

1조

농산물 가격 예측으로 생산자를 위한 재배 스케줄링 제시

2023 01 13



목차

1. 프로젝트 개요

α. 구성원 및 역할

b. 프로젝트 기획 배경 및 목표

2. 프로젝트 수행 절차 및 방법

3. 결과

α. 인사이트 도출

b. 향후 개선 사항

4. 개발 후기 및 느낀 점

프로젝트 개요

THE TEAM

공통 목표 : 소외 되지 않고, 모두가 모든 기능을 구현하는데 일조한다.

김현아

박정현

방승현

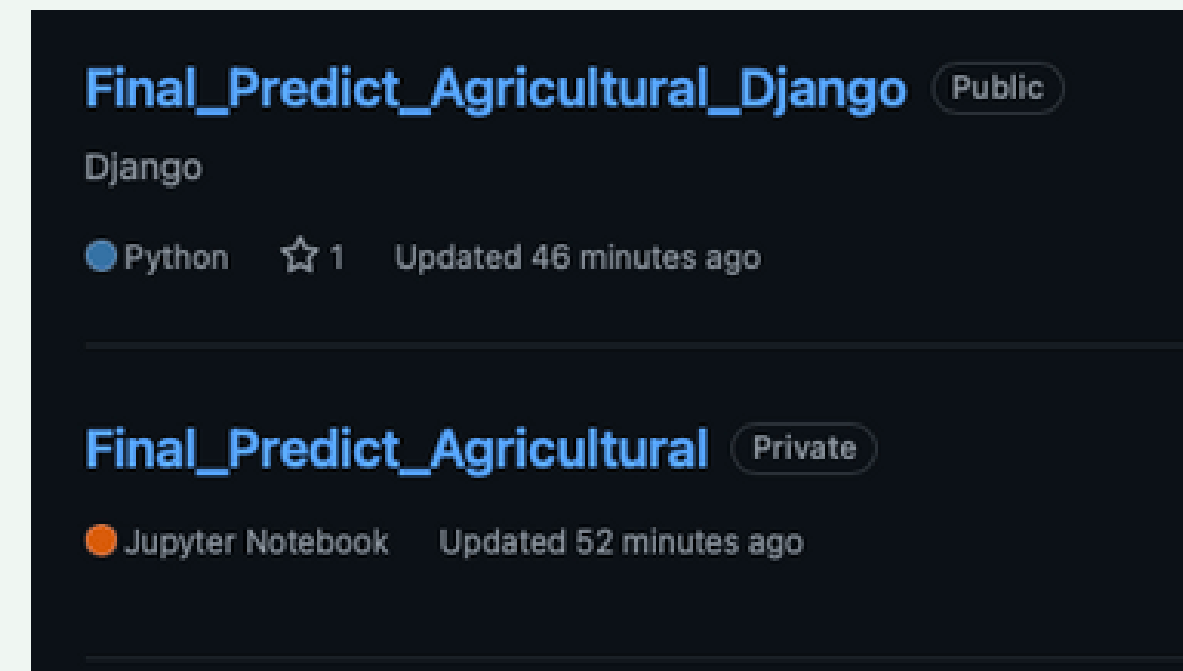
 안진혁

Team project with **GitHub**

Main branch, team members' branch

Model training : Final_Predict_Agricultural

Django : Final_Predict_Agricultural_Django



프로젝트 기획 배경 및 목표

문제 제기

수입엔 '민감', 가격 폭락 시 '깜깜'...물가정책 그만

신정훈 의원, '농산물 최저가격 보장제법' 발의

[축산신문 김수형 기자] 수급에 문제가 발생하면 해외 농축산물을 즉각 수입해 가격을 인위적으로 낮추는 한편 반대로 폭락시 소극적인 시장 대응으로 피해가 고스란히 농가들에게 이어졌던 정부의 가격 정책에 변화가 생길지 관심이 모아지고 있다.

출처 : <http://www.chuksannews.co.kr/news/article.html?no=252806>

농산품의 가격을 조정하는 과정

+

생산 과정에서 발생하는 비용

→ 전부 생산자가 감당

프로젝트 기획 배경 및 목표

문제 제기

"10배의 가격 차, 농수산물 유통 단계 관
찰은지 점검해야"

[이영광의 '온에어' 210] KBS 1TV <시사기획 창> 엄진아 기자

- 실제 소비자에게 판대되는 가격을 알고 놀라는 농민의 목소리로 방송을 시작하셨는데요. 이유가 있을까요.

"제가 여러 가지 사례들을 다 취재했는데 계속 고물가가 최근 이어지고 있죠. 그 모순을 가장 극단적으로 보여줄 수 있는 사례라고 생각했거든요. 생산자와 소비자 사이에서 10배 가까운 가격 차이가 나는 게 일반적인 사례라고 생각하지 않았는데 믿기 어려울 만큼의 가격 차이를 그날 취재하면서 확인하게 됐어요. 중간에 얼마나 가격 거품이 많이 컸는지 단적으로 보여줄 수 있는 사례라고 생각을 한 겁니다."

- 서울 가락동 농수산물 시장 경매가가 농산물 가격의 첫 단추라는데 왜 그런가요?

"모든 제품에는 원가라는 개념이 있죠. 예를 들어서 자전거보다 자동차가 비싼 건 자동차의 원재료도 많이 들어가고 인건비도 많이 들기 때문이죠. 그러나 예외가 되는 분야가 있는데 이게 바로 농산물이라는 생각이 들더라고요. 그러니까 농민이 재배해서 출하할 때까지는 농산물의 가격이 얼마짜리인지 정해지지 않아요. 이게 공릉 도매시장에서 경매하고 도매인이 라고 불리는 상인이 낙찰받을 때 비로소 가격이 붙여지는 것이거든요. 가격이 첫 번째로 붙는다는 의미에서 첫 단추라고 표현을 한 거죠."

생산자와 소비자 사이 가격의 큰 차이
+

도매 시장에서 낙찰됐을 때
정해지는 가격



→ 생산자는 정확한 가격을
판단하기 어려움 (감에 의존)

프로젝트 기획 배경 및 목표

문제 제기

가락시장 도매법인 "담합 맞다" ... 대법원 최종 판결

김한결 기자 | 승인 2022.01.16 18:00 | 댓글 0

2018년부터 이어져 온가락시장 내 도매법인들의 '위탁수수료 담합' 논란에 대해 대법원이 '담합이 맞다'고 최종판결을 내렸다. 지난해 12월 30일 대법원이 '(위탁수수료 담합은) 도매법인 간 가격 경쟁을 직접적으로 감소시킨 것이 분명하다'며 공정거래위원회(위원장 조성욱, 공정위)의 손을 들었다. 공정위의 제재에 불복해 소송을 이어온 도매법인의 담합 혐의가 마침내 모두 인정된 것이다.

출처 : <http://www.ikpnews.net/news/articleView.html?idxno=46402>

도매상의 담합 + 가격 경쟁 의도적 회피
→ 생산물에 대한 불합리한 가격 책정

프로젝트 기획 배경 및 목표

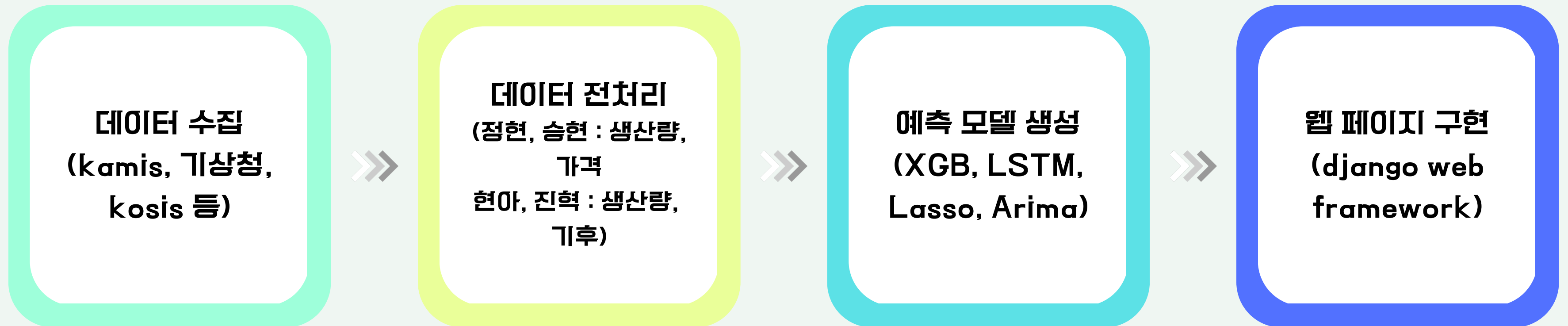
기대 효과 및 목표

1. 생산자의 농지(지역에 따른 기후 등)에서 농산물의 재배 적합 여부와 생산량 제공
  >> 새로운 작물 재배 가능성 + 생산 리스크 감소에 기여
2. 배추 재배가 가능한 경우, 배추 가격을 예측해서 알려줌
  >> 배추의 적정 가격을 제공해, 생산자에게 합리적인 가격 지표 제시
3. 생산자가 얻을 수 있는 최대 수익을 예측 생산량과 예측 수익을 통해 알려준다
  >> 미래 수익 예상 가능(미래의 불확실성에 대비)

프로젝트 수행절차

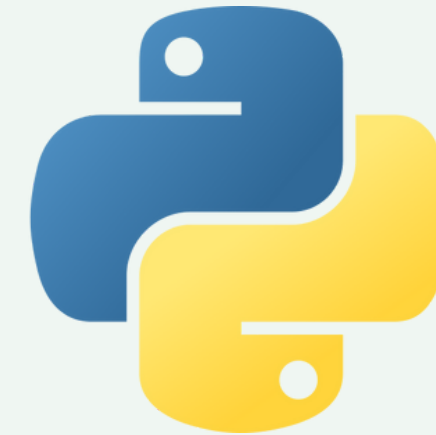
및 방법

프로젝트 수행 절차



기술 분류

사용 언어	Python, HTML, CSS, JavaScript
개발 툴	GitHub, Jupyter Notebook, Visual Studio Code
프레임 워크	Django



자료 수집

생산량 데이터	KOSIS	<p>https://kosis.kr/statHtml/statHtml.do?orgId=101&tblId=DT_1ET_0028&conn_path=I3</p> <ul style="list-style-type: none"> 기간: 2000년~2021년 지역별/품목별로 구별하여 csv 저장
기후 데이터	농업기상관측(AAOS)	<p>https://data.kma.go.kr/data/grnd/selectAgrRltmList.do?pgmNo=72</p> <ul style="list-style-type: none"> 기간 : 2000년 1월 1일부터 2022년 12월 30일로 조회 했으나 실제 데이터는 12월 21일까지 있음 요소 : 기온, 풍속, 습도, 증발량, 지면·초상·지중온도, 토양수분, 복사, 조도, 일사, 일조, 지하수위 등 지역별/관측 장소 별로 나뉘어 있음
가격 데이터	KAMIS	<p>https://www.kamis.or.kr/customer/price/wholesale/period.do</p> <ul style="list-style-type: none"> 기간 : 2000년~2022년 상품/중품별로 구분하여 csv 저장

데이터 수집_크롤링

```
1 url = "https://oasis.krei.re.kr/basicInfo/wholes
2
3 browser.get(url)
4 time.sleep(1)
5
6 for i in range(2000, 2022):
7
8     # <input id="sCalendar" name="sCalendar" title="
9     browser.execute_script("document.getElementB
10     time.sleep(1)
11     # <input id="eCalendar" name="eCalendar" title="
12     browser.execute_script("document.getElementB
13     time.sleep(1)
14
15     # 대분류
16     # /html/body/div[1]/div[2]/form/div[1]/div/d
17     browser.find_element_by_xpath("/html/body/di
18     time.sleep(1)
19
20     # 중분류
```

fig 1_1. 생산량 데이터 크롤링
〈load_quantity.ipynb〉

```
1 def get_url(start_date, end_date, rank):
2     url = "https://www.kamis.or.kr/customer/price/wholesale/pe
3     return url
4
5
6 rank=["1", "2"]
7
8 for i in range(2000, 2022):
9     start_date= str(i)+"-01-01"
10    end_date= str(i)+"-12-31"
11    for k in range(len(rank)):
12        url = get_url(start_date, end_date, rank[k])
13
14        # driver를 이용하여 url을 불러온다.
15        browser.get(url)
16        time.sleep(1)
17        # <a class="common_btn1" onclick="goExcel('1')"><img s
18        browser.find_element_by_css_selector("a.common_btn1").
19        time.sleep(1)
20
21
22 # driver를 종료한다.
23 browser.quit()
```

fig 1_2. 가격 데이터 크롤링
〈load_price.ipynb〉

데이터 전처리_생산량 데이터

시 도 별	2000	2000.1	2000.2	2000.3	2000.4	2000.5	2000.6	2000.7	2000.8	...	2021.15	2021.16	2021.17	2021.18	2021.19	2021.20	2021.21	2021.22
0 시 도 별	업 채 류: 면 적 (ha)	생 산 량 (톤)	배 추: 면 적 (ha)	생 산 량 (톤)	노 지 배 추: 면 적 (ha)	10a당 생 산 량 (kg)	생 산 량 (톤)	노 지 용 배 추: 면 적 (ha)	10a당 생 산 량 (kg)	...	생 산 량 (톤)	노 지 가 용 배 추: 면 적 (ha)	10a당 생 산 량 (kg)	생 산 량 (톤)	노 지 겨 용 배 추: 면 적 (ha)	10a당 생 산 량 (kg)	생 산 량 (톤)	시 설 배 추: 면 적 (ha)
1 계	72925	3744547	51801	3149255	45527	6301	2868690	29114	4301	...	245118	13345	8598	1147465	3274	6920	226584	2443
2 서울특별시	723	15271	208	7906	95	5258	4995	72	3694	...	0	6	12793	771	0	0	0	1
3 부산광역시	1392	77401	921	65562	727	7293	53022	403	4446	...	0	36	11239	4017	0	0	0	1
4 대구광역시	1167	50036	568	38540	316	7376	23307	177	4799	...	0	68	7248	4895	0	0	0	252

