[주제 및 데이터]

1. 주제

제주도 특산물의 가격을 예측하는 AI 모델 개발 및 인사이트 발굴

2. 데이터

1) train.csv: 2019년 01월 01일부터 2023년 03월 03일까지의 유통된 품목의 가격 데이터

- item: 품목 코드 (TG : 감귤 / BC : 브로콜리 / RD : 무 / CR : 당근 / CB : 양배추)

- corporation : 유통 법인 코드 (법인 A부터 F 존재)

- location : 지역 코드 (J : 제주도 제주시, S : 제주도 서귀포시)

- supply(kg) : 유통된 물량, kg 단위

- price(원/kg) : 유통된 품목들의 kg 마다의 가격, 원 단위

2) test.csv: 2023년 03월 04일부터 2023년 03월 31일까지의 데이터

- 예측 대상: price(원/kg)값이 결측치(NaN)로 되어 있으며, 학습된 모델로 예측값을 채워 최종 제출 파일을 만드는 데 사용되었습니다.

[코드 리뷰]

1. 라이브러리 임포트 및 버전 확인

```
import pandas as pd
import numpy as np
import datetime
import random
import os
import sys
import holidays


import sklearn
from sklearn.ensemble import VotingRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error


import xgboost
from xgboost import XGBRegressor
import catboost
from catboost import CatBoostRegressor
```

```
print(f"파이썬 버전 : {sys.version}")
print(f"pandas 버전 : {pd.__version__}")
print(f"numpy 버전 : {np.__version__}")
print(f"sklearn 버전 : {sklearn.__version__}")
print(f"xgboost 버전 확인 : {xgboost.__version__}")
print(f"catboost 버전 : {catboost.__version__}")
```

기본 라이브러리 임포트	데이터를 다루고 기본적인 연산을 수행하기 위해 pandas(pd), numpy(np), 날짜/시간 처리를 위해 datetime, random, os, sys등을 불러온다. holidays 라이브러리는 공휴일 특성 생성에 사용될 것이다.
사이킷런 임포트	머신러닝의 기본 도구들을 불러온다. VotingRegressor는 여러 모델의 예측을 결합하는 앙상블 기법에 사용되며, LabelEncoder는 범주형 데이터를 숫자로 변환하는 전처리에, mean_squared_error는 회귀 모델의 성능 평가 지표로 사용된다.
부스팅 모델 임포트	고성능 예측에 주로 사용되는 부스팅 모델인 XGBRegressor와 CatBoostRegressor를 각각 xgboost와 catboost라이브러리에서 불러온다.
버전 확인	사용 중인 파이썬과 핵심 데이터 과학 라이브러리(pandas, numpy, sklearn, xgboost, catboost)의 버전을 출력한다. 이는 나중에 코드를 재현할 때 발생할 수 있는 오류를 방지하기 위함이다.

2. 시드 고정 (재현성 확보)

: 분석 과정의 무작위성(랜덤성)을 통제하여, 코드를 언제, 어디서 실행하더라도 동일한 결과가 나오도록 재현성을 확보한다.

```
def seed_everything(seed: int = 2024):
    random.seed(seed)
    np.random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
seed_everything(2024)
```

def seed_everything(seed: int = 2024):	시드 값을 고정하는 함수를 정의한다. 기본 시드 값은 2024로 설정되었다.
random.seed(seed)	Python의 기본 random모듈의 시드를 고정한다.
np.random.seed(seed)	NumPy의 무작위 함수 시드를 고정한다. NumPy는 데이터 연산에 광범위하게 사용된다.
os.environ["PYTHONHASHSEED"] = str(seed)	Python의 해시(Hash) 알고리즘 시드를 고정하여, 해시 기반 연산(딕셔너리, 집합 등)의 순서가 변경되지 않도록 한다.
seed_everything(2024)	정의된 함수를 호출하여 프로젝트 전체에 걸쳐 2024 시드를 적용한다.

3. 데이터 로드 및 구조 확인

: 분석에 필요한 훈련 및 테스트 데이터를 불러오고, 훈련 데이터의 초기 구조와 내용을 확인한다.

```
train = pd.read_csv("./train.csv")
test = pd.read_csv("./test.csv")
train
```

	ID	timestamp	item	corporation	location	supply(kg)	price(원/kg)
0	TG_A_J_20190101	2019-01-01	TG	A	J	0.0	0.0
1	TG_A_J_20190102	2019-01-02	TG	A	J	0.0	0.0
2	TG_A_J_20190103	2019-01-03	TG	A	J	60601.0	1728.0
3	TG_A_J_20190104	2019-01-04	TG	A	J	25000.0	1408.0
4	TG_A_J_20190105	2019-01-05	TG	A	J	32352.0	1250.0
...	...	...	...	...	...	...	...
59392	RD_F_J_20230227	2023-02-27	RD	F	J	452440.0	468.0
59393	RD_F_J_20230228	2023-02-28	RD	F	J	421980.0	531.0
59394	RD_F_J_20230301	2023-03-01	RD	F	J	382980.0	574.0
59395	RD_F_J_20230302	2023-03-02	RD	F	J	477220.0	523.0
59396	RD_F_J_20230303	2023-03-03	RD	F	J	427520.0	529.0

