< 아보카도 가격예측 >

팀명: 데이터에 미(美)팀 (2조)

팀원: 김원경, 노민호, 김은경

훈련과정명: 실무 프로젝트 기반 빅데이터 전략 마에스트로 과정

운영기관명: 한국경제신문사/ 한남대학교

Index

1. 프로젝트 배경

1. 프로젝트 배경

프로젝트 명 (주제)
아보카도 가격예측

프로젝트 목적
웰빙 시대에 얼맞는 아보카도의 가격 유추
소비자에게 저렴하게 공급할 수 있는 방안

프로젝트 개요
건강한 한국 사회, 소비자와 판매기업 모두에게 좋은 가격

프로젝트 구조
Python에서 numpy, pandas, seaborn, matplotlib. pyplot 응용하기

기대효과
아보카도 가격변화에 미치는 요인들 파악

1-1. 프로젝트 주제

“ 아보카도 가격예측 ”

아보카도를 선정한 이유 ?

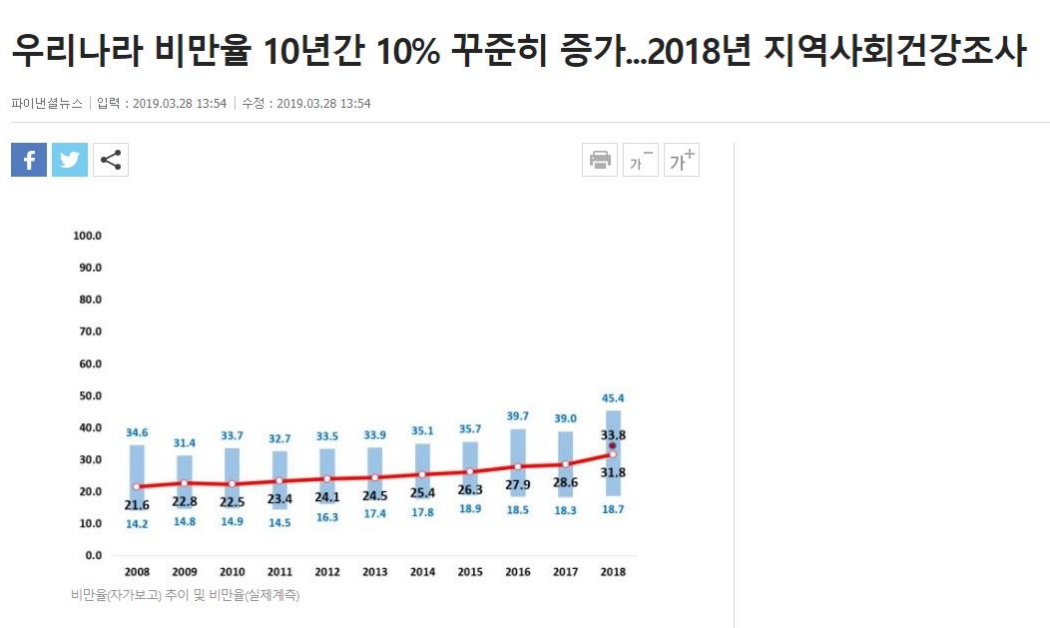
•세계 기네스북에 오른 영양소가 풍부한 식품
아미노산, 11가지 미네랄, 22가지 비타민, 불포화 지방산, 식이섬유 등

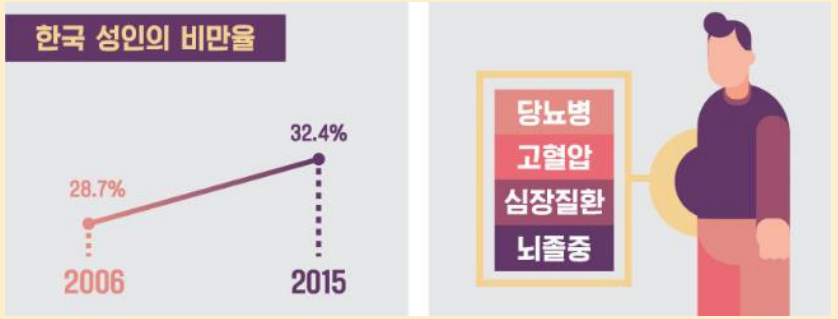
• 미국 타임지가 뽑은 세계 10대 슈퍼푸드
전 세계적 웰빙푸드로 주목

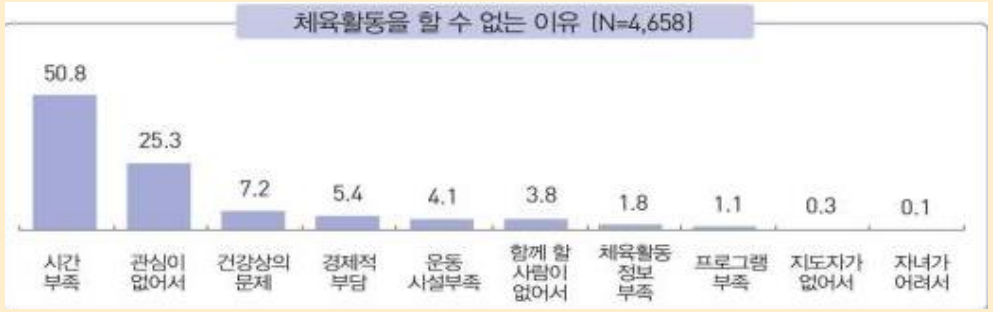
• 국내 수입량 급증

• 아보카도를 찾는 소비자 증가
스타벅스, 써브웨이, 아웃백, CU 편의점 국내유통업계 아보카도활용한 메뉴 출시

1-2. 프로젝트 목적







비만율 증가 / 한국 성인 비만율 증가 / 바쁜 현대인 운동 시간 부족으로 식단관리 중요

21세기에 들어 잘 먹고 잘 사는 '웰빙' 시대의 막이 오르면서,
많은 사람들이 자신이 먹는 음식에 대해 관심을 가짐

1-2. 프로젝트 목적

음료부터 스테이크까지...슈퍼푸드 '아보카도' 전성시대

입력 2018.09.04 13:42 | 수정 2018.09.04 13:49

사진=스타벅스 제공

전 세계적으로 아보카도가 웰빙푸드로 주목
> 국내 수입량도 급증
(남아메리카에서 주로 생산되는 과일로 국내에서는 보기 힘든 과일)

연간 수입액 200만달러(2008년)
> 지난해(2017년) 3000만달러로
→ 10년 사이 1458.3% 증가

올해 (2018년기준) 1~7월 수입액은 3300만달러
(작년 한해 수입액보다 높음)

아보카도를 찾는 소비자 증가
→ 국내 유통업계도 아보카도를 활용한 다양한 메뉴 출시

1-2. 프로젝트 목적

국제유가, 경제 지표 개선에 상승...WTI 3.1%↑

[뉴욕=뉴스핌] 김민정 특파원 = 국제유가가 29일(현지시간) 상승했다. 아시아와 유럽의 경제 지표가 개선되면서 유가는 상승 흐름을 보였다.

뉴욕상업거래소(NYSE)에서 거래된 8월 인도분 서부텍사스산 원유(WTI) 가격은 전 거래일 종가보다 배럴당 1.21달러(3.1%) 오른 39.70달러에 마감했다.

런던 ICE 선물거래소의 국제 벤치마크 브렌트유 8월물은 69센트(1.7%) 상승한 41.71달러를 기록했다.

유가는 전 세계 경제 지표가 개선세를 이어가면서 상승 흐름을 보였다. 유럽연합(EU) 집행위원회의 경기체감지수는 5월 67.5에서 6월 67.5로 역대 최대 상승 폭을 기록했다.

중국에서는 지난 5월 공업이익이 6개월간 처음으로 증가세를 기록하면서 코로나19 이후 경제 회복에 대한 기대를 강화했다.

경제 지표가 개선되면서 유가가 상승

> 유가는 경제지표에 대해 영향을 받는 요소임

> 유가 데이터를 경제적지표로 활용하여 아보카도 데이터를 분석하려 함

1-3. 프로젝트 개요

"몸과 정신이 건강한 한국 사회"

소비자와 판매기업 모두에게

아보카도의 좋은 가격으로 제시할 수 있는 방안모색

1-3. 프로젝트 개요

“판매기업들은 한꺼번에 대량 구입을 해야 하는데 정보가 필요하지 않겠는가?”

↓

월 별 가격 예측 보고서

2. 프로젝트 구성 및 역할

2. 프로젝트 팀 구성 및 역할

조장 1. 김원경

- 데이터분석
- 자료조사
- 데이터수집
- 데이터시각화

조원 2. 노민호

- 데이터분석
- 자료조사
- 데이터수집
- 보고서작성

조원 3. 김은경

- 데이터분석
- 자료조사
- 데이터시각화
- PPT 제작

3. 데이터 설명 및 수정

3-1. 자료 설명

1. 자료 수집 사이트

https://www.kaggle.com/neuromusic/avocado-prices
https://www.kaggle.com/mabusalah/brent-oil-prices
https://www.eia.gov/petroleum/weekly/crude.php
https://hassavocadoboard.com/

- Kaggle 사이트 : 2015/01/04 ~ 2018/03/25 주간 지역별 아보카도 Average Price data, 1987/05/20 ~ 2020/04/21 원유 데이터
- Eia 사이트 : 1990/8/20~ 2020/07/21 가솔린 데이터
- Hass avocado 사이트 : 아보카도 2018~ 2020 데이터 추가.

3-1. 자료 설명

2. 변수 설명

Date	Average Price	Total Volume	4046	4225	4770
주별 날짜	주별 평균 가격	총 판매량	4046제품 판매량	4225제품 판매량	4770제품 판매량

Total Bags	Small Bags	Large Bags	Xlarge Bags	Type	region
총 bags 판매량	총 작은 bags 판매량	총 중간 bags 판매량	총 큰 bags 판매량	유기농(organic), 무기농(conventional)	판매지역

Year	Month	season	Day	gasoline_price	crude_price
연도	월	계절	일	가솔린 가격	원유가격

* Year, Month, season, Day 변수는 새로 생성한 변수

3-2. 자료 수정

1. 데이터 합치기

<Kaggle 사이트>

아보카도 : 2015/01/04 ~ 2018/03/25 주간 지역별 Average Price data
18249행의 데이터
원유 : 1987/05/20 ~ 2020/04/21 원유 가격 데이터

< Eia 사이트>

가솔린 : 1990/8/20~ 2020/07/21 가솔린 가격 데이터

<Hass 사이트>

아보카도 : 2018~ 2020/4/19까지의 데이터

3-2. 자료 수정

1. 데이터 합치기

1. Kaggle 사이트와 Hass 사이트의 아보카도 데이터를 합치고, 겹치는 2018년 data 삭제

2. Kaggle 사이트의 원유 데이터와 Eia 사이트의 가솔린 데이터를 추가

3-2. 자료 수정

1. 데이터 합치기

1) 아보카도의 겹치는 2018년 데이터 제거 후, 2015~2020년 아보카도 데이터 합치기

```
# 겹치는 2018년 데이터 제거하기
av = av[av["year"] != 2018]
```

```
#데이터 합치기
avocado = pd.concat([av] + [av2018] + [av2019]+ [av2020], axis = 0, ignore_index = True)
avocado.dtypes
```

```
Date             object
AveragePrice    float64
Total Volume    float64
4046            float64
4225            float64
4770            float64
Total Bags      float64
Small Bags      float64
Large Bags      float64
XLarge Bags     float64
type             object
year              int64
region           object
dtype: object
```

3-2. 자료 수정

1. 데이터 합치기

2) 가솔린 가격과 원유 가격 합치기

```
price = pd.merge(gas_price,crude_price,left_on = 'Date',right_on = 'Date',how='outer')
price
```

3) 1번의 아보카도 데이터와 합치기

```
av_price = pd.merge(avocado, price, left_on='Date', right_on='Date', how='left')
```

3-2. 자료 수정

2. 새로운 변수 만들기

Year	Month	season	Day
연도	월	계절	일

- Date 변수 -> Year, Month, season, Day 변수 새로 생성

3-2. 자료 수정

2. 새로운 변수 만들기

Year 변수 추가하기

```
# year 변수 추가하기
av2018['year'] = 0
av2019['year'] = 0
av2020['year'] = 0
for i in range(len(av2018)):
    av2018['year'][i] = str(av2018['Date'][i])[0:4]
for i in range(len(av2019)):
    av2019['year'][i] = str(av2019['Date'][i])[0:4]
for i in range(len(av2020)):
    av2020['year'][i] = str(av2020['Date'][i])[0:4]
```

Month 변수 추가하기

```
#month 변수 만들기
avocado['month'] = 0
for i in range(len(avocado)):
    avocado['month'][i]=str(avocado['Date'][i])[5:7]
```

3-2. 자료 수정

2. 새로운 변수 만들기

Season 변수 추가하기

```
#season 변수 만들기
avocado['season'] = 0
boolvec = (avocado['month']==3)|(avocado['month']==4)|(avocado['month']==5)
boolvec1 = (avocado['month']==6)|(avocado['month']==7)|(avocado['month']==8)
boolvec2 = (avocado['month']==9)|(avocado['month']==10)|(avocado['month']==11)
boolvec3 = (avocado['month']==12)|(avocado['month']==1)|(avocado['month']==2)

avocado.loc[boolvec,'season']='spring'
avocado.loc[boolvec1,'season']='summer'
avocado.loc[boolvec2,'season']='fall'
avocado.loc[boolvec3,'season']='winter'
```

Day 변수 추가하기

```
# Date 변수를 datetime 타입으로 변환
avocado['Date'] = pd.to_datetime(avocado['Date'])
avocado.sort_values('Date',ignore_index = True,inplace = True)
avocado['Date'].isnull().value_counts()
```

```
False    29589
Name: Date, dtype: int64
```

```
avocado['Day']=avocado['Date'].apply(lambda x:x.day)
```

3-2. 자료 수정

3. 필요 없는 데이터 제거

Region 변수에서

TotalUS, California, West, Plains, SouthCentral, Southeast, Midsouth, Northeast, GreatLakes 제거

3-2. 자료 수정

3. 필요 없는 데이터 제거

```
# 변수 region에서 TotalUS, California, West, Plains, SouthCentral, Southeast, Midsouth, Northeast, GreatLakes 제거
avocado = avocado[avocado['region'] != 'TotalUS']
avocado = avocado[avocado['region'] != 'California']
avocado = avocado[avocado['region'] != 'West']
avocado = avocado[avocado['region'] != 'Plains']
avocado = avocado[avocado['region'] != 'SouthCentral']
avocado = avocado[avocado['region'] != 'Southeast']
avocado = avocado[avocado['region'] != 'Midsouth']
avocado = avocado[avocado['region'] != 'Northeast']
avocado = avocado[avocado['region'] != 'GreatLakes']
```

3-2. 자료 수정

4. crude_oil, gasoline 3개월 이전의 데이터로 가져오기

crude_oil, gasoline은 선행데이터이므로
3개월 이전의 데이터를 가져온다.

```
import datetime
b = crude_price['Date'] + datetime.timedelta(days=91)
```

```
crude_price['Date'] = b
crude_price
```

```
import datetime
a = gas['Date'] + datetime.timedelta(days=90)
```

```
gas['Date'] = a
gas
```

3-2. 자료 수정

<AVOCADO DATA>

총 24657개의 행과
(2015/01/04~2020/04/19)

19개의 열을 가진다.