5 rows × 509 columns

fig 2. 데이터 전처리 전
17개의 지역이 연생산량 데이터로 들어가 있어 계절별로 분류 작업 필요

데이터 전처리_생산량 데이터

```
1 # 위에코드 함수로 만들기 globals()
2 def make_df(city, season, s_list):
3     globals()[city+"_"+season]= pd.DataFrame()
4     for i in range(len(s_list)):
5         try:
6             globals()[city+"_"+season][s_list[i]]= globals()[city][s_list[i]]
7         except:
8             pass
9
10 # city_list 배열과 season 배열을 이용하여 함수 실행
11 for i in range(len(city_list)):
12     for j in range(len(season)):
13         make_df(city_list[i], season[j], season_list[j])
14
15 # seoul_spring 데이터프레임 확인
16 seoul_spring
```



output_df

- busan_fall_df.csv
- busan_spring_df.csv
- busan_summer_df.csv
- busan_winter_df.csv
- chungbuk_fall_df.csv
- chungbuk_spring_df.csv
- chungbuk_summer_df.csv
- chungbuk_winter_df.csv
- chungnam_fall_df.csv
- chungnam_spring_df.csv
- chungnam_summer_df.csv
- chungnam_winter_df.csv
- daegu_fall_df.csv
- daegu_spring_df.csv
- daegu_summer_df.csv
- daegu_winter_df.csv
- daejeon_fall_df.csv
- daejeon_spring_df.csv
- daejeon_summer_df.csv
- daejeon_winter_df.csv
- gangwon_fall_df.csv

fig 3. 배추 생산량 데이터를 지역별, 계절별로 구분, 전처리
〈baechoo_city_season.ipynb〉

데이터 전처리_기후 데이터

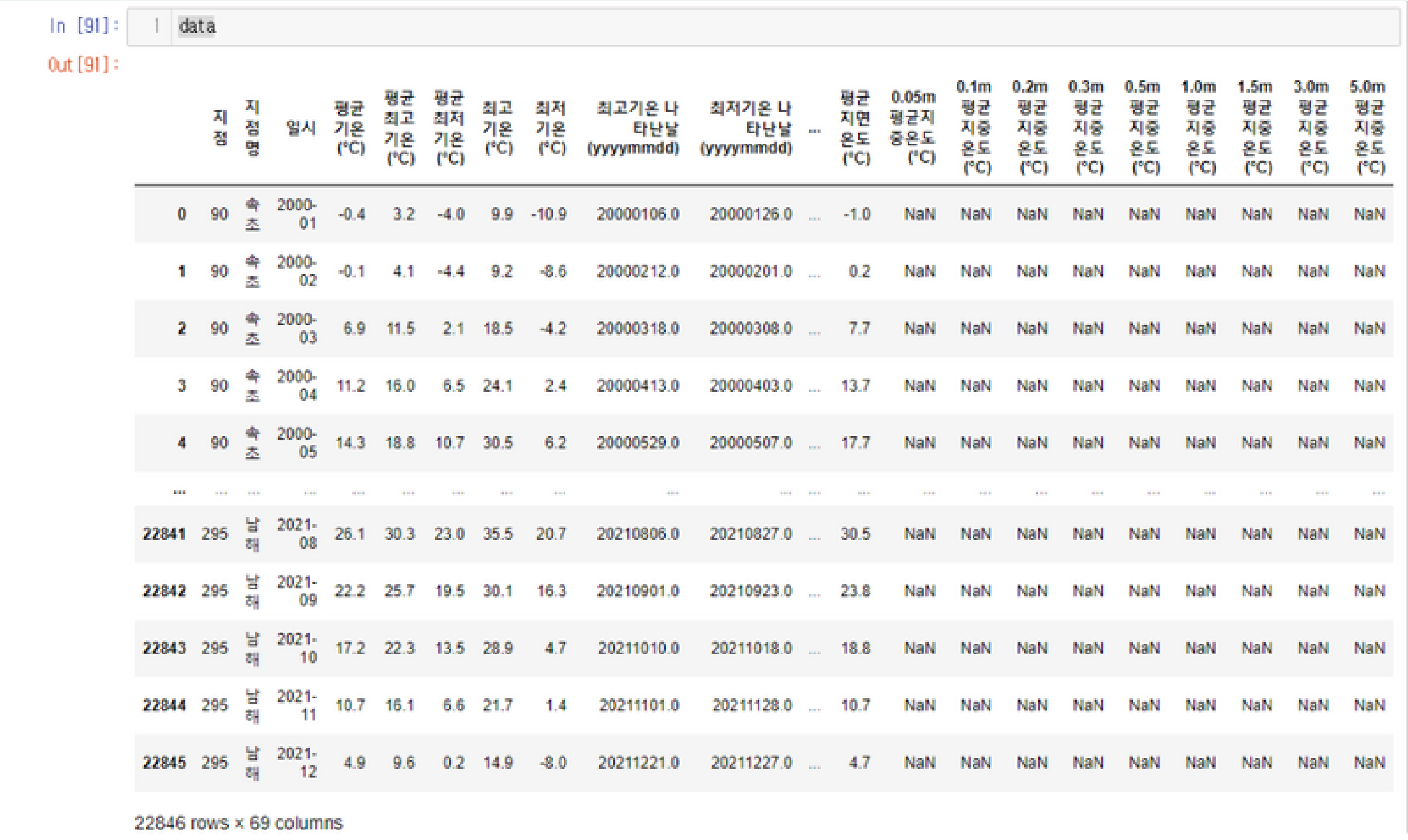


fig 4. 지점이 관측지점으로 나뉘어 있어 17개의 지역으로 라벨링 작업 필요

데이터 전처리_기후 데이터

```
1 empty_array = np.empty((int(len(real_test)/12),60))    # (4488/12, 60)의 구조를 가지는 빈 행렬을 생성
2 empty_array[:] = np.nan
3 empty_frame = pd.DataFrame(empty_array)
4
5 cname = []
6 for i in range(1,13) :
7     L = ['평균기온(°C)', '최고기온(°C)', '최저기온(°C)', '월합강수량(00~24h만)(mm)', '합계 일사량(MJ/m2)']
8     for elem in L :
9         word = elem + "_{}월".format(i)    # 각 컬럼명에 1월부터 12월까지 붙여줌
10        cname.append(word)
11
12 empty_frame.columns = cname    # 변수에 월수가 붙은 리스트를 컬럼명으로 대체
```

fig 5. 17개년의 기후 데이터를 하나의 인덱스에 정렬
〈preprocessing_weather_furoot.ipynb〉

데이터 전처리_가격 데이터

```
1 import jpye
2 import asposecells
3 jpye.startJVM()
4 from asposecells.api import Workbook
5
6 for i in range(2000, 2023) :
7
8     # Load XLS file
9     workbook = Workbook("data/row_price_xls/"+str(i)+"_상품"+"xls")
10
11     # Save as XLSX
12     workbook.save("data/row_price_xlsx/"+str(i)+"_상품"+"xlsx")
13
14 # Shutdown JVM
15 jpye.shutdownJVM()
```

fig 6. xlsx 형식으로 일괄 변경
〈price_change_extender.ipynb〉

```
1 for i in range(2000, 2023) :
2     globals()["df_price_"+str(i)] = pd.read_excel(path+"가격정보_"+str(i)+"_상품.xlsx")
3     globals()["df_price_"+str(i)].drop(columns=globals()["df_price_"+str(i)].columns[1],inplace=True)
4     globals()["df_price_"+str(i)].drop(index=globals()["df_price_"+str(i)].index[3:],inplace=True)
5     globals()["df_price_"+str(i)] = globals()["df_price_"+str(i)].T
6     globals()["df_price_"+str(i)].rename(columns= globals()["df_price_"+str(i)].iloc[0],inplace=True)
7     globals()["df_price_"+str(i)].drop(index=globals()["df_price_"+str(i)].index[0],inplace=True)
8     globals()["df_price_"+str(i)][["구분"]] = globals()["df_price_"+str(i)][["구분"]].apply(lambda x : x)
9     globals()["df_price_"+str(i)][["구분"]] = globals()["df_price_"+str(i)][["구분"]].apply(lambda x : x)
10    globals()["df_price_"+str(i)][["구분"]] = globals()["df_price_"+str(i)][["구분"]].apply(lambda x : x)
11    globals()["df_price_"+str(i)].set_index("구분", inplace=True)
```

```
1 price_baechoo = pd.concat(
2     [globals()[i] for i in year_list],
3     axis=0
4 )
5
6 price_baechoo_high = price_baechoo.drop(columns="평년")
```

fig 7. 각 연도별 상품/종품 가격 df로 저장
〈merge_price.ipynb〉

데이터 전처리_가격 데이터

```
1  # price_baechoo 의 컬럼명 변경
2  price_baechoo.columns = ["중품 평균", "상품 평균"]
3
4  # "중품 평균"과 "상품 평균"의 평균을 구해서 "평균" 컬럼에 저장
5  price_baechoo["평균"] = (price_baechoo["중품 평균"] + price_baechoo["상품 평균"]) / 2
6
7  # "중품 평균"과 "상품 평균" 컬럼 삭제
8  price_baechoo.drop(columns=["중품 평균", "상품 평균"], inplace=True)
9
10 price_baechoo.to_csv("../data/price/price_baechoo.csv")
```

fig 8. 각 일자별 평균가격 csv로 저장

〈merge_price.ipynb〉

학습 모델 생성

- 배추 생산 가능 여부 판단 모델

- 목표 : 사용자가 농지의 위치 정보와 재배하려는 달을 선택하면, 배추 재배가 적합한지 판단해서 알려준다.
- 사용 모델 : XGBoost, RandomForest, Adaboost, GradientBoosting
- 최종 모델 : XGBoost
- 사용 데이터 : 기후 데이터, 생산량 데이터

```
3
4 for train_index, test_index in skfold2.split(X, y) :
5     print("Train :", train_index, "Test :", test_index)
6     X_train, X_test = X[train_index], X[test_index]
7     y_train, y_test = y.values.reshape(-1,1)[train_index], y.values.reshape(-1,1)[test_index]
8
9 xgb2 = XGBClassifier(learning_rate=0.05, gamma=0, max_depth=4, min_child_weight=2, n_estimators=200)
10 xgb2.fit(X_train,y_train)
11
12 y_pred = xgb2.predict(X_test)
13 print("Accuracy:", accuracy_score(y_test, y_pred))
```

