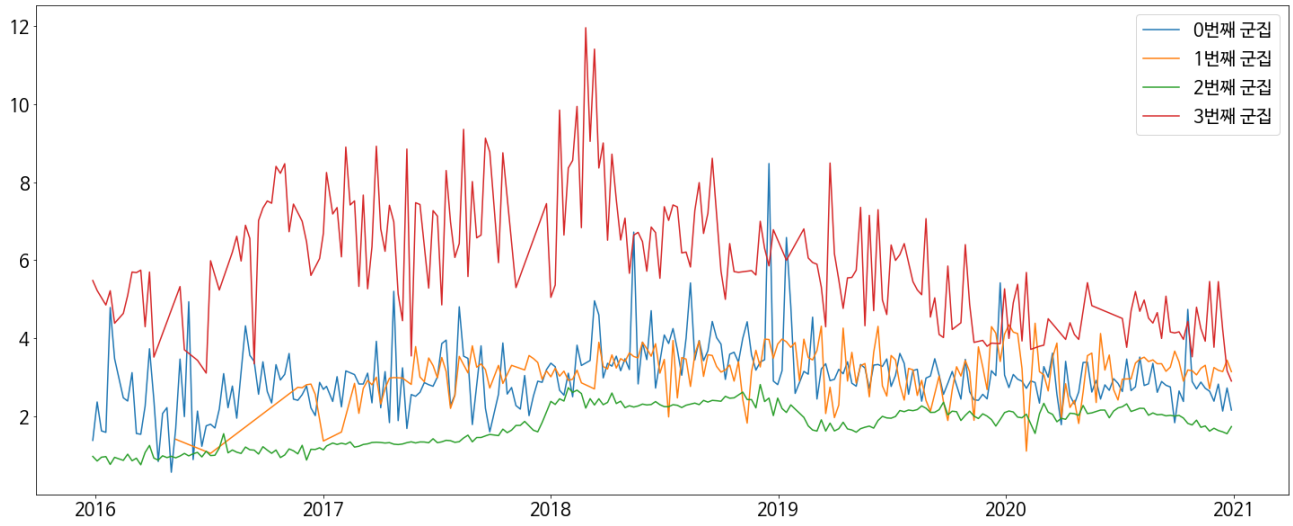
```
## 실루엣 계수 시각화
## 해석 결과 : 오징어의 경우 클러스터 4개가 최적이라 판단
visualize_silhouette([2,3,4,5],train_data_squid_sil)
```




```
## 군집화
## 날짜 및 파생변수(제조국_수입용도_수입형태)별 평균 금액 이용
def kmeans_data_squid(input_data):
    df_C_P_I = input_data.groupby(['ctry_purpose_import_type','REG_DATE'])['P_PRICE'].mean().res
    for i in range(len(df_C_P_I)):
        # 날짜별 평균 금액에 대해 scaling 하여 군집화
        df_C_P_I.iloc[i,1:] = (df_C_P_I.iloc[i,1:] - df_C_P_I.iloc[i,1:].mean())/df_C_P_I.iloc[
    df_C_P_I.fillna(0,inplace=True)
    kmeans = KMeans(n_clusters=4, random_state = 7857)
    km_cluster = kmeans.fit_predict(df_C_P_I.iloc[:,1:])

    df_C_P_I_clust = df_C_P_I.copy()
    df_C_P_I_clust['km_cluster_C_P_I'] = km_cluster

    input_data=input_data.merge(df_C_P_I_clust[['ctry_purpose_import_type','km_cluster_C_P_I']]

    return input_data

```

```
## 오징어 kmeans 실행
train_data_squid=kmeans_data_squid(train_data_squid)

## 오징어 시각화 준비
train_data_squid_cluster=train_data_squid.groupby(['REG_DATE','km_cluster_C_P_I'])['P_PRICE'].me
train_data_squid_cluster.index=train_data_squid_cluster['REG_DATE']
```

```
## 군집별, 날짜별 평균 금액
fig = plt.figure(figsize = (25, 10))

plt.plot(train_data_squid_cluster[train_data_squid_cluster['km_cluster_C_P_I']==0]['P_PRICE'],la
plt.plot(train_data_squid_cluster[train_data_squid_cluster['km_cluster_C_P_I']==1]['P_PRICE'],la
plt.plot(train_data_squid_cluster[train_data_squid_cluster['km_cluster_C_P_I']==2]['P_PRICE'],la
plt.plot(train_data_squid_cluster[train_data_squid_cluster['km_cluster_C_P_I']==3]['P_PRICE'],la

plt.legend(loc='best')
plt.show()
```

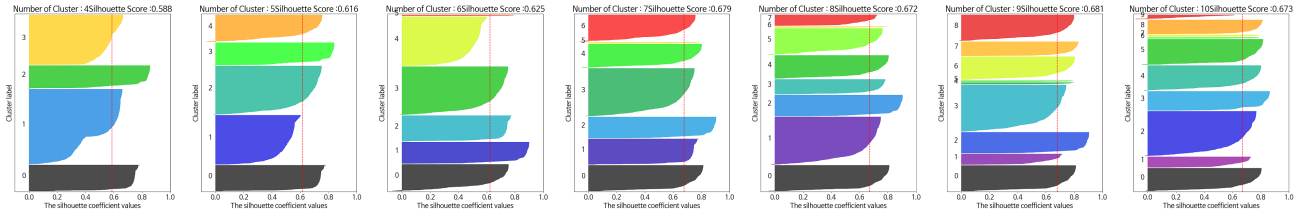


연어 실루엣 검증

```
## 실루엣 계수 사용을 위해 라벨인코딩
train_data_salmon['sil_squid'] = le.fit_transform(train_data_salmon['ctry_purpose_import_type']

## 불필요한 열 제거
train_data_salmon_sil = train_data_salmon.drop(columns=['REG_DATE', 'CTRY_1', 'P_PURPOSE', 'P_NA
```

```
# 실루엣 계수 시각화
# 연어 최적 클러스터 개수는 4개
# 5,6 클러스터로 군집화 하였을때 실루엣 계수는 높지만 평균값(빨간선)에 못미치는 군집이 존재
```

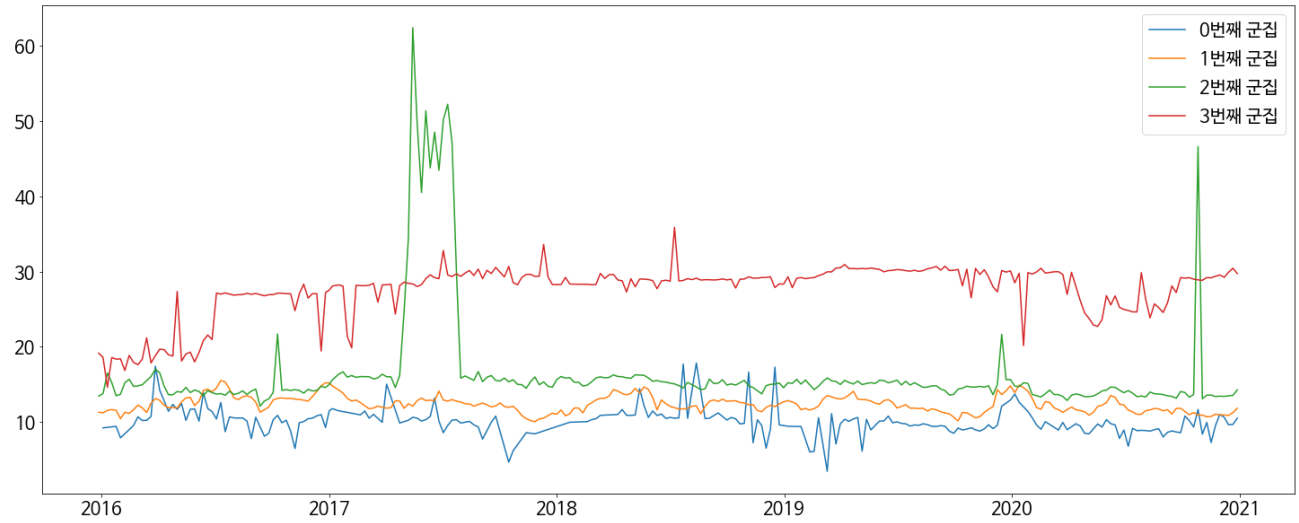
```
4
5 visualize_silhouette([4,5,6,7,8,9,10],train_data_salmon_sil)
```




```
1 ## 군집화
2 def kmeans_data_salmon(input_data):
3     df_C_P_I = input_data.groupby(['ctry_purpose_import_type','REG_DATE'])['P_PRICE'].mean().res
4     for i in range(len(df_C_P_I)):
5         df_C_P_I.iloc[i,1:] = (df_C_P_I.iloc[i,1:] - df_C_P_I.iloc[i,1:].mean())/df_C_P_I.iloc[
6     df_C_P_I.fillna(0,inplace=True)
7     kmeans = KMeans(n_clusters=4, random_state = 7857)
8     km_cluster = kmeans.fit_predict(df_C_P_I.iloc[:,1:])
9
10    df_C_P_I_clust = df_C_P_I.copy()
11    df_C_P_I_clust['km_cluster_C_P_I'] = km_cluster
12
13    input_data=input_data.merge(df_C_P_I_clust[['ctry_purpose_import_type','km_cluster_C_P_I']]
14
15    return input_data
16
```

```
1 ## 연어 kmeans 실행
2 train_data_salmon=kmeans_data_salmon(train_data_salmon)
3
4 ## 연어 시각화 준비
5 train_data_salmon_cluster=train_data_salmon.groupby(['REG_DATE','km_cluster_C_P_I'])['P_PRICE']
6 train_data_salmon_cluster.index=train_data_salmon_cluster['REG_DATE']
```