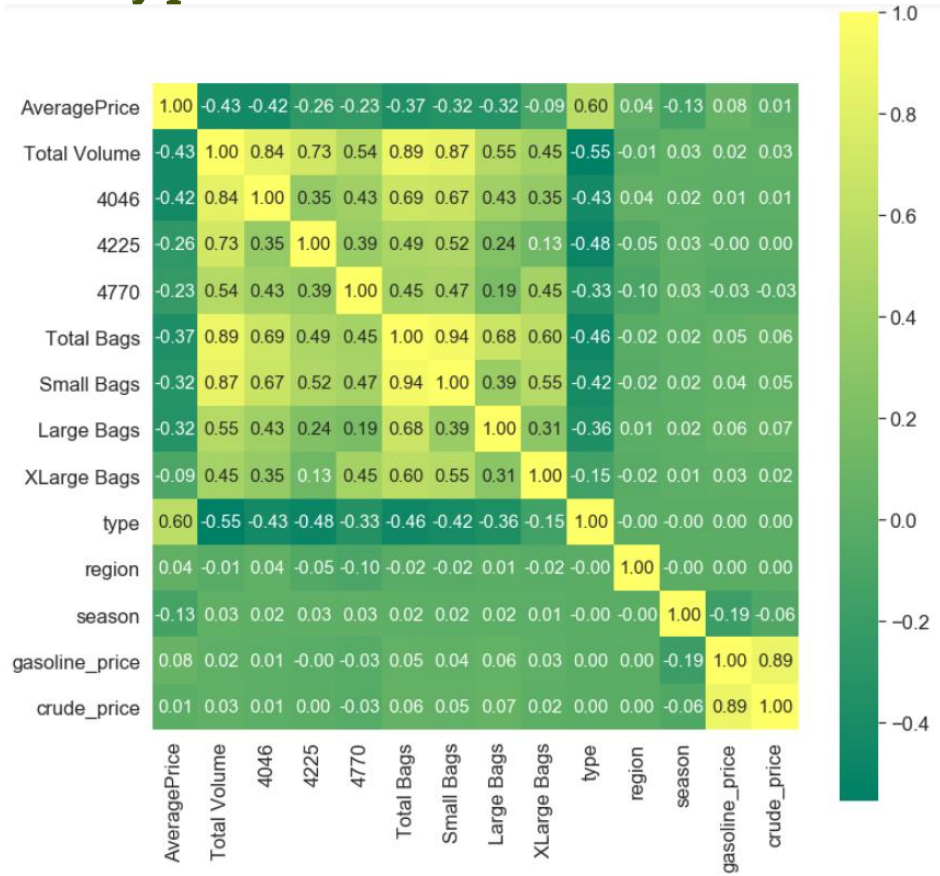
4. EDA

4-1. 가격과 각 변수들 간 그래프 그리기

1) 상관관계 분석 그림

각 변수별로 상관관계 분석 그림 만들기

→ type 변수가 0.6로 굉장히 높은 양의 상관관계를 보여준다.



4-1. 가격과 각 변수들 간 그래프 그리기

2) 가격과 Date

(1) avocado AveragePrice 시계열 그래프

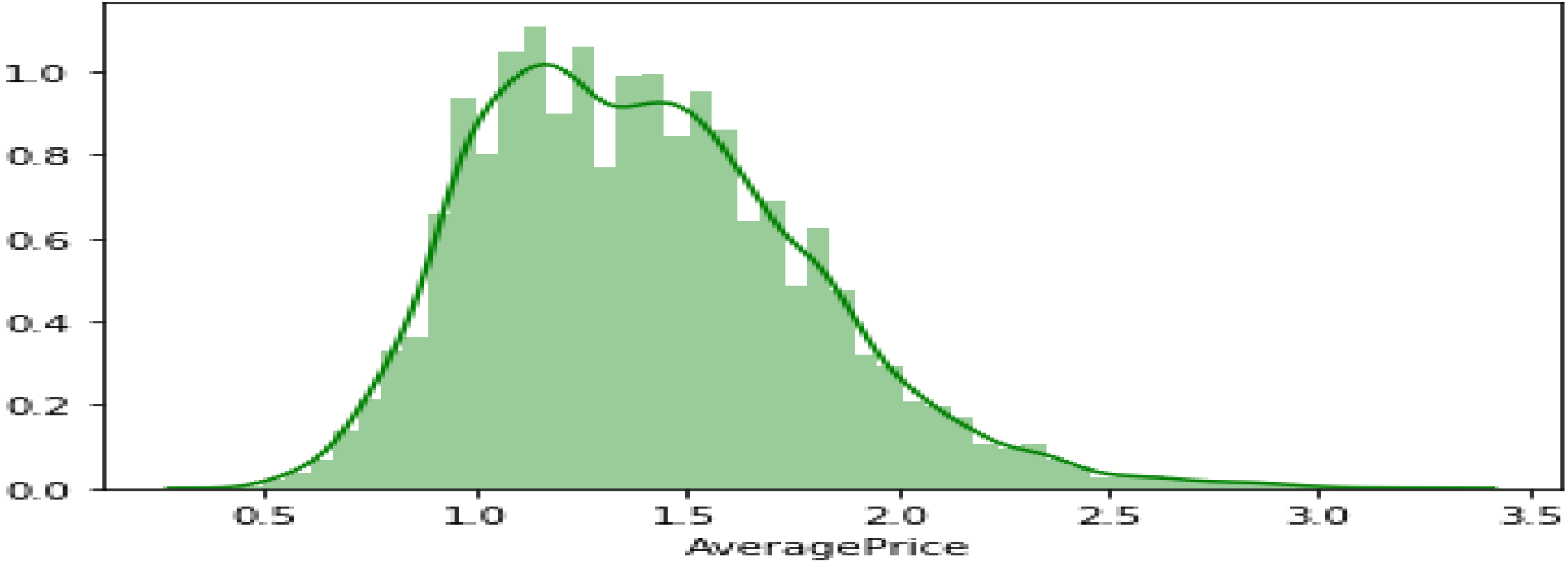
→ avocado의 가격 변화 확인(5년간)

4-1. 가격과 각 변수들 간 그래프 그리기

2) 가격과 Date

2) avocado AveragePrice 히스토그램

→ avocado의 히스토그램 그리기(가격의 형성대 확인(5년간))

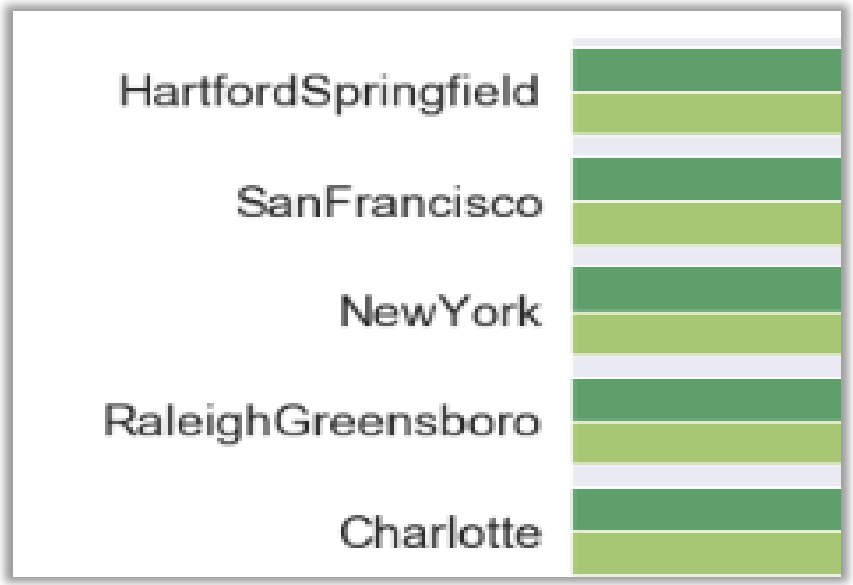


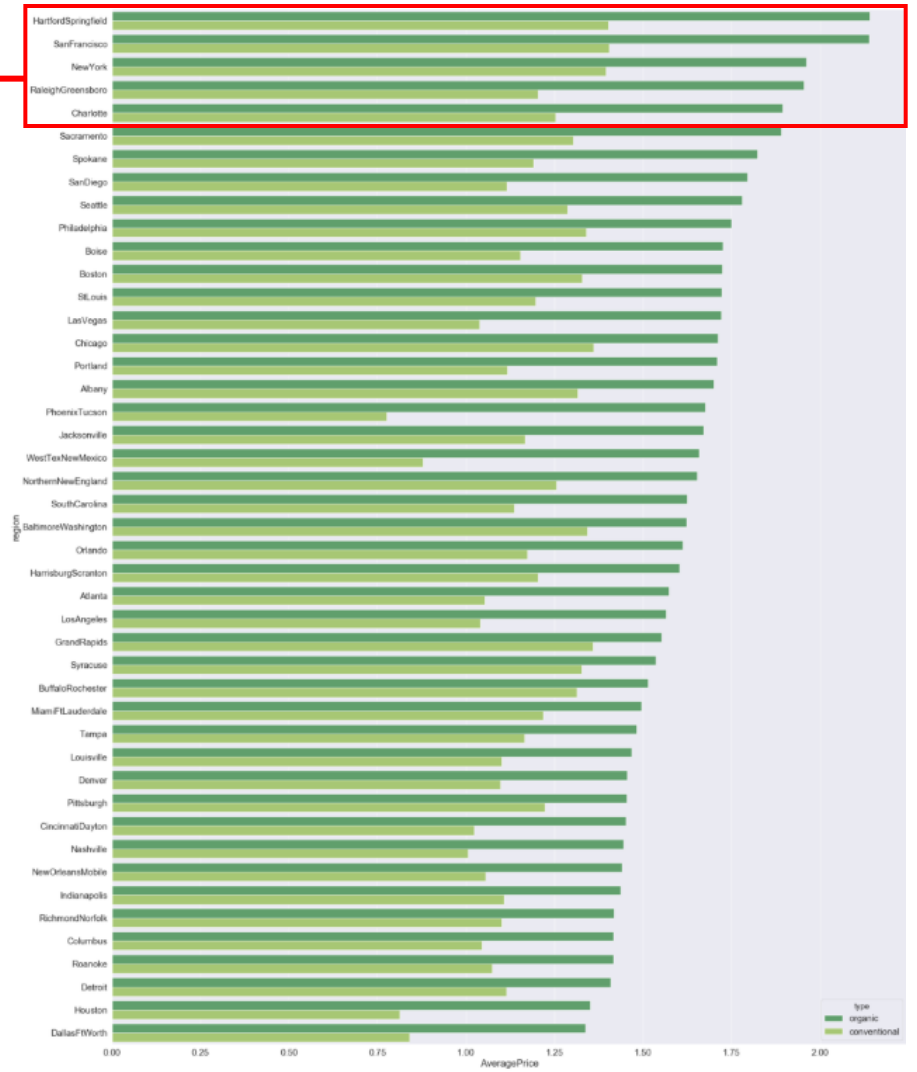
4-1. 가격과 각 변수들 간 그래프 그리기

3) 가격과 지역

1) avocado 지역별 평균 가격 막대 그래프

→ 지역별로 type 마다의 평균가격을 막대그래프로 표시





4-1. 가격과 각 변수들 간 그래프 그리기

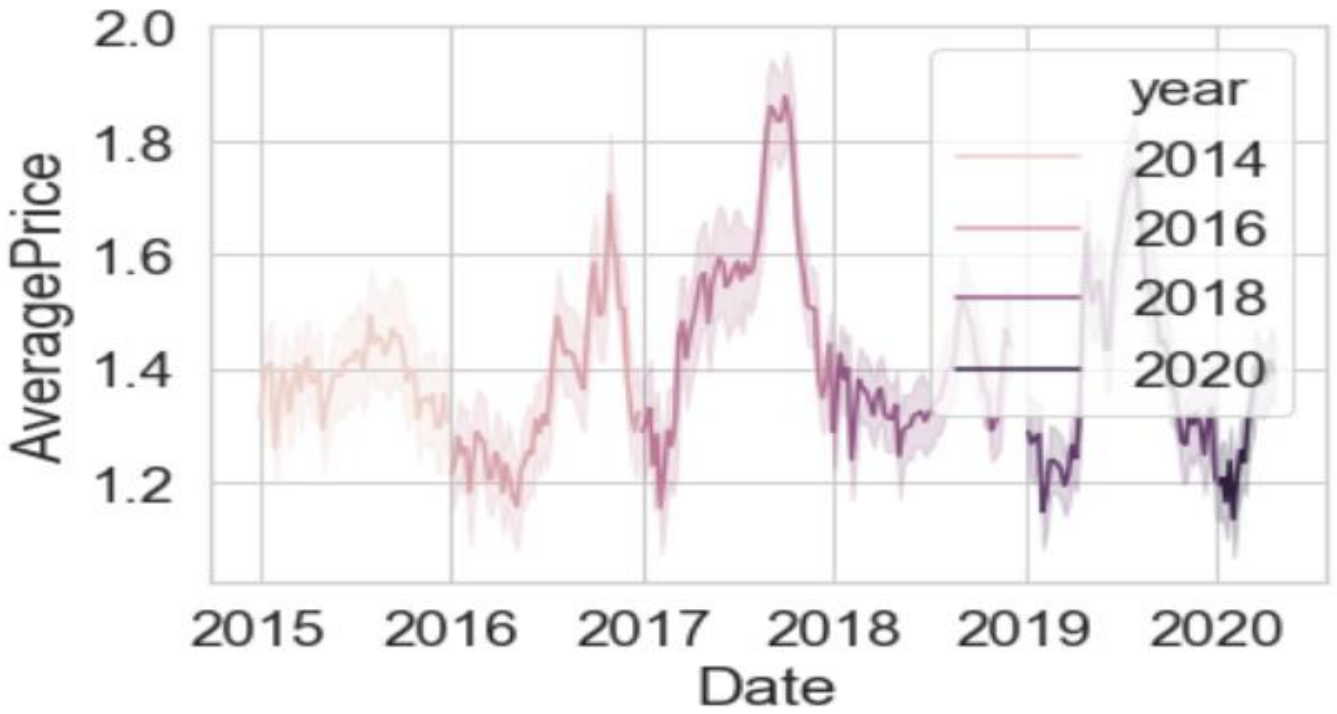
3) 가격과 지역

2) 지역별 평균 가격 지도 시각화

4-1. 가격과 각 변수들 간 그래프 그리기

4) 가격과 year별 그래프

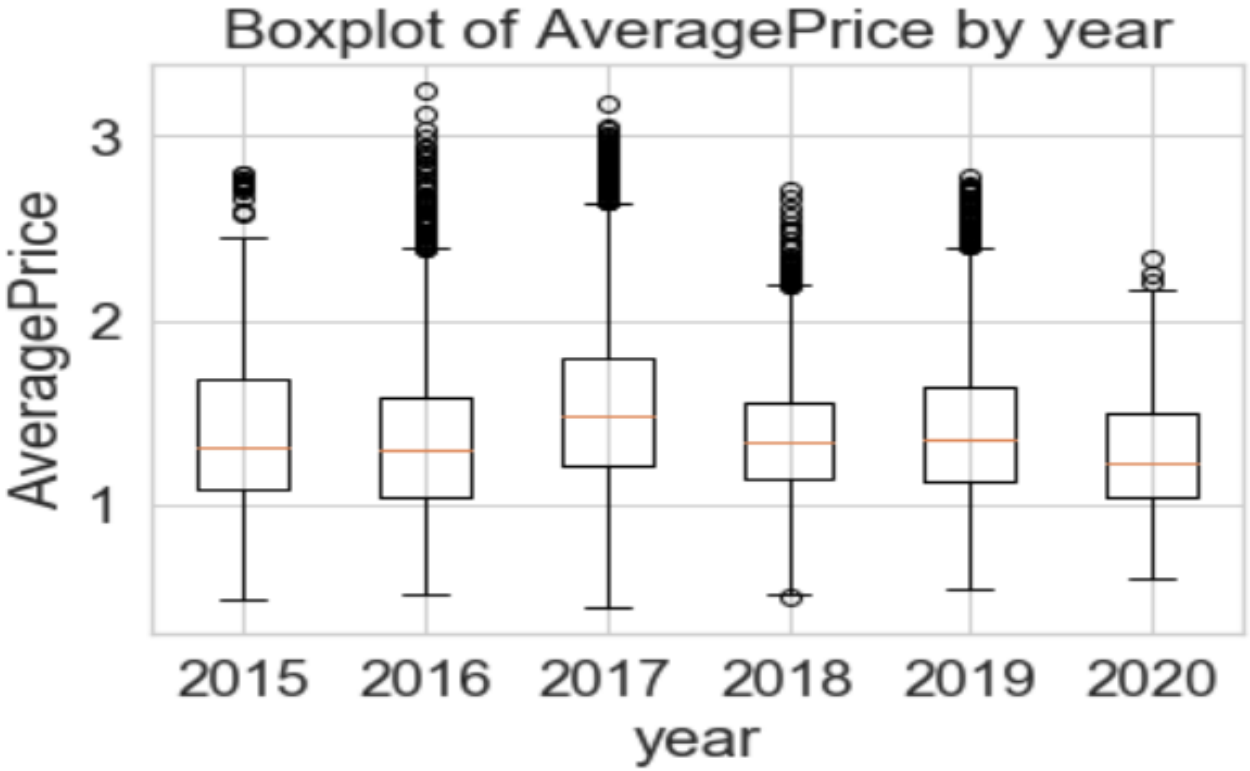
1) avocado AveragePrice 변수를 lineplot으로 그린 후 year별 구분을 해주는 그래프