Accuracy: 0.91

fig 9. XGB 모델

<predict_supply_or_not.ipynb>

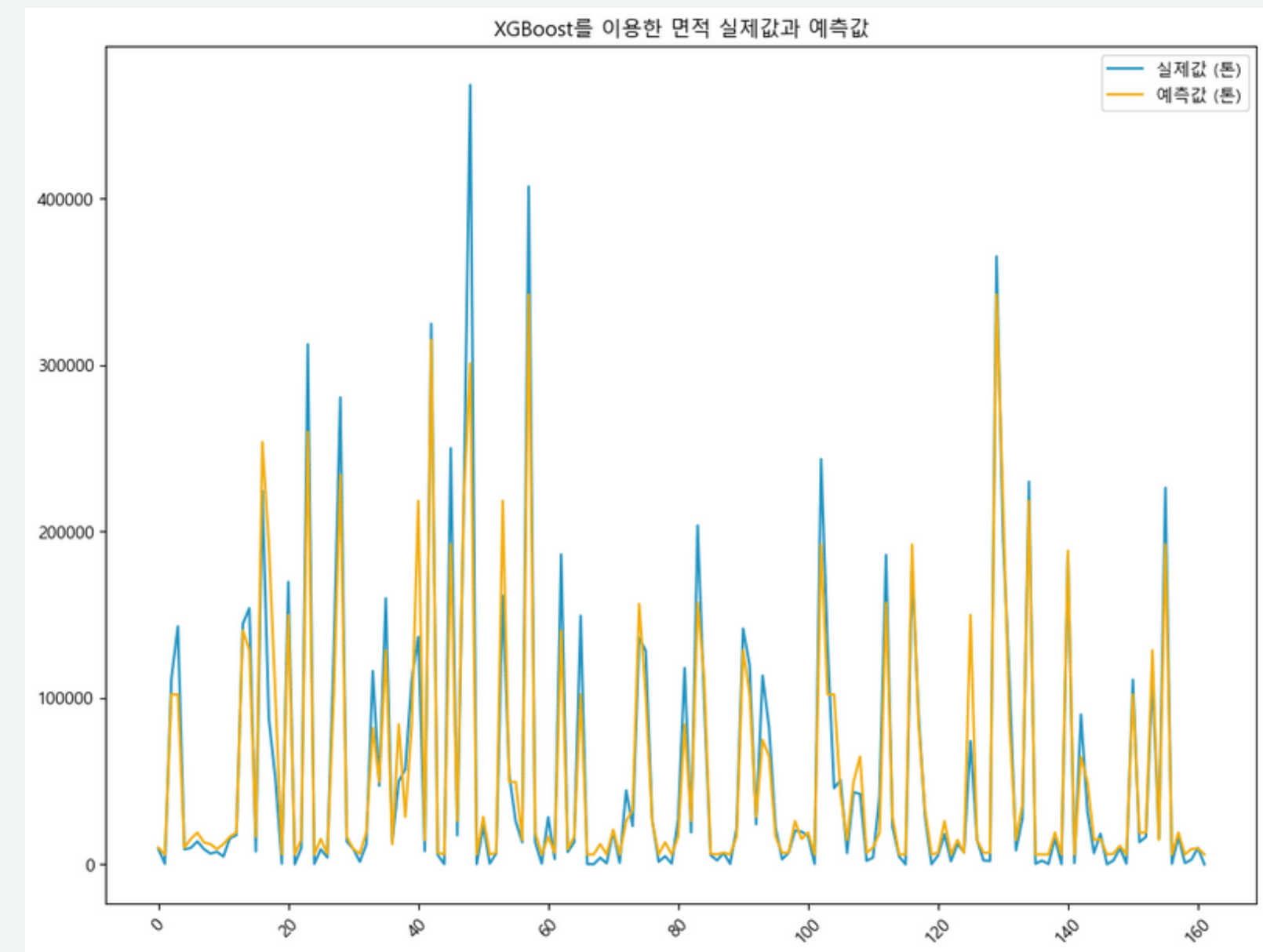
학습 모델 생성

- 생산량 예측 모델

- 목표 : 사용자가 회원가입 때 사용한 면적 정보를 사용하여 생산량을 예측한다.
- 사용 모델 : XGBoost, Lasso, Ridge, LSTM
- 최종 모델 : XGBoost
- 사용 데이터 : 면적 데이터, 생산량 데이터

```
1 grid_ridge = GridSearchCV(estimator=ridge, param_grid=params_r
2 grid_ridge.fit(X_train, y_train)
3
4 print("best score :", grid_ridge.best_score_)
5 print("best parameter :", grid_ridge.best_params_)
6
7 em = grid_ridge.best_estimator_
8 y_pred = em.predict(X_test)
9 print("r2 score :", r2_score(y_test, y_pred))
```

fig 10. GridSearchCV를 이용한 최적의 모델 선정
〈pred_output_with_area.ipynb〉



학습 모델 생성

```
grid_xgb = GridSearchCV(estimator=xgb, param_grid=params_xgb, n_jobs=-1, cv=kfold, scoring="neg_mean_squared_error")
grid_xgb.fit(X_train, y_train)

print("best score :", grid_xgb.best_score_)
print("best parameter :", grid_xgb.best_params_)

em = grid_xgb.best_estimator_
y_pred = em.predict(X_test)
print("r2 score :", r2_score(y_test, y_pred))
```

```
grid_ridge = GridSearchCV(estimator=ridge, param_grid=params_ridge, n_jobs=-1, cv=kfold, scoring="neg_mean_squared_error")
grid_ridge.fit(X_train, y_train)

print("best score :", grid_ridge.best_score_)
print("best parameter :", grid_ridge.best_params_)

em = grid_ridge.best_estimator_
y_pred = em.predict(X_test)
print("r2 score :", r2_score(y_test, y_pred))
```

```
grid_lasso = GridSearchCV(estimator=lasso, param_grid=params_lasso, n_jobs=-1, cv=kfold, scoring="neg_mean_squared_error")
grid_lasso.fit(X_train, y_train)

print("best score :", grid_lasso.best_score_)
print("best parameter :", grid_lasso.best_params_)

em = grid_lasso.best_estimator_
y_pred = em.predict(X_test)
print("r2 score :", r2_score(y_test, y_pred))
```

fig 11. XGB, Lasso, Ridge의 GridSearchCV
〈pred_output_with_area.ipynb〉

Best Score : -0.0924129521745021
Best Parameters : {'eta': 0.1, 'max_depth'

r2 score : 0.89

GridSearchCV를 통한 최적의 파라미터 탐색

scoring 값 + test data의 R2 Score 값을
기준으로 최적의 모델 선정

XGBoost를 최적의 모델로 선정

학습 모델 생성

- 생산량 예측 모델 : LSTM

- 장점 : 트렌드(추세)를 비교적 잘 따라가며, 모델의 높은 r2score (약 92~95%)

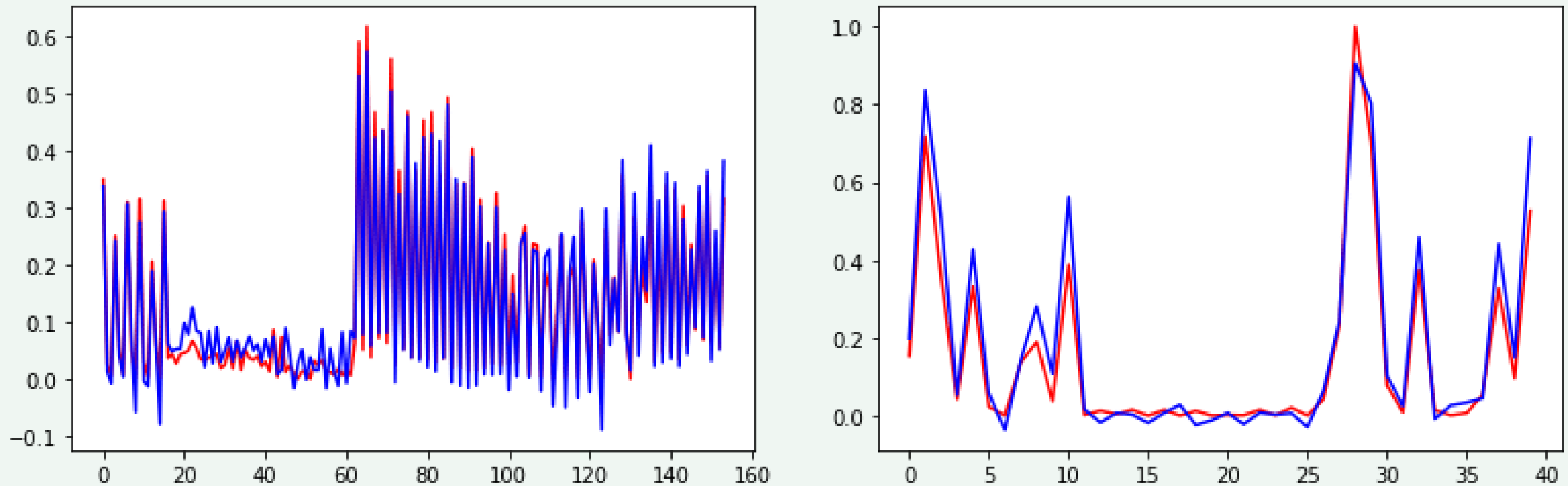


fig 12. LSTM 모델의 실제값과 예측값 비교
〈weather_output.ipynb〉

학습 모델 생성

- 생산량 예측 모델 : LSTM
 - 단점 : 예측가격의 역스케일링 시, 일부 가격이 제대로 예측되지 않음

위 표시와 같이 예측값의 `inverse_transform()`이
일부 데이터를 음수(-)값으로 변환



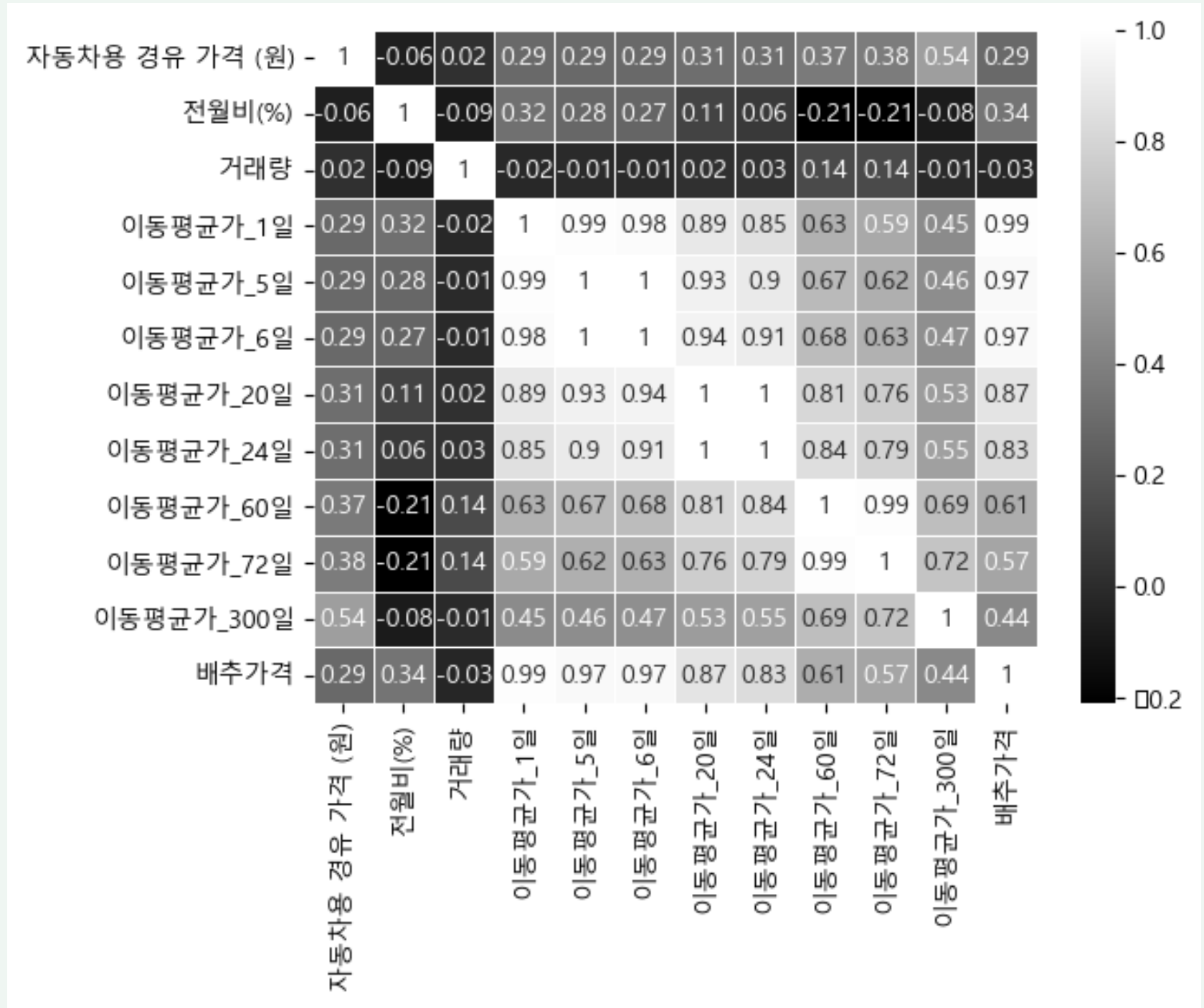
```
pred_y_fin= mms2.inverse_transform(pred_y_fin)

pred_y_fin

Output exceeds the size limit. Open the full output data in a text editor
array([[ 63569.594 ],
       [267825.72  ],
       [162881.22  ],
       [ 17098.488 ],
       [136840.53  ],
       [ 18520.057 ],
       [-11533.3125],
       [ 45830.504 ],
       [ 90018.68  ],
       [ 34826.766 ],
       [180163.02  ],
       [ 5278.1978],
       [-5515.6646],
       [ 2301.0984],
       [ 1066.2372],
       [-5868.5845],
       [ 2435.8987],
       [ 8891.124  ],
       [-7465.4575],
       [-3614.9414],
       [ 2442.5425],
       [-6950.286  ],
       [ 2545.9333],
       [ 977.1482],
       [ 2071.0686],
       ...])
```