59397 rows × 7 columns

4. 데이터 전처리 및 특성 추출

1) 전처리 시작 및 데이터 합치기

```
def pre_all(train, test):
    print(f"전처리 전 train 크기 : {train.shape}")
    print(f"전처리 전 test 크기 : {test.shape}")
    print("=================전처리 중=================")

    # 합쳐서 전처리하기
    train["timestamp"] = pd.to_datetime(train["timestamp"])
    test["timestamp"] = pd.to_datetime(test["timestamp"])
    df = pd.concat([train,test]).reset_index(drop = True)

    df.rename(columns={'supply(kg)':'supply', 'price(원/kg)':'price'},inplace=True)
```

print(f"전처리 전 train 크기 : {train.shape}")	전처리 시작 전 훈련 데이터(train)의 행과 열 크기를 출력
print(f"전처리 전 test 크기 : {test.shape}")	전처리 시작 전 테스트 데이터(test)의 행과 열 크기를 출력
train["timestamp"]=pd.to_datetime(train["timestamp"])	훈련 데이터의 timestamp열을 Pandas datetime 객체로 변환 (문자열 -> 날짜/시간 형식)
test["timestamp"]=pd.to_datetime(test["timestamp"])	테스트 데이터의 timestamp열을 Pandas datetime 객체로 변환
df = pd.concat([train, test]).reset_index(drop = True)	훈련 데이터와 테스트 데이터를 수직으로 합쳐서(concat) df라는 통합 데이터프레임을 만듭니다. reset_index(drop=True)는 합친 후 인덱스를 초기화한다.
df.rename(columns={'supply(kg)':'supply', 'price(원/kg)':'price'},inplace=True)	특성 이름에 있는 괄호(())와 슬래시(/) 같은 특수 문자를 제거하고 간결하게 변경한다. (inplace=True로 원본에 바로 적용)

2) 기본적인 시간 관련 특성 추출

```
#년/월/일 추가
df['year']=df['timestamp'].dt.year
df['month']=df['timestamp'].dt.month
df['day']=df['timestamp'].dt.day

#요일 추가
df['week_day']=df['timestamp'].dt.weekday
```

3) 복합 시계열 특성 추출 및 인코딩

```
# 년-월 변수 추가 : year-month의 형태, 개월단위 누적값
le = LabelEncoder()
df["year_month"] = df["timestamp"].map(lambda x :str(x.year) + "-"+str(x.month))

# 라벨 인코딩
df["year_month"] = le.fit_transform(df["year_month"])


# 주차 변수 추가
df["week"] = df["timestamp"].map(lambda x: datetime.datetime(x.year, x.month, x.day).isocalendar()[1])
```

le = LabelEncoder()	레이블 인코더(LabelEncoder)객체를 생성
df['year_month'] =df["timestamp"].map(lambda x: str(x.year) + "-" + str(x.month))	'연도-월'형태의 새로운 문자열 특성 year_month를 생성합니다. (2019-1, 2019-2등)
df["year_month"]=le.fit_transform(df["year_month"])	생성된 year_month 문자열 특성을 LabelEncoder를 사용하여 0부터 시작하는 순서형 숫자로 변환합니다.
df["week"]=df["timestamp"].map(lambda x: datetime.datetime(x.year,x.month,x.day).isocalendar()[1])	timestamp를 이용해 해당 날짜의 주차(week)를 추출합니다. isocalendar()의 두 번째 요소가 주차를 나타냅니다.

4) 주차(Week) 데이터 보정 및 공휴일 변수 추가

```
# 주차 누적값
week_list=[]
for i in range(len(df['year'])) :
    if df['year'][i] == 2019 :
        week_list.append(int(df['week'][i]))
    elif df['year'][i] == 2020 :
        week_list.append(int(df['week'][i])+52)
    elif df['year'][i] == 2021 :
        week_list.append(int(df['week'][i])+52+53)
    elif df['year'][i] == 2022 :
        week_list.append(int(df['week'][i])+52+53+53)
    elif df['year'][i] == 2023 :
        week_list.append(int(df['week'][i])+52+53+53+52)
df['week_num']= week_list
```

```
# datetime 패키지에서 19년 12월 마지막주가 첫째주로 들어가는거 발견하여 수정
df.loc[df['timestamp']=='2019-12-30','week_num']=52
df.loc[df['timestamp']=='2019-12-31','week_num']=52


# 공휴일 변수 추가
def make_holi(x):
    kr_holi = holidays.KR()

    if x in kr_holi:
        return 1
    else:
        return 0

df["holiday"] = df["timestamp"].map(lambda x : make_holi(x))
```

week_list=[] ... for i in range(len(df['year'])): ...	연도 경계에서 발생하는 주차 오류를 보정하기 위해 수동으로 보정하는 로직입니다. isocalendar() 함수는 연도 시작 주차 계산 시 오류를 발생시키므로, 각 연도에 따라 추가적인 주차 수(52주 혹은 53주)를 더해 연속적인 주차 번호를 생성하고 week_list에 저장합니다.
df["week_num"] = week_list	보정된 연속적인 주차 번호 리스트를 week_num이라는 새로운 특성으로 데이터프레임에 추가합니다.