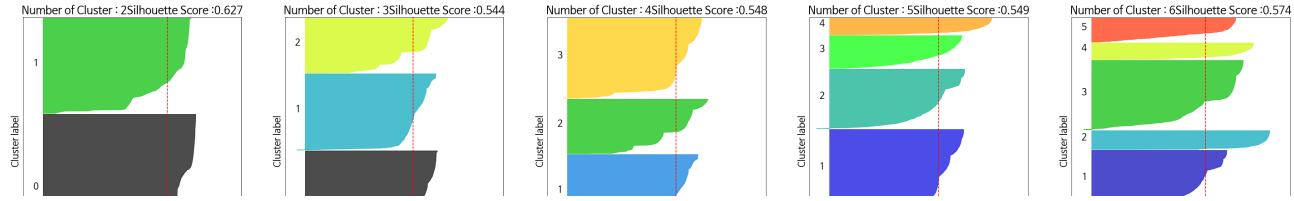
```
1 ## 시각화
2 fig = plt.figure(figsize = (25, 10))
3
4 plt.plot(train_data_salmon_cluster[train_data_salmon_cluster['km_cluster_C_P_I']==0]['P_PRICE']
5 plt.plot(train_data_salmon_cluster[train_data_salmon_cluster['km_cluster_C_P_I']==1]['P_PRICE']
6 plt.plot(train_data_salmon_cluster[train_data_salmon_cluster['km_cluster_C_P_I']==2]['P_PRICE']
7 plt.plot(train_data_salmon_cluster[train_data_salmon_cluster['km_cluster_C_P_I']==3]['P_PRICE']
8
9
10 plt.legend(loc='best')
11 plt.show()
```



흰다리 새우 실루엣 검증

```
## 실루엣 계수 사용을 위해 라벨인코딩
train_data_white_shrimp['sil_squid'] = le.fit_transform(train_data_white_shrimp['ctry_purpose_ir

## 불필요한 열 제거
train_data_white_shrimp_sil = train_data_white_shrimp.drop(columns=['REG_DATE', 'CTRY_1', 'P_PUR
```

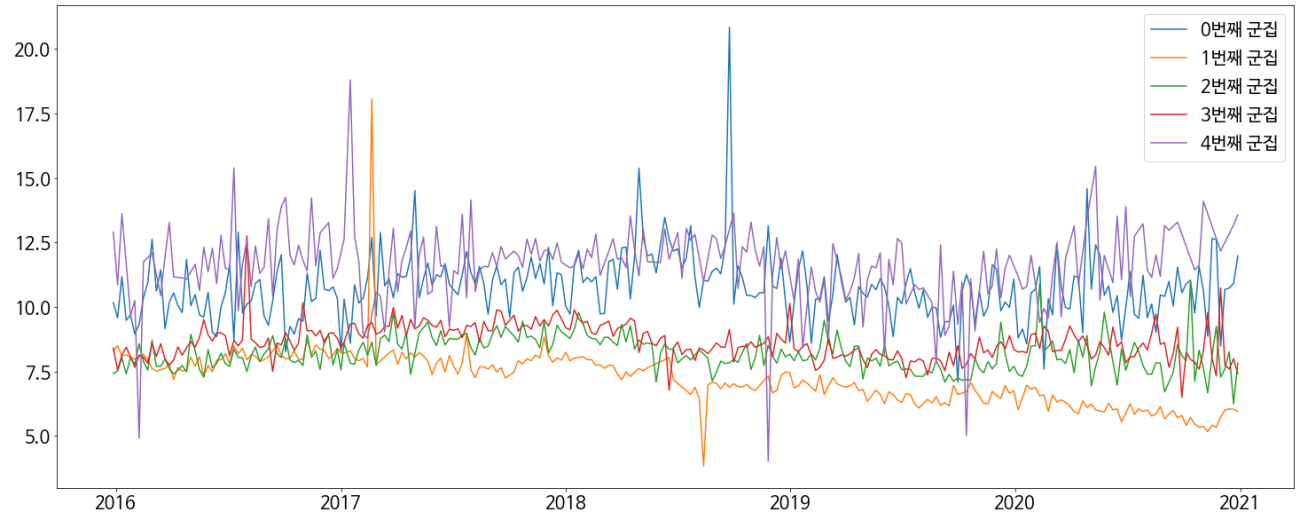
```
## 실루엣 계수 시각화
## 흰다리 새우 최적 클러스터 개수는 5개
## 6개로 클러스터링 하면 실루엣계수는 높지만 0번 클러스터, 1번클러스터에서 평균값을 못넘는 데이터

visualize_silhouette([2,3,4,5,6],train_data_white_shrimp_sil)
```




```
## 군집화
def kmeans_data_white_shrimp(input_data):
    df_C_P_I = input_data.groupby(['ctry_purpose_import_type','REG_DATE'])['P_PRICE'].mean().res
    for i in range(len(df_C_P_I)):
        df_C_P_I.iloc[i,1:] = (df_C_P_I.iloc[i,1:] - df_C_P_I.iloc[i,1:].mean())/df_C_P_I.iloc[
    df_C_P_I.fillna(0,inplace=True)
    kmeans = KMeans(n_clusters=5, random_state = 7857)
    km_cluster = kmeans.fit_predict(df_C_P_I.iloc[:,1:])

    df_C_P_I_clust = df_C_P_I.copy()
    df_C_P_I_clust['km_cluster_C_P_I'] = km_cluster

    input_data=input_data.merge(df_C_P_I_clust[['ctry_purpose_import_type','km_cluster_C_P_I']]

    return input_data

```

```
## 흰다리새우 Kmeans 실행
train_data_white_shrimp=kmeans_data_white_shrimp(train_data_white_shrimp)

## 흰다리 새우 시각화 준비
train_data_white_shrimp_cluster=train_data_white_shrimp.groupby(['REG_DATE','km_cluster_C_P_I']
train_data_white_shrimp_cluster.index=train_data_white_shrimp_cluster['REG_DATE']

```

```
fig = plt.figure(figsize = (25, 10))

plt.plot(train_data_white_shrimp_cluster[train_data_white_shrimp_cluster['km_cluster_C_P_I']==0
plt.plot(train_data_white_shrimp_cluster[train_data_white_shrimp_cluster['km_cluster_C_P_I']==1
plt.plot(train_data_white_shrimp_cluster[train_data_white_shrimp_cluster['km_cluster_C_P_I']==2
plt.plot(train_data_white_shrimp_cluster[train_data_white_shrimp_cluster['km_cluster_C_P_I']==3
plt.plot(train_data_white_shrimp_cluster[train_data_white_shrimp_cluster['km_cluster_C_P_I']==4


plt.legend(loc='best')
plt.show()
```




```
## 상세어종별로 어떤 클러스터링 몇번씩 나타났는지 피벗 만들기위한 준비
train_data_squid_simple=train_data_squid[['REG_DATE','km_cluster_C_P_I']]
train_data_salmon_simple=train_data_salmon[['REG_DATE','km_cluster_C_P_I']]
train_data_white_shrimp_simple=train_data_white_shrimp[['REG_DATE','km_cluster_C_P_I']]


## 상세어종별 피벗테이블
## 오징어
train_data_squid_melt=train_data_squid_simple.melt(id_vars='REG_DATE')
train_data_squid_melt['var_name']=train_data_squid_melt.apply(lambda x : x['variable']+'_'+str(
train_data_squid_pivot=pd.pivot_table(train_data_squid_melt,index='REG_DATE',columns='var_name'
columns=[i[2] for i in train_data_squid_pivot.columns]
train_data_squid_pivot.columns=columns
train_data_squid_pivot.reset_index(inplace=True)

## 연어  train_data_salmon
train_data_salmon_melt=train_data_salmon_simple.melt(id_vars='REG_DATE')
train_data_salmon_melt['var_name']=train_data_salmon_melt.apply(lambda x : x['variable']+'_'+str
train_data_salmon_pivot=pd.pivot_table(train_data_salmon_melt,index='REG_DATE',columns='var_name
columns=[i[2] for i in train_data_salmon_pivot.columns] ## 멀티 컬럼 제거
train_data_salmon_pivot.columns=columns
train_data_salmon_pivot.reset_index(inplace=True)

##흰다리새우  train_data_white_shrimp
train_data_white_shrimp_melt=train_data_white_shrimp_simple.melt(id_vars='REG_DATE')
train_data_white_shrimp_melt['var_name']=train_data_white_shrimp_melt.apply(lambda x : x['variab
train_data_white_shrimp_pivot=pd.pivot_table(train_data_white_shrimp_melt,index='REG_DATE',colu
columns=[i[2] for i in train_data_white_shrimp_pivot.columns]
train_data_white_shrimp_pivot.columns=columns
train_data_white_shrimp_pivot.reset_index(inplace=True)
```