4-1. 가격과 각 변수들 간 그래프 그리기

4) 가격과 year별 그래프

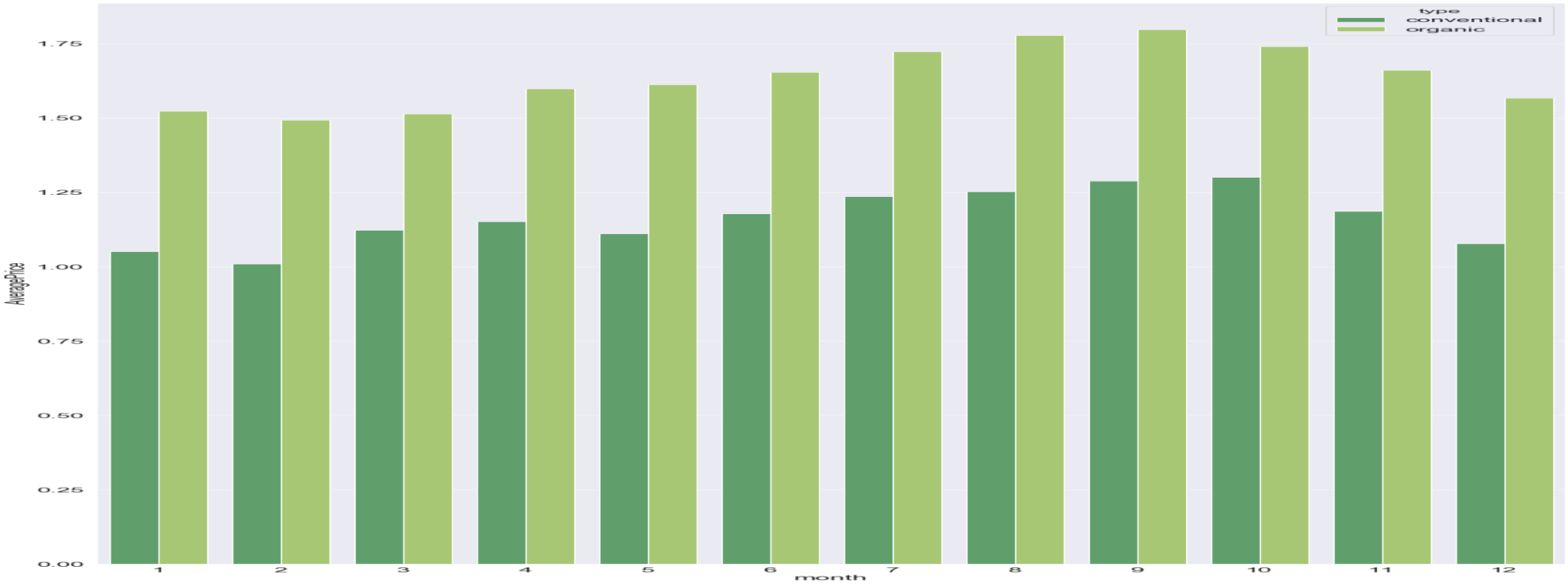
2) boxplot -> year별로 가격의 분포를 보여줌



4-1. 가격과 각 변수들 간 그래프 그리기

5) 가격과 month별 그래프

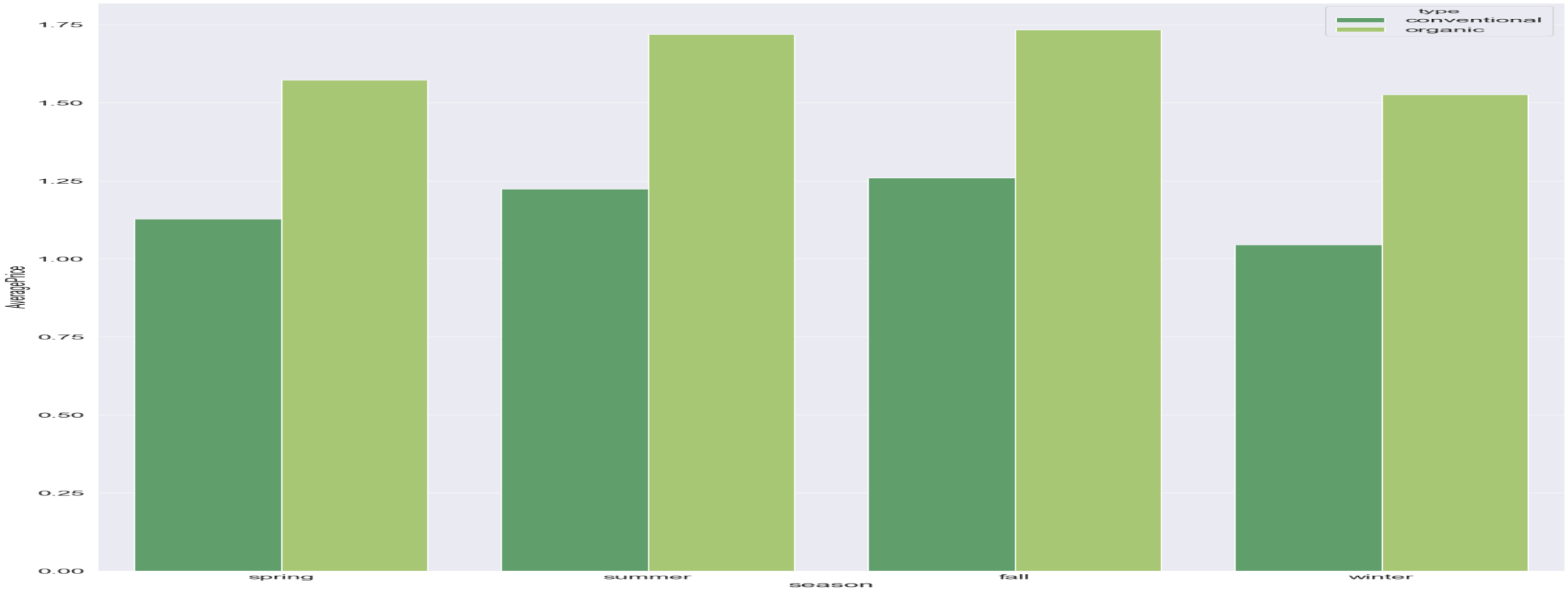
1) month별 평균가격 막대그래프



4-1. 가격과 각 변수들 간 그래프 그리기

6) 가격과 season별 그래프

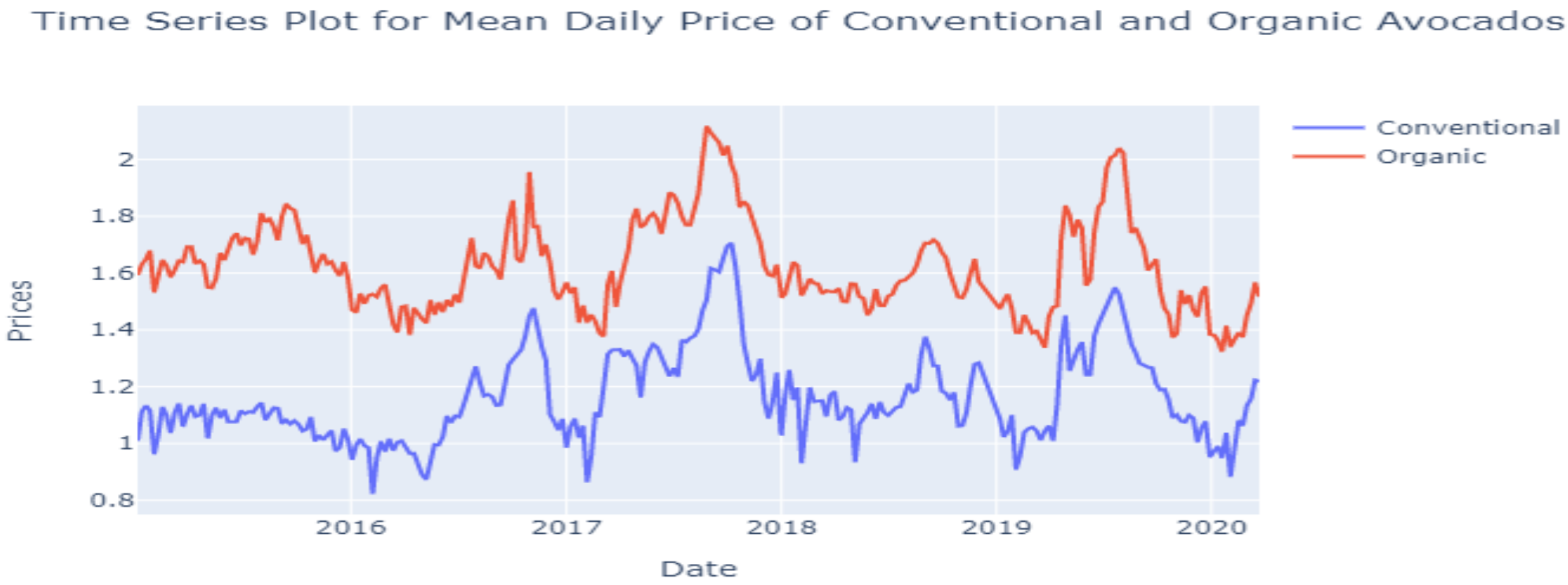
1) season별 type별 평균 가격 barplot



4-1. 가격과 각 변수들 간 그래프 그리기

6) 가격과 type별 그래프 (제일 가격에 영향력이 높은 변수)

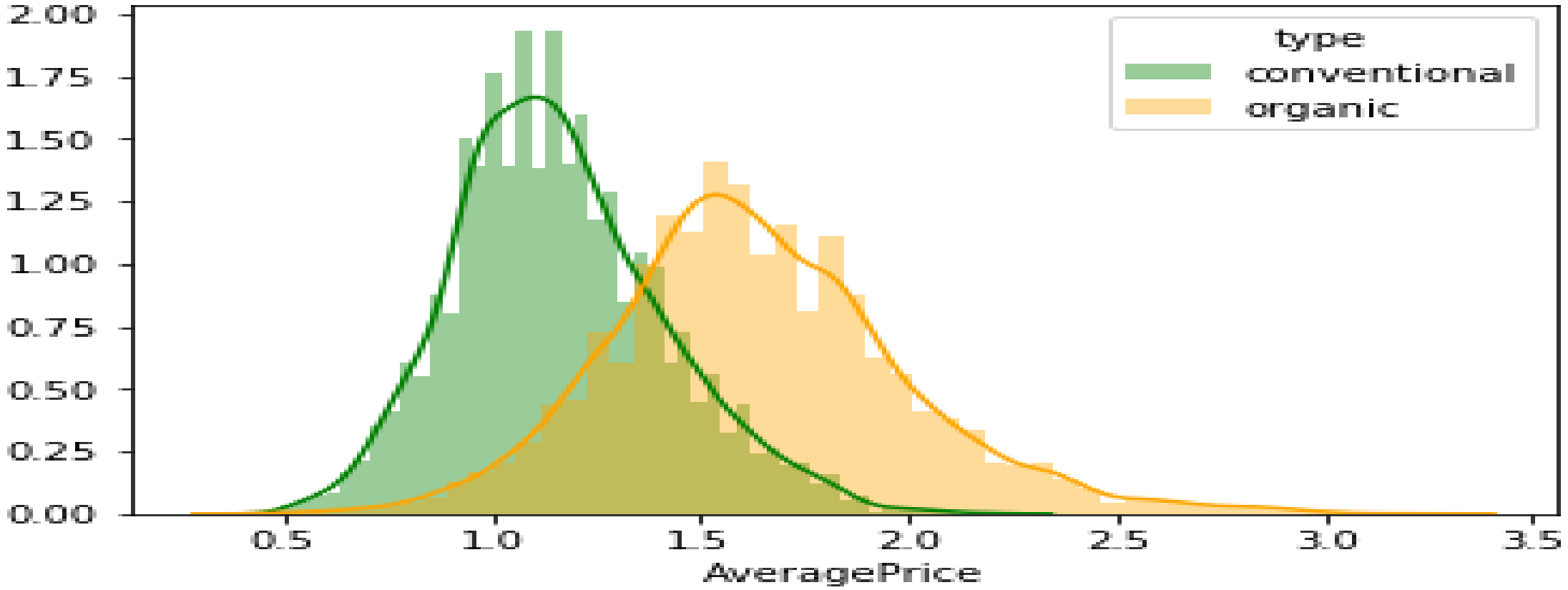
1) type별로 시계열 데이터 그려보기



4-1. 가격과 각 변수들 간 그래프 그리기

6) 가격과 type별 그래프 (제일 가격에 영향력이 높은 변수)

