

대파 가격 예측 모델

201814471_서지희
데이터마이닝 기말 과제



COMPANY NAME / 00. 00. 20XX



CONTENTS

01. 개요

모델을 만든 목적과 기대효과에 대해 설명합니다.

02. 데이터 확인, 데이터 전처리

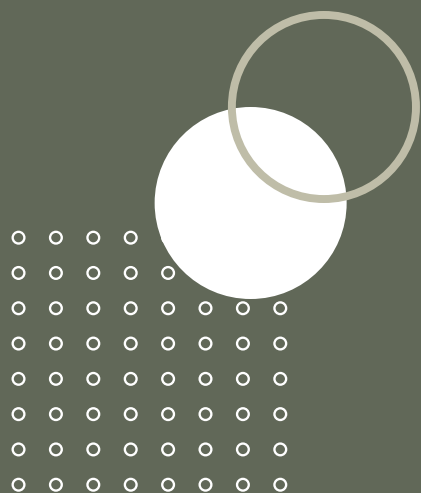
어떤 데이터를 수집했는지 그리고 데이터 수집 방법, 데이터 전처리, 사용한 모델을 간단하게 설명하겠습니다.

04. 모델링

정제한 데이터를 다시 한번 확인하고 모델링을 합니다.

05. 소감 및 출처

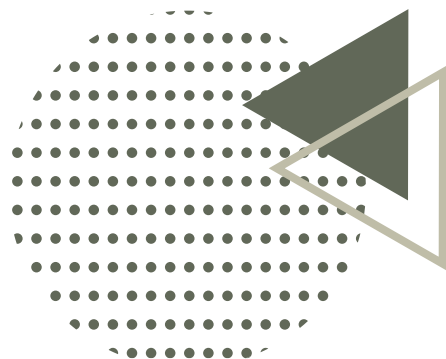
프로젝트를 진행하면서 느꼈던 점이나 아쉬운 점, 출처를 설명합니다.





01

개요



4배 된 파값...소비자물가 14개월 만에 최대 상승



파값 305.8% ↑ 비상 걸린 식탁 물가, 정부 대책은?

유승호 기자 | ④입 뉴스 > 경제 > 경제일반

'金파' 된 대파, 1년 새 3배 넘게 켜쭉...밥상물가 상승 부추겨

뉴스 입력 2021-03-04 10:14 수정 2021-03-04 10:15



삼겹살집 파절이 실종사건...파값 올라도 너무 올랐다

통계청 3월 소비자물가 동향

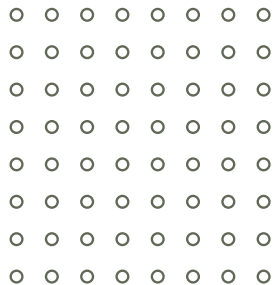
작황 부진·조류독감 여파에
농축산물 물가 오름세 지속
달걀 값도 39% 올라 서민 부담
주부들 "장보기가 무서워요"

공업제품 물가도 상승 전환

'파테크' 탄생시킨 金파 대란...78% 오른

전경윤, 송민근 기자 | 입력 : 2021.04.02 17:29:11 수정 : 2021.04.02 20:50:17

[중앙일보]입력 2021.05.07 05:00



기대효과



가격 예측

기상 변화에 따른 생산량의 변화가 심한 농산물의 가격을 예측하여 원활한 수급이 이루어지도록 한다.
특히 올해 '파테크'라는 단어가 나올 정도로 대파의 가격이 많이 올랐는데 이러한 상황을 예측할 수 있다.



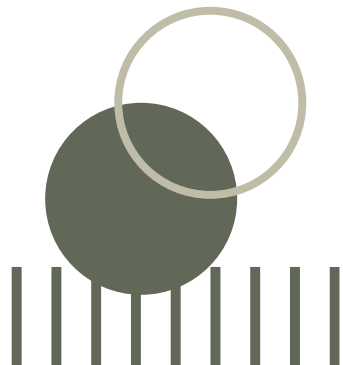
합리적인 가격으로 소비

농산물을 구매하는 소비자는 모델을 확인함으로써, 날씨에 따라 적당한 가격을 예측할 수 있게 되고,
합리적인 가격으로 농산물을 소비할 수 있다.



수입 영향

가격이 오를 것을 예측하여 농산물의 수입량을 정해 가격이 안정화 되도록 한다.



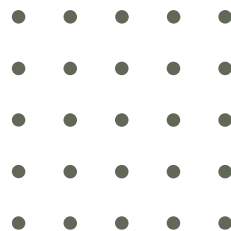


02

데이터 확인



파 가격 예측



2. 파의 재배현황

파의 재배는 거의 노지에서 이루어지고 최근 경기, 충북, 전북 등 겨울에 추위가 다소 심한 지역에서 일부 시설재배를 하고 있다. 도시근교에서는 실파재배, 엇파재배, 대파재배 등 파의 시세에 따라 출하형태가 다양하고, 진도와 부산의 명지 등은 겨울에 대파를 출하하는 유명한 지역이다.

파 재배면적 도별로 생산량 및 재배면적을 비교하여 보면 전남, 경기, 경북, 충남 순으로 파를 많이 생산하고 있으며 시군별로는 진도군, 부산시, 대구시, 보성군, 부안군이 파재배 단지를 형성하고 있다.

3. 특성 및 재배환경

가. 특 성

백합과에 속하는 다년생 초본이지만 재배상 1년생 또는 월년생채소로 취급되고 있다. 잎은 잎새부분과 잎집부분으로 나뉘어진다. 잎새부분은

0 ~ 2020 / 자료문의처 : 042-481-2545 / 기능문의: KOSIS Q&A 게시판

그림 2

시도별 [3/18]

시점 [4/41]

주소정보

행렬전환

분석

2019						
(단위)	파:면적 (ha)	생산량 (톤)	노지파:면적 (ha)	10a당 생산량 (kg)	생산량 (톤)	시설파:면적 (ha)
	▲ ▼ ▢	▲ ▼ ▢	▲ ▼ ▢	▲ ▼ ▢	▲ ▼ ▢	▲ ▼ ▢
0	547	14,481	491	2,695	13,222	
0	232	4,875	223	2,046	4,571	
0	4,750	163,375	4,642	3,481	161,613	

그림 1

그림 1. 파가 많이 나오는 지역 조사

그림 2. 어느 곳에서 정확히 많이 나오는지 확인

-> 진도군에 파가 압도적으로 많이 나오는 것을 확인

-> 진도군의 날씨를 입력데이터로 사용



데이터 수집

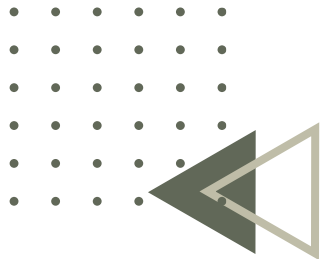


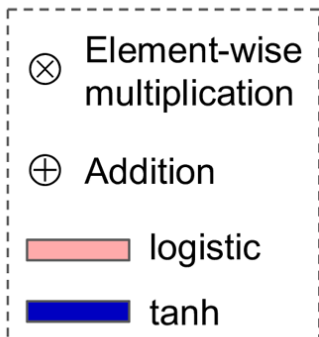
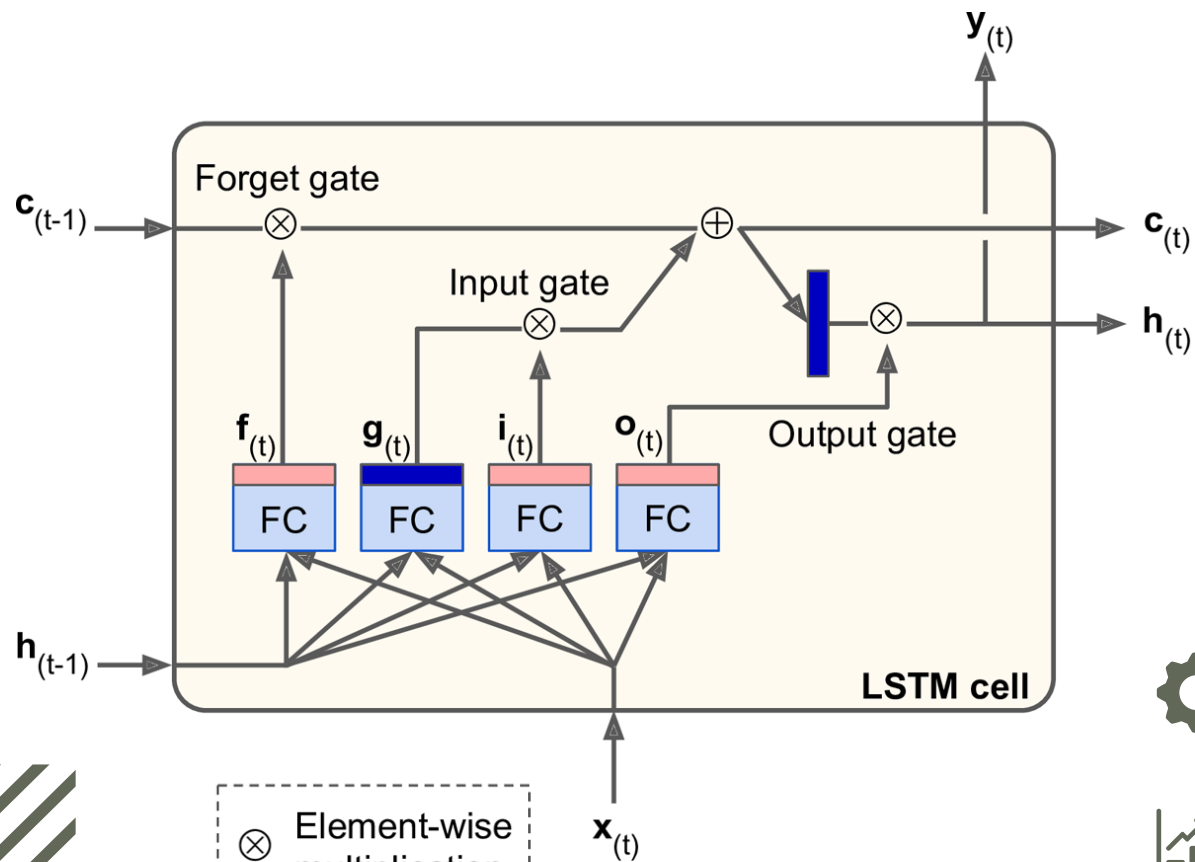
서울시농수산물공사 – 주요 품목 가격 open API