4. 2020년 데이터를 통한 모델 성능평가 및 2021년 클러스터링 예측

XGBoost 이용

```
## 오징어 2015~2020년도 데이터로 2021년도 클러스터링 빈도 예측했을때 몇번씩 등장할것인지 예측
## 학습데이터로 몇번째 전 주를 이용해서 학습시킬껀지 확인
## 13주전이 merror 가장 낮게 나옴 0.355

select_list_colname_squid=['km_cluster_C_P_I_0', 'km_cluster_C_P_I_1','km_cluster_C_P_I_2', 'km_

for input_before_week in range(5,14):
    col_list=select_list_colname_squid
    for i in col_list:
        globals() ['train_data_squid_pivot_{}'.format(i)] = train_data_squid_pivot[['REG_DATE',
        for j in range(1,input_before_week):
            globals() ['train_data_squid_pivot_{}'.format(i)][str(j).zfill(2)+'주전']=train_data
        globals() ['train_data_squid_pivot_{}'.format(i)].dropna(inplace=True)

    list_best_score=[]

    ## 오징어
    for i in tqdm(select_list_colname_squid):
        data=globals()['train_data_squid_pivot_{}'.format(i)].copy()
        data.reset_index(drop=True,inplace=True)

        target=data[i]
        data.drop('REG_DATE',axis=1,inplace=True)

        len_data=data.shape[0]

        feature_columns = list(data.columns.difference(select_list_colname_squid))

        x = data[feature_columns]
        y=target

        x=x.applymap(lambda x : int(x))
        y=y.apply(lambda x : int(x))

        x.reset_index(drop=True,inplace=True)
        y.reset_index(drop=True,inplace=True)

        x_train, x_valid, y_train, y_valid = temporal_train_test_split(x, y, test_size = 0.2) #

        dtrain=xgb.DMatrix(x_train,label=y_train)
        dvalid=xgb.DMatrix(x_valid,label=y_valid)

        params={'objective':'multi:softmax','eval_metric':'merror','random_state':7857,'num_clas

        num_round=500
        watchlist=[(dtrain,'train'),(dvalid,'eval')]
        model=xgb.train(params,dtrain,num_round,evals=watchlist,early_stopping_rounds=20,verbose

        for j in range(0,27):
            dtest=xgb.DMatrix(pd.DataFrame(data.iloc[-1,:].values[0:-1].reshape(1,-1),columns=fe
            result=pd.DataFrame(data.iloc[-1,:].values[0:-1].reshape(1,-1),columns=feature_colu
```

```
            result.insert(0,i,int(np.round(model.predict(dtest),0)))
            data=data.append(result)
            data.reset_index(drop=True,inplace=True)


        globals()['list_cl_{}'.format(i)]=list(data[i].iloc[len_data:].values)

        list_best_score.append(model.best_score)

    print(input_before_week,'주전 변수 있을때 평균 eval-merror:',np.mean(list_best_score))
```

```
100%                                                4/4 [00:00<00:00, 4.88it/s]
5 주전 변수 있을때 평균 eval-merror: 0.35576925
100%                                                4/4 [00:00<00:00, 5.24it/s]
6 주전 변수 있을때 평균 eval-merror: 0.35784324999999995
100%                                                4/4 [00:00<00:00, 5.21it/s]
7 주전 변수 있을때 평균 eval-merror: 0.382353
100%                                                4/4 [00:00<00:00, 4.66it/s]
8 주전 변수 있을때 평균 eval-merror: 0.377451
100%                                                4/4 [00:00<00:00, 4.53it/s]
9 주전 변수 있을때 평균 eval-merror: 0.39705874999999996
100%                                                4/4 [00:00<00:00, 4.64it/s]
10 주전 변수 있을때 평균 eval-merror: 0.36764725
100%                                                4/4 [00:00<00:00, 4.50it/s]
11 주전 변수 있을때 평균 eval-merror: 0.40499999999999997
100%                                                4/4 [00:00<00:00, 4.53it/s]
12 주전 변수 있을때 평균 eval-merror: 0.415
100%                                                4/4 [00:00<00:00, 4.58it/s]
13 주전 변수 있을때 평균 eval-merror: 0.355
```

```
## 2021년 오징어 클러스터링 예측
future_squid_cluster=pd.DataFrame({'km_cluster_C_P_I_0':list_cl_km_cluster_C_P_I_0,
                                   'km_cluster_C_P_I_1':list_cl_km_cluster_C_P_I_1,
                                   'km_cluster_C_P_I_2':list_cl_km_cluster_C_P_I_2,
                                   'km_cluster_C_P_I_3':list_cl_km_cluster_C_P_I_3})
```

```
## 연어 15주전 0.145
select_list_colname_salmon=['km_cluster_C_P_I_0', 'km_cluster_C_P_I_1','km_cluster_C_P_I_2', 'kr


# 연어

for input_before_week in range(5,16):
    col_list=select_list_colname_salmon
    for i in col_list:
        globals() ['train_data_salmon_pivot_{}'.format(i)] = train_data_salmon_pivot[['REG_DATE
        for j in range(1,input_before_week):
            globals() ['train_data_salmon_pivot_{}'.format(i)][str(j).zfill(2)+'주전']=train_da
```

```
        globals() ['train_data_salmon_pivot_{}'.format(i)].dropna(inplace=True)
    list_best_score=[]

    ## 연어
    for i in tqdm(select_list_colname_salmon):
        data=globals()['train_data_salmon_pivot_{}'.format(i)].copy()
        data.reset_index(drop=True,inplace=True)

        target=data[i]
        data.drop('REG_DATE',axis=1,inplace=True)

        len_data=data.shape[0]

        feature_columns = list(data.columns.difference(select_list_colname_salmon))

        x = data[feature_columns]
        y=target

        x=x.applymap(lambda x : int(x))
        y=y.apply(lambda x : int(x))

        x.reset_index(drop=True,inplace=True)
        y.reset_index(drop=True,inplace=True)

        x_train, x_valid, y_train, y_valid = temporal_train_test_split(x, y, test_size = 0.2)

        dtrain=xgb.DMatrix(x_train,label=y_train)
        dvalid=xgb.DMatrix(x_valid,label=y_valid)

        params={'objective':'multi:softmax','eval_metric':'merror','random_state':7857,'num_clas

        num_round=500
        watchlist=[(dtrain,'train'),(dvalid,'eval')]
        model=xgb.train(params,dtrain,num_round,evals=watchlist,early_stopping_rounds=20,verbose

        for j in range(0,27):
            dtest=xgb.DMatrix(pd.DataFrame(data.iloc[-1,:].values[0:-1].reshape(1,-1),columns=fe
            result=pd.DataFrame(data.iloc[-1,:].values[0:-1].reshape(1,-1),columns=feature_colu
            result.insert(0,i,int(np.round(model.predict(dtest),0)))
            data=data.append(result)
            data.reset_index(drop=True,inplace=True)


        globals()['list_cl_{}'.format(i)]=list(data[i].iloc[len_data:].values)

        list_best_score.append(model.best_score)

    print(input_before_week,'주전 변수 있을때 평균 eval-merror:',np.mean(list_best_score))
```

```
100%                                                4/4 [00:00<00:00, 5.66it/s]
5 주전 변수 있을때 평균 eval-merror: 0.17307699999999998
100%                                                4/4 [00:00<00:00, 5.35it/s]
6 주전 변수 있을때 평균 eval-merror: 0.16346175
100%                                                4/4 [00:00<00:00, 5.13it/s]
7 주전 변수 있을때 평균 eval-merror: 0.158654
100%                                                4/4 [00:00<00:00, 5.23it/s]
8 주전 변수 있을때 평균 eval-merror: 0.15196075
100%                                                4/4 [00:00<00:00, 5.17it/s]
9 주전 변수 있을때 평균 eval-merror: 0.15196075
100%                                                4/4 [00:00<00:00, 5.22it/s]
10 주전 변수 있을때 평균 eval-merror: 0.1666665
100%                                                4/4 [00:00<00:00, 4.95it/s]
11 주전 변수 있을때 평균 eval-merror: 0.1666665
100%                                                4/4 [00:00<00:00, 5.12it/s]
12 주전 변수 있을때 평균 eval-merror: 0.17156849999999998
100%                                                4/4 [00:00<00:00, 5.06it/s]
13 주전 변수 있을때 평균 eval-merror: 0.15
```

```
future_salmon_cluster=pd.DataFrame({'km_cluster_C_P_I_0':list_cl_km_cluster_C_P_I_0,
                                    'km_cluster_C_P_I_1':list_cl_km_cluster_C_P_I_1,
                                    'km_cluster_C_P_I_2':list_cl_km_cluster_C_P_I_2,
                                    'km_cluster_C_P_I_3':list_cl_km_cluster_C_P_I_3})
```

```
# 흰다리새우 13주전 0.212
select_list_colname_white_shrimp=['km_cluster_C_P_I_0', 'km_cluster_C_P_I_1','km_cluster_C_P_I_2



for input_before_week in range(5,14):
    col_list=select_list_colname_white_shrimp
    for i in col_list:
        globals() ['train_data_white_shrimp_pivot_{}'.format(i)] = train_data_white_shrimp_pivot
        for j in range(1,input_before_week):
            globals() ['train_data_white_shrimp_pivot_{}'.format(i)][str(j).zfill(2)+'주전']=tra
        globals() ['train_data_white_shrimp_pivot_{}'.format(i)].dropna(inplace=True)
    list_best_score=[]


    for i in tqdm(select_list_colname_white_shrimp):
        data=globals()['train_data_white_shrimp_pivot_{}'.format(i)].copy()
        data.reset_index(drop=True,inplace=True)

        target=data[i]
        data.drop('REG_DATE',axis=1,inplace=True)

        len_data=data.shape[0]

        feature_columns = list(data.columns.difference(select_list_colname_white_shrimp))
```

```
26
27          x = data[feature_columns]
28          y=target
29
30          x=x.applymap(lambda x : int(x))
31          y=y.apply(lambda x : int(x))
32
33          x.reset_index(drop=True,inplace=True)
34          y.reset_index(drop=True,inplace=True)
35
36
37          x_train, x_valid, y_train, y_valid = temporal_train_test_split(x, y, test_size = 0.2)
38
39          dtrain=xgb.DMatrix(x_train,label=y_train)
40          dvalid=xgb.DMatrix(x_valid,label=y_valid)
41
42          params={'objective':'multi:softmax','eval_metric':'merror','random_state':7857,'num_clas
43
44          num_round=500
45          watchlist=[(dtrain,'train'),(dvalid,'eval')]
46          model=xgb.train(params,dtrain,num_round,evals=watchlist,early_stopping_rounds=20,verbose
47
48
49          for j in range(0,27):
50              dtest=xgb.DMatrix(pd.DataFrame(data.iloc[-1,:].values[0:-1].reshape(1,-1),columns=fe
51              result=pd.DataFrame(data.iloc[-1,:].values[0:-1].reshape(1,-1),columns=feature_colu
52              # result.insert(0,'km_cluster_C_P_I_{}'.format(i),int(np.round(model.predict(dtest)
53              result.insert(0,i,int(np.round(model.predict(dtest),0)))
54              data=data.append(result)
55              data.reset_index(drop=True,inplace=True)
56
57
58          globals()['list_cl_{}'.format(i)]=list(data[i].iloc[len_data:].values)
59
60          list_best_score.append(model.best_score)
61
62      print(input_before_week,'주전 변수 있을때 평균 eval-merror:',np.mean(list_best_score))
```

```
100%                                                 5/5 [00:00<00:00, 5.73it/s]
5 주전 변수 있을때 평균 eval-merror: 0.21923079999999998
100%                                                 5/5 [00:00<00:00, 5.76it/s]
6 주전 변수 있을때 평균 eval-merror: 0.22692299999999999
100%                                                 5/5 [00:00<00:00, 5.62it/s]
7 주전 변수 있을때 평균 eval-merror: 0.2230769999999997
```

```
future_white_shrimp_cluster=pd.DataFrame({'km_cluster_C_P_I_0':list_cl_km_cluster_C_P_I_0,
                                          'km_cluster_C_P_I_1':list_cl_km_cluster_C_P_I_1,
                                          'km_cluster_C_P_I_2':list_cl_km_cluster_C_P_I_2,
                                          'km_cluster_C_P_I_3':list_cl_km_cluster_C_P_I_3,
                                          'km_cluster_C_P_I_4':list_cl_km_cluster_C_P_I_4})
```

```
100%                                                 5/5 [00:01<00:00, 5.25it/s]
```