2) type별로 히스토그램 그려보기

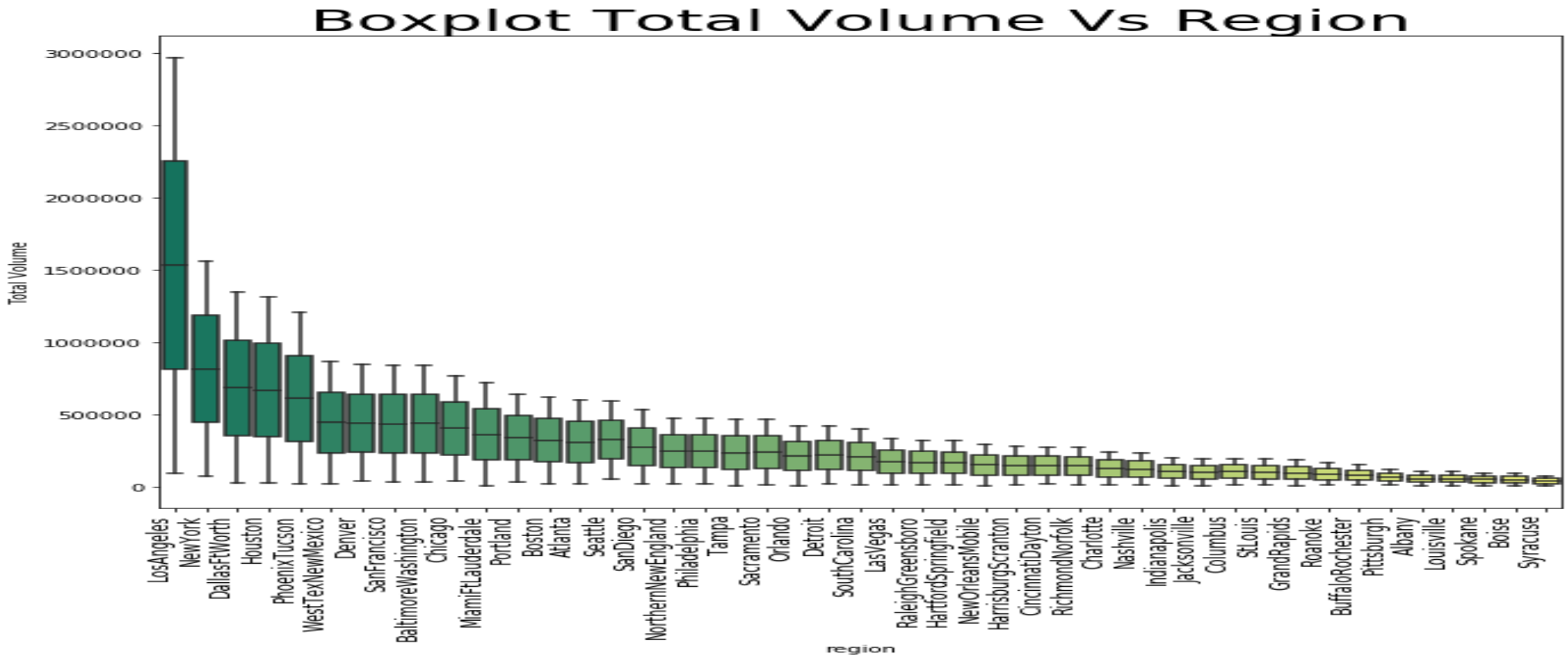


4-2. 판매량과 각 변수들 간 그래프 그리기

1) 판매량과 지역

1) avocado 지역별 판매량 boxplot

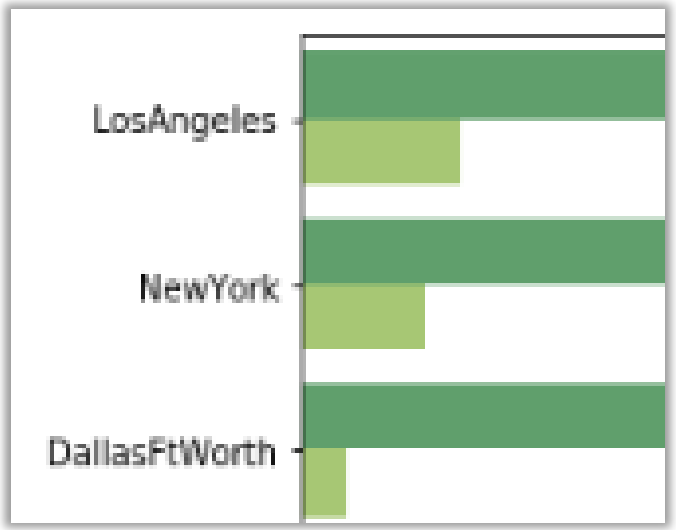
→ 지역별로 판매량을 boxplot으로 표시

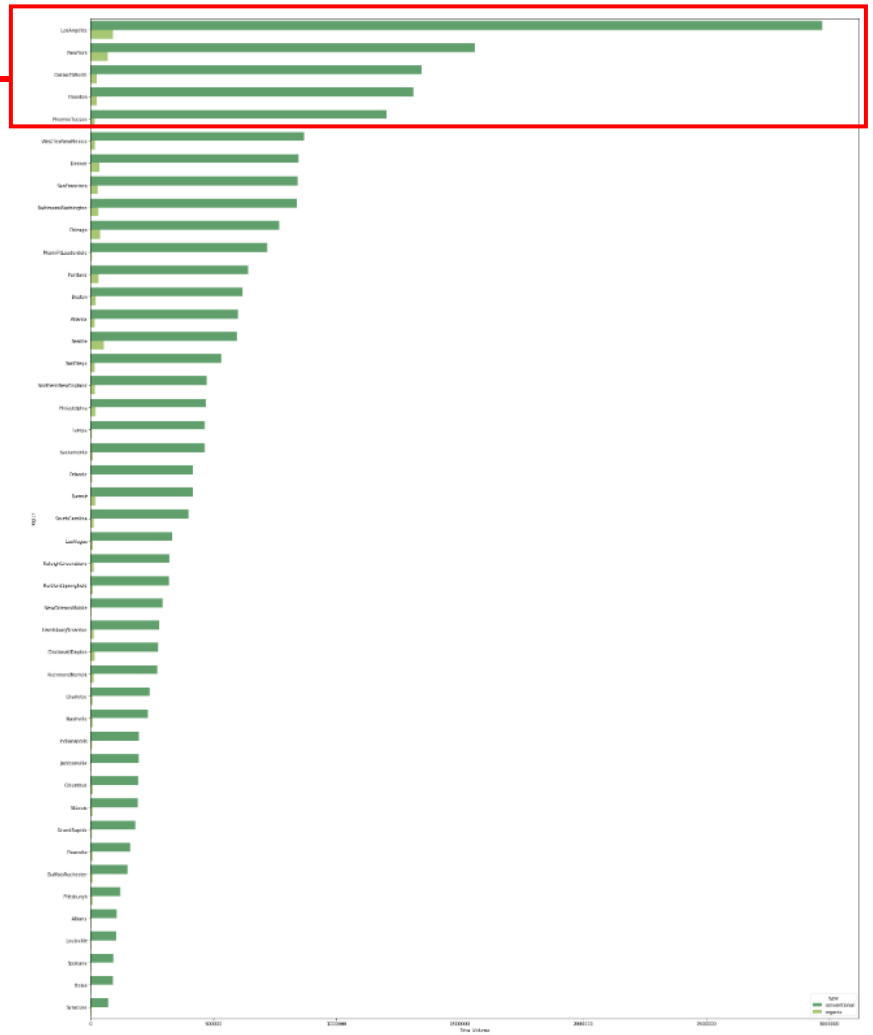


4-2. 판매량과 각 변수들 간 그래프 그리기

1) 판매량과 지역

2) avocado 지역별 type별 barplot

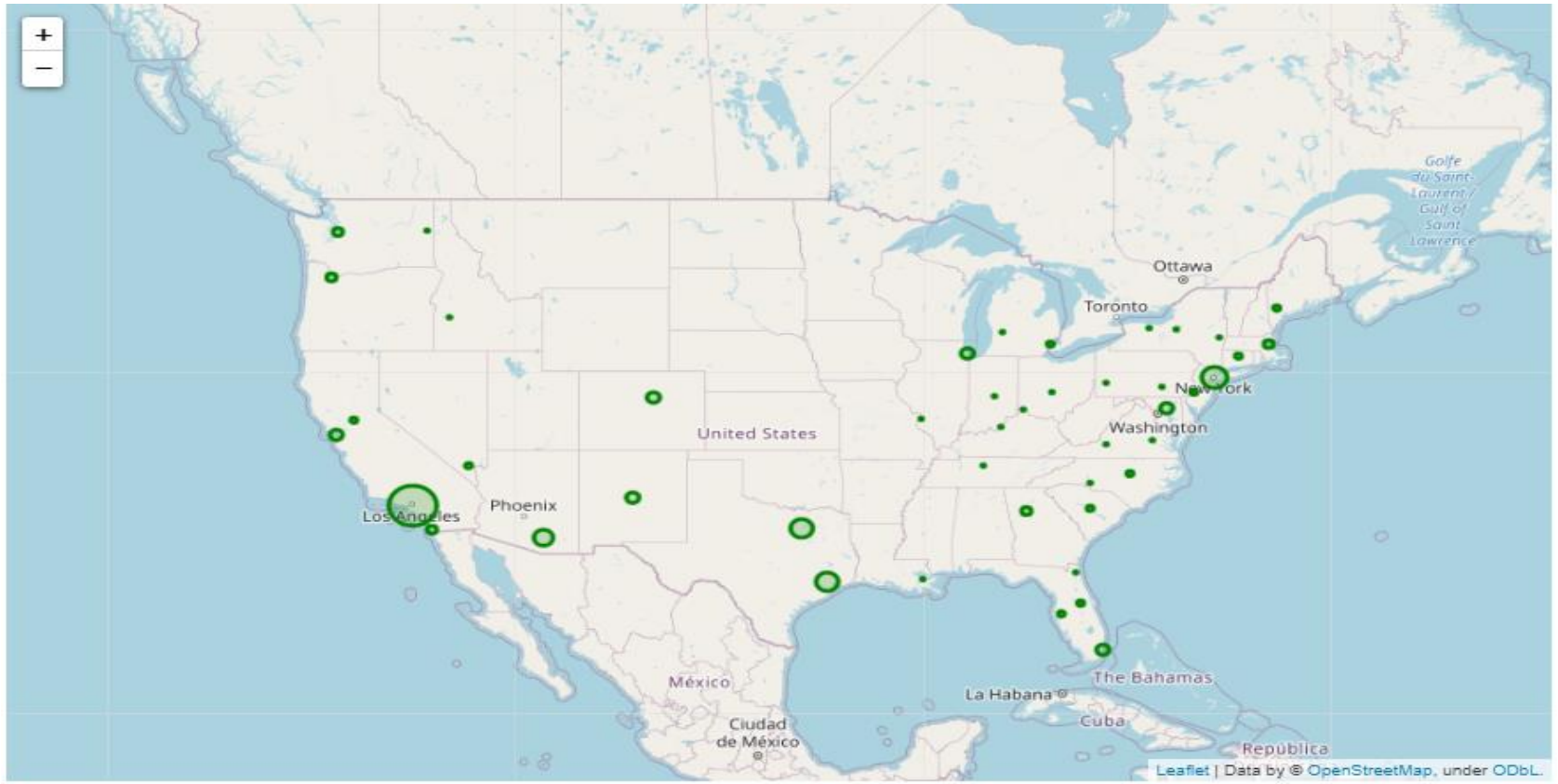




4-2. 판매량과 각 변수들 간 그래프 그리기

1) 지역별 판매량

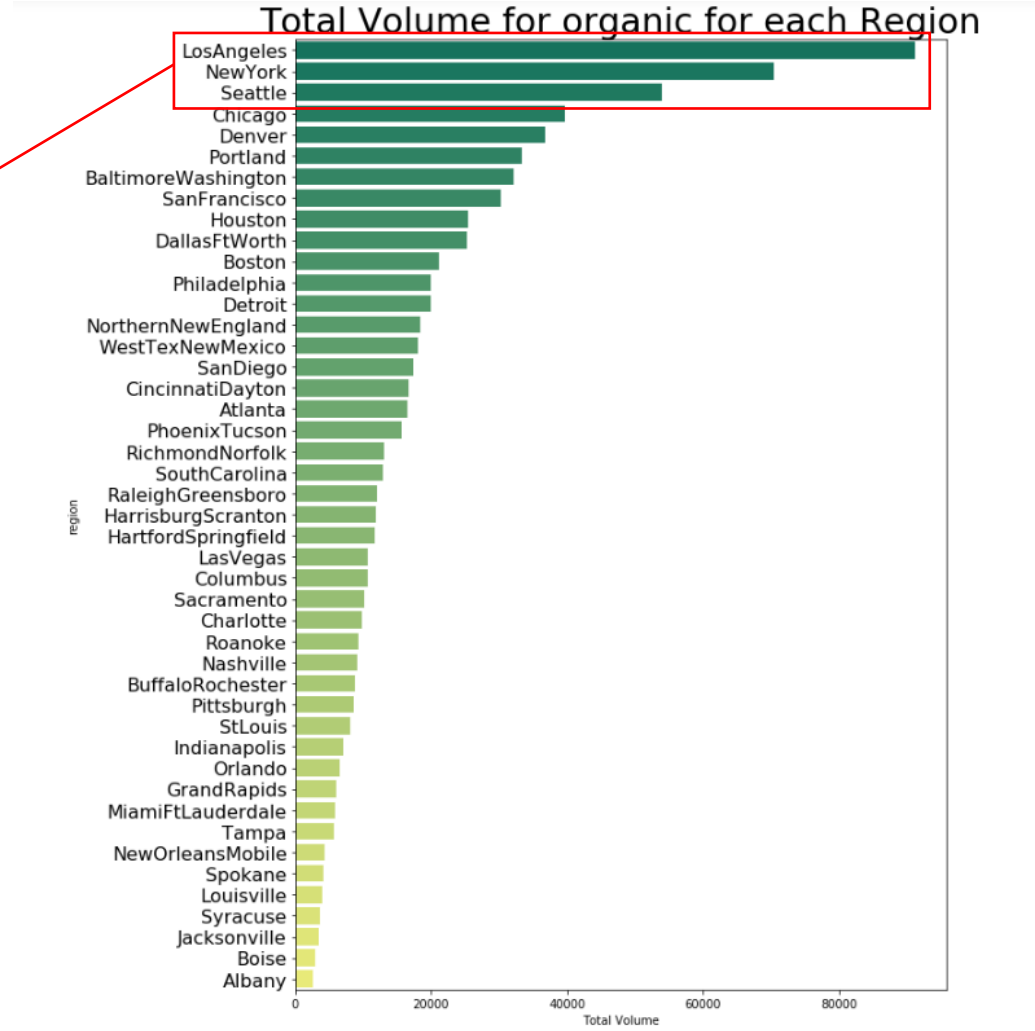
3) 지역별 평균 판매량 지도 시각화



4-2. 판매량과 각 변수들 간 그래프 그리기

2) 판매량과 지역과 type

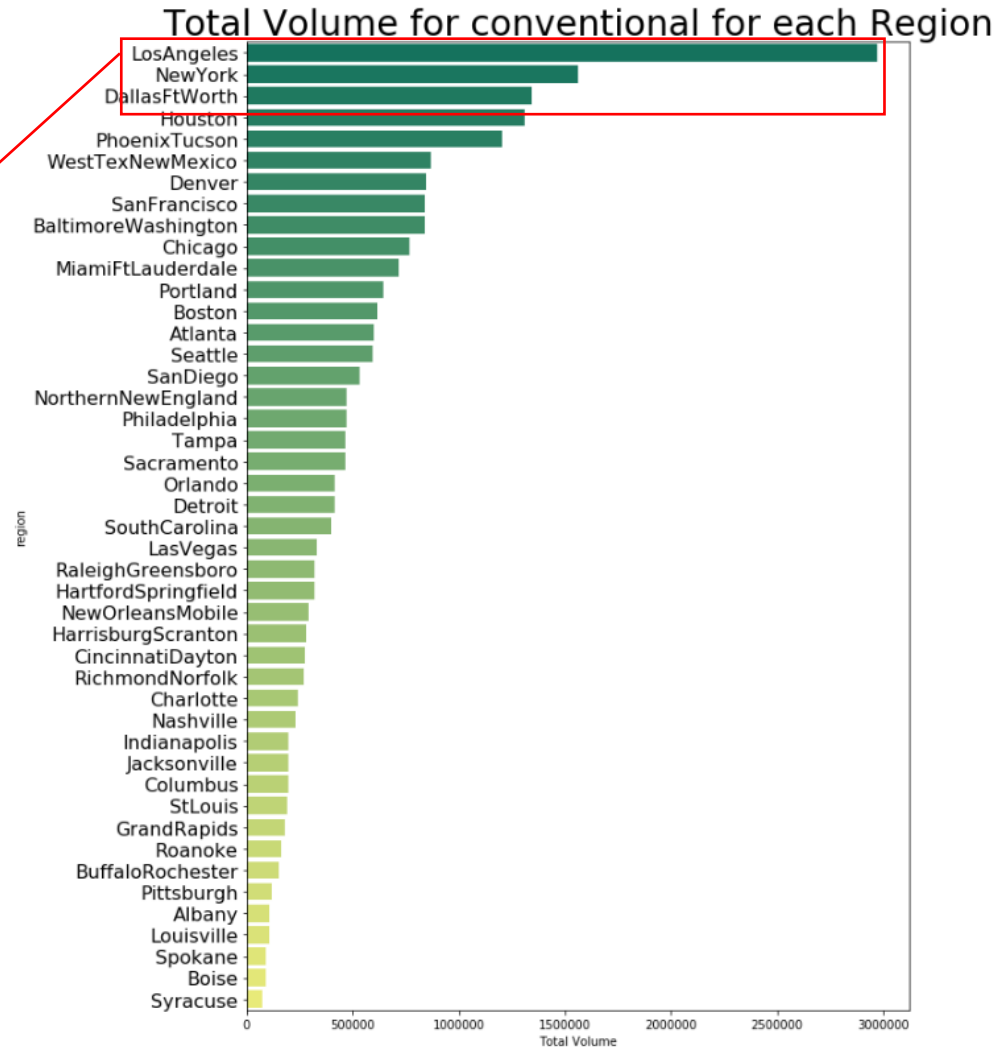
1) type의 organic에 대한 지역별 판매량 barplot



4-2. 판매량과 각 변수들 간 그래프 그리기

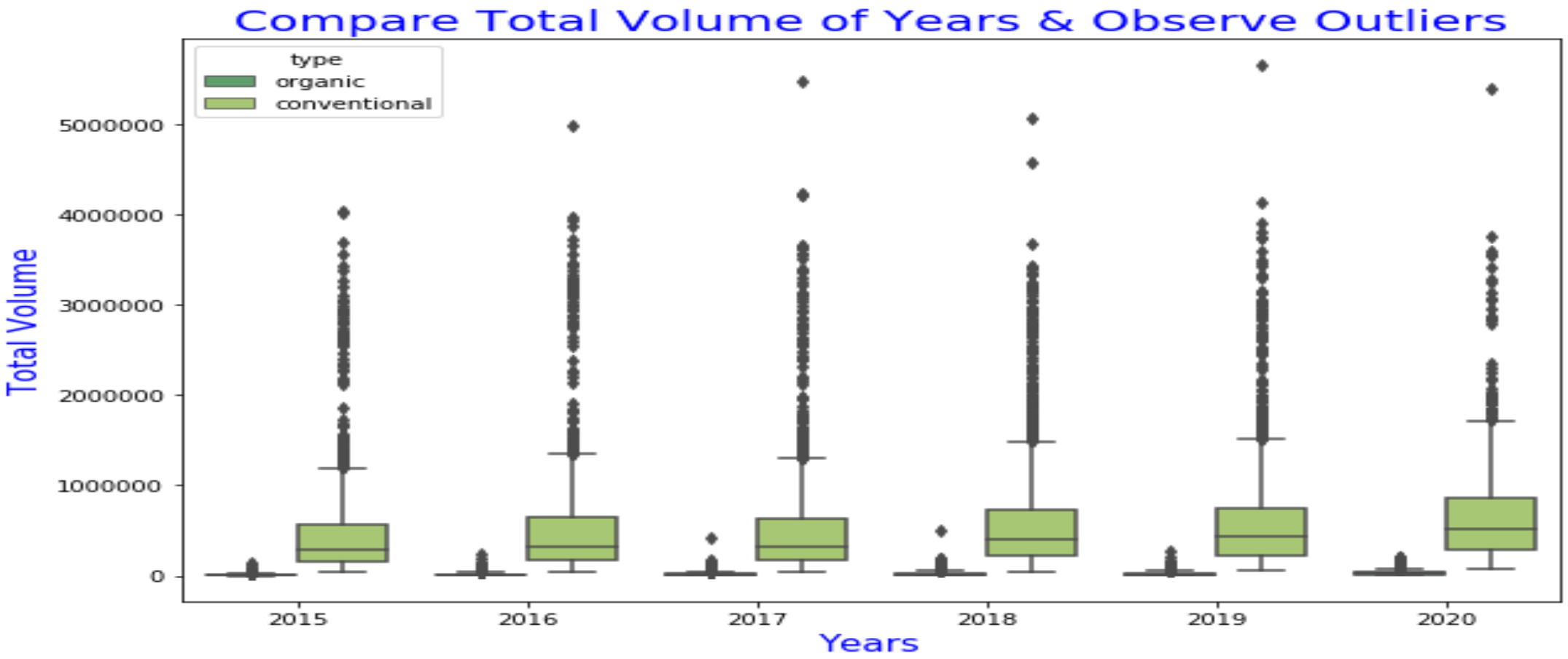
2) 판매량과 지역과 type

2) type의 conventional에 대한 지역별 판매량 barplot



4-2. 판매량과 각 변수들 간 그래프 그리기

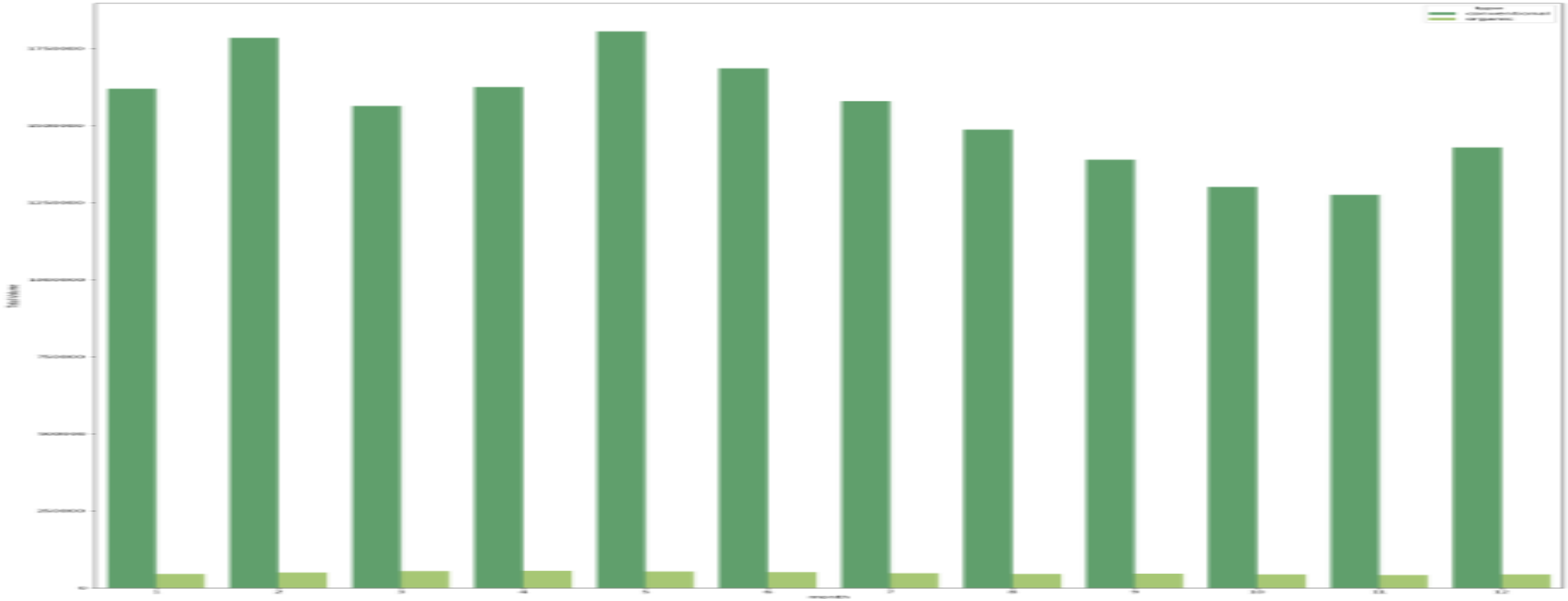
3) 판매량과 type과 year



4-2. 판매량과 각 변수들 간 그래프 그리기

4) 판매량 month별 그래프

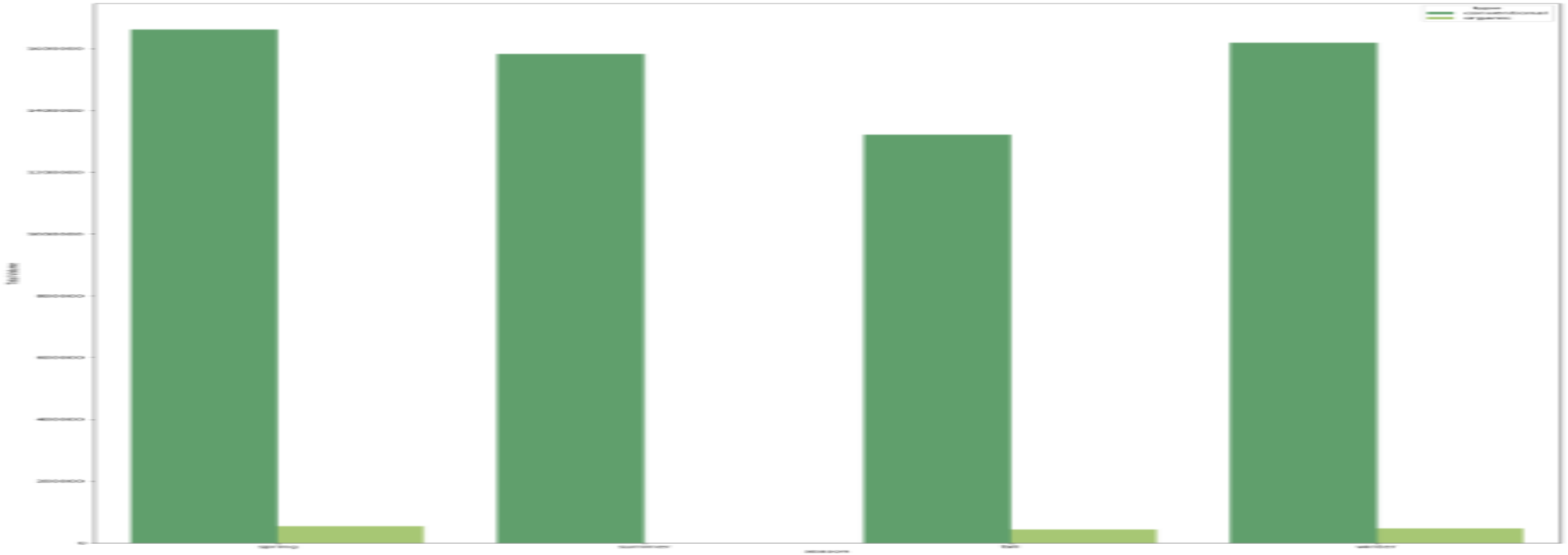
1) month별 평균가격 막대그래프



4-2. 판매량과 각 변수들 간 그래프 그리기

5) 판매량과 season별 그래프