df.loc[df['timestamp'] == '2019-12-30', 'week_num'] = 52	datetime 패키지에서 2019년 12월 마지막 주가 다음 해 첫 주로 들어가는 오류를 발견하고, 특정 날짜의 week_num값을 52로 직접 수정합니다.
def make_holi(x): ... df["holiday"] = df["timestamp"].map(lambda x: make_holi(x))	공휴일 여부를 나타내는 이진(0 또는 1) 특성 holiday를 추가합니다. holidays.KR() 라이브러리를 사용하여 해당 날짜가 공휴일인지 확인하고, 공휴일이면 1, 아니면 0을 반환합니다.

5) 데이터 재분리 및 최종 확인

```
# train, test 분리하기
train = df[~df["price"].isnull()].sort_values("timestamp").reset_index(drop = True)
test = df[df["price"].isnull()].sort_values("timestamp").reset_index(drop=True)


print(f"전처리 후 train 크기 : {train.shape}")
print(f"전처리 후 test 크기 : {test.shape}")

return train, test
```

```
train_pre, test_pre = pre_all(train, test)
```

```
train_pre
```

	ID	timestamp	item	corporation	location	supply	price	year	month	day	week_day	year_month	week	week_num	holiday
0	TG_A_J_20190101	2019-01-01	TG	A	J	0.0	0.0	2019	1	1	1	0	1	1	1
1	CB_A_S_20190101	2019-01-01	CB	A	S	0.0	0.0	2019	1	1	1	0	1	1	1
2	RD_D_J_20190101	2019-01-01	RD	D	J	0.0	0.0	2019	1	1	1	0	1	1	1
3	BC_D_J_20190101	2019-01-01	BC	D	J	0.0	0.0	2019	1	1	1	0	1	1	1
4	CB_F_J_20190101	2019-01-01	CB	F	J	0.0	0.0	2019	1	1	1	0	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59392	CR_E_S_20230303	2023-03-03	CR	E	S	0.0	0.0	2023	3	3	4	50	9	219	0
59393	BC_A_S_20230303	2023-03-03	BC	A	S	3776.0	2875.0	2023	3	3	4	50	9	219	0
59394	CB_E_J_20230303	2023-03-03	CB	E	J	0.0	0.0	2023	3	3	4	50	9	219	0
59395	BC_D_J_20230303	2023-03-03	BC	D	J	1776.0	3059.0	2023	3	3	4	50	9	219	0
59396	RD_F_J_20230303	2023-03-03	RD	F	J	427520.0	529.0	2023	3	3	4	50	9	219	0

59397 rows × 15 columns

```
test_pre
```

	ID	timestamp	item	corporation	location	supply	price	year	month	day	week_day	year_month	week	week_num	holiday
0	TG_A_J_20230304	2023-03-04	TG	A	J	NaN	NaN	2023	3	4	5	50	9	219	0
1	TG_E_S_20230304	2023-03-04	TG	E	S	NaN	NaN	2023	3	4	5	50	9	219	0
2	BC_B_J_20230304	2023-03-04	BC	B	J	NaN	NaN	2023	3	4	5	50	9	219	0
3	TG_E_J_20230304	2023-03-04	TG	E	J	NaN	NaN	2023	3	4	5	50	9	219	0
4	BC_B_S_20230304	2023-03-04	BC	B	S	NaN	NaN	2023	3	4	5	50	9	219	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1087	TG_A_J_20230331	2023-03-31	TG	A	J	NaN	NaN	2023	3	31	4	50	13	223	0
1088	RD_D_J_20230331	2023-03-31	RD	D	J	NaN	NaN	2023	3	31	4	50	13	223	0
1089	CR_D_J_20230331	2023-03-31	CR	D	J	NaN	NaN	2023	3	31	4	50	13	223	0
1090	TG_E_J_20230331	2023-03-31	TG	E	J	NaN	NaN	2023	3	31	4	50	13	223	0
1091	RD_F_J_20230331	2023-03-31	RD	F	J	NaN	NaN	2023	3	31	4	50	13	223	0