<https://www.garak.co.kr/publicdata/selectPageListPublicData.do>

기상자료개방포털 – 농산물 재배지의 평균기온, 일강수량

<https://data.kma.go.kr/data/grnd/selectAsosRltmList.do?pgmNo=36>





LSTM

사용할 딥러닝 모델



RNN input의 순서와 지속성을 고려한 모델



장기 의존성을 학습할 수 있는 RNN의 한 종류



오랫동안 정보를 기억하는 기능을 가지고 있음

- Forget Gate Layer: 어떤 정보를 유지할지, 버릴지
- 어떤 새로운 정보를 cell state에 저장할지
- 새로운 cell state로 갱신하여 무엇을 출력할지 결정

Open API 데이터 확인

- 대파를 검색했을 때 화면
제품의 상태 (특, 상, 중, 하), 몇 키로 인지
오늘 가격, 어제 가격, 전년 가격 출력
- 체크 표시: 모델링 하기 위한 데이터 추출
- API 신청을 하고 아래 쿼리스트링에 필요한
정보 입력
[https://www.garak.co.kr/publicdata/dataOpen.do?id=아이디&passwd=비밀번호&dataid=data4&pagesize=10&pageidx=1&portaltemplet=false&p_ymd=검색일자\(8자리\)&p_jymd=전일일자\(8자리\)&d_cd=2는 청과 3은 수산&p_jjymd=전년도날짜\(8자리\)&p_pos_gubun=9는 도매가격&pum_nm=품목명](https://www.garak.co.kr/publicdata/dataOpen.do?id=아이디&passwd=비밀번호&dataid=data4&pagesize=10&pageidx=1&portaltemplet=false&p_ymd=검색일자(8자리)&p_jymd=전일일자(8자리)&d_cd=2는 청과 3은 수산&p_jjymd=전년도날짜(8자리)&p_pos_gubun=9는 도매가격&pum_nm=품목명)

```
<PUM_NM_A>
  <![CDATA[ 대파(일반) ]]>
</PUM_NM_A>
<PUM_CD>
  <![CDATA[ 24501 ]]>
</PUM_CD>
<G_NAME_A>
  <![CDATA[ 특 ]]>
</G_NAME_A>
<UNIT_QTY>
  <![CDATA[ 1 ]]>
</UNIT_QTY>
<U_NAME>
  <![CDATA[ 1키로단 ]]>
</U_NAME>
<AV_P_A>
  <![CDATA[ 3601 ]]>
</AV_P_A>
<PAV_P_A>
  <![CDATA[ 3662 ]]>
</PAV_P_A>
<PAV_PY_A>
  <![CDATA[ 1868 ]]>
</PAV_PY_A>
<A_B>
  <![CDATA[ 98.3342435827416712179137083560895685418 ]]>
</A_B>
<A_C>
  <![CDATA[ 192.773019271948608137044967880085653105 ]]>
</A_C>
<E_NAME>
  <![CDATA[ 특 ]]>
</E_NAME>
<F_NAME>
  <![CDATA[ 키로단 ]]>
</F_NAME>
<GRADE_CD>
  <![CDATA[ 0 ]]>
</GRADE_CD>
<UNIT_CD>
  <![CDATA[ 29 ]]>
</UNIT_CD>
</list>
<list>
  <PUM_NM_A>
    <![CDATA[ 대파(일반) ]]>
  </PUM_NM_A>
  <PUM_CD>
```

Import urllib.request

```
def query_sender(date, date_1, last_year, kinds):
    url='http://www.garak.co.kr/publicdata/data0pen.do?id=2956&passwd=qwert1324!&' \
        'dataid=data4&pagesize=1000&pageidx=1&portal.template=false&'

    kinds = urllib.parse.quote(kinds)
    code_info='p_ymd=' + date+'&p_jymd='+date_1+'&d_cd=2&p_jjymd='+last_year
    code_info_2='&p_pos_gubun=9&pum_nm='+kinds

    request = urllib.request.Request(url+code_info+code_info_2)
    request.get_method=lambda : 'GET'
    response_body=urllib.request.urlopen(request).read()
    u=str(response_body, 'utf-8')
    return u
```

쿼리를 보내는 함수

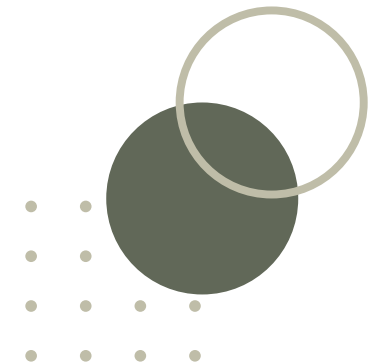
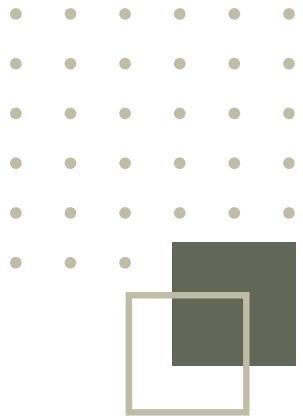
오른쪽 표의 파라미터 타입에 맞춰
알맞은 정보를 입력하고, 쿼리를 보냄

파라미터명	타입	필수여부	내용	비고
id	String	Y	승인시 자동생성	수정불가
passwd	String	Y	승인시 자동생성(인증번호)	수정불가
p_ymd	String	Y	검색일자	8자리
p_jymd	String	Y	전일날짜	8자리
d_cd	String	Y	부류구분	청과:2, 수산:3
p_jjymd	String	Y	전년도날짜	8자리
p_pos_gubun	String	Y	시장구분	가락시장:1, 양곡시장:2, 강서시장(경매):3, 강서시장(시장도매인):9
pum_nm	String		품목명	예: 사과,배

xml을 데이터 프레임으로 저장

```
def xml_to_item_list(xml_string, date):
    result=[]
    root = ET.fromstring(xml_string)
    for child in root:
        item_list=[]
        item_list.append(date)
        for list in child:
            if list.tag in ['PUM_NM_A', 'AV_P_A', 'PAV_P_A', 'PAV_PY_A', 'E_NAME']:
                item_list.append(list.text)
                #print(list.text)
        result.append(item_list)

    return result
```



flag=False

for date in pandas.date_range(start='20170101', end='20210527'):

반복문을 date_range로 한 이유는 숫자형으로 하면 20179999 이런 식으로 갈 수 있기 때문이다.

date1: 오늘, date2: 전날, date3: 전년도 오늘

date1 = str(date.strftime("%Y%m%d"))

date2 = str((date-timedelta(days=1)).strftime("%Y%m%d"))

date3 = str((date-timedelta(days=365)).strftime("%Y%m%d"))

query=query_sender(date1, date2, date3, '대파')

i_list=xml_to_item_list(query, date1)

df = pd.DataFrame(i_list)

df=df.drop(df.index[0])

csv파일에 저장할 때 첫번째 열에 column name을 넣어주기 위해 만약 파일이 없으면

첫 번째 파일을 넣어주고 파일이 있으면 그 뒤로는 데이터를 계속 넣어준다.

if not flag:

flag = True

col=['날짜', '품목명', '평균가격', '전일평균가격', '전년가격', '등급']

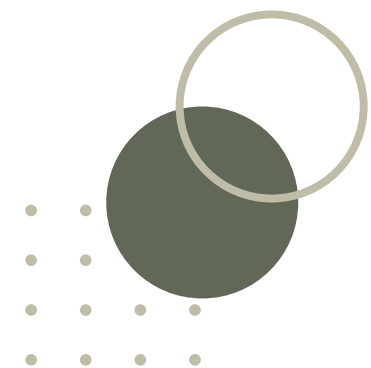
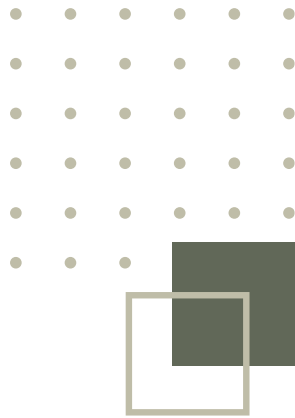
df.columns=col

df.to_csv('./data/test_Gonion.csv',encoding='utf-8-sig')

else:

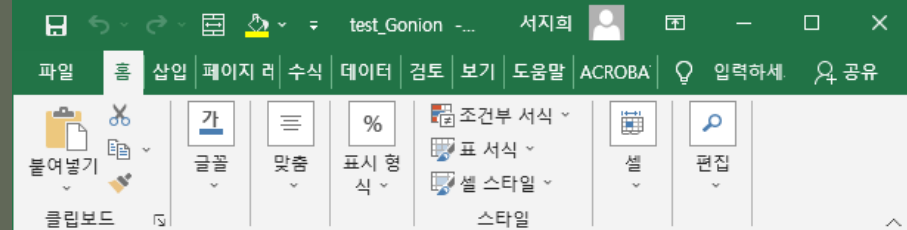
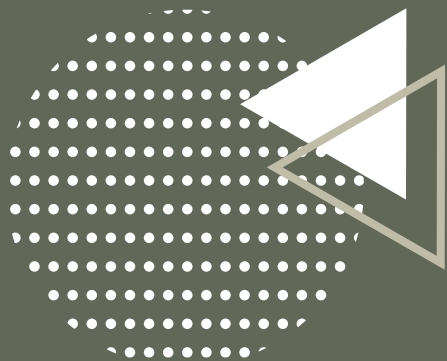
df.to_csv('./data/test_Gonion.csv',encoding='utf-8-sig',mode='a', header=False)

print('끝났다!@@@@@@@@@@@@@@@@@@@@!!!!!!!!!!!!!!!!!!!!!!!!@@@@@@@@@@@@')