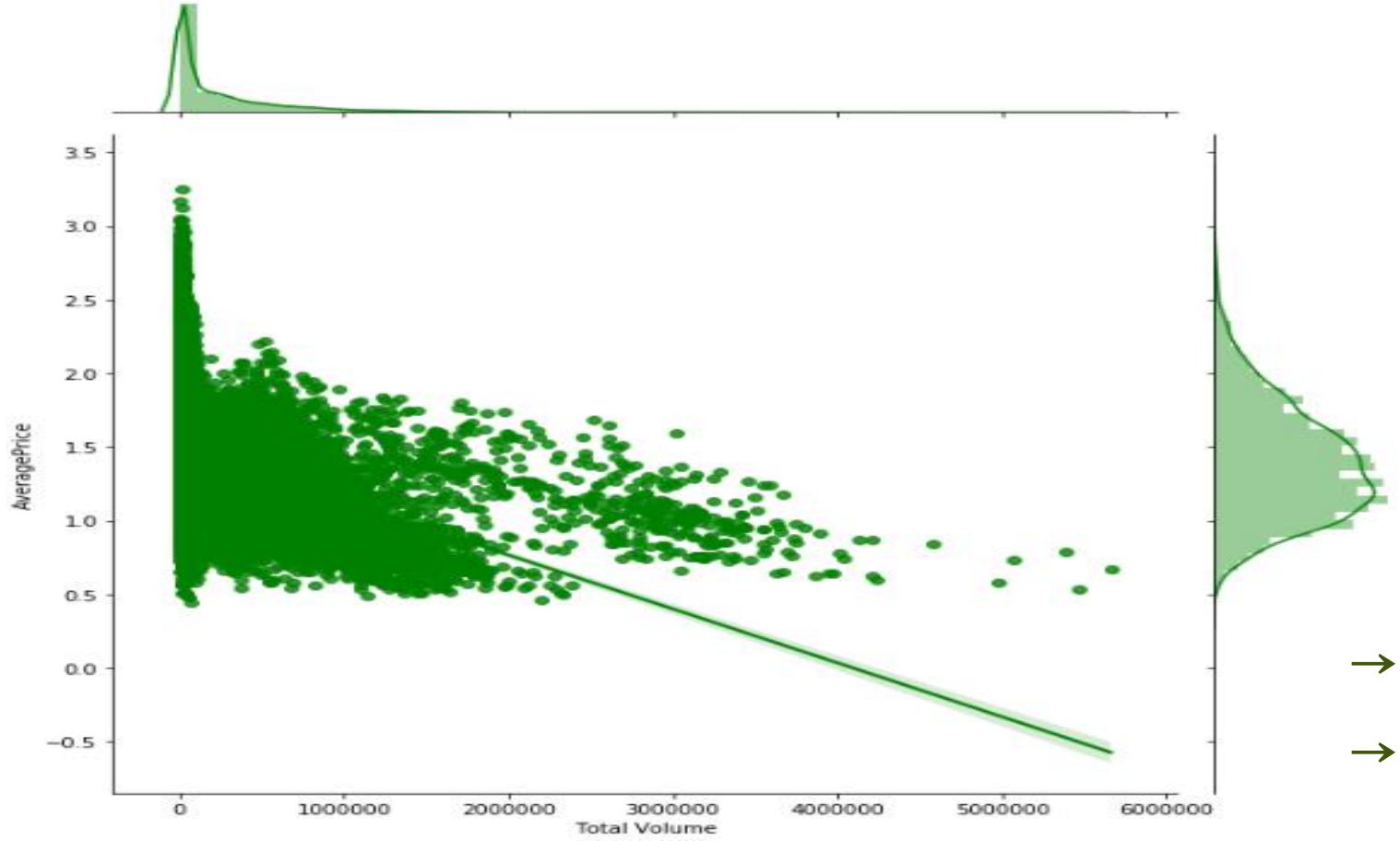
1) season별 type별 평균 판매량 barplot



4-3. 가격과 판매량 간 그래프 그리기

1) 가격과 판매량의 상관관계

1) 가격과 판매량에 대한 상관관계



→ - 0.43

→ 판매량이 ↑ , 평균가격↓

4-3. 가격과 판매량 간 그래프 그리기

2) 가격과 판매량의 KDE그래프

1) 지역에 대해 평균가격과 평균판매량 데이터 만들기

```
df0=pd.DataFrame({'region_list':region_list,'average_price':average_price})
new_index=df0.average_price.sort_values(ascending=False).index.values
sorted_data=df0.reindex(new_index)
```

```
df0.head()
```

	region_list	average_price
0	Columbus	1.232296
1	GreatLakes	1.343153
2	StLouis	1.427930
3	RichmondNorfolk	1.295637
4	Indianapolis	1.272870

```
region_list=list(df1.region.unique())
average_total_volume=[]

for i in region_list:
    x=df1[df1.region==i]
    average_total_volume.append(sum(x['Total Volume'])/len(x))
df2=pd.DataFrame({'region_list':region_list,'average_total_volume':average_total_volume})

new_index=df2.average_total_volume.sort_values(ascending=False).index.values
sorted_df1=df2.reindex(new_index)
```