1092 rows × 15 columns

5. TG 제외 나머지 농산물

1) 전처리

(1) 이상치(극단값) 식별 및 제거

품목별로 비정상적으로 높은 가격을 보이는 데이터 포인트(극단값)를 식별하고, 해당 가격을 해당 품목의 정상 가격 평균값으로 대체하여 처리합니다.

```
# 극 이상치 제거
tg_idx = train_pre[(train_pre["item"]=="TG") & (train_pre["price"]>20000)].index
rd_idx = train_pre[(train_pre["item"]=="RD") & (train_pre["price"]>5000)].index
bc_idx = train_pre[(train_pre["item"]=="BC") & (train_pre["price"]>8000)].index
cb_idx = train_pre[(train_pre["item"]=="CB") & (train_pre["price"]>2300)].index

train_pre.loc[tg_idx,"price"] = train_pre[(train_pre["item"]=="TG") & (train_pre["price"]!=0)]["price"].mean()
train_pre.loc[rd_idx,"price"] = train_pre[(train_pre["item"]=="RD") & (train_pre["price"]!=0)]["price"].mean()
train_pre.loc[bc_idx,"price"] = train_pre[(train_pre["item"]=="BC") & (train_pre["price"]!=0)]["price"].mean()
train_pre.loc[cb_idx,"price"] = train_pre[(train_pre["item"]=="CB") & (train_pre["price"]!=0)]["price"].mean()
```

(2) TG 제외

(a) 특성(데이터) 분포의 이질성 (Heterogeneity)

- 데이터 분석에서 품목별로 가격 결정 요인이나 데이터의 특성이 크게 다를 수 있습니다.

- 감귤(TG)의 특수성:감귤은 주로 제주도(지역 J, S)에서 생산되며, 계절성이 매우 강하고, 다른 채소 품목(무, 당근 등)과는 유통 구조, 수요 패턴, 저장성등이 근본적으로 다릅니다.

- 모델 간섭 최소화:만약 모든 품목을 하나의 모델로 학습시키면, 감귤의 특이한 가격 변동 패턴이 다른 품목(예: 무, 당근)의 가격 예측에 "노이즈"로 작용하여 전반적인 예측 정확도를 떨어뜨릴 수 있습니다. 데이터를 분리함으로써 각 품목 그룹의 고유한 패턴에만 집중할 수 있습니다.

(b) 이상치 및 전처리 전략의 차별화

- 각 품목이 허용하는 가격 범위나 데이터의 이상치 기준이 다릅니다.

이상치 기준의 차이:코드에서 보았듯이, 감귤(TG)은 가격 20,000 초과를 이상치로 처리했지만, 당근(CR)은 5,000 초과, 양배추(CB)는 2,300 초과를 기준으로 삼았습니다.

모든 품목에 동일한 이상치 처리 기준을 적용할 수 없기 때문에, 데이터를 분리하여 품목의 특성에 맞는 맞춤형 전처리를 적용하는 것이 필수적입니다.

```
# 감귤이 아닌것
print(f"train의 컬럼 : {train_pre.columns}")
print(f"test의 컬럼 : {test_pre.columns}")

train_notg = train_pre[train_pre["item"] !="TG"]
test_notg = test_pre[test_pre["item"] != "TG"]
```

(3) 범주형 변수 인코딩

머신러닝 모델이 이해할 수 있도록 문자열로 된 범주형 특성을 숫자 형태로 변환(인코딩)합니다. 여기서는 get_dummies를 사용하여 원-핫 인코딩(One-Hot Encoding)을 수행합니다.

```
#인코딩
Xy = pd.get_dummies(train_notg.sort_values(by = ["timestamp"]).reset_index(drop=True).drop(columns = ["supply"]), columns = ["item","corporatic
answer_notg = pd.get_dummies(test_notg.drop(columns = ["timestamp","supply","price"]), columns = [ "item","corporation","location"])
print(Xy.columns)
```

Xy=pd.get_dummies(train_notg.sort_values(by=["timestamp"])...)	훈련 데이터에 대해 원-핫 인코딩을 수행하고 결과를 Xy에 저장합니다.
sort_values(by=["timestamp"])	인코딩 전에 데이터를 시간(timestamp) 순서로 정렬합니다.
drop_columns=["supply"]	최종 특성 셋에서 supply(공급량)열을 제거합니다.
columns=["item", "corporation", "location"]	item, corporation, location세 개의 범주형 특성에 대해서만 원-핫 인코딩을 수행합니다.
answer_notg=pd.get_dummies(test_notg.drop(columns=["timestamp", "supply", "price"]), columns=["item", "corporation", "location"])	테스트 데이터에 대해서도 동일하게 원-핫 인코딩을 수행합니다. 이때 테스트 데이터는 목표 변수(price)가 없으므로 해당 열과 timestamp, supply열을 제거한 후 인코딩합니다.

```
train의 컬럼 : Index(['ID', 'timestamp', 'item', 'corporation', 'location', 'supply', 'price',
       'year', 'month', 'day', 'week_day', 'year_month', 'week', 'week_num',
       'holiday'],
      dtype='object')
test의 컬럼 : Index(['ID', 'timestamp', 'item', 'corporation', 'location', 'supply', 'price',
       'year', 'month', 'day', 'week_day', 'year_month', 'week', 'week_num',
       'holiday'],
      dtype='object')
Index(['ID', 'timestamp', 'price', 'year', 'month', 'day', 'week_day',
       'year_month', 'week', 'week_num', 'holiday', 'item_BC', 'item_CB',
       'item_CR', 'item_RD', 'corporation_A', 'corporation_B', 'corporation_C',
       'corporation_D', 'corporation_E', 'corporation_F', 'location_J',
       'location_S'],
      dtype='object')
```