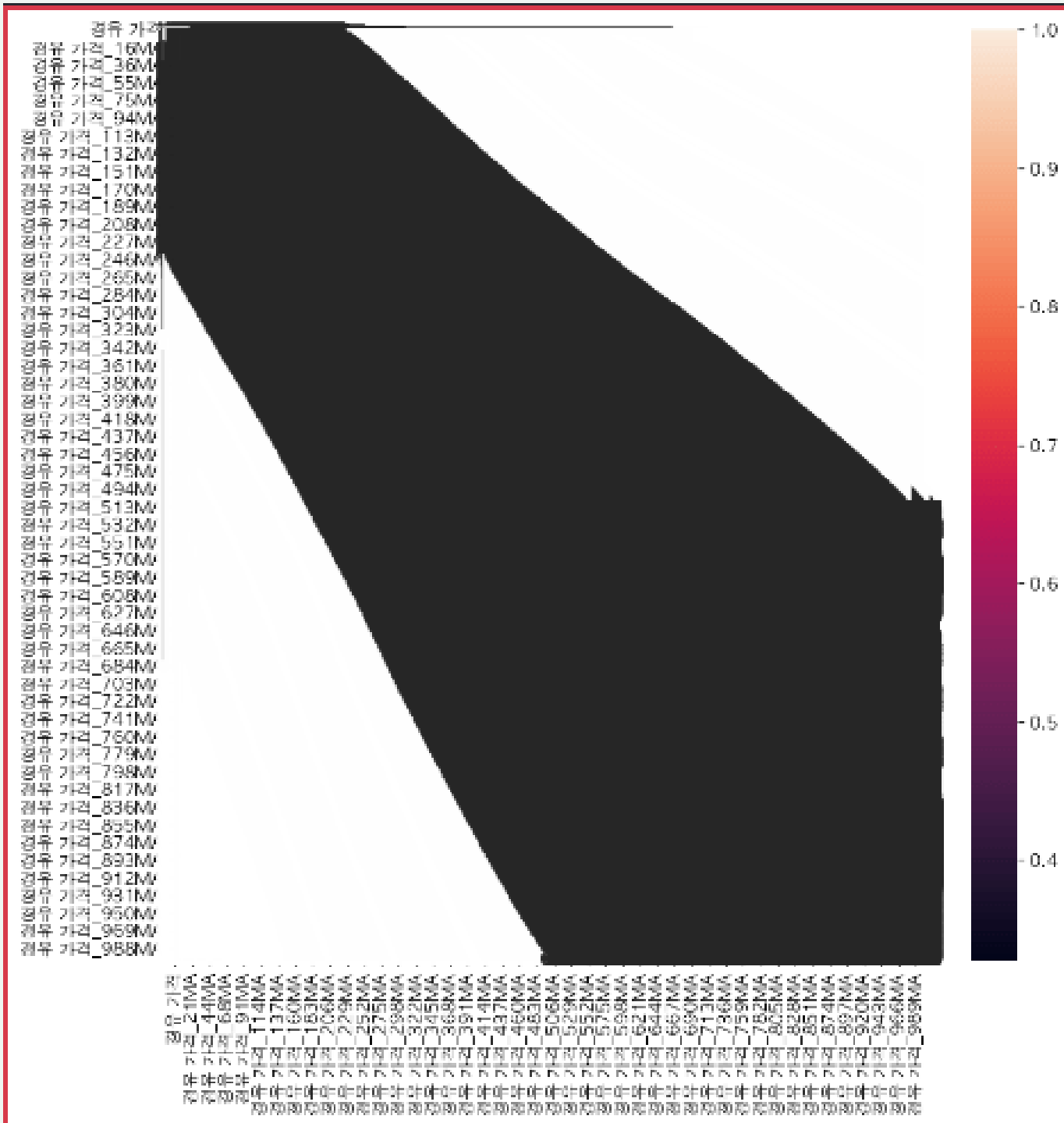
fig 13. 예측값을 역스케일링
〈weather_output.ipynb〉

학습 모델 생성



변수 간 상관 관계를 Heatmap으로
시각화하여 직관적으로 판단

fig 14. 독립변수와 종속변수 간 상관관계 heatmap
<model_pred_price_xgb_dlearn.ipynb>



상관성이 낮은 변수를 제거하거나
+
변수들의 차원을 축소

학습 모델 생성

- 가격 예측 모델

- 목표 : 사용자가 배추 판매를 원하는 시점에 예측 가격을 알려준다.
- 사용 모델 : XGBoost, LSTM
- 사용 데이터 : 배추 가격 데이터

- 모델 예시 : LSTM

```
1 # lstm 모델 생성
2 model = Sequential()
3 model.add(LSTM(50, input_shape=(4,1)))
4 model.add(Dense(1))
5 model.compile(loss='mean_squared_error', optimizer='adam')
6 model.summary()
7
8 # 모델 학습 early stopping 적용
9 early_stop = EarlyStopping(monitor='loss', patience=1, verbose=1)
10 model.fit(train_feature, train_target, epochs=100, batch_size=32, verbose=1)
```

fig 20.<predict_price_LSTM_withPrice.ipynb>

학습 모델 생성

- 가격 예측 모델 : LSTM
 - 과적합
 - 낮은 예측률

```
# 모델 성능 평가
train_predict = model.predict(train_feature)
test_predict = model.predict(test_feature)

# r2 score
print('train r2 score :', r2_score(train_target, train_predict))
print('test r2 score :', r2_score(test_target, test_predict))
```

```
train r2 score : 0.99997
test r2 score : 0.999956
```

fig 20. <predict_price_LSTM_withPrice.ipynb>

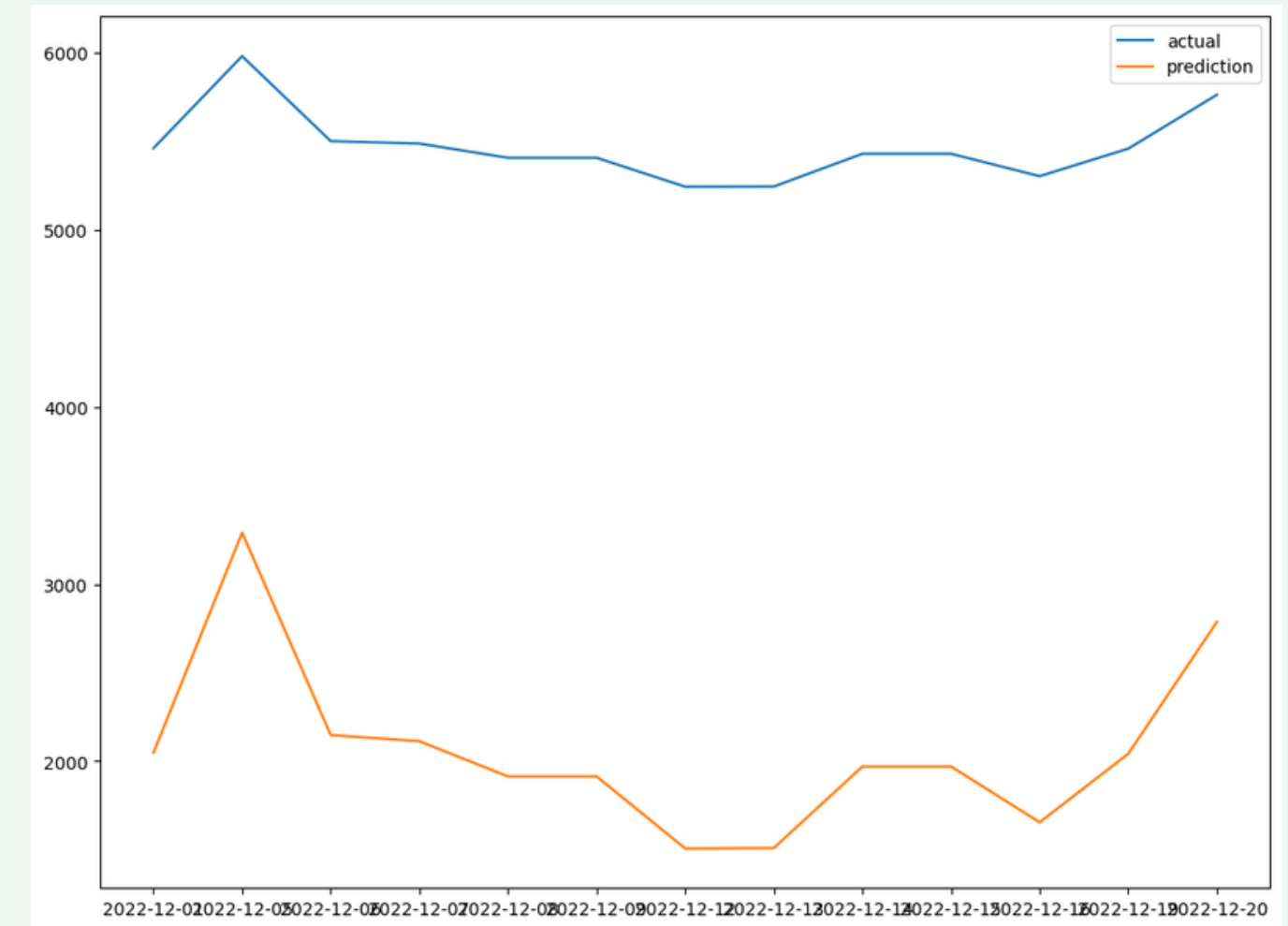


fig 21. 학습하지 않은 데이터 모델 예측 시
<predict_price_LSTM_withPrice.ipynb>

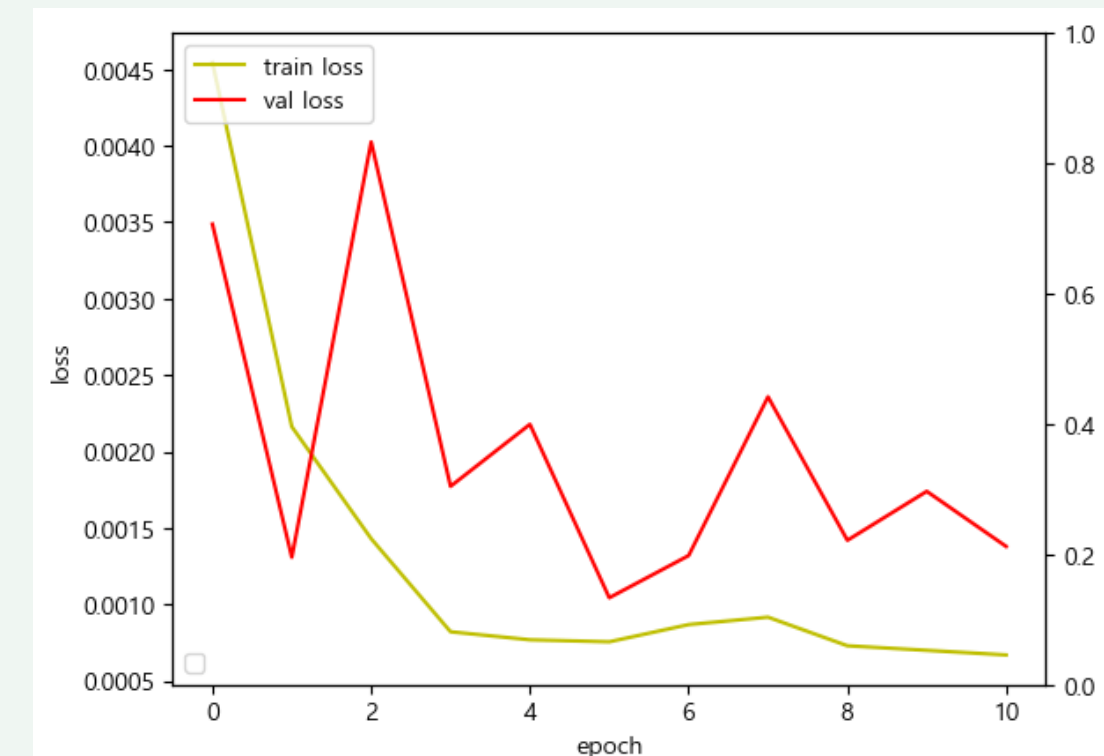


fig 22. train,
validation data의
epoch-loss
<model_pred_price
_xgb_dlearn.ipynb>

학습 모델 생성

- 가격 예측 모델 : XGBoost

- 이동 평균을 이용한 다음날 가격 예측
- 예측값을 기반으로 이후 배추 가격을 예측하고자 함

(ex. 1월 13일 가격을 예측하기 위해, 1월 11일 가격의 예측값을 통해 12일을 예측하고, 12일 예측값을 통해 13일 가격을 예측)

↔ 가격의 변화를 예측하지 못 해서 모델을 사용하지 못 함

	이동평균가_5일	이동평균가_20일	이동평균가_60일	이동평균가_300일
거래년월일				
2001-01-11	1904.0	1826.5	2088.683333	4683.080000
2001-01-12	1912.0	1823.0	2062.933333	4670.746667
2001-01-13	1924.0	1824.0	2040.266667	4658.413333
2001-01-15	1908.0	1829.5	2020.933333	4646.013333
2001-01-16	1928.0	1844.0	2004.850000	4633.813333
2001-01-17	1980.0	1864.5	1992.016667	4622.113333
2001-01-18	2058.0	1890.5	1981.475000	4610.513333
2001-01-19	2168.0	1922.0	1975.641667	4599.380000
2001-01-20	2322.0	1967.5	1975.241667	4589.146667
2001-01-22	2524.0	2032.0	1982.008333	4580.546667