데이터 확인

더 수정이 필요해 보이지만 일단 데이터가 잘 들어간 모습을 확인할 수 있음



test_Gonion

	A	B	C	D	E	F	G	H
1		날짜	품목명	평균가격	전일평균가	전년가격	등급	
2	1	20170101	대파(일반)	0	2833	0	특	
3	2	20170101	대파(일반)	0	2164	0	상	
4	3	20170101	대파(일반)	0	1747	0	중	
5	4	20170101	대파(일반)	0	1214	0	하	
6	1	20170102						
7	1	20170103	대파(일반)	2833	0	2761	특	
8	2	20170103	대파(일반)	2164	0	1926	상	
9	3	20170103	대파(일반)	1747	0	1611	중	
10	4	20170103	대파(일반)	1214	0	1434	하	
11	1	20170104	대파(일반)	2797	2833	2957	특	
12	2	20170104	대파(일반)	2100	2164	2126	상	
13	3	20170104	대파(일반)	2000	1747	1797	중	
14	4	20170104	대파(일반)	1838	1214	1532	하	
15	1	20170105	대파(일반)	2797	2797	2621	특	
16	2	20170105	대파(일반)	2100	2100	1848	상	
17	3	20170105	대파(일반)	2000	2000	1707	중	
18	4	20170105	대파(일반)	1838	1838	1498	하	
19	1	20170106	대파(일반)	2797	2797	2599	특	
20	2	20170106	대파(일반)	2100	2100	1865	상	
21	3	20170106	대파(일반)	2000	2000	1566	중	
22	4	20170106	대파(일반)	1838	1838	1419	하	
23	1	20170107	대파(일반)	0	2797	2442	특	
24	2	20170107	대파(일반)	0	2100	1755	상	
25	3	20170107	대파(일반)	0	2000	1529	중	
26	4	20170107	대파(일반)	0	1838	1367	하	
27	1	20170108						
28	1	20170109	대파(일반)	2797	0	0	특	
29	2	20170109	대파(일반)	2100	0	0	상	
30	3	20170109	대파(일반)	2000	0	0	중	
31	4	20170109	대파(일반)	1838	0	0	하	
32	1	20170110	대파(일반)	2527	2797	2278	특	

test_Gonion



데이터 정제



	날짜	품목명	평균가격	전일평균가격	전년가격	등급
1	20170101	대파(일반)	0.0	2833.0	0.0	특
2	20170101	대파(일반)	0.0	2164.0	0.0	상
3	20170101	대파(일반)	0.0	1747.0	0.0	중
4	20170101	대파(일반)	0.0	1214.0	0.0	하
1	20170102	NaN	NaN	NaN	NaN	NaN
...
4	20210526	대파(일반)	1180.0	1392.0	1208.0	하
1	20210527	대파(일반)	3699.0	3601.0	1952.0	특
2	20210527	대파(일반)	1554.0	2152.0	1639.0	상
3	20210527	대파(일반)	1247.0	1572.0	1500.0	중
4	20210527	대파(일반)	1110.0	1180.0	1257.0	하

6390 rows × 6 columns

대파 가격 데이터

	지점	지점명	일시	평균기온(°C)	일강수량(mm)	일 최심신적설(cm)
0	268	진도군	20170101	4.3	NaN	NaN
1	268	진도군	20170102	8.1	0.0	NaN
2	268	진도군	20170103	5.1	NaN	NaN
3	268	진도군	20170104	4.4	NaN	NaN
4	268	진도군	20170105	7.5	0.5	NaN
...
1590	268	진도군	20210523	19.7	0.0	NaN
1591	268	진도군	20210524	17.6	0.0	NaN
1592	268	진도군	20210525	17.3	NaN	NaN
1593	268	진도군	20210526	17.1	0.7	NaN
1594	268	진도군	20210527	16.9	0.6	NaN

1595 rows × 6 columns

기온과 강수량 데이터



날씨 데이터의 NaN 값을 0으로 바꿔준다.

dfw=dfw.fillna(0)

#%%

인덱스 12341234 되어 있는걸 바꿔줌

index = range(df2.shape[0])

df2.index = index

display(df2)

날씨 데이터와 파 가격 데이터를 한 데이터프레임으로 만드는 과정

데이터프레임에 리스트를 붙이려면 한 열이 완전하게 만들어져 있어야됨

한 열을 리스트로 만드는 작업

temp_list = []

rain_list = []

day_list = []

count = 0

for i1 in df2.index:

 for i2 in dfw.index:

 if df2['날짜'][i1] != dfw['일시'][i2]:

 continue

 temp_list.append(dfw['평균기온(°C)'][i2])

 rain_list.append(dfw['일강수량(mm)'][i2])

 day_list.append(dfw['일시'][i2])

 count +=1

print('temp_list', temp_list)

print(len(temp_list))

print('rain_list', rain_list)

print(len(rain_list))



바로 위에 보면 열이 6390개인데 여기서는 6338개가 나옴!
바로 데이터 확인 함(차집합 이용)

```
dd=np.array(df2['날짜'].tolist())  
dl=np.array(day_list)  
intersection = np.array(list(set(dd)-set(dl)))  
print(intersection)  
print(len(intersection))
```

13*4 해서 누락된 52개의 데이터를 확인함.
엑셀로 확인해 보니 저 날씨 데이터가 누락되어있었음

```
[20170720 20170721 20170722 20190529  
20191202 20191017 20191018 20171116 20171221  
20171222 20171031 20171130 20200702] 13
```

기상청 기록이 13개가 없던 것
그래서 리스트에 매핑하려고 했는데 수가 없어가지고 아마 오류가 났을것이다.
누락된 52개 데이터 삭제
tf=[]

```
for i in df2.index:  
    if df2['날짜'][i] in intersection:  
        df2= df2.drop(index=i, axis=0)
```

df2를 df22로 옮기기
df22= df2.copy()
누락된 데이터가 잘 삭제되었는지 확인
print(len(df22.index))

```
# 누락된 데이터가 잘 삭제되었는지 확인  
print(len(df22.index))
```

6338

6390 rows x 6 columns

```
temp_list [4.3, 4.3, 4.3, 4.3, 8.1, 5.1, 5.1, 5.1, 5.1, 4.4  
6338  
rain_list [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0  
6338
```

200	268	진도군	20170718	
201	268	진도군	20170719	
202	268	진도군	20170723	28.7
203	268	진도군	20170724	29.3



```
# df22에 데이터를 붙여주기  
df22['평균기온'] = temp_list  
df22['강수량'] = rain_list  
display(df22)
```

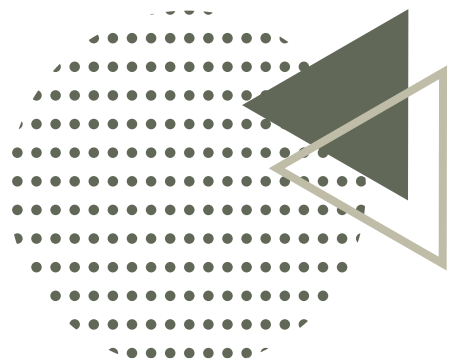
	날짜	품목명	평균가격	전일평균가격	전년가격	등급	평균기온	강수량
0	20170101	대파(일반)	0.0	2833.0	0.0	특	4.3	0.0
1	20170101	대파(일반)	0.0	2164.0	0.0	상	4.3	0.0
2	20170101	대파(일반)	0.0	1747.0	0.0	중	4.3	0.0
3	20170101	대파(일반)	0.0	1214.0	0.0	하	4.3	0.0
4	20170102	NaN	NaN	NaN	NaN	NaN	8.1	0.0
...
6385	20210526	대파(일반)	1180.0	1392.0	1208.0	하	17.1	0.7
6386	20210527	대파(일반)	3699.0	3601.0	1952.0	특	16.9	0.6
6387	20210527	대파(일반)	1554.0	2152.0	1639.0	상	16.9	0.6
6388	20210527	대파(일반)	1247.0	1572.0	1500.0	중	16.9	0.6
6389	20210527	대파(일반)	1110.0	1180.0	1257.0	하	16.9	0.6

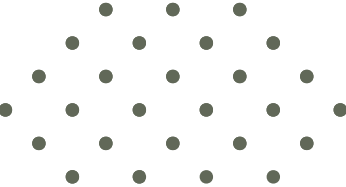
6338 rows × 8 columns



03

데이터 전처리





```
In [16]: df22.shape
```

(6338, 8)

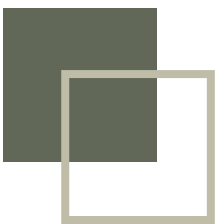
```
In [17]: # 누락된 데이터 삭제  
df22.isna().sum()
```

```
날짜      0  
품목명    14  
평균가격  14  
전일평균가격  14  
전년가격  14  
등급      14  
평균기온   0  
강수량     0  
dtype: int64
```

```
In [19]: df22 = df22.dropna()  
display(df22)
```

	날짜	품목명	평균가격	전일평균가격	전년가격	등급	평균기온	강수량
0	20170101	대파(일반)	0.0	2833.0	0.0	특	4.3	0.0
1	20170101	대파(일반)	0.0	2164.0	0.0	상	4.3	0.0
2	20170101	대파(일반)	0.0	1747.0	0.0	중	4.3	0.0
3	20170101	대파(일반)	0.0	1214.0	0.0	하	4.3	0.0
5	20170103	대파(일반)	2833.0	0.0	2761.0	특	5.1	0.0
...
6385	20210526	대파(일반)	1180.0	1392.0	1208.0	하	17.1	0.7
6386	20210527	대파(일반)	3699.0	3601.0	1952.0	특	16.9	0.6
6387	20210527	대파(일반)	1554.0	2152.0	1639.0	상	16.9	0.6
6388	20210527	대파(일반)	1247.0	1572.0	1500.0	중	16.9	0.6
6389	20210527	대파(일반)	1110.0	1180.0	1257.0	하	16.9	0.6

6324 rows × 8 columns



```
In [20]: # 데이터 정보 확인
df22.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6324 entries, 0 to 6389
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   날짜        6324 non-null   int64
 1   품목명      6324 non-null   object
 2   평균가격    6324 non-null   float64
 3   전일평균가격 6324 non-null   float64
 4   전년가격    6324 non-null   float64
 5   등급        6324 non-null   object
 6   평균기온    6324 non-null   float64
 7   강수량      6324 non-null   float64
dtypes: float64(5), int64(1), object(2)
memory usage: 444.7+ KB
```

```
In [21]: df22.describe()
```

	날짜	평균가격	전일평균가격	전년가격	평균기온	강수량
count	6.324000e+03	6324.000000	6324.000000	6324.000000	6324.000000	6324.000000
mean	2.018813e+07	1581.423941	1582.323371	1334.657337	13.753574	3.705187
std	1.277048e+04	1106.623326	1105.753259	857.593694	8.481000	15.402556
min	2.017010e+07	0.000000	0.000000	0.000000	-8.200000	0.000000
25%	2.018022e+07	946.000000	946.750000	866.000000	6.700000	0.000000
50%	2.019032e+07	1423.000000	1425.000000	1329.500000	13.700000	0.000000
75%	2.020043e+07	2072.500000	2074.000000	1862.000000	21.000000	0.400000
max	2.021053e+07	10749.000000	10749.000000	10749.000000	29.500000	305.000000