2) 모델링 & 훈련예측

(1) 개별 기본 모델(Base Model) 정의

앙상블에 사용할 두 가지 강력한 부스팅 모델인 CatBoost와 XGBoost의 하이퍼파라미터를 설정하고 정의

```
cat = CatBoostRegressor(random_state = 2024,
                        n_estimators = 1000,
                        learning_rate = 0.01,
                        depth = 10,
                        l2_leaf_reg = 3,
                        metric_period = 1000)

xgb = XGBRegressor(n_estimators = 1000, random_state = 2024, learning_rate = 0.01, max_depth = 10)
```

cat = CatBoostRegressor(random_state = 2024, n_estimators = 1000, learning_rate = 0.01, depth = 10, l2_leaf_reg = 3, metric_period = 1000)	CatBoost Regressor 모델을 정의합니다. 1000개의 트리를 사용하고, 낮은 학습률(0.01)로 정밀하게 학습하며, 트리 깊이(10)를 지정하여 모델의 복잡도를 설정합니다. random_state로 재현성을 확보합니다.
xgb = XGBRegressor(n_estimators = 1000, random_state = 2024, learning_rate = 0.01, max_depth = 10)	XGBoost Regressor 모델을 정의합니다. CatBoost와 유사하게 1000개의 트리, 낮은 학습률(0.01), 최대 깊이(10)를 설정합니다.

(2) 보팅 회귀 모델 정의 및 학습

```
# voting
vote_model = VotingRegressor(
    estimators =[("cat",cat), ("xgb", xgb)]
)

vote_model.fit(Xy.drop(columns = ["timestamp", "ID","price"]), Xy["price"])
```

vote_model=VotingRegressor(estimators=[("cat",cat), ("xgb", xgb)])	Voting Regressor객체를 생성합니다. estimators 리스트에는 ("모델 이름", 모델 객체)쌍으로 CatBoost와 XGBoost 모델을 포함시켜 두 모델의 평균 예측값을 최종 결과로 사용하도록 설정합니다.
vote_model.fit(Xy.drop(columns=["timestamp", "ID", "price"]), Xy["price"])	앙상블 모델을 학습시킵니다. 훈련 데이터(Xy)에서 목표 변수(price)와 불필요한 열(timestamp, ID)을 제외한 나머지 모든 특성(X)을 사용하여 모델을 훈련합니다.

(3) 최종 예측 및 결과 처리

학습된 앙상블 모델을 사용하여 테스트 데이터의 가격을 예측하고, 예측 결과에 대한 후처리를 수행한다.

```
pred = vote_model.predict(answer_notg.drop(columns = ["ID"]))
for idx in range(len(pred)):
    if pred[idx]<0:
        pred[idx]= 0
answer_notg["answer"] = pred

answer_notg[["ID","answer"]]
```

pred=vote_model.predict(answer_notg.drop(columns=["ID"]))	학습된 vote_model을 사용하여 테스트 데이터(answer_notg)의 가격을 예측합니다. 테스트 데이터에서 ID열을 제외한 특성을 입력으로 사용합니다. 예측 결과는 pred에 저장됩니다.
for idx in range(len(pred)):	예측값(pred)의 길이만큼 반복문을 실행합니다.
if pred[idx] < 0:	예측값중 0보다 작은 값이 있는지 확인합니다. (가격 예측 문제에서 가격이 음수일 수는 없으므로)
pred[idx] = 0	만약 예측값이 0보다 작다면, 해당 예측값을 0으로 강제 보정합니다.
answer_notg["answer"] = pred	보정된 최종 예측값(pred)을 테스트 데이터(answer_notg)에 answer라는 새로운 열로 추가합니다.
answer_notg[["ID", "answer"]]	최종적으로 제출을 위해 필요한 ID와 예측값(answer)열만 선택하여 출력합니다.

	ID	answer
2	BC_B_J_20230304	2376.672901
4	BC_B_S_20230304	97.678967
6	BC_C_J_20230304	2385.800075
7	BC_A_S_20230304	2763.041513
10	BC_D_J_20230304	2771.454688
...	...	...
1085	RD_D_S_20230331	476.193654
1086	CR_C_J_20230331	1696.957199
1088	RD_D_J_20230331	387.570045
1089	CR_D_J_20230331	1818.944640
1091	RD_F_J_20230331	492.058492