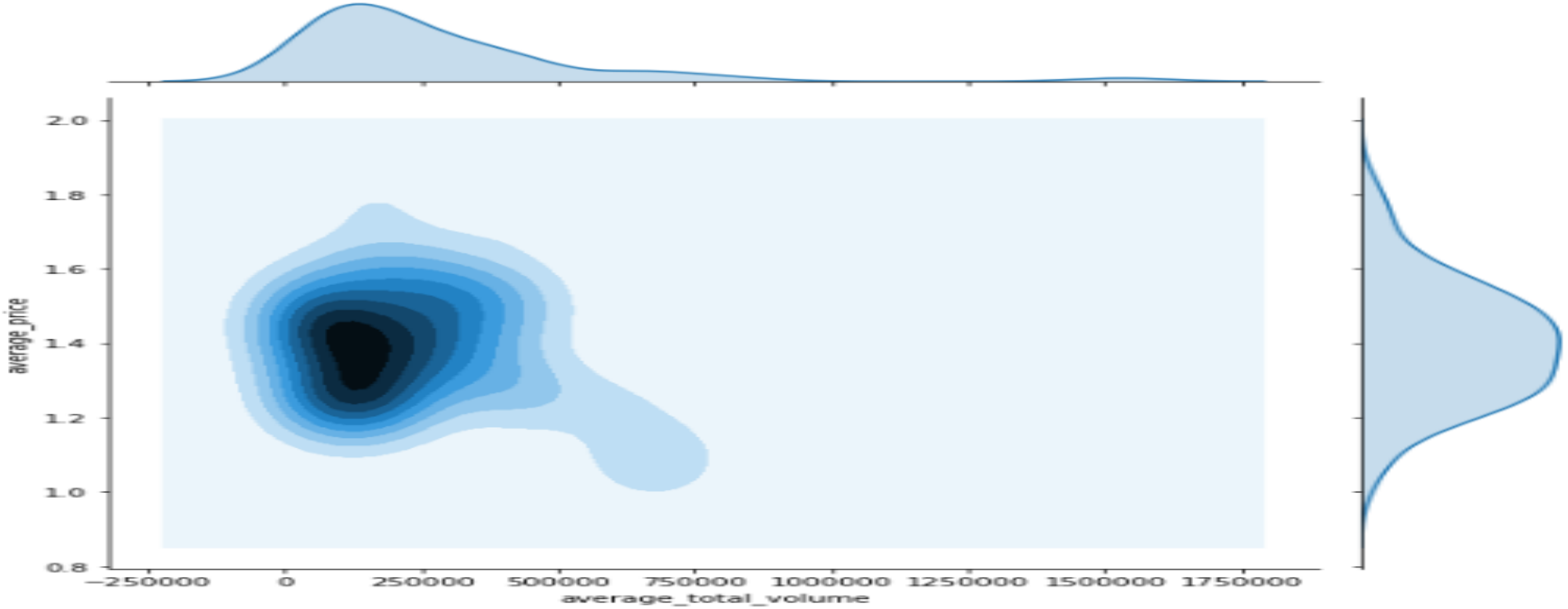
```
df2.head()
```

	region_list	average_total_volume
0	SanDiego	274039.968394
1	RaleighGreensboro	166234.681861
2	SouthCarolina	206177.799708
3	Nashville	121025.995657
4	Atlanta	307191.015328

4-3. 가격과 판매량 간 그래프 그리기

2) 가격과 판매량의 KDE그래프

2) joint plot을 이용한 평균가격과 평균판매량에 대한 kde그래프



4-4. 지역과 Bags 간 그래프 그리기

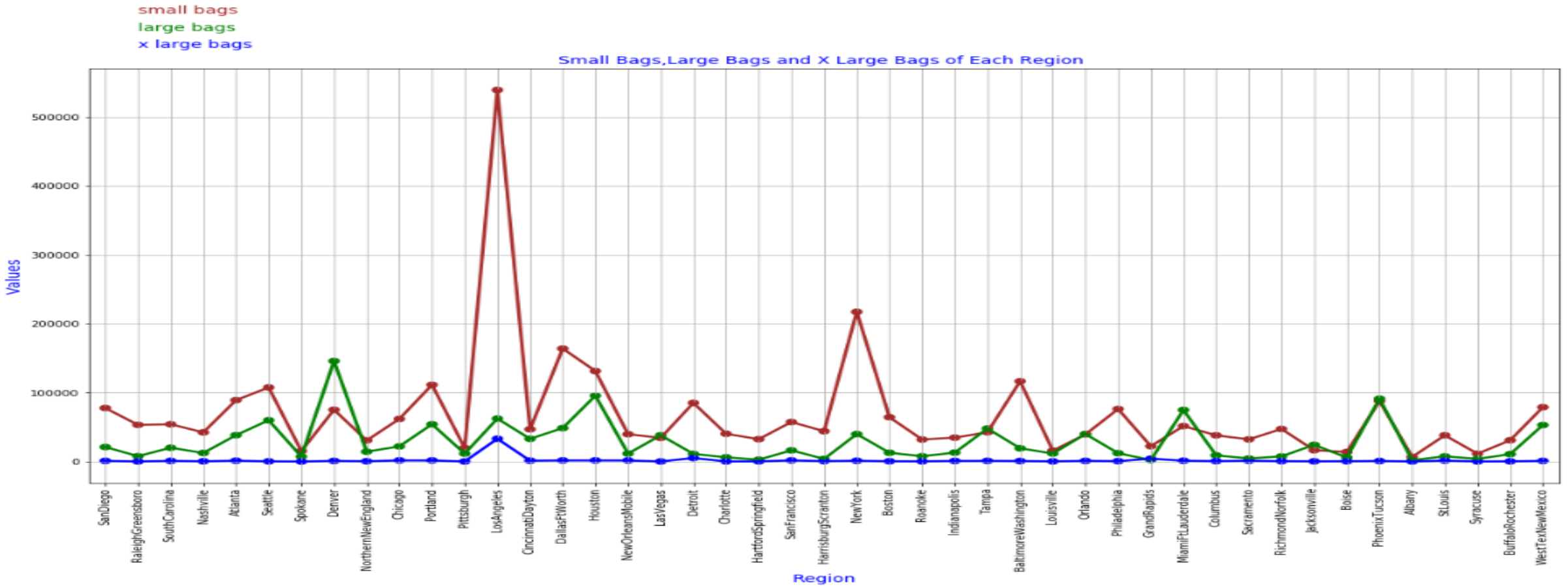
1) 지역간 Bags의 그래프

```
small=[]
large=[]
xlarge=[]

for i in region_list:
    x=df[df.region==i]
    small.append(sum(x['Small Bags'])/len(x))
    large.append(sum(x['Large Bags'])/len(x))
    xlarge.append(sum(x['XLarge Bags'])/len(x))
df5=pd.DataFrame({'region_list':region_list,'small':small,'large':large,'xlarge':xlarge})

f,ax1=plt.subplots(figsize=(20,10))
sns.pointplot(x=region_list,y=small,data=df5,color='brown',alpha=0.7)
sns.pointplot(x=region_list,y=large,data=df5,color='green',alpha=0.7)
sns.pointplot(x=region_list,y=xlarge,data=df5,color='blue',alpha=0.7)

plt.xticks(rotation=90)
plt.text(1,650000,'small bags',color='brown',fontsize=14)
plt.text(1,625000,'large bags',color='green',fontsize=14)
plt.text(1,600000,'x large bags', color='blue',fontsize=14)

plt.xlabel('Region',color='blue',fontsize=14)
plt.ylabel('Values',color='blue',fontsize=14)
plt.title('Small Bags,Large Bags and X Large Bags of Each Region ',color='blue',fontsize=14)
plt.grid()
```

4-3. 가격과 판매량 간 그래프 그리기

1) 지역간 Bags의 그래프

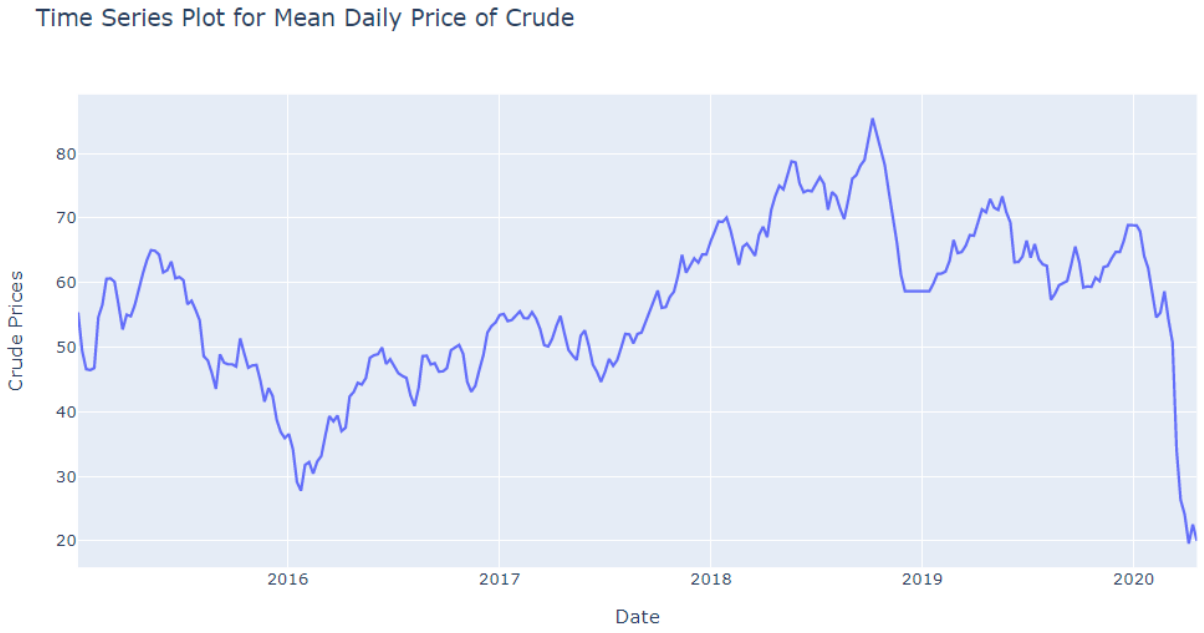
LosAngeles는 small bags를 주로 판매하는 것을 알 수 있음