: 등급에 따른 그룹화

- # 가격 데이터를 보면 하루에 특, 상, 중, 하 4개로 나누어져 있으므로,
- # 이 4개의 데이터가 섞이지 않도록 하기 위해 groupby로 등급별로 나눠준다.
- `print('## groupby (등급) ##')`
- `grouped = df22.groupby('등급')`

#그룹 확인

```
for name, group in grouped:  
    print('groupname: ', name)  
    print('groupdata')  
    print(group)
```

#그룹 추출

```
adf = grouped.get_group(name='특')  
bdf = grouped.get_group(name='상')  
cdf = grouped.get_group(name='중')  
ddf = grouped.get_group(name='하')
```

groupby (등급)

groupname: 상

groupdata

	날짜	품목명	평균가격	전일평균가격	전년가격	등급	평균기온	강수량
6	20170103	대파(일반)	2164.0	0.0	1926.0	상	5.1	0.0
10	20170104	대파(일반)	2100.0	2164.0	2126.0	상	4.4	0.0
14	20170105	대파(일반)	2100.0	2100.0	1848.0	상	7.5	0.5
18	20170106	대파(일반)	2100.0	2100.0	1865.0	상	7.2	0.2
27	20170109	대파(일반)	2100.0	0.0	0.0	상	5.6	0.0
...
6367	20210522	대파(일반)	1883.0	2020.0	1556.0	상	17.9	0.0
6375	20210524	대파(일반)	1804.0	0.0	0.0	상	17.6	0.0
6379	20210525	대파(일반)	2153.0	1804.0	1614.0	상	17.3	0.0
6383	20210526	대파(일반)	2152.0	2153.0	1623.0	상	17.1	0.7
6387	20210527	대파(일반)	1554.0	2152.0	1639.0	상	16.9	0.6

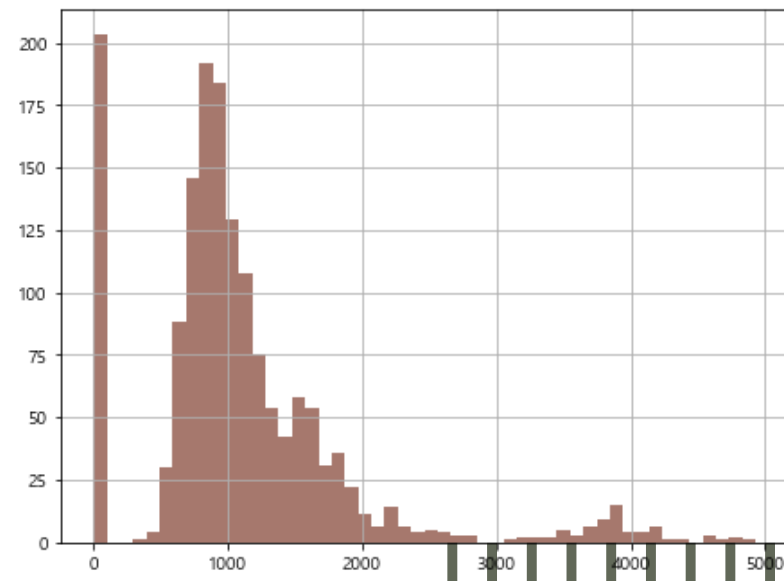
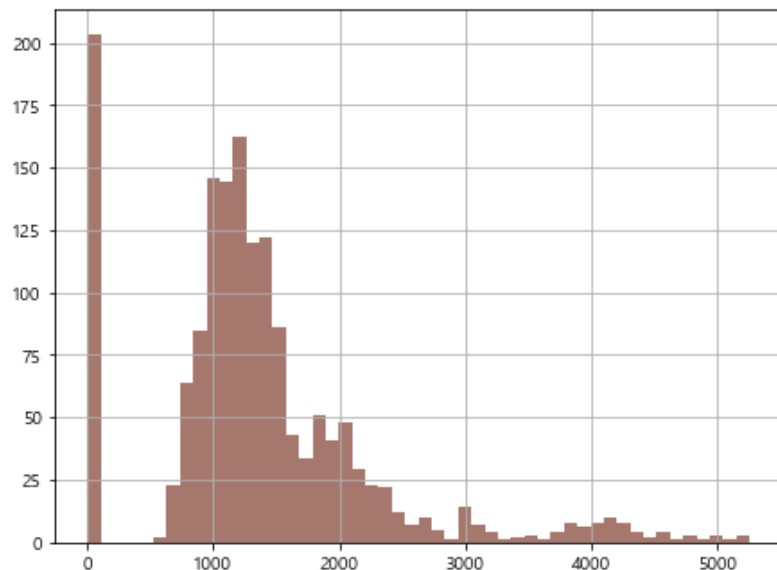
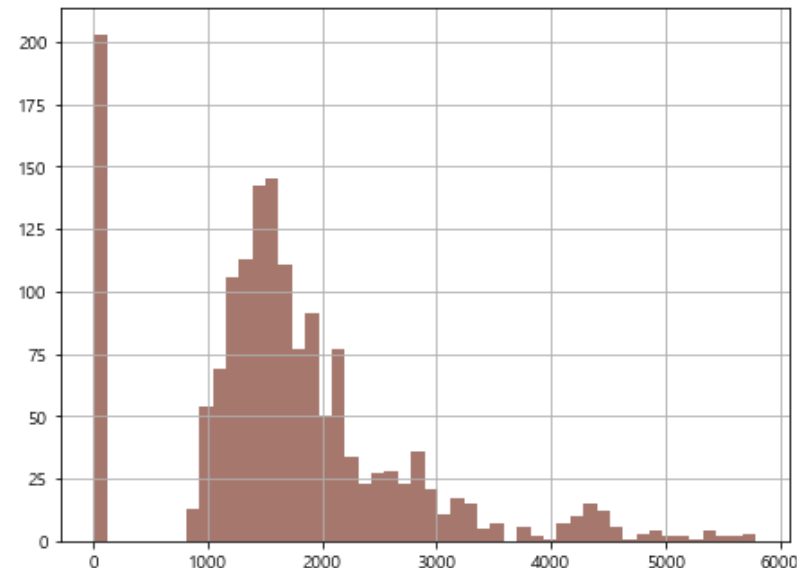
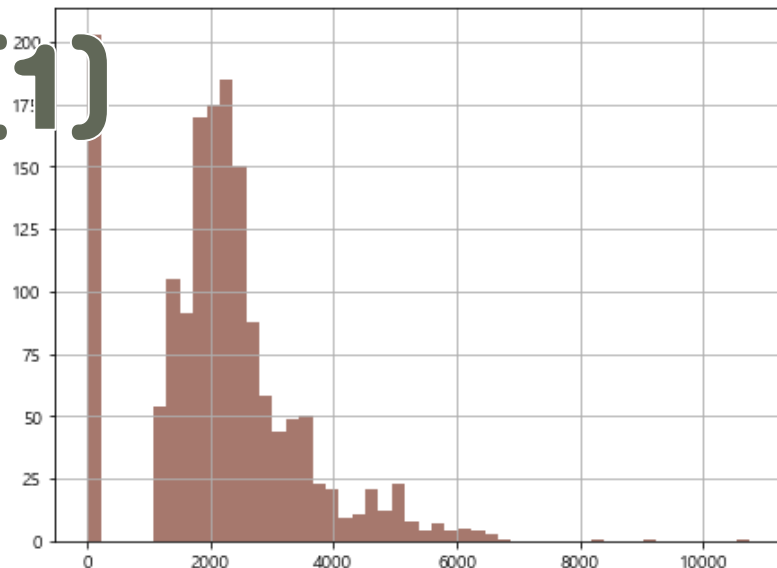
데이터 확인 (1)

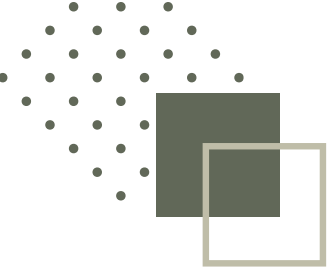
히스토그램

평균 가격 시각화

0인 데이터가 많은
것을 확인

결측값 수정





```
# 가격 데이터가 없는 곳에는 0이라 되어 있으므로 0 -> NaN
# 나눈 상태에서 NaN 값에 평균을 넣어 줘야 하는구나,,,
def zeroToNaN (list):
    list['평균가격'] = list['평균가격'].replace(0, np.NaN)
    display('평균가격 0.0 -> NaN',list.head())

    list = list.fillna(list.mean())
    list = list.round(1)
    display('평균가격 NaN -> 열의 평균',list.head())
```