812 rows × 2 columns

6. TG(1)

1) 전처리

(1) 초기 함수 호출 및 공휴일 특성 수정

```
train_pre, test_pre = pre_all(train, test)

# 공휴일이지만 안쉬는 날 제외하기
no_holi = list(train_pre[(train_pre["item"] =="TG") &(train_pre["holiday"]==1) & (train_pre["price"]!=0)].groupby("timestamp").count().reset_index()[
noholi_idx = train_pre[train_pre["timestamp"].isin(no_holi)]["holiday"].index
for idx in noholi_idx:
    train_pre.loc[idx, "holiday"] = 0
```

train_pre, test_pre = pre_all(train, test)	이전에 정의된 전처리 함수 pre_all을 호출하여 훈련(train) 및 테스트(test) 데이터를 전처리하고, 그 결과를 train_pre와 test_pre에 저장합니다.
no_holi= list(train_pre[(train_pre["item"]=="TG") &(train_pre["holiday"]==1)&(train_pre["price"]!=0)].groupby("timestamp").count().reset_index()["timestamp"])	"공휴일이지만 안 쉬는 날 제외하기"위한 복잡한 조건 필터링입니다. 품목이 'TG'이고, holiday가 1(공휴일)이며, 가격이 0이 아닌 경우의 데이터를 선택합니다. 이를 timestamp별로 그룹화하여 날짜 목록을 추출하고 no_holi에 저장합니다. (즉, 가격이 0이 아니었으므로 실제로 쉬지 않은 것으로 간주될 수 있는 공휴일 날짜 목록입니다.)
noholi_idx=train_pre[(train_pre["timestamp"].isin(no_holi))]['holiday'].index	위에서 찾은 쉬지 않은 공휴일 날짜(no_holi)에 해당하는 훈련 데이터의 holiday특성 인덱스를 noholi_idx에 저장합니다.
for idx in noholi_idx:	noholi_idx에 저장된 모든 인덱스에 대해 반복문을 실행합니다.
train_pre.loc[idx, "holiday"] = 0	해당 인덱스의 holiday특성 값을 0으로 수정합니다. (실제로는 가격 거래가 있었으므로, 공휴일 특성을 0으로 처리하여 모델이 해당 날짜를 공휴일로 인식하지 않게 합니다.)

(2) 품목 TG 데이터 분리 및 정렬

```
# train 및 test 시간 순서로 정렬하기
train_tg = train_pre[train_pre["item"] == "TG"].sort_values(by = ["timestamp"]).reset_index(drop= True)
test_tg = test_pre[test_pre["item"] == "TG"].sort_values(by = ["timestamp"]).reset_index(drop= True)
```

(3) TG 데이터 인코딩 및 최종 준비

분리된 'TG' 데이터에 대해 원-핫 인코딩을 적용하고, 목표 변수를 로그 변환합니다.

```
Xy = pd.get_dummies(train_tg, columns = [ "item","corporation","location"]).drop(columns = ["supply"])
answer_tg1 = pd.get_dummies(test_tg, columns = [ "item","corporation","location"]).drop(columns = ["timestamp","supply","price"])
print(f"train의 컬럼 : {Xy.columns}")
print(f"test의 컬럼 : {answer_tg1.columns}")
Xy["price"] = np.sqrt(Xy["price"])
```

Xy = pd.get_dummies(train_tg, columns = ["item","corporation", "location"]).drop(columns = ["supply"])	훈련 데이터(train_tg)에 대해 item,corporation, location범주형 특성을 원-핫 인코딩합니다. 그리고 supply열을 특성에서 제외합니다. 결과는 Xy에 저장됩니다.
answer_tg1= pd.get_dummies(test_tg, columns =["item","corporation","location"]).drop(columns = ["timestamp", "supply", "price"])	테스트 데이터(test_tg)에 대해 동일하게 원-핫 인코딩을 수행합니다. 테스트 데이터이므로 timestamp, supply, price열은 제외합니다.
print(f"train의 컬럼 : {Xy.columns}")	훈련 데이터(Xy)의 최종 특성(컬럼) 목록을 출력합니다.
print(f"test의 컬럼 : {answer_tg1.columns}")	테스트 데이터(answer_tg1)의 최종 특성(컬럼) 목록을 출력합니다.
Xy["price"] = np.sqrt(Xy["price"])	훈련 데이터의 목표 변수(가격, price)에 제곱근 변환(qrt)을 적용합니다. 이는 예측 대상의 분포를 정규화하여 모델의 성능을 향상시키기 위한 일반적인 기법입니다.

```
전처리 전 train 크기 : (59397, 7)
전처리 전 test 크기 : (1092, 5)
================전처리 중================
전처리 후 train 크기 : (59397, 15)
전처리 후 test 크기 : (1092, 15)
train의 컬럼 : Index(['ID', 'timestamp', 'price', 'year', 'month', 'day', 'week_day',
       'year_month', 'week', 'week_num', 'holiday', 'item_TG', 'corporation_A',
       'corporation_B', 'corporation_C', 'corporation_D', 'corporation_E',
       'location_J', 'location_S'],
      dtype='object')
test의 컬럼 : Index(['ID', 'year', 'month', 'day', 'week_day', 'year_month', 'week',
       'week_num', 'holiday', 'item_TG', 'corporation_A', 'corporation_B',
       'corporation_C', 'corporation_D', 'corporation_E', 'location_J',
       'location_S'],
      dtype='object')
```