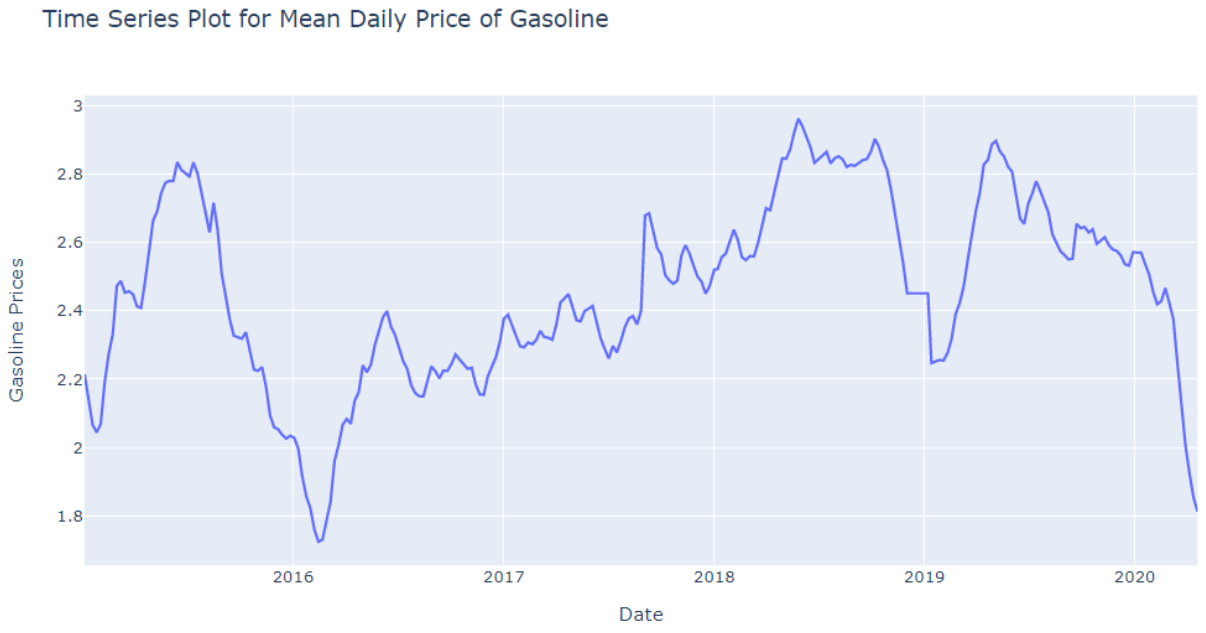
5. 유가와 가솔린 가격 그래프 그리기

1) 유가와 가솔린의 가격 시계열 그래프 그리기

유가 가격



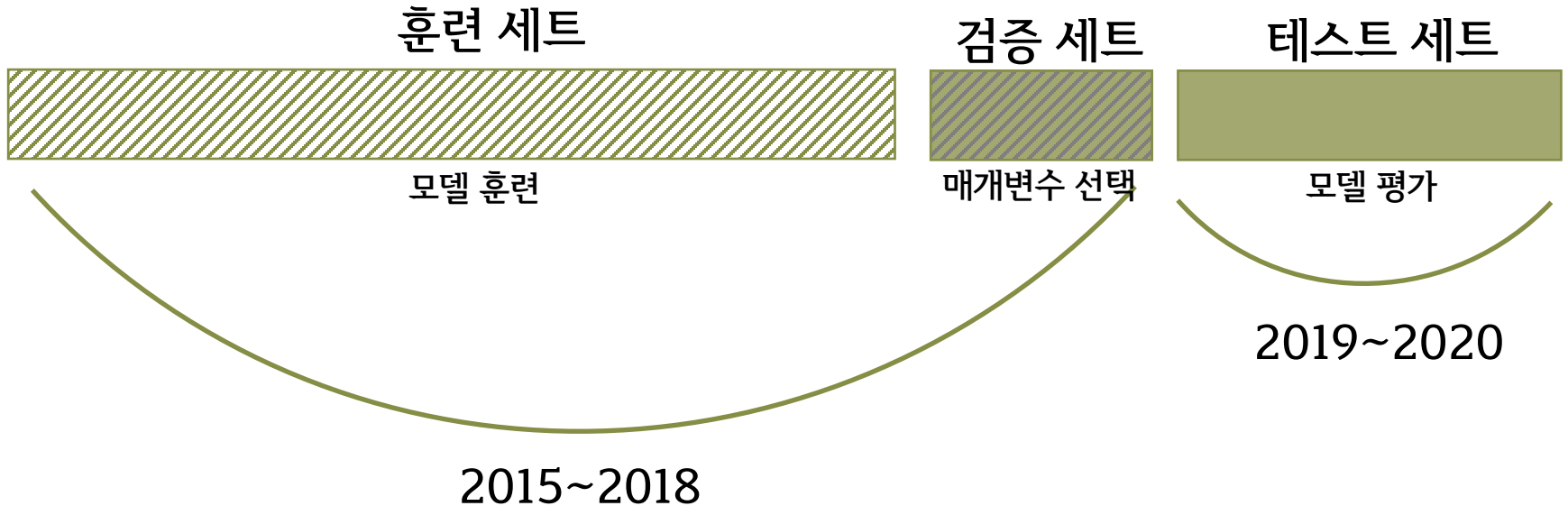
가솔린 가격



5. 모델링

5-1. Traindata와 Testdata 나누기

1) Traindata와 Testdata 나누기



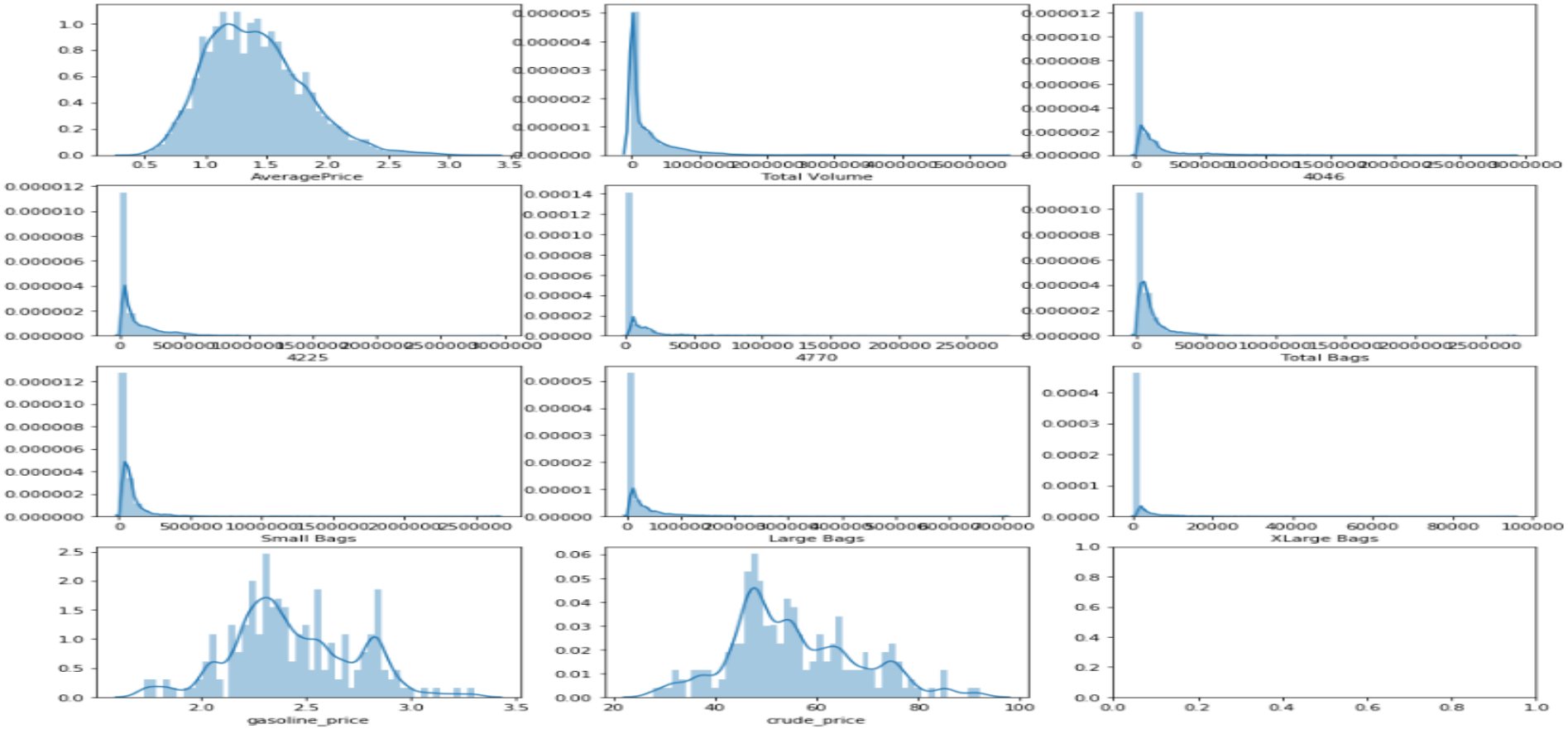
5-2. 연속형 특성 히스토그램 그리기

1) 연속형 특성 히스토그램 그리기

```
names =['AveragePrice', 'Total Volume', '4046', '4225', '4770',
       'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags','gasoline_price', 'crude_price']

import matplotlib.pyplot as plt
import seaborn as sns
fig, axes = plt.subplots(4,3)
fig.set_size_inches(15,15)
axes = axes.ravel()

for names,i in zip(names,range(11)):
    sns.distplot(avo1[names],ax=axes[i])
```

5-2. 연속형 특성 히스토그램 그리기

1) 연속형 특성 히스토그램 그리기



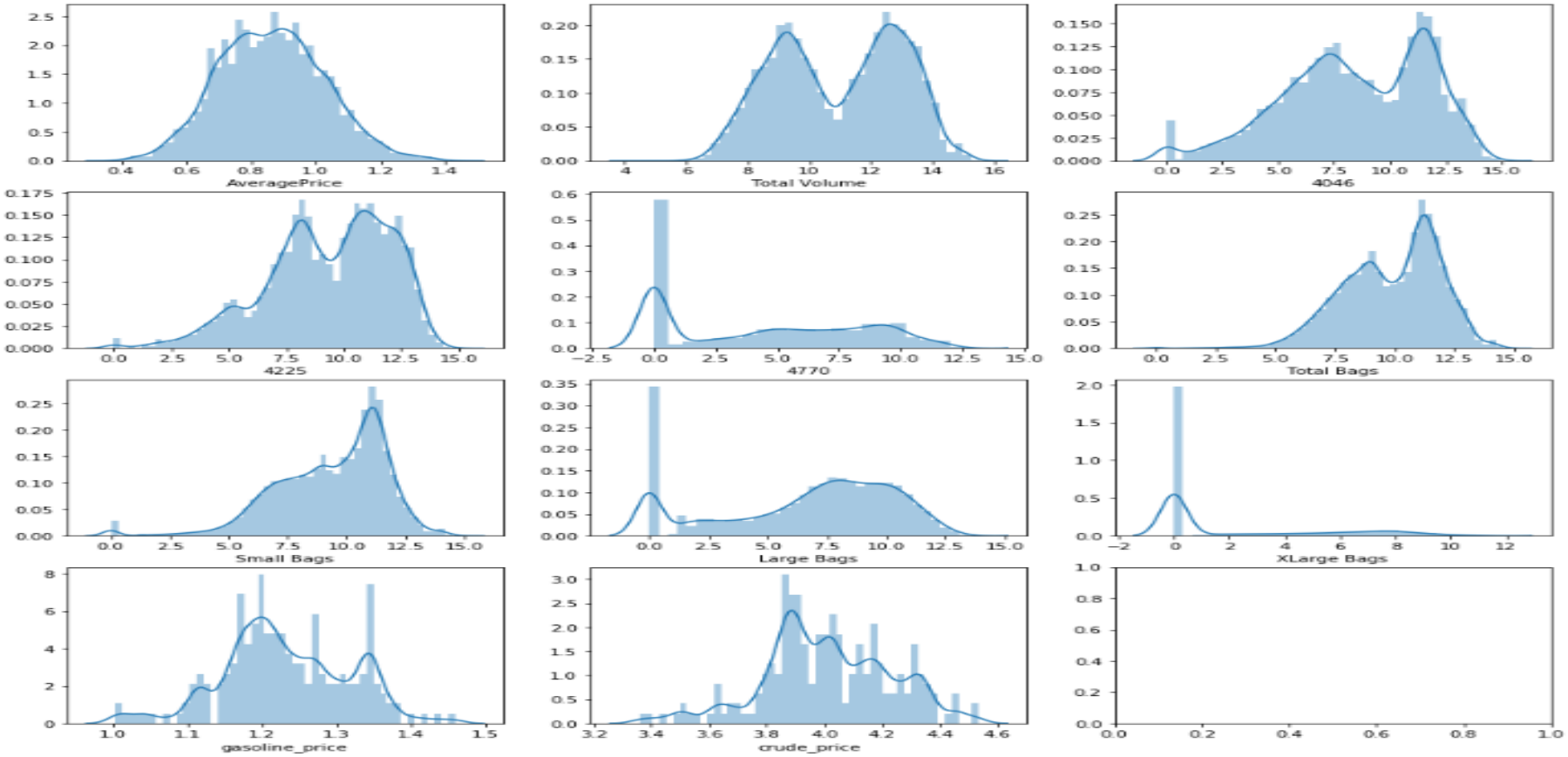
5-2. 연속형 특성 히스토그램 그리기

2) 연속형 변수 특성에 log 정규화

```
import numpy as np
X_names = ['Total Volume', '4046', '4225', '4770',
       'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags','gasoline_price', 'crude_price']
# X 값 log 씌우기
for names in X_names:
    X[names] = np.log1p(X[names])
# Y 값 log 씌우기
y = np.log1p(y)
```

5-2. 연속형 특성 히스토그램 그리기

2) 연속형 변수 특성에 log 정규화



5-3. 회귀 모델 평가 지표

1) 회귀 모델 평가 지표

평가 지표	설명	수식
MAE	Mean Absolute Error(MAE)이며 실제 값과 예측값의 차이를 절댓값으로 변환해 평균한 것입니다	$MAE = \frac{1}{n}\sum_{i=1}^{n}\|Yi - \hat{Y}i\|$
MSE	Mean Squared Error(MSE)이며 실제 값과 예측값의 차이를 제곱해 평균한 것입니다.	$MSE = \frac{1}{n}\sum_{i=1}^{n}(Yi - \hat{Y}i)^2$
MSLE	MSE에 로그를 적용한 것입니다. 결정값이 클 수록 오류값도 커지기 때문에 일부 큰 오류값들로 인해 전체 오류값이 커지는 것을 막아줍니다.	Log(MSE)
RMSE	MSE 값은 오류의 제곱을 구하므로 실제 오류 평균보다 더 커지는 특성이 있으므로 MSE에 루트를 씌운 것이 RMSE(Root Mean Squared Error)입니다	$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Yi - \hat{Y}i)^2}$
RMSLE	RMSE에 로그를 적용한 것입니다. 결정값이 클 수록 오류값도 커지기 때문에 일부 큰 오류값들로 인해 전체 오류값이 커지는 것을 막아줍니다.	Log(RMSE)
R^2	분산 기반으로 예측 성능을 평가합니다. 실제 값의 분산 대비 예측값의 분산 비율을 지표로 하며, 1에 가까울수록 예측 정확도가 높습니다.	$R^2 = \frac{\text{예측값 } Variance}{\text{실제 값 } Variance}$

5-4. 교차 검증

1) 여러 가지 모델 만들기

- LinearRegression
- Ridge
- Lasso
- DecisionTree
- RandomForest
- GradientBoostingTree
- SVM
- XGBoost
- LightGBM

5-4. 교차 검증

2) 임의 분할 교차 검증

- 임의 분할 교차 검증은 반복 횟수를 훈련 세트나 테스트 세트의 크기와 독립적으로 조절해야 할 때 유용.
- train_size 와 test_size의 합을 전체와 다르게 함으로써 전체 데이터의 일부만 사용할 수 있음.
- 데이터를 부분 샘플링하는 방식 : 대규모 데이터 셋으로 작업할 때 도움
- 데이터에서 무작위로 추출하여 훈련 세트와 테스트 세트를 만드므로 random_state 매개변수를 지정하지 않으면 실행할 때마다 결과가 다르게 다오므로 random_state를 설정해줘야 함.

5-4. 교차 검증

3) 임의 분할 교차 검증 함수 만들기

```
def get_model_cv_prediction(model, X, y):
    shuffle_split = ShuffleSplit(test_size=.5,train_size = .5,n_splits= 10,random_state=0)
    Reg = cross_validate(model, X, y, scoring =["neg_mean_squared_error","r2"] ,cv= shuffle_split,return_train_score=True)
    rmse_scores = np.sqrt(-1*Reg['test_neg_mean_squared_error'])
    avg_rmse = np.mean(rmse_scores)
    avg_train = np.mean(Reg['train_r2'])
    avg_r2 = np.mean(Reg['test_r2'])
    print('##### ',model.__class__.__name__ , ' #####')
    print(' 5교차 검증 개별 RMSE :{0:.3f}'.format(np.round(avg_rmse, 3)))
    print(' 5교차 train 수정결정계수 R2 :{0:.3f}'.format(avg_train))
    print(' 5교차 test 수정결정계수 R2 :{0:.3f} '.format(avg_r2))
```

5-4. 교차 검증

4) 모델 별 검증 데이터 비교

LGBM Regressor이

RMSE가 가장 낮고

수정결정계수R2가 높게 나옴

```
#####  LGBMRegressor  #####
 5교차 검증 개별 RMSE :0.045
 5교차 train 수정결정계수 R2 :0.979
 5교차 test 수정결정계수 R2 :0.924
```

```
#####  LinearRegression  #####
 5교차 검증 개별 RMSE :0.086
 5교차 train 수정결정계수 R2 :0.727
 5교차 test 수정결정계수 R2 :0.721
#####  Ridge  #####
 5교차 검증 개별 RMSE :0.087
 5교차 train 수정결정계수 R2 :0.721
 5교차 test 수정결정계수 R2 :0.714
#####  Lasso  #####
 5교차 검증 개별 RMSE :0.088
 5교차 train 수정결정계수 R2 :0.714
 5교차 test 수정결정계수 R2 :0.708
#####  DecisionTreeRegressor  #####
 5교차 검증 개별 RMSE :0.082
 5교차 train 수정결정계수 R2 :1.000
 5교차 test 수정결정계수 R2 :0.746
#####  RandomForestRegressor  #####
 5교차 검증 개별 RMSE :0.055
 5교차 train 수정결정계수 R2 :0.984
 5교차 test 수정결정계수 R2 :0.884
#####  GradientBoostingRegressor  #####
 5교차 검증 개별 RMSE :0.049
 5교차 train 수정결정계수 R2 :0.971
 5교차 test 수정결정계수 R2 :0.910
#####  SVR  #####
 5교차 검증 개별 RMSE :0.135
 5교차 train 수정결정계수 R2 :0.315
 5교차 test 수정결정계수 R2 :0.312
#####  XGBRegressor  #####
 5교차 검증 개별 RMSE :0.053
 5교차 train 수정결정계수 R2 :0.943
 5교차 test 수정결정계수 R2 :0.895
#####  LGBMRegressor  #####
 5교차 검증 개별 RMSE :0.045
 5교차 train 수정결정계수 R2 :0.979
 5교차 test 수정결정계수 R2 :0.924
```