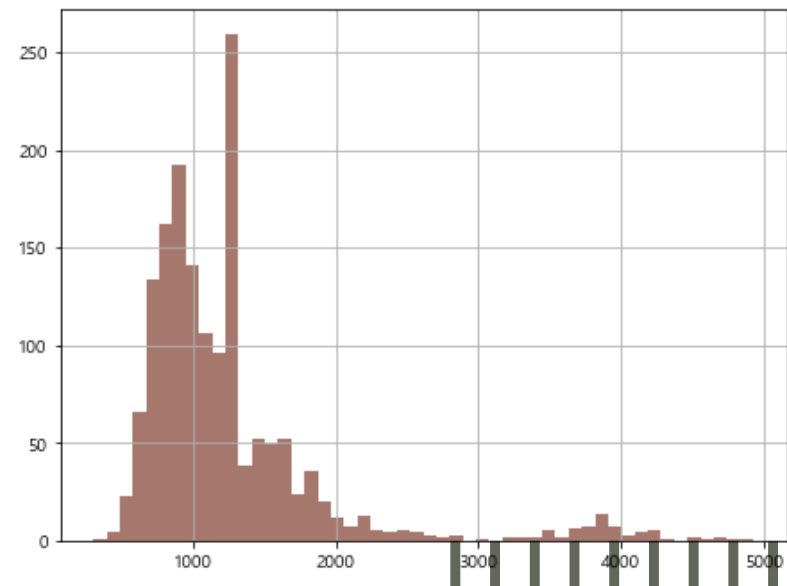
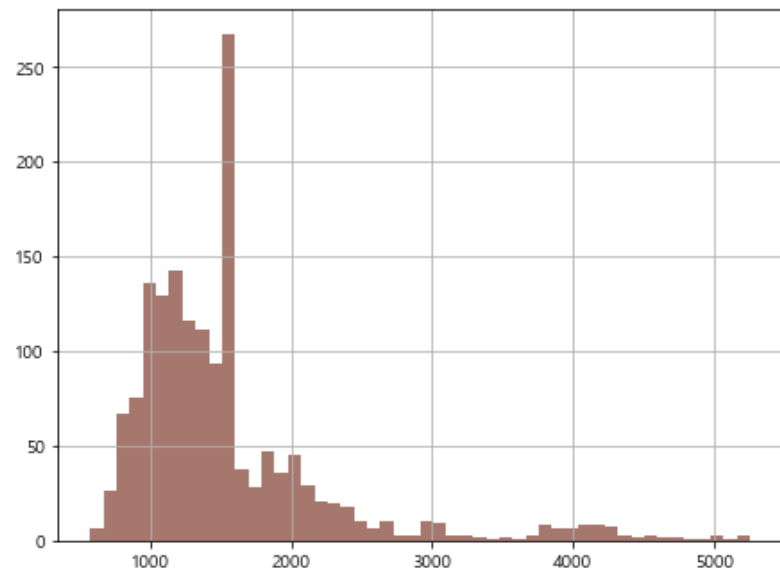
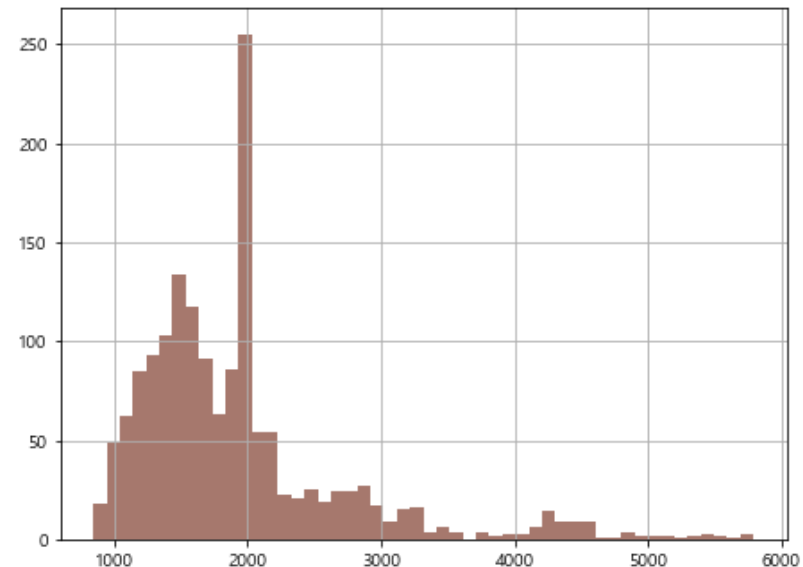
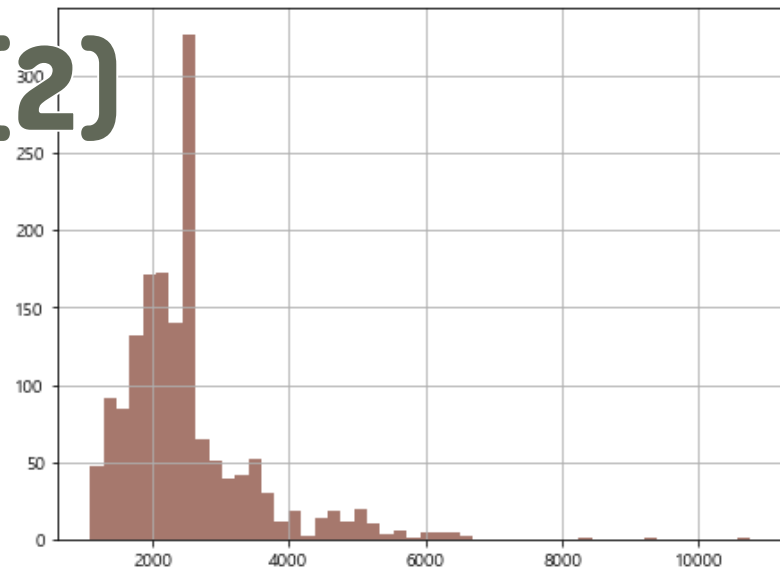
```
zeroToNaN(adf)
zeroToNaN(bdf)
zeroToNaN(cdf)
zeroToNaN(ddf)
```

```
plot_price(adf, bdf,cdf,ddf)
```



데이터 확인 (2)

히스토그램
수정된 데이터
확인



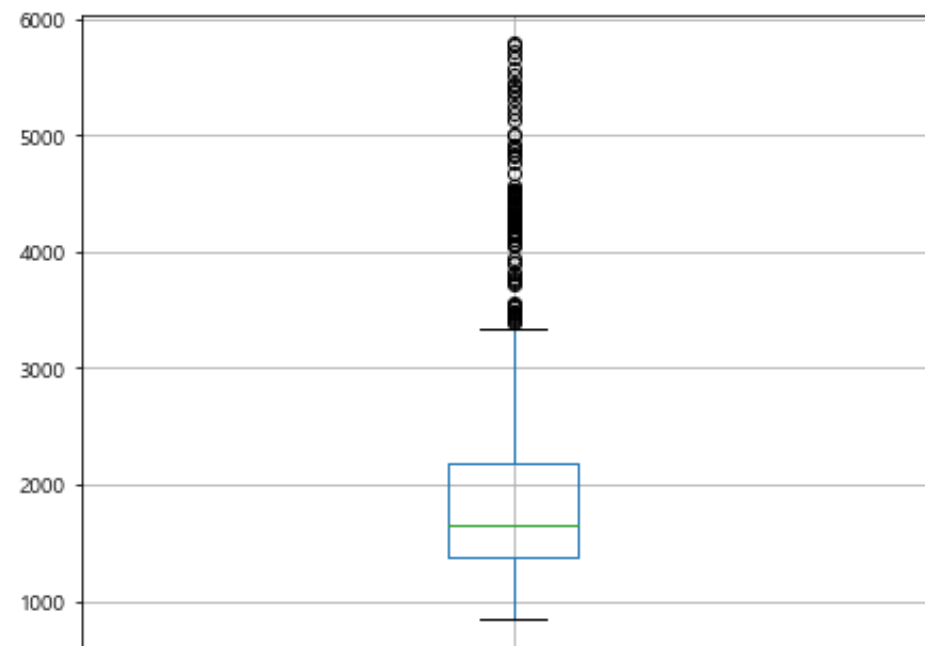
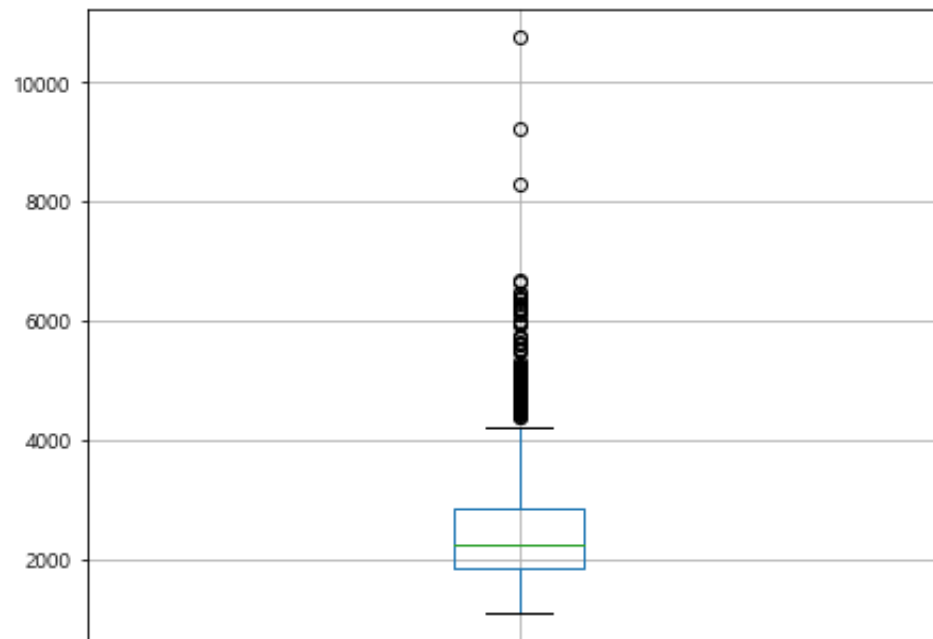
데이터 확인 (3)

Box Plot

초록선: 중앙값

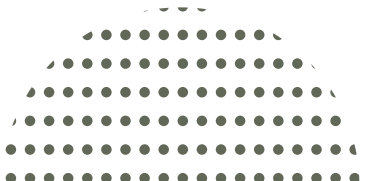
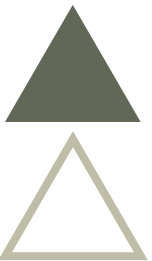
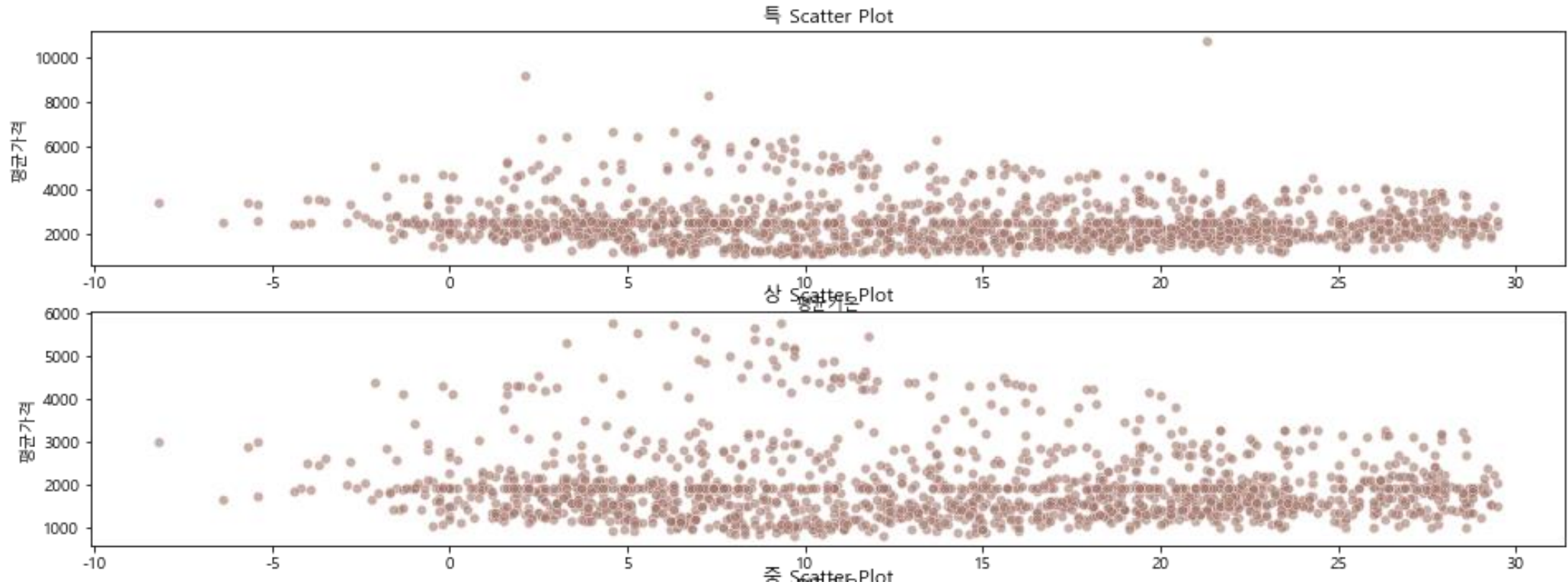
사각형: 1~3 사분위 수