2) 모델링& 훈련예측

```
# # 모델 정의
cat = CatBoostRegressor(random_state = 2024,
                        n_estimators = 1000,
                        learning_rate = 0.01,
                        depth = 10,
                        l2_leaf_reg = 3,
                        metric_period = 1000)

xgb = XGBRegressor(n_estimators = 1000, random_state = 2024, learning_rate = 0.01, max_depth = 10)


# voting
vote_model = VotingRegressor(
    estimators =[("cat",cat), ("xgb", xgb)]
)

vote_model.fit(Xy.drop(columns = ["timestamp", "ID","price"]), Xy["price"])

pred = vote_model.predict(answer_tg1.drop(columns = ["ID"]))
for idx in range(len(pred)):
    if pred[idx]<0:
        pred[idx]= 0
answer_tg1["answer"] = np.power(pred,2)

answer_tg1[["ID","answer"]]
```

vote_model=VotingRegressor(estimators=[("cat",cat), ("xgb", xgb)])	CatBoost와 XGBoost모델 객체를 포함시켜 Voting Regressor객체(vote_model)를 생성합니다. 이 앙상블은 두 모델의 예측을 결합하여 최종 예측값을 도출합니다.
vote_model.fit(Xy.drop(columns=["timestamp", "ID", "price"]), Xy["price"])	앙상블 모델을 훈련시킵니다. 훈련 데이터(Xy)에서 목표 변수(price)와 불필요한 열(timestamp, ID)을 제외한 나머지 특성(X)을 입력으로 사용하여 모델을 학습시킵니다.
pred=vote_model.predict(answer_tg1.drop(columns=["ID"]))	학습된 vote_model을 사용하여 테스트 데이터(answer_tg1)의 가격을 예측합니다. 입력 특성에서 ID열을 제외합니다
for idx in range(len(pred)):	예측값(pred) 전체에 대해 반복문을 실행합니다.
answer_tg1["answer"] = np.power(pred, 2)	보정된 예측값(pred)을 제곱하여 answer열을 생성합니다. 이는 전처리 단계에서 목표 변수에 적용했던 제곱근 변환 sqrt(x)을 원래의 가격 스케일로 되돌리는 역변환입니다.
answer_tg1[["ID", "answer"]]	최종적으로 제출 형식에 맞는 ID와 역변환된

	예측값(answer)열만 선택하여 출력합니다.

	ID	answer
0	TG_A_J_20230304	2751.208449
1	TG_E_S_20230304	3381.528438
2	TG_E_J_20230304	529.638514
3	TG_D_S_20230304	3764.708357
4	TG_D_J_20230304	407.141038
...	...	...
275	TG_D_J_20230331	1870.228988
276	TG_D_S_20230331	5046.532954
277	TG_A_S_20230331	5182.915243
278	TG_E_S_20230331	4690.096292
279	TG_E_J_20230331	1526.626858

280 rows × 2 columns

7. TG (2) : 일반화를 위한 추가 모델링

1) 전처리

```
train_tg2 = train_pre[train_pre["item"] =="TG"]
test_tg2 = test_pre[test_pre["item"] == "TG"]

Xy2 = pd.get_dummies(train_tg2.sort_values(by = ["timestamp", "corporation","location"]).reset_index(drop=True).drop(columns = ["item","supply"
answer_tg2 = pd.get_dummies(test_tg2.drop(columns = ["timestamp","supply","price","item"]), columns = [ "corporation","location"])
print(Xy2.columns)

# 종속변수 루트값
Xy2["price"] = np.sqrt(Xy2["price"])
```

```
Index(['ID', 'timestamp', 'price', 'year', 'month', 'day', 'week_day',
       'year_month', 'week', 'week_num', 'holiday', 'corporation_A',
       'corporation_B', 'corporation_C', 'corporation_D', 'corporation_E',
       'location_J', 'location_S'],
      dtype='object')
```

2) 모델링 & 훈련 예측

```
n_estimators =1000
lrs = 0.05
max_depths = 10
l2_leaf_reg = 3

cat = CatBoostRegressor(random_state = 2024,
                        n_estimators = n_estimators,
                        learning_rate = lrs,
                        depth = max_depths,
                        l2_leaf_reg = l2_leaf_reg,
                        metric_period = 1000)

cat.fit(Xy2.drop(columns = ["timestamp", "ID","price"]), Xy2["price"])

pred2 = cat.predict(answer_tg2.drop(columns = ["ID"]))
for idx in range(len(pred2)):
    if pred2[idx]<0:
        pred2[idx]= 0
answer_tg2["answer"] = np.power(pred2,2)

answer_tg2[["ID","answer"]]
```

	ID	answer
0	TG_A_J_20230304	2742.172780
1	TG_E_S_20230304	3354.003496
3	TG_E_J_20230304	760.581106
5	TG_D_S_20230304	3769.615969
8	TG_D_J_20230304	53.451592
...	...	...
1074	TG_D_S_20230331	4727.895634
1077	TG_A_S_20230331	5403.967583
1079	TG_E_S_20230331	4715.201735
1087	TG_A_J_20230331	6813.935154
1090	TG_E_J_20230331	2810.642810

280 rows × 2 columns

8. TG 앙상블

품목 'TG' 데이터와 'TG 외' 데이터에 대한 개별 예측 결과를 하나의 최종 예측 파일로 통합(앙상블하는 과정입니다. 두 개의 독립적인 최종 예측 세트(total1, total2)를 ID 기준으로 합친 다음, 예측값들을 단순 평균하여 최종적으로 가장 안정적이고 일반화된 모델의 예측 결과를 도출하는 앙상블 기법을 구현하고 있습니다.