fig 22. 이동평균가격을 독립 변수로
〈model_pred_price_xgb_dlearn.ipynb〉

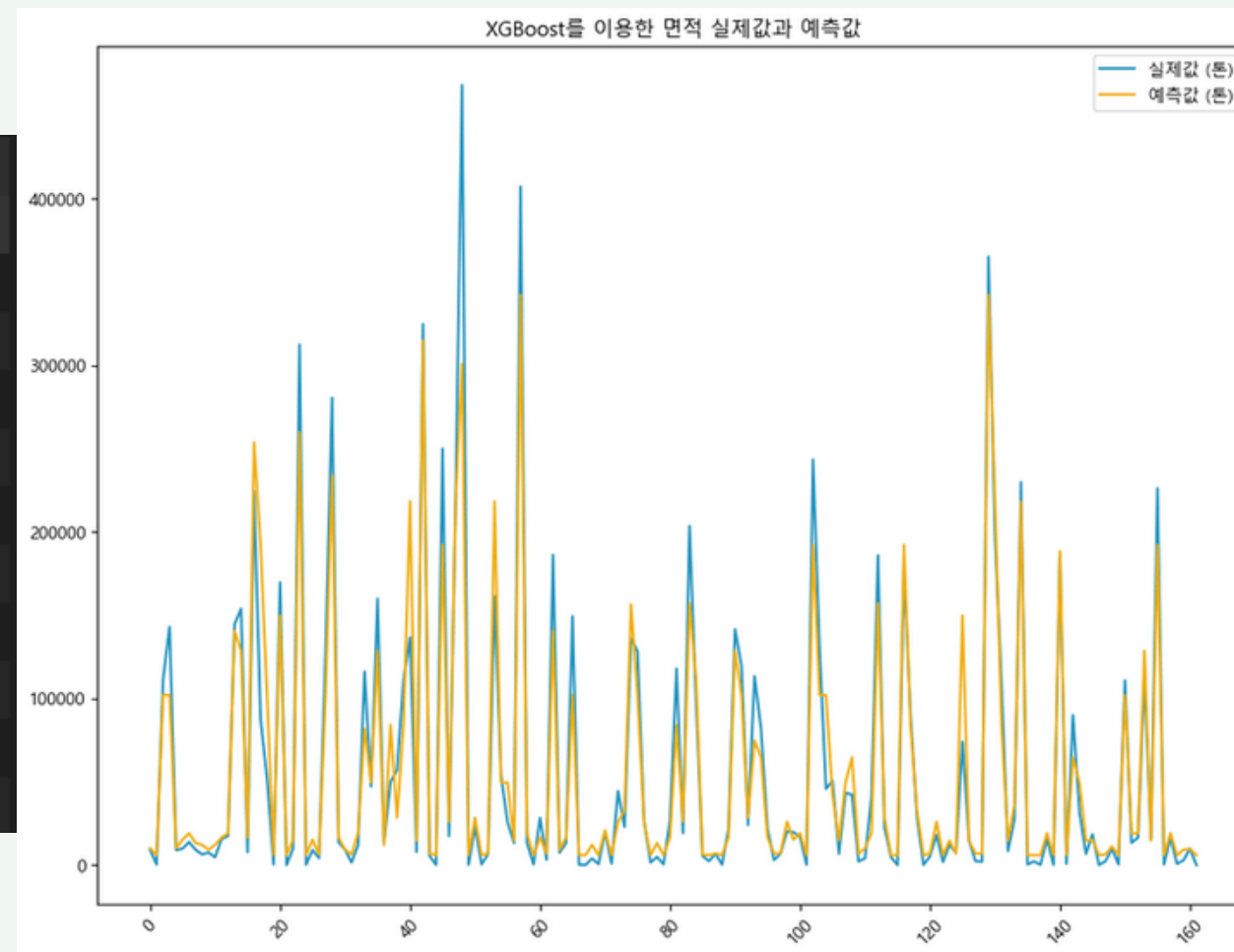


fig 22. 실제값과 예측값 비교
〈model_pred_price_xgb_dlearn.ipynb〉

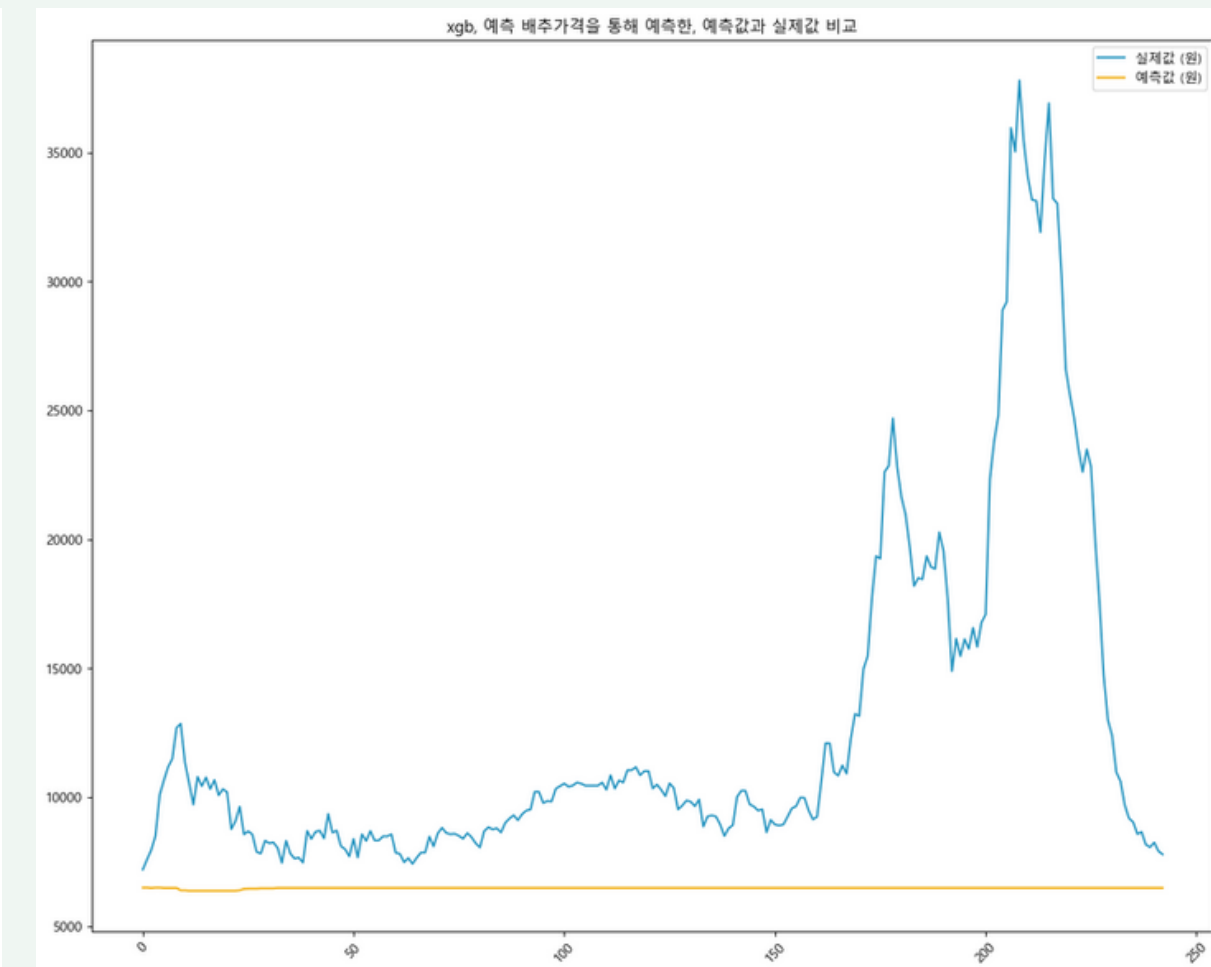


fig 22. 예측값을 이용한 예측값과 실제값 비교 그래프
〈model_pred_price_xgb_dlearn.ipynb〉

학습 모델 생성

- 가격 예측 모델

- 디테일 : 캔들스틱의 형태인 시가, 고가, 저가, 종가 및 다음날 종가(종가_shift) 컬럼 생성하여, 특정 window size를 설정하여 가격 예측 실행.

```
def create_candles(df, group_sizes):
    candles = {}
    for group_size in group_sizes:
        candle_df = pd.DataFrame(columns=['시가', '고가', '저가', '종가', '일자'])
        for i in range(len(df)-group_size):
            temp = df.iloc[i:i+group_size+1]
            open_price = temp.iloc[0]['가격']
            high_price = temp['가격'].max()
            low_price = temp['가격'].min()
            close_price = temp.iloc[-1]['가격']
            date = temp.iloc[-1].name
            candle_df = candle_df.append({'시가': open_price, '고가': high_price, '저가': low_price, '종가': close_price, '일자': date})
        candles[group_size] = candle_df

    for key, df in candles.items():
        for group_size in group_sizes:
            if key == group_size:
                df[str(key) + '종가_shift'] = df['종가'].shift(-key)

    for df in candles.values():
        df.set_index('일자', inplace=True)
        df.dropna(inplace=True)
```

캔들스틱 (시가, 고가, 저가, 종가)

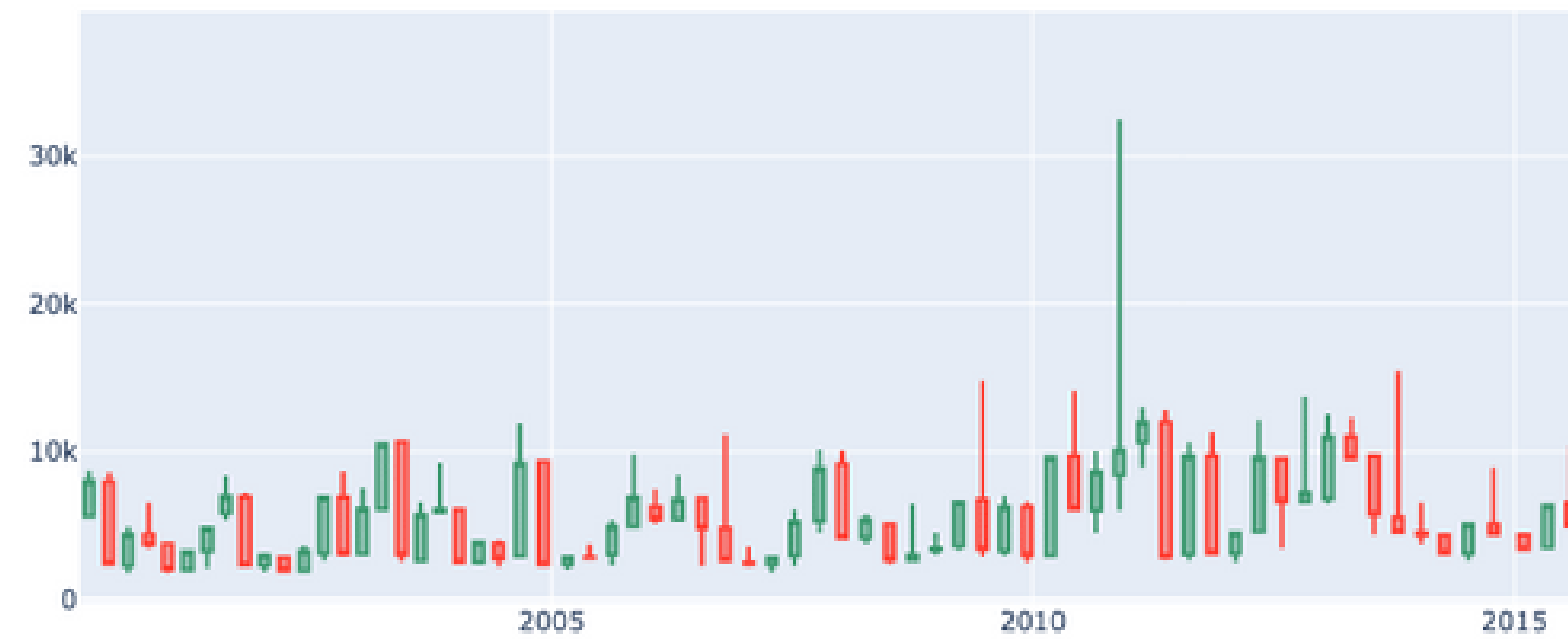


fig 15. <price_candle_XGBoost_continuous.ipynb>

학습 모델 생성

- 가격 예측 모델

- 목표 : 사용자가 배추 판매를 원하는 시점에 예측 가격을 알려준다.
- 사용 모델 : XGBoost
- 사용 데이터 : 배추 가격 데이터
- 디테일 : 캔들스틱의 형태인 시가, 고가, 저가, 종가 및 다음날 종가(종가_shift) 컬럼 생성하여, 특정 window size를 설정하여 가격 예측 실행.

	시가	고가	저가	종가	5종가_shift
일자					
2022-11-08	7350.0	7381.5	6835.5	7193.5	6684.0
2022-11-09	6835.5	7381.5	6760.0	6760.0	6900.0
2022-11-10	7142.0	7381.5	6760.0	6952.0	6432.0
2022-11-11	7381.5	7381.5	6760.0	6992.0	7018.0
2022-11-14	7330.0	7330.0	6724.0	6724.0	6300.0
2022-11-15	7193.5	7193.5	6684.0	6684.0	6260.0
2022-11-16	6760.0	6992.0	6684.0	6900.0	6100.0
2022-11-17	6952.0	6992.0	6432.0	6432.0	6100.0
2022-11-18	6992.0	7018.0	6432.0	7018.0	6100.0
2022-11-21	6724.0	7018.0	6300.0	6300.0	5664.0

2022-11-28 가격 예측

생성한 캔들스틱 : 1,2,3,4,5,10,20,60,120 일

	Actual	Predicted	Error
가격	5664.0	6093.022949	7.574558
가격	5664.0	6108.940918	7.855595
가격	5664.0	5890.554688	3.999906
가격	5664.0	6197.649414	9.421776
가격	5664.0	6343.410645	11.995244
가격	5664.0	6374.698730	12.547647
가격	5664.0	6805.323730	20.150490
가격	5664.0	7819.958984	38.064248
가격	5664.0	7220.296387	27.476984

실제 값과 예측값의 오차율

학습 모델 생성

가격	
날짜	
2022-11-01	7350.0
2022-11-02	6835.5
2022-11-03	7142.0
2022-11-04	7381.5
2022-11-07	7330.0
2022-11-08	7193.5
2022-11-09	6760.0
2022-11-10	6952.0
2022-11-11	6992.0
2022-11-14	6724.0
2022-11-15	6684.0

static

날짜					
날짜	시가	고가	저가	종가	증가_shift
2000-01-13	5600.0	5790.0	5600.0	5630.0	5910.0
2000-01-25	5630.0	5910.0	5630.0	5910.0	6970.0
2000-02-08	5930.0	6970.0	5930.0	6970.0	8370.0
2000-02-19	7230.0	8690.0	7230.0	8370.0	7630.0
2000-03-03	8030.0	8030.0	7430.0	7630.0	7990.0

continuous

일자					
일자	시가	고가	저가	종가	5종가_shift
2022-11-08	7350.0	7381.5	6835.5	7193.5	6684.0
2022-11-09	6835.5	7381.5	6760.0	6760.0	6900.0
2022-11-10	7142.0	7381.5	6760.0	6952.0	6432.0
2022-11-11	7381.5	7381.5	6760.0	6992.0	7018.0
2022-11-14	7330.0	7330.0	6724.0	6724.0	6300.0

캔들스틱 생성 방법

2022-11-08 일 5종가_shift 생성 시 전 5일에 데이터를 사용한다.