선 끝: 최대, 최소

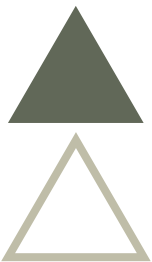
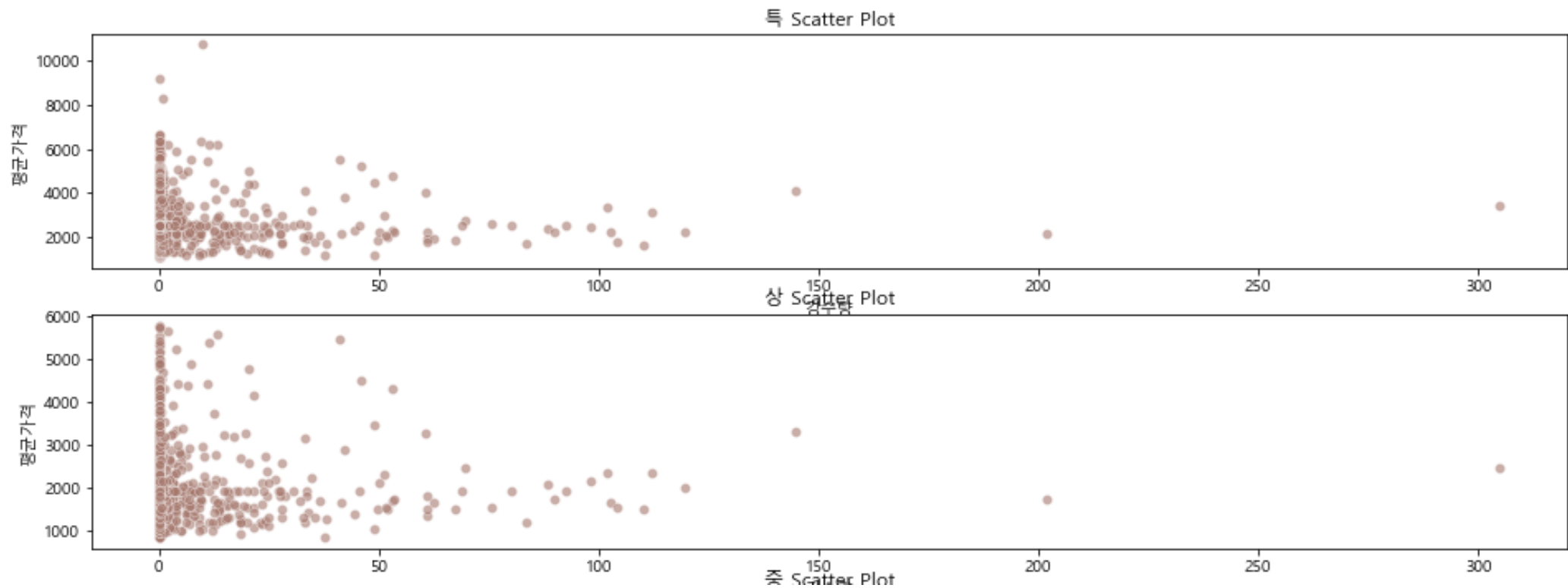




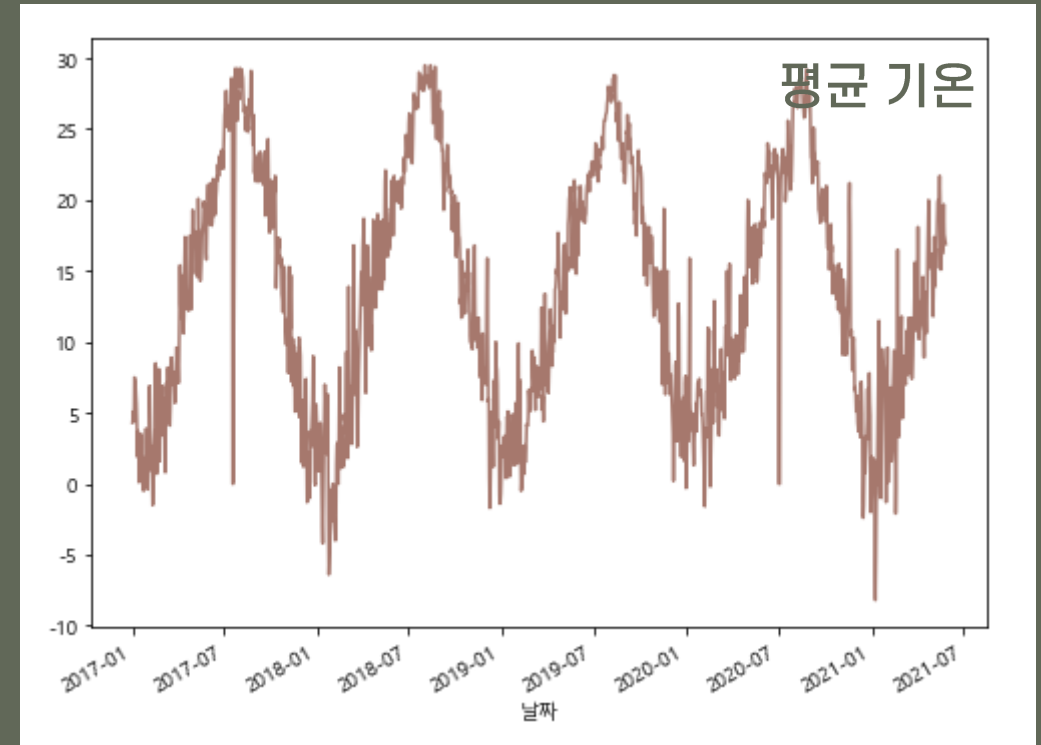
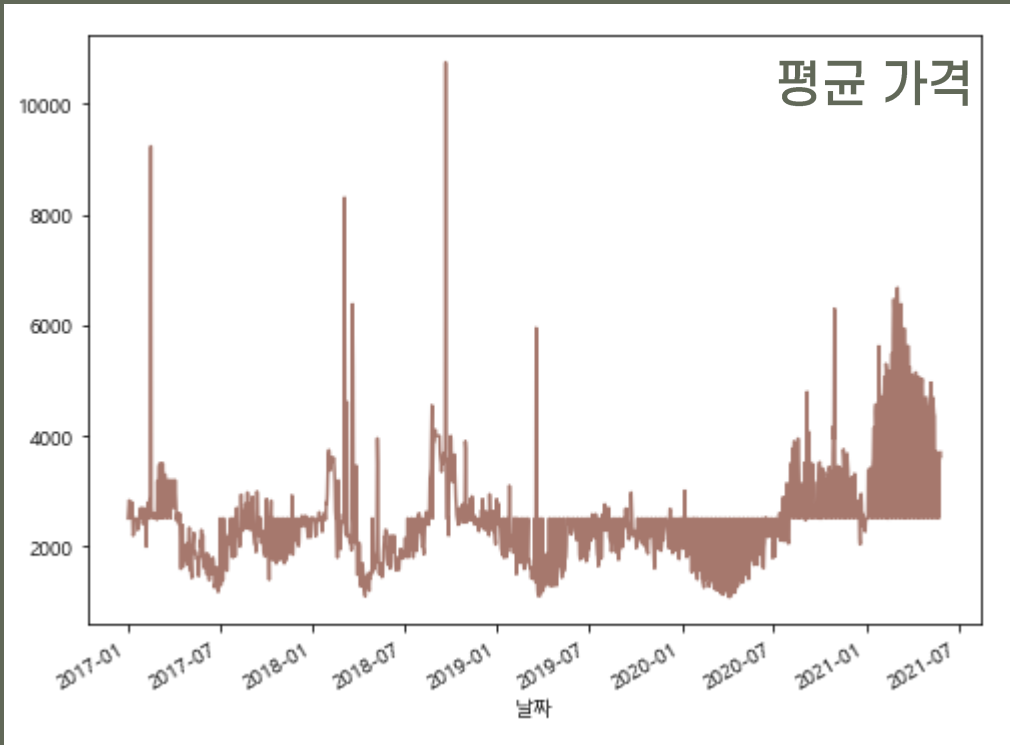
평균 기온과 평균 가격 사이의 관계 확인



강수량과 평균 가격 사이의 관계 확인



입력 데이터

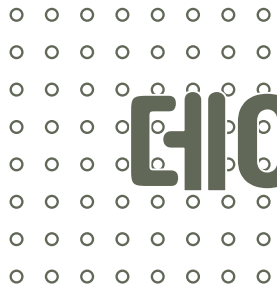




04

모델링





데이터 정규화

```
from sklearn.preprocessing import MinMaxScaler
```

```
df.sort_index(ascending=False).reset_index(drop=True)
```

```
scaler1 = MinMaxScaler()
scaler2 = MinMaxScaler()
scaler3 = MinMaxScaler()
scale_cols=['평균가격', '평균기온', '강수량']
dfs1 = scaler1.fit_transform(df[['평균가격']])
dfs2 = scaler2.fit_transform(df[['평균기온']])
dfs3 = scaler3.fit_transform(df[['강수량']])
```