5. 외부데이터 전처리

```
11 주전 변수 있을때 평균 eval-merror: 0.2235296
```

환율데이터

```
100%                                                 5/5 [00:01<00:00, 4.87it/s]
```

```
## 환율 데이터 경로
ex_can = './외부데이터/USD_CAD 내역.csv' ## 캐나다
ex_CHL = './외부데이터/USD_CLP 내역.csv' ## 칠레
ex_CH = './외부데이터/USD_CNH 내역.csv' ## 중국
ex_ENG = './외부데이터/USD_GBP 내역.csv' ## 영국
ex_KOR = './외부데이터/USD_KRW 내역.csv' ## 대한민국
ex_MAL = './외부데이터/USD_MYR 내역.csv' ## 말레이시아
ex_NOR = './외부데이터/USD_NOK 내역.csv' ## 노르웨이
ex_PE = './외부데이터/USD_PEN 내역.csv' ## 페루
ex_TH = './외부데이터/USD_THB 내역.csv' ## 태국
ex_VN = './외부데이터/USD_VND 내역.csv' ## 베트남


## 국가별 환율 데이터 불러오기
train_full_data=pd.DataFrame()
ex_CAN=pd.read_csv(ex_can, parse_dates=['날짜'])
ex_CHL=pd.read_csv(ex_CHL, parse_dates=['날짜'])
ex_CH=pd.read_csv(ex_CH, parse_dates=['날짜'])
ex_ENG=pd.read_csv(ex_ENG, parse_dates=['날짜'])
ex_KOR=pd.read_csv(ex_KOR, parse_dates=['날짜'])
ex_MAL=pd.read_csv(ex_MAL, parse_dates=['날짜'])
ex_NOR=pd.read_csv(ex_NOR, parse_dates=['날짜'])
ex_PE=pd.read_csv(ex_PE, parse_dates=['날짜'])
ex_TH=pd.read_csv(ex_TH, parse_dates=['날짜'])
ex_VN=pd.read_csv(ex_VN, parse_dates=['날짜'])
```

```
# 문자열 형태의 날짜 칼럼 datetime 형식으로 변환
def str_to_date(str_1):

    str_2 = str_1[:4] + '-' + str_1[6:8]+'-'+str_1[10:12]   # 'YYYY년 MM월 DD일' 형태의 데이터를

```

```
    date_return = datetime.strptime(str_2, '%Y-%m-%d')
    return date_return

# 문자열 형태의 종가 칼럼을 분석에 맞게 float형태로 변환
# 대한민국용
def str_to_float_kor(str_1):

    str_2 = str_1[0] +str_1[2:] # 0,000.00 형태에서 1000의자리 소수점 제거

    rate_return = float(str_2)
    return rate_return

# 베트남용
def str_to_float_vn(str_1):

    str_2 = str_1[:2] +str_1[3:] # 00,000.00 형태에서 1000의자리 소수점 제거

    rate_return = float(str_2)
    return rate_return
```

```
## 국가 데이터프레임 이름으로 된 리스트 선언
ctry = ['ex_CAN','ex_CHL','ex_CH','ex_ENG','ex_KOR','ex_MAL','ex_NOR','ex_PE','ex_TH','ex_VN']

## 각 국가 데이터프레임의 날짜칼럼 datetime 데이터타입으로 변환
for i in ctry:
  globals() [i]['날짜'] = globals() [i]['날짜'].apply(lambda x: str_to_date(x))

## 종가 칼럼이 문자열로 이루어진 한국과 베트남 데이터프레임에 대해서 문자열 데이터를 float형 데
ex_KOR['종가'] = ex_KOR['종가'].apply(lambda x: str_to_float_kor(x))
ex_VN['종가'] = ex_VN['종가'].apply(lambda x: str_to_float_vn(x))

## 대한민국데이터의 종가 칼럼을 KRW종가로 변경
ex_KOR.rename(columns={'종가':'KRW종가'},inplace=True)

## 각 데이터에서 불필요한 칼럼(오픈, 고가, 저가. 변동) 제거
for i in ctry:
  globals() [i].drop(columns=['오픈','고가','저가','변동 %'],inplace=True)


# 모델에서 분석할 대한민국을 제외한 나머지 국가들 데이터만 추출
ctry_concat = ['ex_CAN','ex_CHL','ex_CH','ex_ENG','ex_MAL','ex_NOR','ex_PE','ex_TH','ex_VN']

## 기존의 달러대비 환율을 통해 원화대비 환율을 새로 계산한 후 새로운 칼럼으로 생성
for i in ctry_concat:
  globals() [i] = pd.merge(globals()[i],ex_KOR,how='left',left_on='날짜',right_on='날짜')
  globals() [i]['원화대비 환율'+'_'+i[3:]] = globals() [i]['KRW종가']/globals() [i]['종가']
```

유가 데이터

```
## 유가 데이터 경로 및 불러오기
oil = './외부데이터/국제유가2021-09-09.csv' ## 유가
oil_price = pd.read_csv(oil, parse_dates=['Date'])
```

외부 데이터 전처리 작업

```
## 각 어종별 데이터프레임을 따로 생성

## 오징어
## 페루, 칠레, 중국
ctry_squid=['ex_PE','ex_CHL','ex_CH']
for i in ctry_squid:
  globals() [i].drop(columns=['종가','KRW종가'],inplace=True)              # 불필요한 칼럼 제거

## 연어
## 노르웨이, 영국, 캐나다
ctry_salmon=['ex_NOR','ex_ENG','ex_CAN']
for i in ctry_salmon:
  globals() [i].drop(columns=['종가','KRW종가'],inplace=True)              # 불필요한 칼럼 제거

## 흰다리새우
## 베트남, 태국, 말레이시아
ctry_white_shrimp=['ex_VN','ex_TH','ex_MAL']
for i in ctry_white_shrimp:
  globals() [i].drop(columns=['종가','KRW종가'],inplace=True)              # 불필요한 칼럼 제거

```

```
## 어종에 따라 같은 어종 국가는 하나로 병합

## 오징어 주요 수출국 환율데이터
ex_squid = pd.merge(ex_PE,ex_CHL, how='left',left_on='날짜', right_on='날짜')
ex_squid = pd.merge(ex_squid,ex_CH, how='left',left_on='날짜', right_on='날짜')
ex_squid = pd.merge(ex_squid,ex_KOR, how='left',left_on='날짜', right_on='날짜')

## 연어 주요 수출국 환율데이터
ex_salmon = pd.merge(ex_NOR,ex_ENG, how='left',left_on='날짜', right_on='날짜')
ex_salmon = pd.merge(ex_salmon,ex_CAN, how='left',left_on='날짜', right_on='날짜')
ex_salmon = pd.merge(ex_salmon,ex_KOR, how='left',left_on='날짜', right_on='날짜')


## 흰다리새우 주요 수출국 환율데이터
ex_white_shrimp = pd.merge(ex_VN,ex_TH, how='left',left_on='날짜', right_on='날짜')
ex_white_shrimp = pd.merge(ex_white_shrimp,ex_MAL, how='left',left_on='날짜', right_on='날짜')
ex_white_shrimp = pd.merge(ex_white_shrimp,ex_KOR, how='left',left_on='날짜', right_on='날짜')