학습 모델 생성

정확도 비교

continuous

static

5일

```
Best hyperparameters for time period 5: {'colsamp1  
'reg_alpha': 0, 'reg_lambda': 0.1, 'subsample': 0.  
Best score for time period 5: 0.8946459699251825  
RMSE for time period 5: 0.0713846079588744  
r2 score for time period 5: 0.7713629508940741
```

```
Best hyperparameters for time period 5: {'colsamp1  
'reg_alpha': 0.01, 'reg_lambda': 0, 'subsample': 0  
Best score for time period 5: 0.8966602505936994  
RMSE for time period 5: 0.07078456636595712  
r2 score for time period 5: 0.8057506611795663
```

20일

```
Best hyperparameters for time period 20: {'colsamp  
100, 'reg_alpha': 0.1, 'reg_lambda': 0.5, 'subsamp  
Best score for time period 20: 0.3616222794143953  
RMSE for time period 20: 0.137260082265927  
r2 score for time period 20: 0.1586220491588307
```

```
Best hyperparameters for time period 20: {'colsamp  
500, 'reg_alpha': 0.1, 'reg_lambda': 0.5, 'subsamp  
Best score for time period 20: 0.3078528740200751  
RMSE for time period 20: 0.17357028427155774  
r2 score for time period 20: 0.18415116400730724
```


학습 모델 생성

- 가격 예측 모델 : XGBoost
 - 비교적 높지 않은 오차율
 - 최대 120일까지 예측 가능
 - 시도한 방법 : 캔들스틱 데이터 프레임 생성시 종가_shift 컬럼 생성을 위해서 static, continuous 방법을 시도 하였다.
 - static
 - 캔들스틱을 만들때 비연속적으로 생성한다.
 - 예시) 크기가 5인, 2022-1-13일 캔들스틱을 생성할때 전 거래일 5개로 만들고, 사용한 5개에 데이터는 사라진다.
 - N일 종가_shift 컬럼을 기준으로 날짜가 줄어들어, 행 수가 줄어든다. 줄어든 행수로 인해 학습시 정확도가 떨어진다.
 - 예시) 120일 종가_shift의 경우 원본 df을 캔들스틱으로 만들 시 전체 행수가 120으로 나눈어져서, 데이터 수가 현저하게 떨어진다. 이로인해 정확도가 떨어진 것을 확인할 수 있다.
 - continuous
 - 캔들스틱을 만들때 연속적으로 생성한다
 - 예시) 크기가 5인, 2022-1-13일 캔들스틱을 생성할때 전 거래일 5개로 만들고, 사용한 5개에 데이터는 사라지지 않는다.
 - 전체 데이터의 행수가 온전히 유지되어, 학습 시 정확도가 static 방법보다 높다.

웹사이트 구현

- 사용 도구: 장고 & 비주얼 스튜디오 코드
- 구현 페이지 및 기능
 - 회원 가입
 - 로그인/로그아웃(기능)
 - 마이 페이지
 - 게시판 (POST 기능)
 - 생산 적합 여부 판단
 - 생산량 & 가격 예측
 - 면적에 따른 예상 수익(기능)
- 데이터를 DB에 저장
 - 지역별 연도별 기후 데이터 (생산량, 면적 추가)
- 머신러닝 모델 장고와 연동
 - 배추 생산 가능 여부 판단 모델 (XGBoost)
 - 배추 생산량 예측 모델 (XGBoost)
 - 배추 예측 가격 모델 (XGBoost)

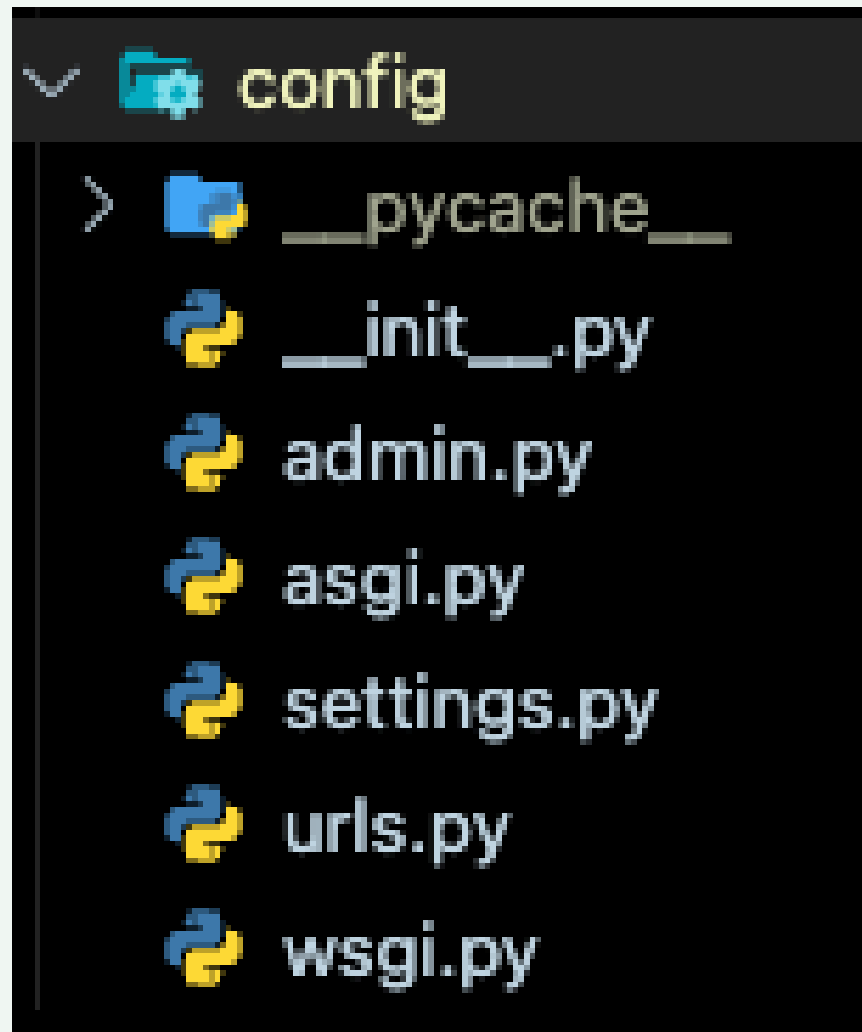
장고 생성 앱

✓ FINAL_PREDICT_AGRICULTURAL_DJANGO

- > .vscode
- > api_checker
- > common
- > config
- > model
- > output
- > predict
- > pybo
- > rawdata
- > recommend
- > save_csv
- > static
- > templates

- api_checker : api 연동
- common : 회원가입, 로그인, 로그아웃, 마이페이지
- model : 머신러닝 모델 (XGBoost) 저장
- output : 생산량 예측
- pybo : 질문, 답변 게시판
- recommend : 생산 적합 여부 판단
- save_csv : csv를 모델로 저장
- process.py : db에 csv 저장

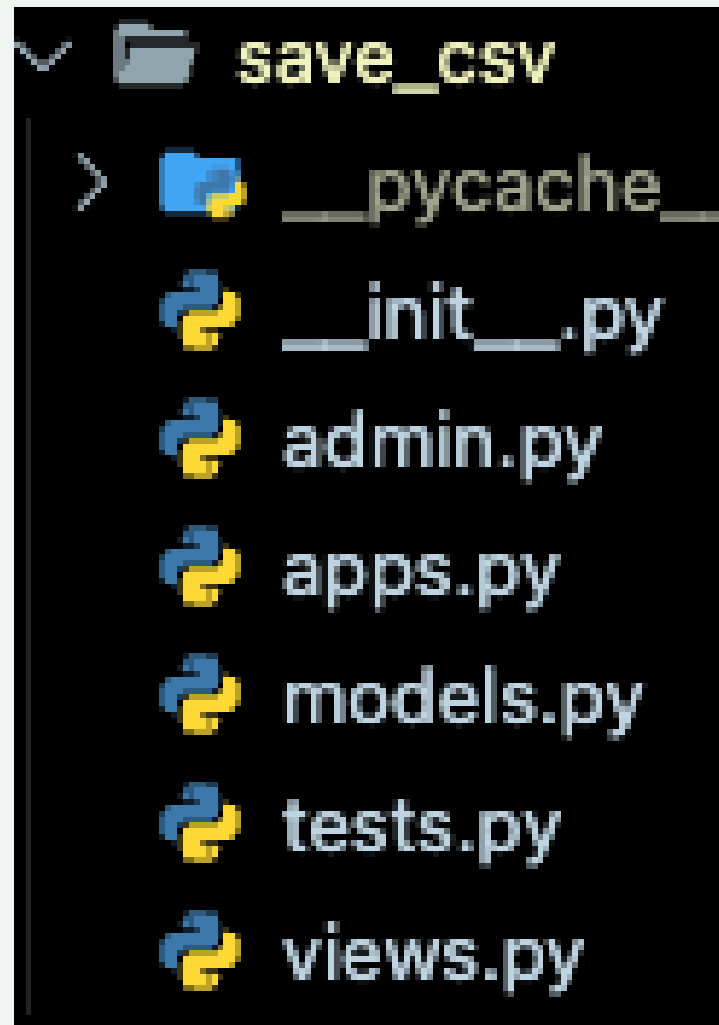
CONFIG



```
INSTALLED_APPS = [  
    "rest_framework",  
    "corsheaders",  
    "common.apps.CommonConfig",  
    "pybo.apps.PyboConfig",  
    "django.contrib.admin",  
    "django.contrib.auth",  
    "django.contrib.contenttypes",  
    "django.contrib.sessions",  
    "django.contrib.messages",  
    "django.contrib.staticfiles",  
    "save_csv",  
    "recommend",  
    "output",  
    "predict",  
    "api_checker",  
]
```