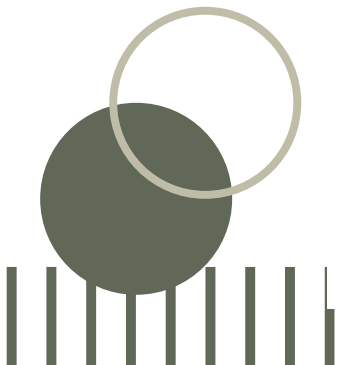
```
df_scaled = pd.DataFrame()
df_scaled=pd.DataFrame(dfs1)
df_scaled.columns=['평균가격']
df_scaled['평균기온'] = pd.DataFrame(dfs2)
df_scaled['강수량'] = pd.DataFrame(dfs3)
```

```
df_scaled.columns = scale_cols
```

```
print(df_scaled)
```

가격과 기온, 강수량이 모두 다르므로 각각 정규화 함수를 적용함

	평균가격	평균기온	강수량
0	0.147325	0.331565	0.000000
1	0.180793	0.352785	0.000000
2	0.177067	0.334218	0.000000
3	0.177067	0.416446	0.001639
4	0.177067	0.408488	0.000656
...
1576	0.147325	0.740053	0.000000
1577	0.202939	0.684350	0.000000
1578	0.266584	0.676393	0.000000
1579	0.260271	0.671088	0.002295
1580	0.270413	0.665782	0.001967
[1581 rows x 3 columns]			





시계열 데이터셋 분리 (1)

```
TEST_SIZE = 200  
WINDOW_SIZE = 20
```

```
train = df_scaled[: -TEST_SIZE]  
test = df_scaled[-TEST_SIZE:]
```

```
#  
def make_dataset(data, label, window_size = 20):  
    feature_list = []  
    label_list = []  
    for i in range(len(data) - window_size):  
  
        feature_list.append(np.array(data.iloc[i:i+window_size]))  
  
        label_list.append(np.array(label.iloc[i+window_size]))  
    return np.array(feature_list), np.array(label_list)
```

```
from sklearn.model_selection import train_test_split
```

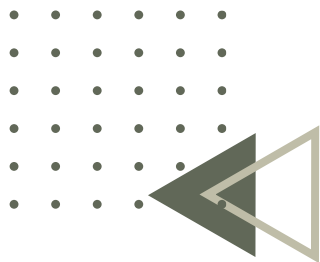
```
feature_cols = ['평균기온', '강수량']  
label_cols = ['평균가격']
```

```
train_feature = train[feature_cols]  
train_label = train[label_cols]
```

```
train_feature, train_label =  
make_dataset(train_feature, train_label, 20)
```

```
x_train, x_valid, y_train, y_valid =  
train_test_split(train_feature, train_label, test_size=0.2)  
print(x_train.shape, x_valid.shape)
```

```
(1088, 20, 2) (273, 20, 2)
```





시계열 데이터셋 분리 (2)

```
test_feature = test[feature_cols]
test_label = test[label_cols]
```

```
test_feature.shape, test_label.shape
```

###

```
test_feature, test_label = make_dataset(test_feature, test_label, 20)
test_feature.shape, test_label.shape
```

window size에 맞춰서 3차원 데이터로 바꿔줌

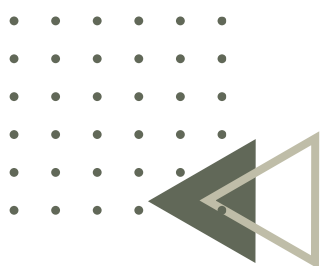
```
215 test_feature = test[feature_cols]
    test_label = test[label_cols]

    test_feature.shape, test_label.shape
```

```
215 ((200, 2), (200, 1))
```

```
216 test_feature, test_label = make_dataset(test_feature, test_label, 20)
    test_feature.shape, test_label.shape
```

```
216 ((180, 20, 2), (180, 1))
```



모델 학습 (1)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import LSTM
import tensorflow as tf
```

```
tf.random.set_seed(715)
```

```
###
```

```
lstmodel = Sequential()
lstmodel.add(LSTM(16,
                  input_shape=(train_feature.shape[1],
                                train_feature.shape[2]),
                  activation='relu',
                  return_sequences=False))
```

```
lstmodel.add(Dense(1))
```

```
###
```

```
print(train_feature.shape[1])
print(train_feature.shape[2])
```

```
###
```

```
lstmodel.compile(loss='mean_squared_logarithmic_error',
                 optimizer='adam')
early_stop = EarlyStopping(monitor='val_loss', patience=5)
```

```
history = lstmodel.fit(x_train, y_train,
                       epochs=200,
                       batch_size=16,
                       validation_data=(x_valid, y_valid),
                       callbacks=early_stop)
```

모델 학습 (2)

train loss: 0.0030

validation loss: 0.0044

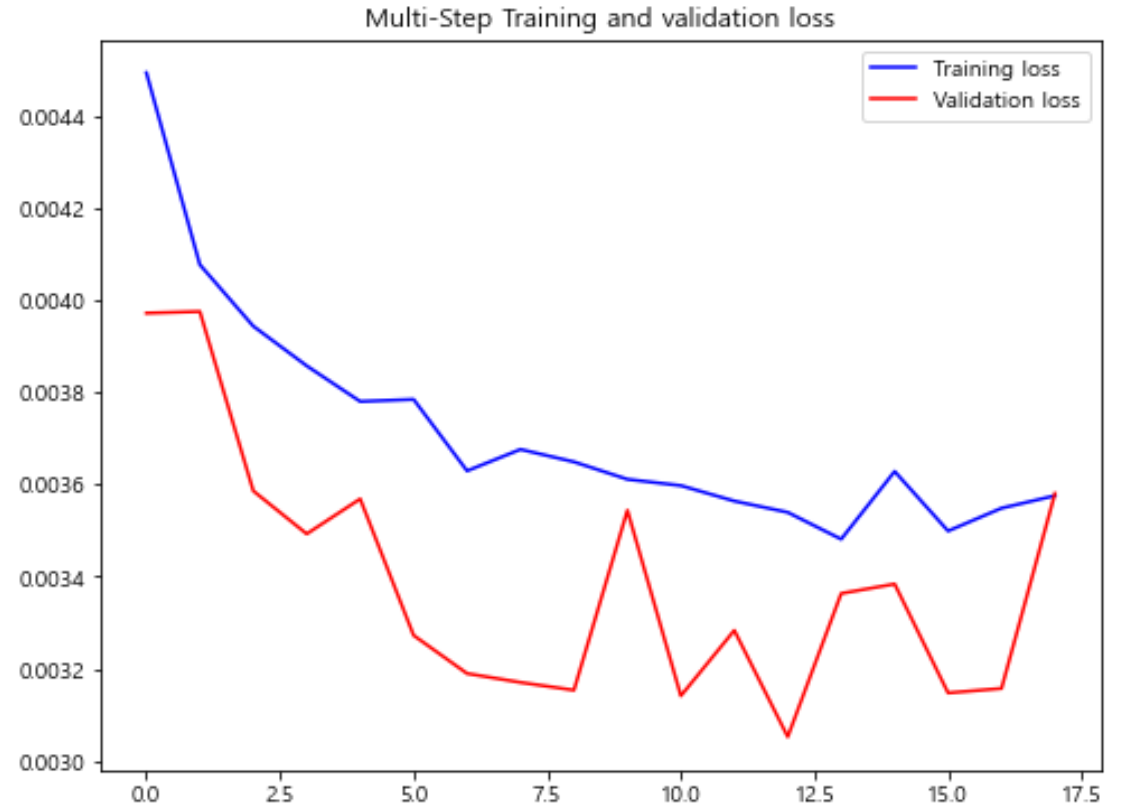
Early Stopping으로 모델
이 일찍 종료됨

```
Epoch 1/200
68/68 [=====] - 1s 5ms/step - loss: 0.0040 - val_loss: 0.0049
Epoch 2/200
68/68 [=====] - 0s 3ms/step - loss: 0.0032 - val_loss: 0.0047
Epoch 3/200
68/68 [=====] - 0s 3ms/step - loss: 0.0038 - val_loss: 0.0044
Epoch 4/200
68/68 [=====] - 0s 3ms/step - loss: 0.0046 - val_loss: 0.0043
Epoch 5/200
68/68 [=====] - 0s 3ms/step - loss: 0.0035 - val_loss: 0.0048
Epoch 6/200
68/68 [=====] - 0s 3ms/step - loss: 0.0035 - val_loss: 0.0043
Epoch 7/200
68/68 [=====] - 0s 3ms/step - loss: 0.0031 - val_loss: 0.0043
Epoch 8/200
68/68 [=====] - 0s 3ms/step - loss: 0.0033 - val_loss: 0.0042
Epoch 9/200
68/68 [=====] - 0s 3ms/step - loss: 0.0031 - val_loss: 0.0042
Epoch 10/200
68/68 [=====] - 0s 3ms/step - loss: 0.0030 - val_loss: 0.0042
Epoch 11/200
68/68 [=====] - 0s 3ms/step - loss: 0.0031 - val_loss: 0.0043
Epoch 12/200
68/68 [=====] - 0s 3ms/step - loss: 0.0029 - val_loss: 0.0042
Epoch 13/200
68/68 [=====] - 0s 3ms/step - loss: 0.0035 - val_loss: 0.0042
Epoch 14/200
68/68 [=====] - 0s 3ms/step - loss: 0.0030 - val_loss: 0.0041
Epoch 15/200
68/68 [=====] - 0s 3ms/step - loss: 0.0030 - val_loss: 0.0042
Epoch 16/200
68/68 [=====] - 0s 3ms/step - loss: 0.0028 - val_loss: 0.0042
Epoch 17/200
68/68 [=====] - 0s 3ms/step - loss: 0.0031 - val_loss: 0.0041
Epoch 18/200
68/68 [=====] - 0s 3ms/step - loss: 0.0032 - val_loss: 0.0041
Epoch 19/200
68/68 [=====] - 0s 3ms/step - loss: 0.0030 - val_loss: 0.0044
```



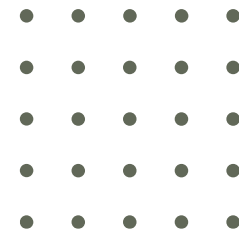
: loss 값 출력