```

```
## 오징어
## 오징어 피벗 테이블과 유가데이터 병합
data_squid_oil = pd.merge(train_data_squid_pivot, oil_price, how='left', left_on='REG_DATE', ri

## 오징어 피벗 데이블 + 유가 + 환율(오징어 주요 수출국)
data_squid_oil_ex = pd.merge(data_squid_oil, ex_squid, how='left', left_on='REG_DATE', right_on=

```

```
## 연어
## 연어 피벗 테이블과 유가데이터 병합
data_salmon_oil = pd.merge(train_data_salmon_pivot, oil_price, how='left', left_on='REG_DATE', r

## 연어 피벗 데이블 + 유가 + 환율(연어 주요 수출국)
data_salmon_oil_ex = pd.merge(data_salmon_oil, ex_salmon, how='left', left_on='REG_DATE', right_


## 흰다리새우
## 흰다리새우 피벗 테이블과 유가데이터 병합
data_white_shrimp_oil = pd.merge(train_data_white_shrimp_pivot, oil_price, how='left', left_on=

## 흰다리새우 피벗 데이블 + 유가 + 환율(연어 주요 수출국)
data_white_shrimp_oil_ex = pd.merge(data_white_shrimp_oil, ex_white_shrimp, how='left', left_on=
```

```
## 각 어종별 데이터에서 REG_DATE를 제외한 datetime 타입 칼럼 제거

data_squid_oil_ex.drop(columns=['Date','날짜'],inplace=True)
data_salmon_oil_ex.drop(columns=['Date','날짜'],inplace=True)
data_white_shrimp_oil_ex.drop(columns=['Date','날짜'],inplace=True)
```

```
## 결측치 처리
data_squid_oil_ex.interpolate(method="pad", inplace=True)
data_salmon_oil_ex.interpolate(method="pad", inplace=True)
data_white_shrimp_oil_ex.interpolate(method="pad", inplace=True)
```

6. 제품별 평균 단가 예측 성능 평가

target data : 2020년도 데이터 성능파악결과 예측 방법론에 문제 없다고 판단하여 2021년도 데이터 예측 진행

```
## 가격예측에 맞게 불필요한 열 제거
train_data_squid_cluster_price_grp = train_data_squid.drop(columns=['CTRY_1','P_PURPOSE', 'P_NAM
train_data_salmon_cluster_price_grp = train_data_salmon.drop(columns=['CTRY_1','P_PURPOSE', 'P_N
train_data_white_shrimp_price_grp = train_data_white_shrimp.drop(columns=['CTRY_1','P_PURPOSE',
```

```
## 평가용 데이터 날짜 불러오기
REG_DATE_2021=pd.read_excel('./제공데이터/2021 빅콘테스트_데이터분석분야_챔피언리그_수산Biz_평가
eval=REG_DATE_2021[['일자']]
eval.rename(columns={'일자':'REG_DATE'},inplace=True)
```

```
## 2021년 클러스터링
eval_squid = pd.merge(eval, future_squid_cluster, how='left',left_index=True, right_index=True
eval_salmon = pd.merge(eval, future_salmon_cluster, how='left',left_index=True, right_index=Tru
eval_white_shrimp = pd.merge(eval, future_white_shrimp_cluster, how='left',left_index=True, righ
```

```
## 유가 및 환율 데이터 연걸
eval_squid_oil = pd.merge(eval_squid, oil_price, how='left',left_on='REG_DATE', right_on='Date'
eval_squid_oil_ex = pd.merge(eval_squid_oil, ex_squid, how='left', left_on='REG_DATE', right_on
```

```
eval_squid_oil_ex = pd.merge(eval_squid_oil, ex_squid, how='left' , left_on='REG_DATE' , right_on=

eval_salmon_oil = pd.merge(eval_salmon, oil_price, how='left',left_on='REG_DATE', right_on='Date
eval_salmon_oil_ex = pd.merge(eval_salmon_oil, ex_salmon, how='left', left_on='REG_DATE', right_

eval_white_shrimp_oil = pd.merge(eval_white_shrimp, oil_price, how='left',left_on='REG_DATE', r
eval_white_shrimp_oil_ex = pd.merge(eval_white_shrimp_oil, ex_white_shrimp, how='left', left_on=

eval_squid_oil_ex.drop(columns=['Date','날짜'],inplace=True)
eval_salmon_oil_ex.drop(columns=['Date','날짜'],inplace=True)
eval_white_shrimp_oil_ex.drop(columns=['Date','날짜'],inplace=True)
```

```
## 2015 ~ 2020년도 데이터에 가격 붙이기
train_data_squid_cluster_price_grp = train_data_squid.drop(columns=['CTRY_1','P_PURPOSE', 'P_NAI
train_data_salmon_cluster_price_grp = train_data_salmon.drop(columns=['CTRY_1','P_PURPOSE', 'P_
train_data_white_shrimp_cluster_price_grp = train_data_white_shrimp.drop(columns=['CTRY_1','P_PU

data_squid_oil_ex_price = pd.merge(data_squid_oil_ex, train_data_squid_cluster_price_grp, how='
data_salmon_oil_ex_price = pd.merge(data_salmon_oil_ex, train_data_salmon_cluster_price_grp, hov
data_white_shrimp_oil_ex_price = pd.merge(data_white_shrimp_oil_ex, train_data_white_shrimp_clus
```

```
## 결측치 보간법으로 대체
data_squid_oil_ex_price.interpolate(method="pad", inplace=True)
data_salmon_oil_ex_price.interpolate(method="pad", inplace=True)
data_white_shrimp_oil_ex_price.interpolate(method="pad", inplace=True)
```

가격 예측 모델링

```
## mse 계산하는 생성함수
def get_rmse(y,pred):
    mse = mean_squared_error(y,pred)
    rmse = np.sqrt(mse)
    print('RMSE 값:', np.round(rmse,3))
    return rmse
```

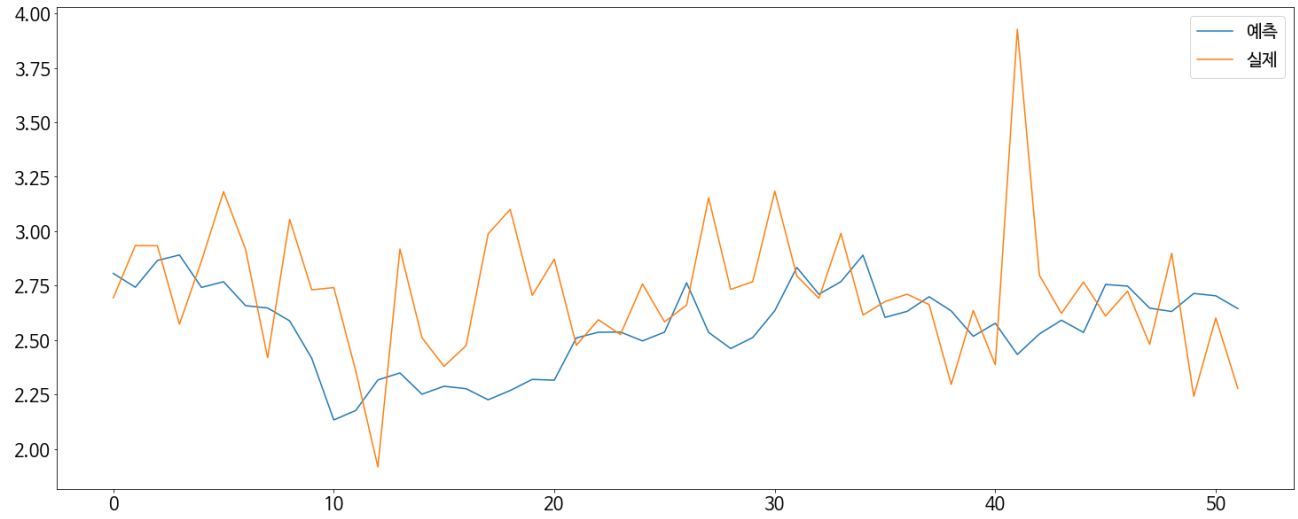
```
## 2021년도까지 연결한 데이터에서 2021년도 제거
data_squid_price = pd.concat([data_squid_oil_ex_price, eval_squid_oil_ex], ignore_index=True)
data_salmon_price = pd.concat([data_salmon_oil_ex_price, eval_salmon_oil_ex], ignore_index=True
data_white_shrimp_price = pd.concat([data_white_shrimp_oil_ex_price, eval_white_shrimp_oil_ex],

data_squid_price_test=data_squid_price.copy()
data_salmon_price_test=data_salmon_price.copy()
data_white_shrimp_price_test=data_white_shrimp_price.copy()

data_squid_price.dropna(axis=0,inplace=True)
data_salmon_price.dropna(axis=0,inplace=True)
data_white_shrimp_price.dropna(axis=0,inplace=True)
```