1) 개별 예측 결과 통합

```
total1 = pd.concat([answer_tg1[["ID","answer"]],answer_notg[["ID","answer"]]])
total2 = pd.concat([answer_tg2[["ID","answer"]],answer_notg[["ID","answer"]]])
```

- 품목 'TG'에 대한 예측 결과(answer_tg1)와 'TG 외' 품목에 대한 예측 결과(answer_notg)에서 ID와 answer열만을 선택하여 수직으로 합쳐(concat)total1에 저장합니다.

- 품목 'TG 외'의 다른 모델링 결과(answer_tg2)와 'TG 외' 품목에 대한 예측 결과(answer_notg)에서 ID와 answer열만을 선택하여 수직으로 합쳐 total2에 저장합니다.

2) 평균 앙상블

```
# TG 앙상블 (평균)

df = pd.merge(total1, total2, how = "inner", on="ID")
df["answer"] = (df["answer_x"]+df["answer_y"])/2
df["item"] = df["ID"].map(lambda x :x.split("_")[0])
```

df = pd.merge(total1, total2, how = "inner", on = "ID")	total1과 total2두 데이터프레임을 ID열을 기준으로 내부 조인하여 병합합니다. 결과 데이터프레임 df는 ID별로 두 예측값(answer_x와 answer_y)을 포함하게 됩니다.
df["answer"]= (df["answer_x"] + df["answer_y"]) / 2	total1의 예측값(answer_x)과 total2의 예측값(answer_y)을 더한 후 2로 나누어평균을 계산합니다. 이 평균값이 최종 앙상블 예측값으로 df["answer"]에 저장됩니다.
df["item"]= df["ID"].map(lambda x: x.split('_')[0])	ID열의 문자열을 언더바(_)를 기준으로 분리하고, 그중 첫 번째 요소를 추출하여 item열을 생성합니다.

9. 후처리

최종 앙상블 예측값에 대한 후처리(Post-processing)를 수행하고, 불필요한 열을 제거하여 최종 제출 파일을 정리한다. 후처리는 예측값이 비현실적인 최소값 이하로 나왔을 때 해당 값을 0 또는 특정 값으로 강제 보정하는 작업입니다.

1) 최소값 기반 후처리 (값 보정)

품목별로 예측값(answer)이 특정 최소 기준보다 낮을 경우, 해당 값을 0으로 강제 보정하는 작업

```
## 전체 min값 | 3월의 min값 확인
df.loc[(df['item']=='TG')&(df['answer']<400),'answer'] =0 # 551   #3월 675
df.loc[(df['item']=='CB')&(df['answer']<50),'answer'] =0 # 162  # 3월 200
df.loc[(df['item']=='RD')&(df['answer']<10),'answer'] =0 # 50     # 3월 124
df.loc[(df['item']=='CR')&(df['answer']<150),'answer'] =0 # 250   # 3월 450
df.loc[(df['item']=='BC')&(df['answer']<100),'answer'] =0 #205 3월 205.0
```

2) 불필요한 열 제거

answer_x와 answer_y(앙상블 시 사용된 두 모델의 예측값), 그리고 임시로 생성했던 item열을 최종 데이터프레임 df에서 제거합니다.

```
df = df.drop(columns = ["answer_x","answer_y", "item"])
df
```

	ID	answer
0	TG_A_J_20230304	2746.690615
1	TG_E_S_20230304	3367.765967
2	TG_E_J_20230304	645.109810
3	TG_D_S_20230304	3767.162163
4	TG_D_J_20230304	0.000000
...	...	...
1087	RD_D_S_20230331	476.193654
1088	CR_C_J_20230331	1696.957199
1089	RD_D_J_20230331	387.570045
1090	CR_D_J_20230331	1818.944640
1091	RD_F_J_20230331	492.058492

1092 rows × 2 columns

[차별점 및 배울점]

1. 데이터 이질성에 대한 전략적 대응

감귤의 가격 변동 요인이 다른 채소들과 근본적으로 다르다는 점을 인식하고 감귤 데이터만 따로 분리하여 해당 품목의 고유 패턴에 최적화된 모델링을 수행했다. 이를 통해 모든 품목에 동일한 기준을 적용하지 않고, 품목별 가격 범위를 고려한 맞춤형 이상치(극단값) 처리 기준을 설정하여 데이터의 품질을 극대화했다.

2. 데이터 배경 이해의 중요성

단순히 캘린더상의 공휴일 정보(holiday=1)를 믿는 대신, 실제 유통 데이터를 통해 공휴일 데이터를 수정하였다. 이를 통해 분석의 오류를 줄이고 최종 예측 성능을 안정적으로 높일 수 있었다.