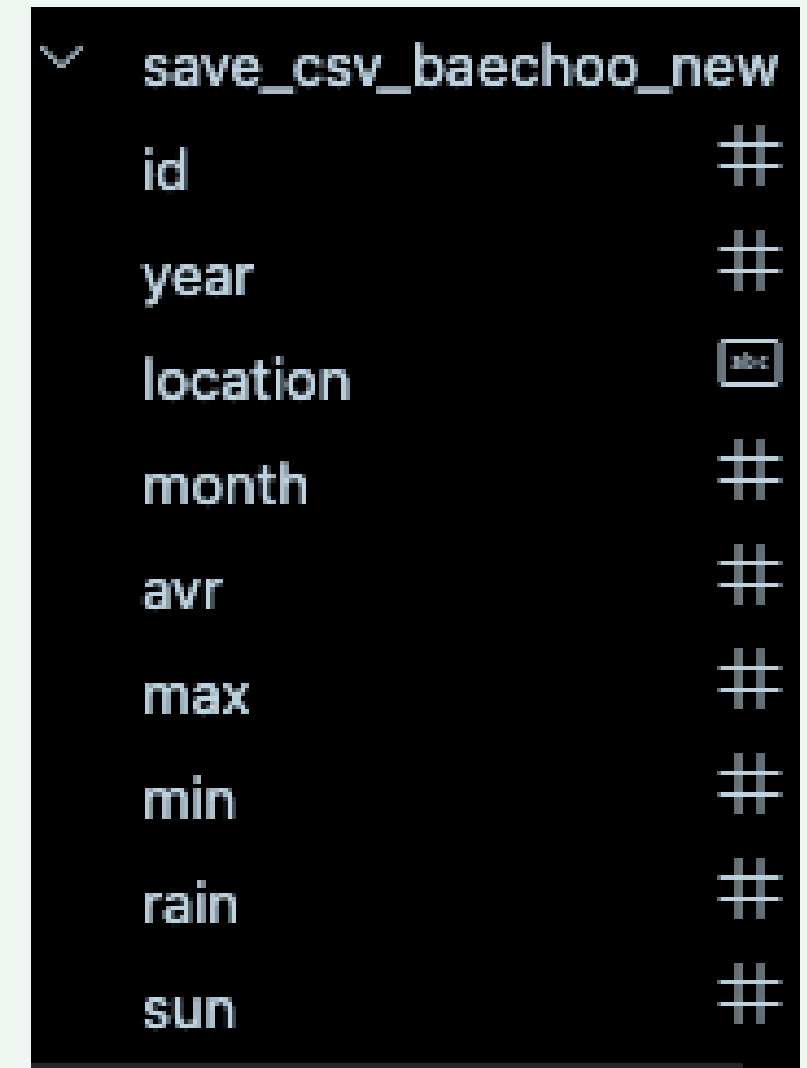
INSTALLED_APPS : 사용한 app 추가

SAVE_CSV



- 앱 기능 : 배추가격 csv 파일을 전처리 한 후 DB에 저장

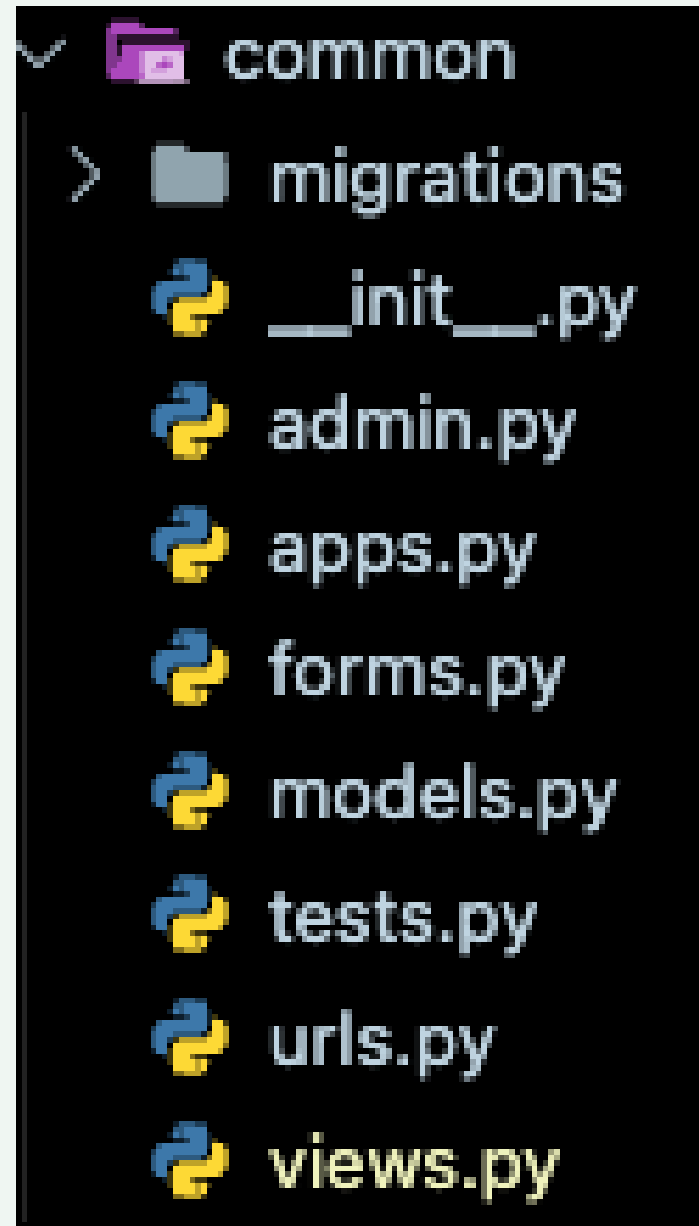
- 생성된 DB



A screenshot of a database table named 'save_csv_baechoo_new'. The table has the following columns:

id	#
year	#
location	abc
month	#
avr	#
max	#
min	#
rain	#
sun	#

COMMON



- 앱 기능 : 회원가입 및 로그인 기능 구현, 마이페이지 생성
- `views.py` : 로그인, 로그아웃, 마이페이지에 모델 DB를 렌더링 해준다.
- 연결 페이지 : `common/login.html`, `common/mypage.html`

COMMON

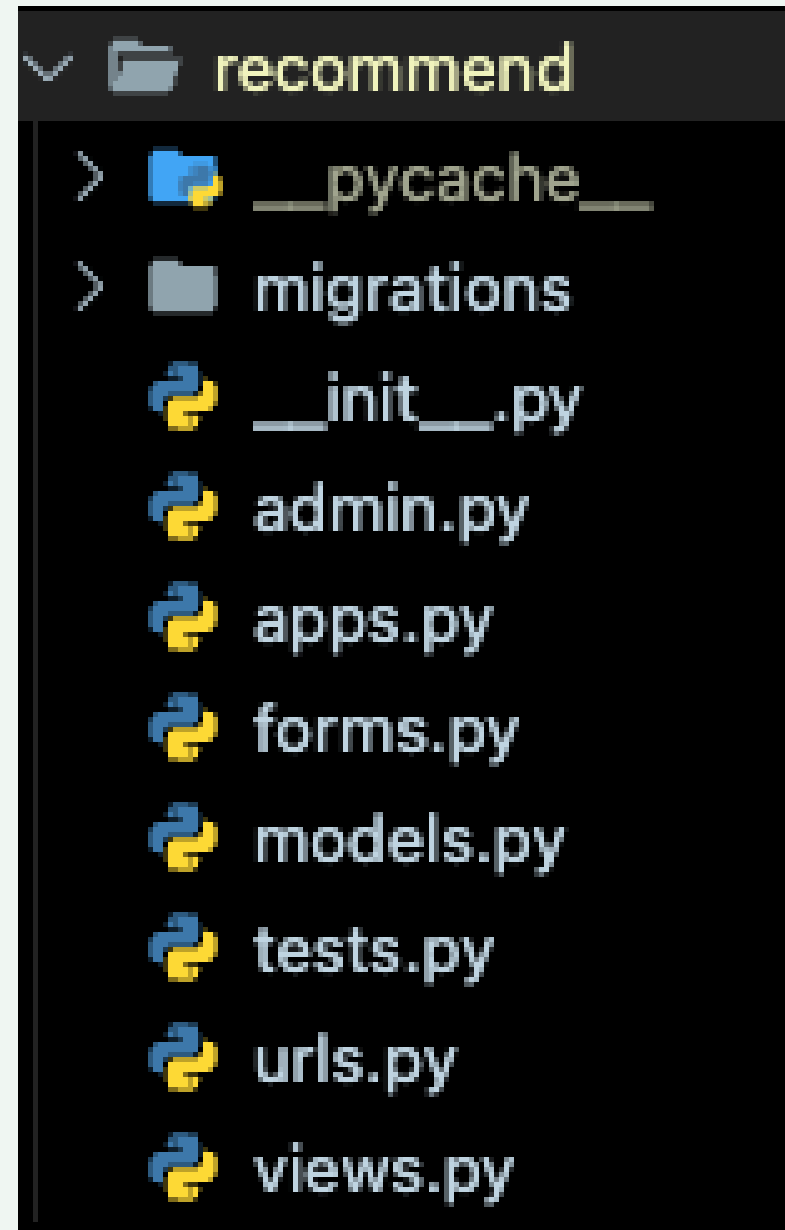
```
def signup(request):
    if request.method == "POST":
        form = UserForm(request.POST)
        if form.is_valid():
            user = form.cleaned_data.get("username")
            phone = form.cleaned_data.get("phone")
            area = form.cleaned_data.get("area")
            location = form.cleaned_data.get("location")
            # NOT NULL constraint failed common_userinfo.user_id
            UserInfo.objects.create(
                user=user, phone=phone, area=area, location=location
            )
            auth_login(request, user)
            return redirect("pybo:index")
        else:
            form = UserForm()
    return render(request, "common/signup.html", {"form": form})
```

```
def logout(request):
    auth_logout(request)
    return redirect("pybo:index")
```

```
def login(request):
    if request.method == "POST":
        form = AuthenticationForm(request, request.POST)
        if form.is_valid():
            auth_login(request, form.get_user())
            return redirect("pybo:index")
        else:
            form = AuthenticationForm()
    return render(request, "common/login.html", {"form": form})
```

<views.py>
로그인, 로그아웃, 회원가입 정의

RECOMMEND



- 앱 기능 : 배추 재배 적합 여부 파악
- models.py : 지역과 월을 정의
- views.py :
 - 사용자가 입력한 지역과 월 모델DB 저장
 - DB 저장된 기후 데이터 이용 XGBoost 모델 통한 적합 여부 판단
 - 적합 여부 결과, 사용자 입력 지역과 월을 페이지에 렌더링
- 연결 페이지 : [recommend/predict.html](#), [recommend/result.html](#)

RECOMMEND

```
def predict(request):
    if request.method == "POST":
        form = PredictForm(request.POST)
        if form.is_valid():
            location = form.cleaned_data.get("location")
            month = form.cleaned_data.get("month")
            PredictionInput.objects.create(location=location, month=month)
            obj = PredictionInput.objects.last()

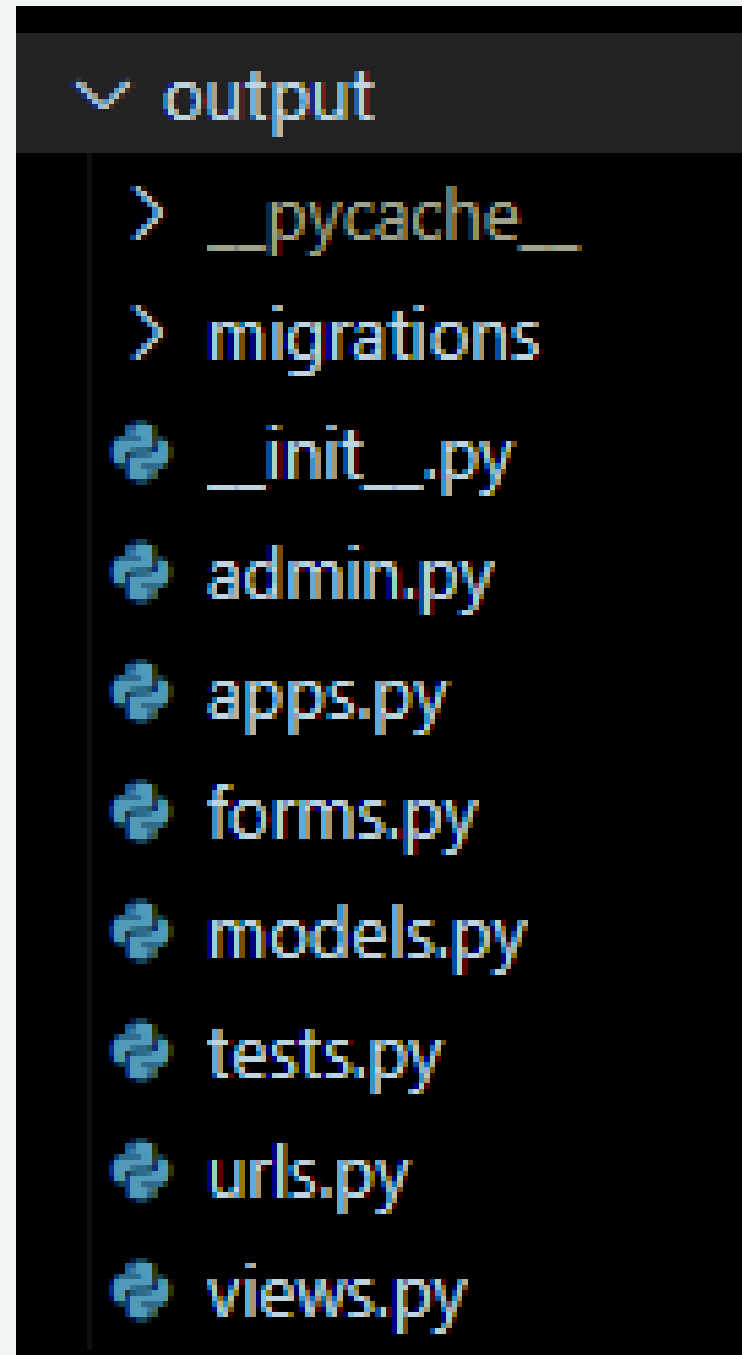
            if obj.month==11:
                bae_1= baechoo_new.objects.get(location= obj.location, month=obj.month)
                bae_2= baechoo_new.objects.get(location= obj.location, month=obj.month+1)
                bae_3= baechoo_new.objects.get(location= obj.location, month=1)
```

```
        return render(request, "common/result.html", context)
    else:
        form = PredictForm()
        user = request.user
        user_info = UserInfo.objects.get(user=user)
        context = {
            "user_info": user_info,
            "form": form,
            "y_p": None,
        }
        return render(request, "common/recommend.html", context)
```

<views.py>

predict : 배추재배 적합 여부 함수 생성 후 랜더링

OUTPUT



- 앱 기능 : 재배면적을 입력하면 생산량 데이터 추출
- views.py : 로그인, 로그아웃, 마이페이지에 모델 DB를 렌더링 해준다.
- 연결 페이지 : output/result_output, common/mypage.html

OUTPUT

```
def predict(request):
    if request.method == "POST":
        form = PredictForm(request.POST)
        if form.is_valid():
            area = form.cleaned_data.get("area")
            PredictionOutput.objects.create(area=area, output=0)
            obj = PredictionOutput.objects.last()

            obj_list = [obj.area]

            model_input_list = np.array(obj_list).reshape(1, -1)

            with open("model/pred_xgb_output_with_area_sc_f.pkl", "rb") as f:
                scaler_f = joblib.load(f)
                feature = scaler_f.transform(model_input_list)
```