오징어 가격 예측 성능평가

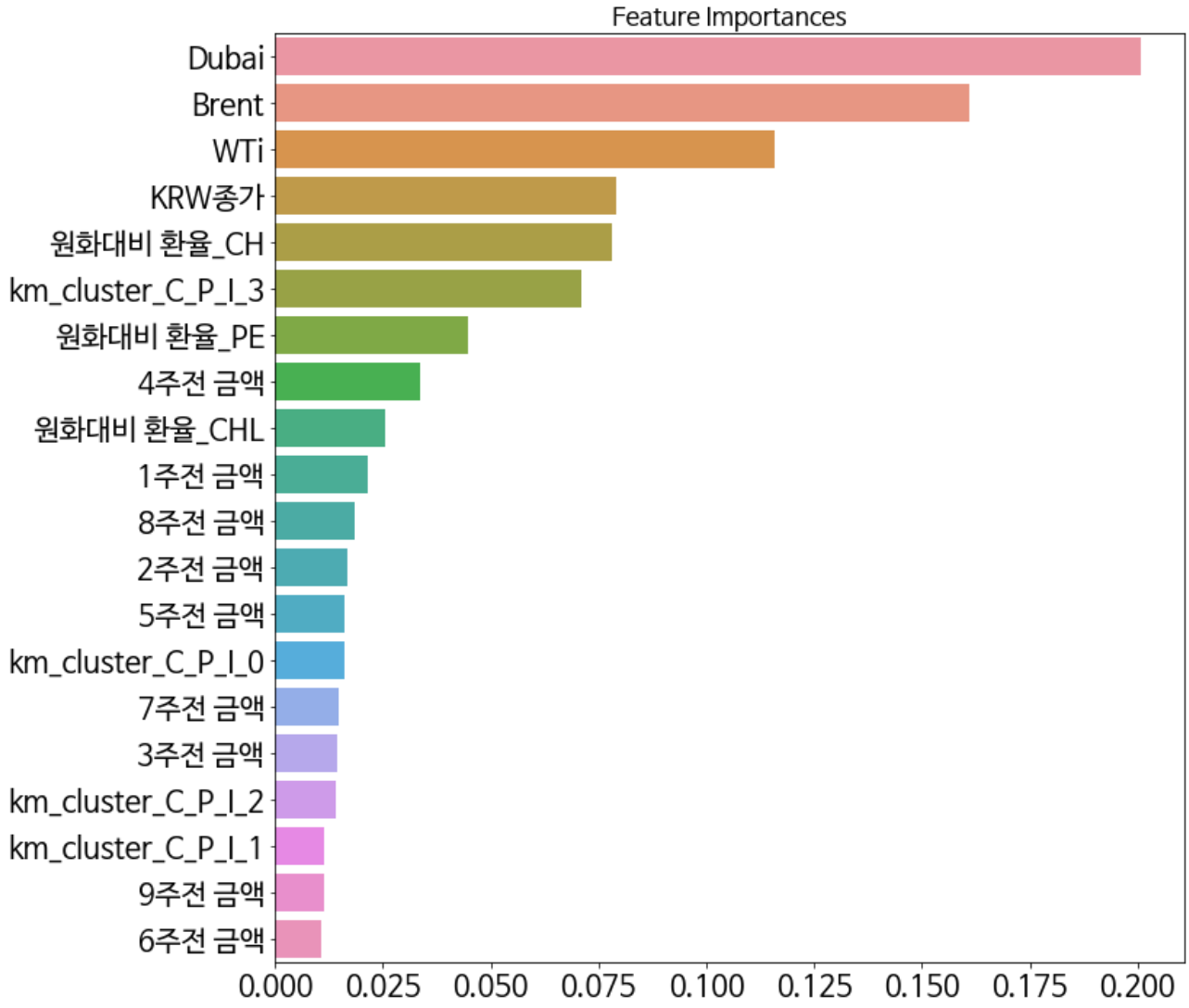
```
## 1주 ~ 12주 전 가격을 각각 하나의 칼럼을 생성하여 저장
for i in range(1,13):
    data_squid_price['{}주전 금액'.format(i)]=data_squid_price['P_PRICE'].shift(i)


rmse_list =[] ## 평균 rmse 계산


## 2020년 이전 데이터는 Train 데이터로
## 2020년 데이터는 Test 데이터로 저장
data_squid_20_train = data_squid_price.head(-52)
data_squid_20_test = data_squid_price.tail(52)

## 2020년 가격 예측 결과를 저장할 P_PRICE_pred
data_squid_20_train['P_PRICE_pred'] = data_squid_20_train['P_PRICE']
data_squid_20_test['P_PRICE_pred'] = np.nan          # 2020년 데이터의 P_PRICE_pred 칼럼은 모두

# 2020년 데이터를 한 행씩 예측 후 Train 데이터에 연결 후 연결된 새로운 데이터를 바탕으로 다시 여
for i in tqdm(range(208,260)):
    test = data_squid_20_test[data_squid_20_test.index == i]


    X_train = data_squid_20_train.drop(['P_PRICE','REG_DATE','P_PRICE_pred'], axis=1, inplace=Fa
    y_train = data_squid_20_train['P_PRICE_pred']

    X_test = data_squid_20_test.drop(['P_PRICE','REG_DATE','P_PRICE_pred'], axis=1, inplace=Fals
    y_test = data_squid_20_test['P_PRICE']

    my_model = XGBRegressor(n_estimators=10000,learning_rate=0.01,seed=7857,max_depth=30,colsamp
    my_model.fit(X_train, y_train,  verbose=False)
    pred = my_model.predict(X_test)
    data_squid_20_test['P_PRICE_pred'] = pred # 예측 결과 값은 P_PRICE_pred 칼럼에 저장

    data_squid_20_train = pd.concat([data_squid_20_train, data_squid_20_test])
    rmse=get_rmse(y_test, pred)
    rmse_list.append(rmse)     # 각 행마다 RMSE 출력

    for i in range(1,13):
        data_squid_20_train['{}주전 금액'.format(i)]=data_squid_20_train['P_PRICE_pred'].shift(


rmse_avg = np.array(rmse_list)
print('평균 RMSE 값은 :',rmse_avg.mean())
```

```
100%                                                52/52 [08:31<00:00, 17.73s/it]
RMSE 값: 0.433
RMSE 값: 0.402
RMSE 값: 0.395
RMSE 값: 0.395
RMSE 값: 0.389
RMSE 값: 0.392
RMSE 값: 0.392
RMSE 값: 0.388
RMSE 값: 0.387
RMSE 값: 0.387
RMSE 값: 0.387
RMSE 값: 0.385
RMSE 값: 0.382
RMSE 값: 0.381
RMSE 값: 0.383
RMSE 값: 0.385
RMSE 값: 0.387
RMSE 값: 0.389
RMSE 값: 0.387
RMSE 값: 0.39
RMSE 값: 0.389
RMSE 값: 0.39
RMSE 값: 0.39
RMSE 값: 0.389
RMSE 값: 0.389
RMSE 값: 0.39
RMSE 값: 0.392
RMSE 값: 0.392
RMSE 값: 0.39
RMSE 값: 0.389
RMSE 값: 0.39
RMSE 값: 0.388
RMSE 값: 0.387
RMSE 값: 0.388
RMSE 값: 0.39
RMSE 값: 0.387
RMSE 값: 0.388
RMSE 값: 0.387
RMSE 값: 0.386
RMSE 값: 0.386
RMSE 값: 0.387
RMSE 값: 0.386
RMSE 값: 0.385
RMSE 값: 0.385
RMSE 값: 0.385
RMSE 값: 0.384
```

```
## 시각화
fig = plt.figure(figsize = (25, 10))

plt.plot(my_model.predict(X_test),label='예측')
plt.plot(y_test.values,label='실제')

plt.legend(loc='best')
plt.show()

```




```
## 중요도 파악
## 외부데이터가 중요한것으로 판단
def get_top_features(model):
    ftr_importances_values = model.feature_importances_
    ftr_importances = pd.Series(ftr_importances_values, index=X_train.columns  )
    ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]
    return ftr_top20

def visualize_ftr_importances(models):
    # 2개 회귀 모델의 시각화를 위해 2개의 컬럼을 가지는 subplot 생성
    fig, axs = plt.subplots(figsize=(10,10),nrows=1, ncols=1)
    fig.tight_layout()
    # 입력인자로 받은 list객체인 models에서 차례로 model을 추출하여 피처 중요도 시각화.

    # 중요도 상위 20개의 피처명과 그때의 중요도값 추출
    ftr_top20 = get_top_features(models)
    plt.title('Feature Importances',size=17)
    sns.barplot(x=ftr_top20.values, y=ftr_top20.index ) # , ax=axs[1]


visualize_ftr_importances(my_model)
```