```
def plot_train_history(history, title):  
    loss = history.history['loss']  
    val_loss = history.history['val_loss']  
  
    epochs = range(len(loss))  
  
    plt.figure()  
  
    plt.plot(epochs, loss, 'b', label = 'Training loss')  
    plt.plot(epochs, val_loss, 'r', label = 'Validation loss')  
    plt.title(title)  
    plt.legend()  
  
    plt.show()  
  
plot_train_history(history, 'Multi-Step Training and validation loss')
```



테스트 (1)

최근 날씨 정보를 이용하여 input 데이터 입력하여 예측



```
test = pd.read_csv('./data/testdata2.csv',  
encoding='CP949')  
display(test)
```

```
# 날씨 데이터의 NaN 값을 0으로 바꿔준다.  
test=test.fillna(0)
```

```
###
```

```
# 정규화
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
test.sort_index(ascending=False).reset_index(drop=True)
```

```
scaler = MinMaxScaler()  
scale_cols=['평균기온(°C)', '일강수량(mm)']  
test_scaled=scaler.fit_transform(test[scale_cols])  
test_scaled = pd.DataFrame(test_scaled)  
test_scaled.columns = scale_cols
```

```
display(test_scaled)
```

```
feature_list = []
```

```
for i in range(len(test_scaled) - 20):
```

```
feature_list.append(np.array(test_scaled.iloc[i:i+20]))
```

```
test_input = np.array(feature_list)  
display(test_input)
```

```
###
```

```
test_pred = lstmmodel.predict(test_input)  
test_pred.shape
```

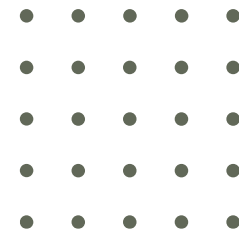
```
###
```

```
result = scaler1.inverse_transform(test_pred)  
print(result)
```



테스트 (2)

최근 날씨 정보를 이용하여 input 데이터 입력하여 예측



```
flag=False
# 여기로 쿼리를 불러와서 확인하기!
for date in pandas.date_range(start='20210526',
end='20210606'):

    date1 = str(date.strftime("%Y%m%d"))
    date2 = str((date-timedelta(days=1)).strftime("%Y%m%d"))
    date3 = str((date-
timedelta(days=365)).strftime("%Y%m%d"))
    query1=query_sender(date1,
                        date2,
                        date3,
                        '대파')

    i_list1=xml_to_item_list(query1, date1)
    check = pd.DataFrame(i_list1)
    check=check.drop(check.index[0])
    if not flag:
        flag = True
        col=['날짜', '품목명', '평균가격', '전일평균가격',
'전년가격', '등급']
        check.columns=col
        check.to_csv('./data/check.csv',encoding='utf-8-
```

```
sig')
    else:
        check.to_csv('./data/check.csv',encoding='utf-8-
sig',mode='a', header=False)

    check = pd.read_csv('./data/check.csv', index_col=0)

    ##

    print('## groupby (등급) ##')
    cgrouped = check.groupby('등급')

    #그룹 확인
    result_g = cgrouped.get_group(name='특')
    result_g['결과'] = result
    display(result_g)
```



	날짜	품목명	평균가격	전일평균가격	전년가격	등급	결과
1	20210526	대파(일반)	3601	3662	1868	특	3743.333252
1	20210527	대파(일반)	3699	3601	1952	특	3608.868164
1	20210528	대파(일반)	3755	3699	1992	특	3509.707275
1	20210529	대파(일반)	2317	3755	1769	특	3355.578369
1	20210530	대파(일반)	0	2317	1672	특	3248.082275
1	20210531	대파(일반)	2202	0	0	특	3101.629639
1	20210601	대파(일반)	2892	2202	2044	특	2964.048584
1	20210602	대파(일반)	2837	2892	1992	특	2832.143555
1	20210603	대파(일반)	2355	2837	1855	특	2711.469482
1	20210604	대파(일반)	2268	2355	1865	특	3058.038330
1	20210605	대파(일반)	1785	2268	1974	특	2985.660400
1	20210606	대파(일반)	0	1785	1930	특	2717.155518

전반적으로 결과가 잘 나오지만 가격 데이터가 없는 날(일요일)을 기준으로 가격 차이가 많이 나는 모습을 확인할 수 있음

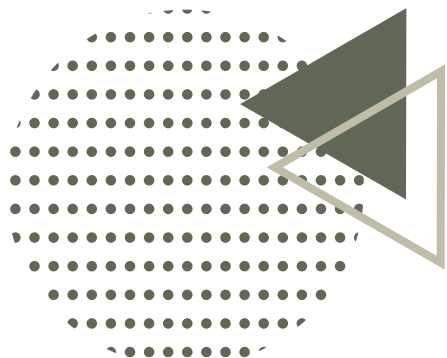
최신 날짜일수록 정확도가 많이 떨어짐

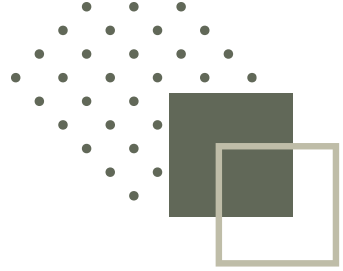
결과 해석



05

소감 및 출처





결과 및 소감

전체적으로 데이터를 수집하고 정제하는데 시간이 너무 오래걸렸습니다.

쿼리스트링과 쿼리스트링의 반환인 XML을 몰라 데이터를 이해하는 데에도 시간이 오래 걸렸습니다.

항상 정리된 데이터만 사용하다가 직접 데이터를 정리해 보니 내가 필요한 데이터가 왜 많이 없는지 알게 되었습니다.

여러 농산물을 측정하고 싶었는데 대파 하나 만으로 이렇게 시간이 오래 걸릴 줄 몰랐습니다. 시험이 끝난다면 다른 데이터도 학습시켜 예측해 보고 싶습니다.

01.

쿼리문에 다른 농산물 정보를 넣고 충분히 데이터를 정제한다면 이 모델을 대파 뿐만 아니라 다른 농산물 및 수산물 가격 예측에 이용 할 수 있을 것 같습니다.

03.

가격 정보가 소매 가격이 아닌 도매 가격 정보로 모델을 만든 점이 아쉽습니다.

데이터 수집을 할 때 소개 가격을 알 수 있는 KAMIS 사이트 접속이 잘 안 되서 도매가격으로 했는데, 실질적인 도움이 안되는 것 같아 아쉬웠습니다.

02.

아쉬운 점은 데이터가 5년밖에 제공이 안되고, 그 중에서도 누락된 데이터가 많아 학습 시킬 수 있는 데이터가 얼마 없는 점입니다.

lstm layer를 쌓을 때 한 층으로 모델을 만들었는데, 한층을 더 쌓으려고 하면 바로 과적합 되어버리는 문제가 발생했었습니다.

04.

입력 데이터로 진도군의 날씨만 넣은 것이 아쉽다.

자신이 사는 곳에서 농산물의 소비가 이뤄지는데, 내가 사는 곳의 날씨가 아닌 재배지의 날씨만을 본 것이 아쉽습니다.



참고자료

쿼리스트링, 파라미터

<https://ysoh.tistory.com/entry/Query-String>

GET, POST의 차이

<https://ysoh.tistory.com/entry/Query-String>

XML API 구조

<https://docs.python.org/ko/3/library/xml.etree.elementtree.html#module-xml.etree.ElementTree>

API to CSV

<https://superelement.tistory.com/16?category=711682>

To_csv 인코딩 문제

<https://zel0rd.tistory.com/50>

날짜 반복문

<https://hashcode.co.kr/questions/3118/python-%EC%9D%BC%EC%A0%95-%EA%B8%B0%EA%B0%84%EC%9D%98-%EB%82%A0%EC%A7%9C-%EC%B6%9C%EB%A0%A5-%EB%B0%A9%EB%B2%95>

Dataframe 을 csv파일로 누적 저장하기

<https://hogni.tistory.com/10>

강의자료

[도서] 파이썬 라이브러리를 활용한 머신러닝

[도서] 핸즈온 머신러닝

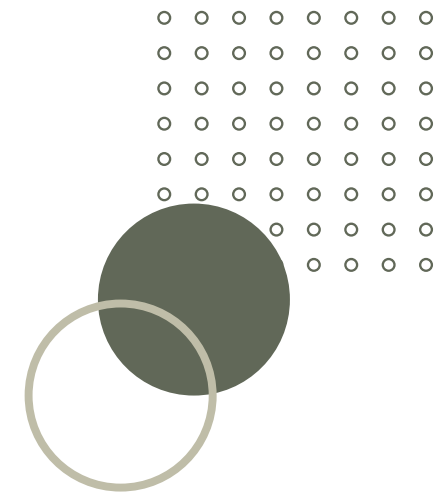
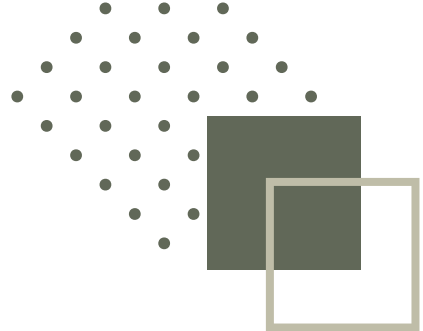
파이썬 차집합

https://zetawiki.com/wiki/Python_%EB%A6%AC%EC%8A%A4%ED%8A%B8_%ED%95%A9%EC%A7%91%ED%95%A9,%EA%B5%90%EC%A7%91%ED%95%A9,%EC%B0%A8%EC%A7%91%ED%95%A9,%EB%8C%80%EC%B9%AD%EC%B0%A8

모델 참고

https://dschloe.github.io/python/python_edu/07_deeplearning/deep_learning_lstm/

<https://roboreport.co.kr/%EB%94%A5%EB%9F%AC%EB%8B%9Dlstm%EC%9C%BC%EB%A1%9C-%EC%95%84%ED%8C%8C%ED%8A%B8-%EC%A7%80%EC%88%98-%EC%98%88%EC%B8%A1%ED%95%98%EA%B8%B0-2-lstm-%EC%8B%A4%ED%97%98%ED%95%98%EA%B8%B0/>



감사합니다 !