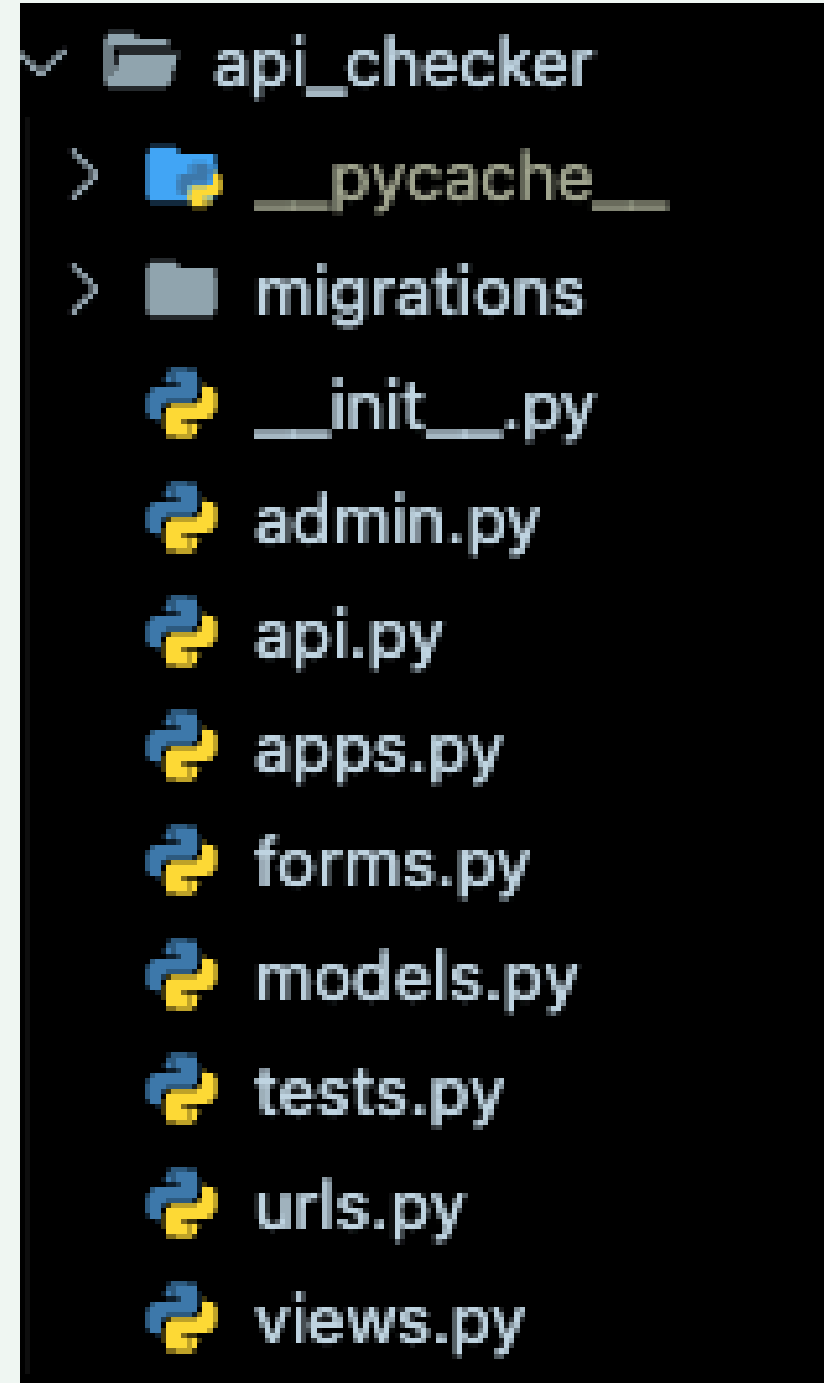
<views.py>

predict : 면적 입력시 배추 생산량 예측

```
        return render(request, "common/result_output.html", context)
```

```
    else:
        form = PredictForm()
        user = request.user
        user_info = UserInfo.objects.get(user=user)
        context = {
            "user_info": user_info,
            "form": form,
        }
        return render(request, "common/recommend_op.html", context)
```

API_CHECKER



- 앱 기능 : 실시간 API 연동, 머신러닝 모델 (XGBoost)를 활용한 가격 예측 기능 구현, DB에 예측가격 저장
- `api.py` : 사용자의 요청에 따른 API 호출, 캔들스틱 형식으로 데이터 전처리 (함수 정의)
- `views.py` : XGBoost 모델을 통한 가격 예측, 모델 DB에 저장 후 페이지에 렌더링
- 연결 페이지 : `api_checker/detail.html`

API_CHECKER

```
def check_api():
    url = "http://www.kamis.or.kr/service/price/xml.do"
    returnType = "json"
    ID = 3067
    p_itemcategorycode = 200
    p_itemcode = 211
    today = datetime.datetime.today()
    start_date = (today - datetime.timedelta(365)).strftime("%Y-%m-%d")
    end_date = today.strftime("%Y-%m-%d")
    p_productrankcodes = ["04", "05"]

    for rank in p_productrankcodes:

        queryParams = "?" + urlencode(
            {
                quote_plus("action"): "periodProductList",
                quote_plus("p_startday"): start_date,
                quote_plus("p_endday"): end_date,
                quote_plus("p_itemcategorycode"): p_itemcategorycode,
                quote_plus("p_itemcode"): p_itemcode,
                quote_plus("p_productrankcode"): rank,
                quote_plus("p_cert_key"): serviceKeyDecoded,
                quote_plus("p_cert_id"): ID,
                quote_plus("p_returntype"): returnType,
            }
        )
        res = requests.get(url + queryParams)
```

<api.py>

1. check_api : api 연동 후 데이터 추출

```
    }
)
res = requests.get(url + queryParams)

contents = res.text
json_ob = json.loads(contents)
choice = json_ob["data"]["item"]

year = []
date = []
price = []
for dct in choice:
    if dct["countname"] == "평균":
        year.append(dct["yyyy"])
        date.append(dct["regday"])
        price.append(dct["price"])

globals()["df_" + rank] = pd.DataFrame(zip(year, date, price))
globals()["df_" + rank].columns = ["year", "date", "price_" + rank]

df = pd.merge(df_04, df_05, on=["year", "date"], how="inner")
df["price_04"] = df["price_04"].str.replace(",", "").astype("int")
df["price_05"] = df["price_05"].str.replace(",", "").astype("int")

df["가격"] = df[["price_04", "price_05"]].mean(axis="columns").astype("int")
df.drop(columns=["price_04", "price_05"], inplace=True)

cols = ["year", "date"]
df["날짜"] = df[cols].apply(lambda row: "/".join(row.values.astype(str)), axis=1)
df["날짜"] = pd.to_datetime(df["날짜"])
df = df.drop(labels=["year", "date"], axis=1)

df = df.set_index("날짜")

return df
```

API_CHECKER

```
def create_candles(df, group_sizes):
    candles = {}
    for group_size in tqdm(group_sizes):
        candle_df = pd.DataFrame(columns=["시가", "고가", "저가", "종가", "일자"])
        for i in range(len(df) - group_size):
            temp = df.iloc[i : i + group_size + 1]
            open_price = temp.iloc[0]["가격"]
            high_price = temp["가격"].max()
            low_price = temp["가격"].min()
            close_price = temp.iloc[-1]["가격"]
            date = temp.iloc[-1].name
            candle_df = candle_df.append(
                {
                    "시가": open_price,
                    "고가": high_price,
                    "저가": low_price,
                    "종가": close_price,
                    "일자": date,
                },
                ignore_index=True,
            )
        candles[group_size] = candle_df

    for key, df in candles.items():
        for group_size in group_sizes:
            if key == group_size:
                df[str(key) + "종가_shift"] = df["종가"].shift(-key)

    for df in candles.values():
        df.set_index("일자", inplace=True)
        df.dropna(inplace=True)

    return candles
```

```
def get_candle_df():
    df_origin = check_api()
    df_origin_list = df_origin["가격"].tolist()

    candles = create_candles(df_origin, [1, 2, 3, 4, 5, 10, 20, 60, 120])
    (
        candel_df_1,
        candel_df_2,
        candel_df_3,
        candel_df_4,
        candle_df_5,
        candle_df_10,
        candle_df_20,
        candle_df_60,
        candle_df_120,
    ) = (candles[size] for size in [1, 2, 3, 4, 5, 10, 20, 60, 120])

    candle_df_last = {key: df.iloc[-1] for key, df in candles.items()}

    for df in candle_df_last.values():
        df = df.T.dropna()

    return candle_df_last, df_origin_list
```

〈api.py〉

2. create_candles : 모델에 맞춰 api 데이터 캔들 스틱 생성

3. get_candle_df : 일자별로 추출

API_CHECKER

```
def predict_price(days=1):

    # Load the model and scalers
    model = pickle.load(
        open("model/price_candle_XGBoost_continuous_{}.pkl".format(days), "rb")
    )
    scaler1 = pickle.load(open("model/price_candle_scaler1_{}.pkl".format(days), "rb"))
    scaler2 = pickle.load(open("model/price_candle_scaler2_{}.pkl".format(days), "rb"))

    # load df
    candle_df_last, df_organ_list = get_candle_df()

    return {
        "common/api_detail.html",
        {
            "context": context,
            "item_5": Result.objects.get(id=obj.id).item_5,
            "item_20": Result.objects.get(id=obj.id).item_20,
            "item_365": Result.objects.get(id=obj.id).item_365,
            "output": PredictionOutput.objects.get(id=obj.id).output,
            "area": PredictionOutput.objects.get(id=obj.id).area,
```

<views.py>

4. predict_price : XGBoost 모델 사용하여 가격 예측

2. detail : 예측한 가격 저장해두어 api의 값이 바뀌기 전엔 저장한 결과 반환으로 시간을 단축 html에 필요한 값 렌더링

결과

인사이트 도출

생산품목

생산량

미래 가격

생산량 결정 요인 : 면적, 기상 등

실제 생산량과 출하량의 차이
(인위적 생산량 조절)
(생산량을 통한 가격 조절)



예측 생산량과 미래 가격 정보를 기반
으로 생산자가 가격의 변화를 예측하고
합리적인 가격을 인지

배추의 적절한 생산량을 조정하며
효율적인 이윤 전략을 수립할 수 있도록
함



한계점

- 배추 가격변동에 영향을 끼치는 요인이 다양

1) 기상 변화 : 태풍, 비, 홍수 등의 재난으로 생산량을 감소

2) 정책 변화 : 배추는 식용으로 사용되는 중요한 식량 하나로 정책적으로 여러가지가 시행 되었다.

- 배추 재배 확대(지원금 지급이나 재배 기술 개발) =

- 수입 배추 규제(수입 배추가 생산량에 미치는 영향을 최소화 하기 위하여 규제 정책 시행)

- 소비자 지원(배추를 저렴하게 구매할 수 있도록 지원 정책 시행)

3) 배추 생산량 급등 : 다수의 생산자가 서비스를 이용하면 생산물의 공급이 많아져 오히려 가격이 하락될 것으로 예상

- 모델을 학습시킬 때 직면한 문제점

1) 데이터의 수 : 데이터의 수가 많지 않아 (약 6000건) 신뢰도가 높은 모델을 구현하는데 어려움이 존재했다.

- 20년 전 데이터는 재배 기술 차이가 영향을 미쳤을 것이라 생각해 배제

- 수익을 예측할 때 직면한 문제점

1) 원가 데이터 : 농기계, 퇴비 등 각 개인에 따라 다른 조건을 가지고 있는 경우, 이를 취득하거나 개별로 적용하는 데 어려움이 있다.

개발 후기 및 느낌점

김현아

프로젝트를 하며 발생하는 많은 오류를 통해 모든 부분을 해결할 순 없었지만 해결되는 부분을 보며 뿌듯함을 느꼈고 앞으로도 문제가 발생한다면 포기하지 않고 다양한 방면으로 해결해가며 그 속에서도 배움을 챙겨야겠다고 생각했다. 추후 공부를 통해 모자란 부분을 채워나가야겠다.

방승현

의도치 않은 부분에서 오류들을 많이 맞닥뜨리며, 스스로도 몰랐던 부분들을 많이 배운 것 같다. 장고나 머신러닝 또는 딥러닝을 조금 더 이해한 것 같지만, 앞으로도 많은 연습과 이해가 필요할 것 같다.

박정현

해결되지 못한 현실의 문제를 찾아서, 해결방안을 도출하는 과정에서, 수업 시간에 배운 데이터 전처리, 머신러닝, 장고를 적극 활용할 수 있었다. 유의미한 결과를 내어 자신감이 생겼고, 코드의 완결성을 위해 시간을 내어 더 수정할 것이다.

안진혁

가격 예측이라는 쉽지 않은 분야에 도전한 것에 큰 의의를 두고 싶다. 머신러닝 모델의 원리를 이해하고 일을 진행한다면 더욱 신뢰할 만한 모델을 만들 것이라 생각된다. 공부 후 프로젝트를 더욱 발전시킬 미래의 내 모습이 약간? 기대된다.

개선 방향

- 배추 이외의 다양한 농산물 예측 추가 필요
- 업데이트 되는 기후, 면적, 생산량, 가격 데이터를 새로이 갱신하고, 모델에 학습시켜 모델 신뢰도 향상에 기여
- 특정 작물의 가격 예측 시행 횟수가 많을 경우, 가격 하락의 가능성을 경고

For all purpose

THANK YOU

**FOR
LISTENING**

2023 Jan 13

1조

안진혁, 김현아, 방승현, 박정현



Fu;Root