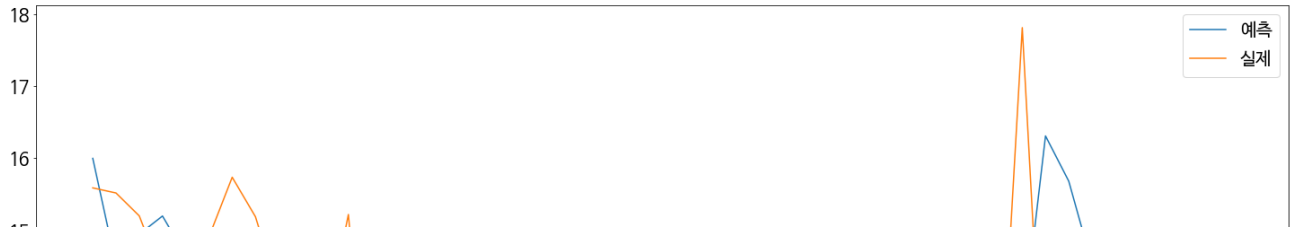
연어 가격 예측 성능평가

```
 1
 2 ## 1주 ~ 12주 전 가격을 각각 하나의 칼럼을 생성하여 저장
 3 for i in range(1,13):
 4     data_salmon_price['{}주전 금액'.format(i)]=data_salmon_price['P_PRICE'].shift(i)
 5
 6 rmse_list =[]
 7
 8 ## 2020년 이전 데이터는 Train 데이터로
 9 ## 2020년 데이터는 Test 데이터로 저장
10 data_salmon_20_train = data_salmon_price.head(-50)
11 data_salmon_20_test = data_salmon_price.tail(50)
12
13 ## 2020년 가격 예측 결과를 저장할 P_PRICE_pred
14 data_salmon_20_train['P_PRICE_pred'] = data_salmon_20_train['P_PRICE']
15 data_salmon_20_test['P_PRICE_pred'] = np.nan          # 2020년 데이터의 P_PRICE_pred 칼럼은 모두
16
```

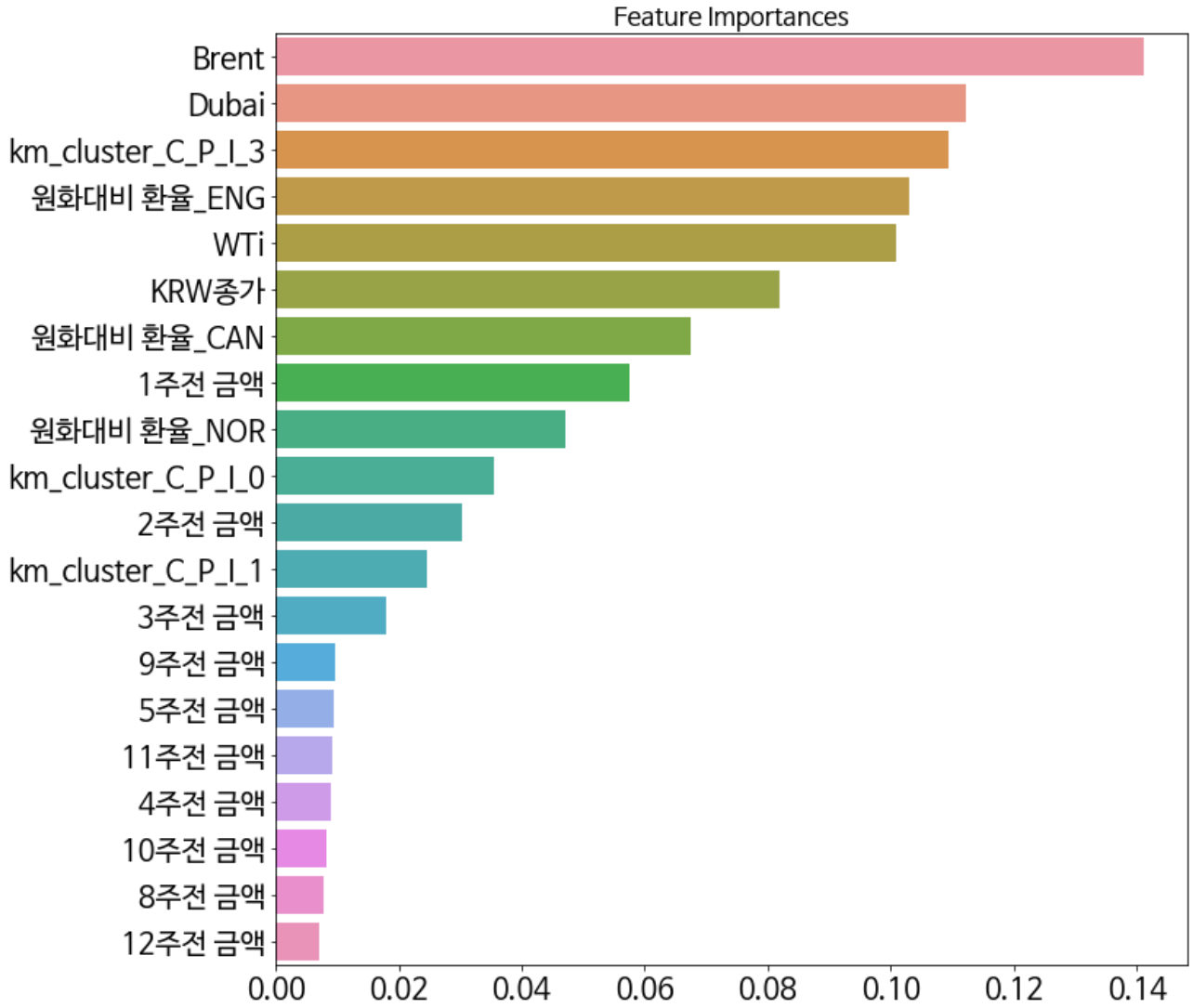
```
# 2020년 데이터를 한 행씩 예측 후 Train 데이터에 연결 후 연결된 새로운 데이터를 바탕으로 다시 여
for i in tqdm(range(210,260)):
    test = data_salmon_20_test[data_salmon_20_test.index == i]


    X_train = data_salmon_20_train.drop(['P_PRICE','REG_DATE','P_PRICE_pred'], axis=1, inplace=F
    y_train = data_salmon_20_train['P_PRICE_pred']

    X_test = data_salmon_20_test.drop(['P_PRICE','REG_DATE','P_PRICE_pred'], axis=1, inplace=Fa
    y_test = data_salmon_20_test['P_PRICE']

    my_model = XGBRegressor(n_estimators=10000,learning_rate=0.01,seed=7857,max_depth=30,colsamp
    my_model.fit(X_train, y_train,  verbose=False)
    pred = my_model.predict(X_test)
    data_salmon_20_test['P_PRICE_pred'] = pred # 예측 결과 값은 P_PRICE_pred 칼럼에 저장

    data_salmon_20_train = pd.concat([data_salmon_20_train, data_salmon_20_test])
    rmse=get_rmse(y_test, pred)
    rmse_list.append(rmse)  # 각 행마다 RMSE 출력

    for i in range(1,13):
        data_salmon_20_train['{}주전 금액'.format(i)]=data_salmon_20_train['P_PRICE_pred'].shif

rmse_avg = np.array(rmse_list)
print('평균 RMSE 값은 :',rmse_avg.mean())
```

```
100%                                                  50/50 [08:37<00:00, 18.54s/it]
RMSE 값: 1.14
RMSE 값: 1.196
RMSE 값: 1.214
RMSE 값: 1.236
RMSE 값: 1.249
RMSE 값: 1.246
RMSE 값: 1.241
RMSE 값: 1.256
RMSE 값: 1.246
RMSE 값: 1.261
RMSE 값: 1.245
RMSE 값: 1.251
RMSE 값: 1.245
RMSE 값: 1.254
RMSE 값: 1.26
RMSE 값: 1.274
RMSE 값: 1.27
RMSE 값: 1.269
RMSE 값: 1.268
RMSE 값: 1.276
RMSE 값: 1.284
RMSE 값: 1.286
```

```
1 fig = plt.figure(figsize = (25, 10))
2
3 plt.plot(my_model.predict(X_test),label='예측')
4 plt.plot(y_test.values,label='실제')
5
6 plt.legend(loc='best')
7 plt.show()
8
```




```
1 visualize_ftr_importances(my_model)
```



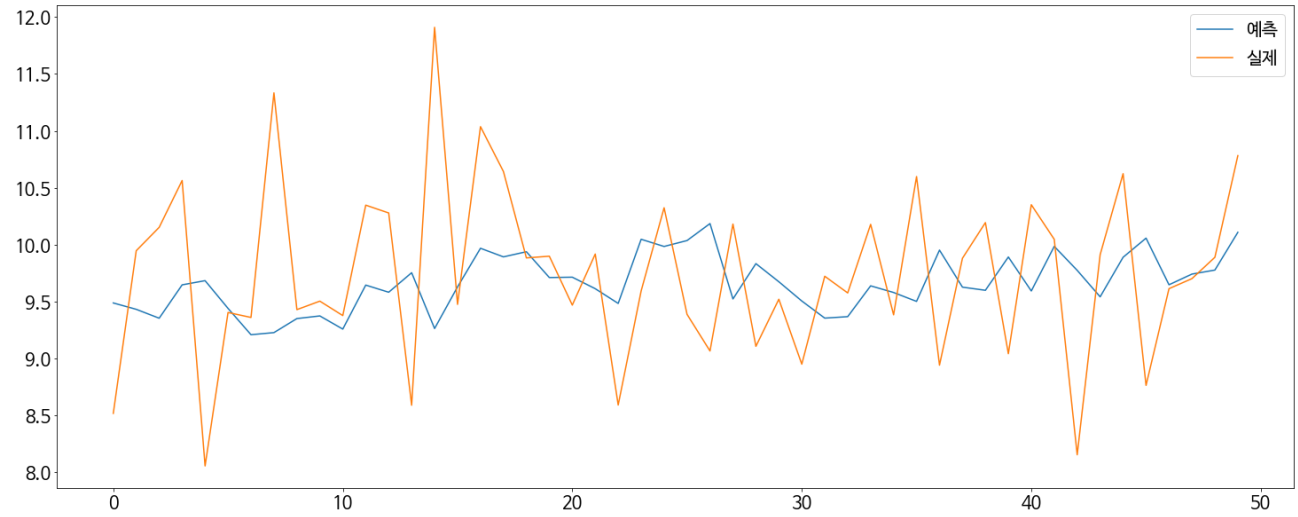
흰다리새우 가격 예측 성능평가

```
1 ## 1주 ~ 12주 전 가격을 각각 하나의 칼럼을 생성하여 저장
2 for i in range(1,13):
3     data_white_shrimp_price['{}주전 금액'.format(i)]=data_white_shrimp_price['P_PRICE'].shift(i
4
5 rmse_list =[]
```

```
rmse_list = []

## 2020년 이전 데이터는 Train 데이터로
## 2020년 데이터는 Test 데이터로 저장
data_white_shrimp_20_train = data_white_shrimp_price.head(-50)
data_white_shrimp_20_test = data_white_shrimp_price.tail(50)

## 2020년 가격 예측 결과를 저장할 P_PRICE_pred
data_white_shrimp_20_train['P_PRICE_pred'] = data_white_shrimp_20_train['P_PRICE']
data_white_shrimp_20_test['P_PRICE_pred'] = np.nan # 2020년 데이터의 P_PRICE_pred 칼럼은 모두 Nu

# 2020년 데이터를 한 행씩 예측 후 Train 데이터에 연결 후 연결된 새로운 데이터를 바탕으로 다시 여
for i in tqdm(range(210,260)):
    test = data_white_shrimp_20_test[data_white_shrimp_20_test.index == i]


    X_train = data_white_shrimp_20_train.drop(['P_PRICE','REG_DATE','P_PRICE_pred'], axis=1, inp
    y_train = data_white_shrimp_20_train['P_PRICE_pred']

    X_test = data_white_shrimp_20_test.drop(['P_PRICE','REG_DATE','P_PRICE_pred'], axis=1, inpla
    y_test = data_white_shrimp_20_test['P_PRICE']

    my_model = XGBRegressor(n_estimators=10000,learning_rate=0.01,seed=7857,max_depth=30,colsamp
    my_model.fit(X_train, y_train,  verbose=False)
    pred = my_model.predict(X_test)
    data_white_shrimp_20_test['P_PRICE_pred'] = pred # 예측 결과 값은 P_PRICE_pred 칼럼에 저장

    data_white_shrimp_20_train = pd.concat([data_white_shrimp_20_train, data_white_shrimp_20_tes
    rmse=get_rmse(y_test, pred)
    rmse_list.append(rmse)     # 각 행마다 RMSE 출력

    for i in range(1,13):
        data_white_shrimp_20_train['{}주전 금액'.format(i)]=data_white_shrimp_20_train['P_PRICE_


rmse_avg = np.array(rmse_list)
print('평균 RMSE 값은 :',rmse_avg.mean())
```

```
100%                                                  50/50 [08:30<00:00, 18.23s/it]
RMSE 값: 0.853
RMSE 값: 0.845
RMSE 값: 0.847
RMSE 값: 0.849
RMSE 값: 0.847
RMSE 값: 0.842
RMSE 값: 0.848
RMSE 값: 0.847
RMSE 값: 0.855
RMSE 값: 0.854
RMSE 값: 0.852
RMSE 값: 0.853
RMSE 값: 0.853
RMSE 값: 0.853
RMSE 값: 0.848
RMSE 값: 0.846
RMSE 값: 0.844
RMSE 값: 0.846
RMSE 값: 0.844
RMSE 값: 0.845
RMSE 값: 0.843
RMSE 값: 0.847
RMSE 값: 0.846
RMSE 값: 0.847
RMSE 값: 0.848
RMSE 값: 0.846
RMSE 값: 0.85
RMSE 값: 0.846
RMSE 값: 0.849
RMSE 값: 0.847
RMSE 값: 0.849
RMSE 값: 0.851
RMSE 값: 0.85
RMSE 값: 0.844
```

```
1 fig = plt.figure(figsize = (25, 10))
2
3 plt.plot(my_model.predict(X_test),label='예측')
4 plt.plot(y_test.values,label='실제')
5
6 plt.legend(loc='best')
7 plt.show()
8
```




```
1 visualize_ftr_importances(my_model)
```

7. 2021년도 예측

위의 방법을 이용해서 2020년 까지 데이터로 2021년 예측


```
## 12주전 까지 데이터 생성

## 오징어
for i in range(1,13):
    data_squid_price_test['{}주전 금액'.format(i)]=data_squid_price_test['P_PRICE'].shift(i)

## 연어
for i in range(1,13):
    data_salmon_price_test['{}주전 금액'.format(i)]=data_salmon_price_test['P_PRICE'].shift(i)

## 흰다리새우
for i in range(1,13):
    data_white_shrimp_price_test['{}주전 금액'.format(i)]=data_white_shrimp_price_test['P_PRICE
```


```
## 2021년은 타겟 데이터
data_squid_test = data_squid_price_test.iloc[260:]
data_salmon_test = data_salmon_price_test.iloc[260:]
data_white_shrimp_test = data_white_shrimp_price_test.iloc[260:]

## 2021년 이전은 훈련 데이터
data_squid_train = data_squid_price_test.iloc[:260]
data_salmon_train = data_salmon_price_test.iloc[:260]
data_white_shrimp_train = data_white_shrimp_price_test.iloc[:260]


# 훈련데이터에서 1~12주 전의 데이터가 없는 행은 제거
data_squid = data_squid_train.dropna()
data_salmon = data_salmon_train.dropna()
data_white_shrimp = data_white_shrimp_train.dropna()
```

```
## 2021년 오징어 가격 예측

for i in tqdm(range(260,286)):
    test = data_squid_test[data_squid_test.index == i]

    y_train = data_squid['P_PRICE']
    X_train = data_squid.drop(['P_PRICE','REG_DATE'], axis=1, inplace=False)
    X_test = test.drop(['P_PRICE','REG_DATE'], axis=1, inplace=False)

    my_model = XGBRegressor(n_estimators=10000,learning_rate=0.001,seed=7857,max_depth=30,colsar
    my_model.fit(X_train, y_train,  verbose=False)
    pred = my_model.predict(X_test)
    test['P_PRICE'] = pred

```

```
    data_squid = pd.concat([data_squid, test])

    for i in range(1,13):
        data_squid['{}주전 금액'.format(i)]=data_squid['P_PRICE'].shift(i)

```

```
100%                                                  26/26 [02:57<00:00, 7.09s/it]
```

```
## 2021년 연어 가격 예측

for i in tqdm(range(262,288)):
    test = data_salmon_test[data_salmon_test.index == i]

    y_train = data_salmon['P_PRICE']
    X_train = data_salmon.drop(['P_PRICE','REG_DATE'], axis=1, inplace=False)
    X_test = test.drop(['P_PRICE','REG_DATE'], axis=1, inplace=False)


    my_model = XGBRegressor(n_estimators=10000,learning_rate=0.001,seed=7857,max_depth=30,colsar
    my_model.fit(X_train, y_train,  verbose=False)
    pred = my_model.predict(X_test)
    test['P_PRICE'] = pred

    data_salmon = pd.concat([data_salmon, test])

    for i in range(1,13):
        data_salmon['{}주전 금액'.format(i)]=data_salmon['P_PRICE'].shift(i)

```

```
100%                                                  26/26 [02:45<00:00, 6.64s/it]
```

```
## 2021년 흰다리새우 가격 예측

for i in tqdm(range(262,288)):
    test = data_white_shrimp_test[data_white_shrimp_test.index == i]

    y_train = data_white_shrimp['P_PRICE']
    X_train = data_white_shrimp.drop(['P_PRICE','REG_DATE'], axis=1, inplace=False)
    X_test = test.drop(['P_PRICE','REG_DATE'], axis=1, inplace=False)


    my_model = XGBRegressor(n_estimators=10000,learning_rate=0.001,seed=7857,max_depth=30,colsar
    my_model.fit(X_train, y_train,  verbose=False)
    pred = my_model.predict(X_test)
    test['P_PRICE'] = pred

    data_white_shrimp = pd.concat([data_white_shrimp, test])

    for i in range(1,13):
        data_white_shrimp['{}주전 금액'.format(i)]=data_white_shrimp['P_PRICE'].shift(i)

```

100% 26/26 [03:03<00:00, 7.53s/it]

8. 최종결과물

```
sub_data_squid=data_squid[['REG_DATE','P_PRICE']].tail(26)
sub_data_squid.reset_index(drop=True,inplace=True)
sub_data_squid.to_excel('/content/drive/MyDrive/TAVE_빅콘테스트/제출데이터/2021_오징어_단가예측
```

```
sub_data_salmon=data_salmon[['REG_DATE','P_PRICE']].tail(26)
sub_data_salmon.reset_index(drop=True,inplace=True)
sub_data_salmon.to_excel('/content/drive/MyDrive/TAVE_빅콘테스트/제출데이터/2021_연어_단가예측.x
```

```
sub_data_white_shrimp=data_white_shrimp[['REG_DATE','P_PRICE']].tail(26)
sub_data_white_shrimp.reset_index(drop=True,inplace=True)
sub_data_white_shrimp.to_excel('/content/drive/MyDrive/TAVE_빅콘테스트/제출데이터/2021_흰다리새
```

```
1
```

✓ 0초 오후 10:58에 완료됨