5-5. 적합한 scale 조정 방법 찾기

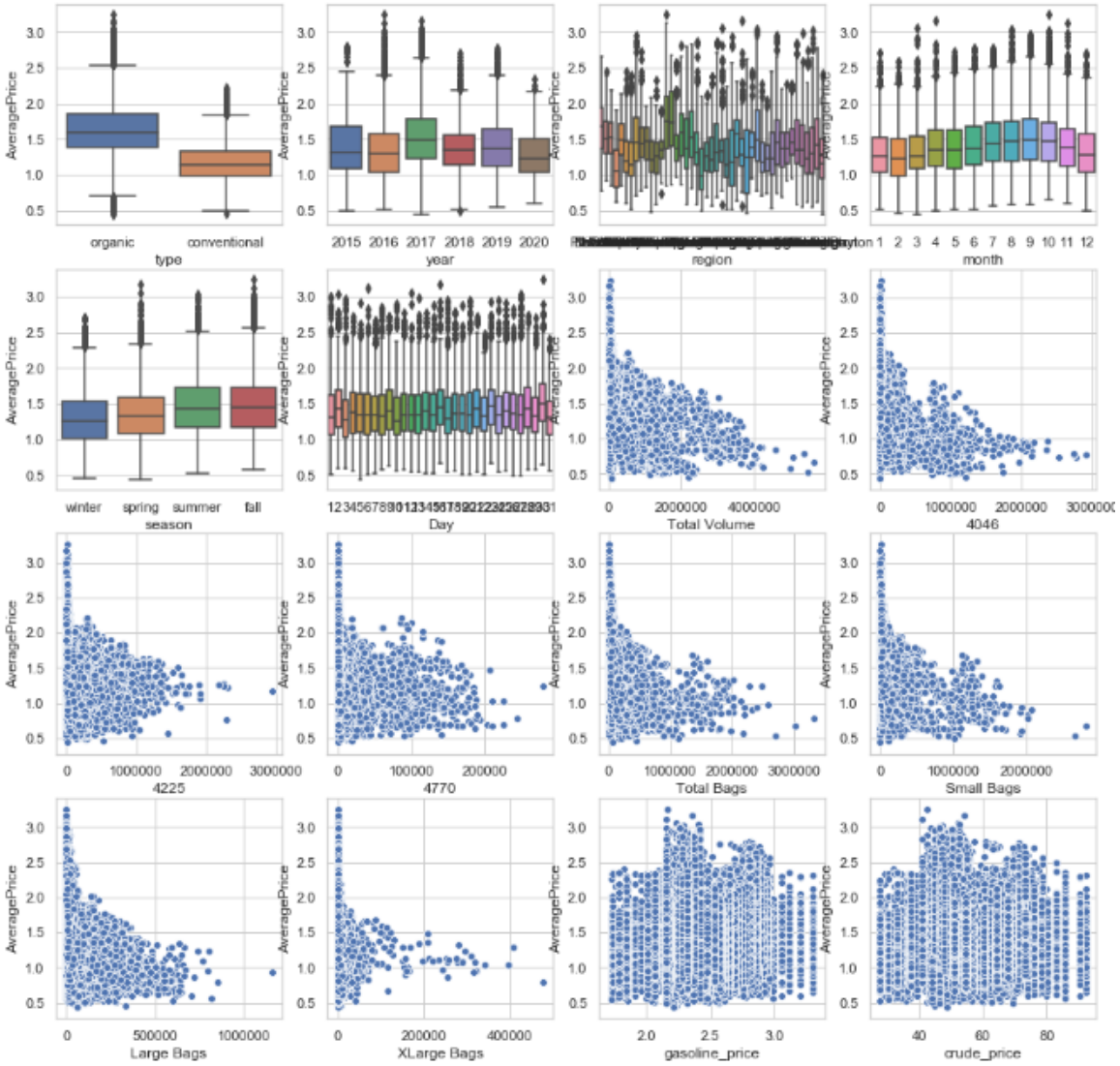
1) 이상치 유무 확인하기

```
conti = ['Total Volume', '4046', '4225','4770',
         'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags',
         'gasoline_price', 'crude_price']

categ=['type', 'year', 'region', 'month', 'season','Day']
```

```
fig = plt.figure(figsize=(15,15))

for i in range(0,len(categ)):
    fig.add_subplot(4,4,i+1)
    sns.boxplot(x= categ[i],y='AveragePrice', data = av_price)

for col in conti:
    fig.add_subplot(4,4,i+2)
    sns.scatterplot(x = av_price[col],y='AveragePrice', data = av_price)
    i+=1

plt.show()
fig.clear()
```

5-5. 적합한 scale 조정 방법 찾기

1) 이상치 유무 확인하기



연속형data와 범주형data의 이상치가 많이 관측된 것으로 확인

-> Robust Scaler 나 Quantile Transformer 사용 결정

5-5. 적합한 scale 조정 방법 찾기

2) 이상치에 영향을 받지 않는 스케일 조정 방법

- Robust Scaler

중앙값과 IQR사용. 아웃라이어의 영향 최소화

- Quantile Transformer

1000개의 분위를 사용하여 데이터를 균등하게 분포 시킴. 이상치에 민감하지 않고 전체데이터를 0과 1사이로 압축

5-6. scale 조정

1) scale 조정

선형 모델과 SVR만 확인
트리모형은 영향 X

```
rb = RobustScaler()
X_rb = rb.fit_transform(X)

qt= QuantileTransformer()
X_qt = qt.fit_transform(X)

qt_normal= QuantileTransformer(output_distribution ='normal')
X_qt_normal = qt_normal.fit_transform(X)
```

5-6. scale 조정

1) scale 조정

```
models = [lr,ridge,lasso,svr]
for model in models:
    get_model_cv_prediction(model, X_rb, y)
get_model_cv_prediction(gbrt,X,y)
```

```
#####  LinearRegression  #####
 5교차 검증 개별 RMSE :0.086
 5교차 train 수정결정계수 R2 :0.727
 5교차 test 수정결정계수 R2 :0.721
#####  Ridge  #####
 5교차 검증 개별 RMSE :0.087
 5교차 train 수정결정계수 R2 :0.721
 5교차 test 수정결정계수 R2 :0.714
#####  Lasso  #####
 5교차 검증 개별 RMSE :0.088
 5교차 train 수정결정계수 R2 :0.714
 5교차 test 수정결정계수 R2 :0.708
#####  SVR  #####
 5교차 검증 개별 RMSE :0.061
 5교차 train 수정결정계수 R2 :0.881
 5교차 test 수정결정계수 R2 :0.857
```

```
models = [lr,ridge,lasso,svr]
for model in models:
    get_model_cv_prediction(model, X_qt, y)
get_model_cv_prediction(gbrt,X,y)
```

```
#####  LinearRegression  #####
 5교차 검증 개별 RMSE :0.087
 5교차 train 수정결정계수 R2 :0.716
 5교차 test 수정결정계수 R2 :0.710
#####  Ridge  #####
 5교차 검증 개별 RMSE :0.088
 5교차 train 수정결정계수 R2 :0.712
 5교차 test 수정결정계수 R2 :0.706
#####  Lasso  #####
 5교차 검증 개별 RMSE :0.089
 5교차 train 수정결정계수 R2 :0.703
 5교차 test 수정결정계수 R2 :0.696
#####  SVR  #####
 5교차 검증 개별 RMSE :0.063
 5교차 train 수정결정계수 R2 :0.870
 5교차 test 수정결정계수 R2 :0.848
```

```
models = [lr,ridge,lasso,svr]
for model in models:
    get_model_cv_prediction(model, X_qt_normal, y)
get_model_cv_prediction(gbrt,X,y)
```

```
#####  LinearRegression  #####
 5교차 검증 개별 RMSE :0.093
 5교차 train 수정결정계수 R2 :0.682
 5교차 test 수정결정계수 R2 :0.675
#####  Ridge  #####
 5교차 검증 개별 RMSE :0.093
 5교차 train 수정결정계수 R2 :0.681
 5교차 test 수정결정계수 R2 :0.674
#####  Lasso  #####
 5교차 검증 개별 RMSE :0.093
 5교차 train 수정결정계수 R2 :0.677
 5교차 test 수정결정계수 R2 :0.669
#####  SVR  #####
 5교차 검증 개별 RMSE :0.072
 5교차 train 수정결정계수 R2 :0.829
 5교차 test 수정결정계수 R2 :0.804
```

5-7. Base model 선정

1) Base model 선정 – LGBM

LGBM Regressor

```
import lightgbm as lgb
lgb = lgb.LGBMRegressor(objective='regression',
                        learning_rate=0.05, n_estimators=1000)

lgb.fit(X,y)
```

```
LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
              importance_type='split', learning_rate=0.05, max_depth=-1,
              min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
              n_estimators=1000, n_jobs=-1, num_leaves=31,
              objective='regression', random_state=None, reg_alpha=0.0,
              reg_lambda=0.0, silent=True, subsample=1.0,
              subsample_for_bin=200000, subsample_freq=0)
```

```
get_model_cv_prediction(lgb, X, y)
```

```
#####  LGBMRegressor  #####
 5교차 검증 개별 RMSE :0.045
 5교차 train 수정결정계수 R2 :0.979
 5교차 test 수정결정계수 R2 :0.924
```

간단한 LGBM Regressor 모델 설명

- 더 빠른 학습과 예측 수행 시간
- 더 작은 메모리 사용량
- 카테고리형 피처의 자동 변환과 최적 분할
  (원-핫 인코딩을 사용하지 않고도
  카테고리형 피처를 최적으로 변환하고
  이에 따른 노드 분할 수행)

5-7. Base model 선정

2) 중요 매개변수 선정

- learning_rate
  : 학습률. 0에서 1 사이의 값을 지정하여 부스팅 스텝을 반복적으로 수행할 때 업데이트되는 학습률 값.

- n_estimators
  : 약한 학습기의 개수(반복 수행 횟수)

5-7. Base model 선정

3) 최적의 파라미터 선정

learning_rate : [0.1, 0.001, 0.05, 0.01, 0.005],
최적의 파라미터 : 0.1
n_estimators : [100,500,1000],
최적의 파라미터 : 1000

5교차 검증 개별 RMSE : 0.044
5교차 train 수정결정계수 R2 : 0.993
5교차 test 수정결정계수 R2 : 0.928

5-8. Base model 비교 모델

1) XGBoost 매개변수

- n_estimators
  : 트리의 개수, 값을 키우면 앙상블에 트리가 더 많이 추가되어 모델의 복잡도가 커지고 훈련 세트에서의 실수를 바로잡을 기회가 많아진다.

- max_depth
  : 트리의 최대 깊이, 값을 줄이면 과대적합을 줄일 수 있다.

5-8. Base model 비교 모델

1) XGBoost 매개변수

- min_child_weight
  : 필요한 모든 관측치의 최소 가중치 합계를 정의 [default=1]

- Gamma
  : 분할에 필요한 최소 손실 감소를 지정 [default=0]

- Subsample
  : 관측치의 비율이 각 나무에 대해 무작위로 표본임을 나타냄 [default=1]

5-8. Base model 비교 모델

2) XGBoost 최적의 파라미터 선정

max_depth : range (3,10,2)
최적의 파라미터 : 7
min_child_weight : range(1,6,2)
최적의 파라미터 : 3
gamma : [i/10.0 for I in range(0,5)]
최적의 파라미터 : 0.0
subsample : [i/100.0 for I in range(75,90,5)]
최적의 파라미터 : 0.8
reg_alpha : [0, 0.001, 0.005, 0.01, 0.05]
최적의 파라미터 : 0

5교차 검증 개별 RMSE :0.047
5교차 train 수정결정계수 R2 :0.979
5교차 test 수정결정계수 R2 :0.915

5-9. 가격 예측하기

1) 2019~2020 Average Price 예측하기

```
avo2 = avocado[avocado['Date'] > '2019-01-01']
X_test = avo2.drop(['Date','AveragePrice'],axis=1)
X_test = pd.get_dummies(X_test)
y2 = avo2['AveragePrice']
```

```
import numpy as np
X_names = ['Total Volume', '4046', '4225', '4770',
       'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags',
           'gasoline_price', 'crude_price']
# X 값 log 씌우기
for names in X_names:
    X_test[names] = np.log1p(X_test[names])
# Y 값 log 씌우기
y2 = np.log1p(y2)
```

5-9. 가격 예측하기

1) 2019~2020 Average Price 예측하기

```
rb = RobustScaler()
X_rb = rb.fit_transform(X)
X_test_rb = rb.transform(X_test)
```

```
from sklearn.linear_model import LinearRegression
linear_final = LinearRegression()
```

```
from sklearn.ensemble import RandomForestRegressor
forest_final = RandomForestRegressor(n_estimators = 100, random_state=0,n_jobs=-1)
```

```
xgb_model_best.fit(X,y)
lgb_final.fit(X,y)
linear_final.fit(X_rb,y)
forest_final.fit(X,y)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=-1, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

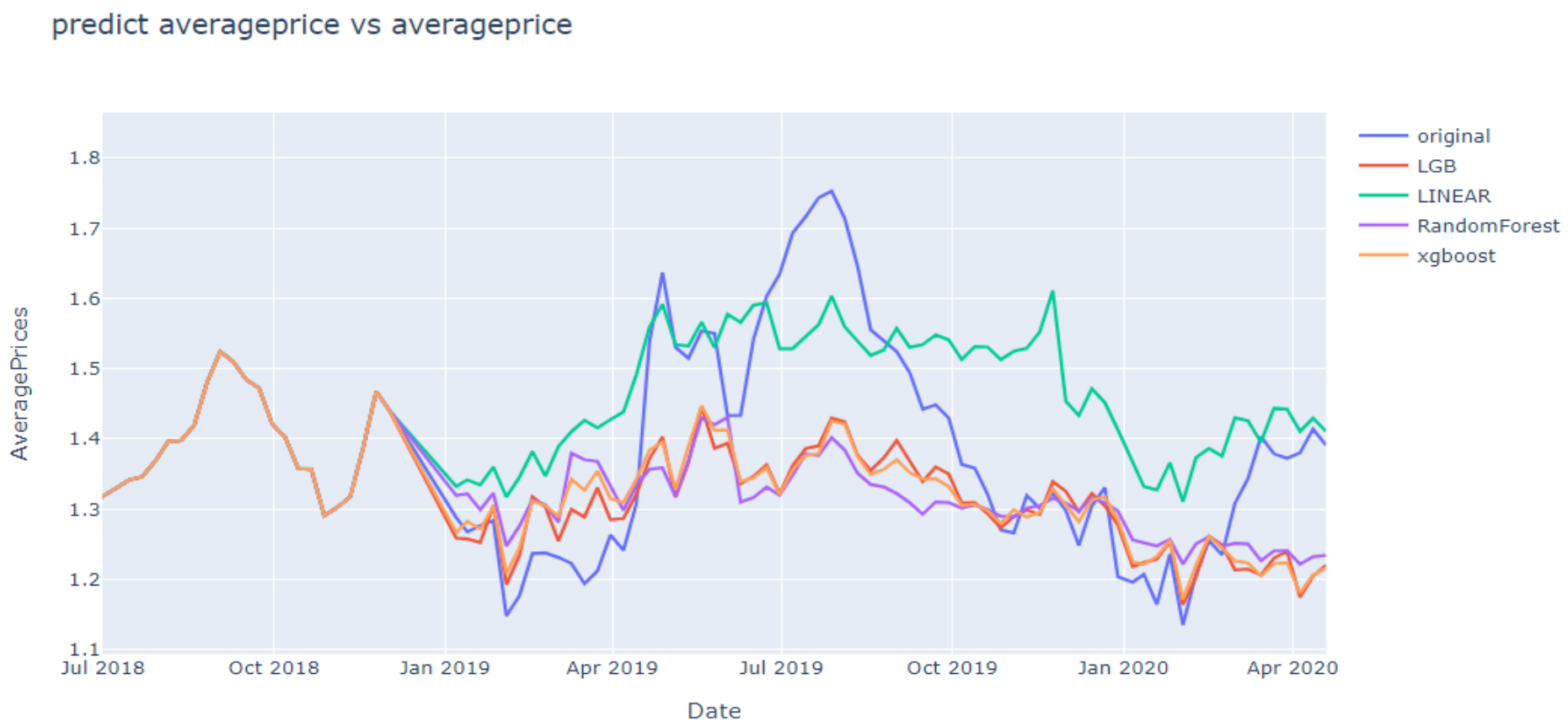
5-9. 가격 예측하기

2) 시계열 그래프로 실제 가격과 예측모델 그래프 비교

```
import plotly
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
init_notebook_mode(connected=True)

#원본자료
groupBy_whole = avocado.groupby('Date').mean()
avocado_scatter = go.Scatter(x=groupBy_whole.AveragePrice.index, y=groupBy_whole.AveragePrice,name ='original')

#예측자료
avo2['AveragePrice'] = np.expm1(linear_final.predict(X_test_rb))
linear_avocado = pd.concat([avo1] + [avo2], axis = 0, ignore_index = True)
groupBy_linear = linear_avocado.groupby('Date').mean()
LINEAR = go.Scatter(x=groupBy_linear.AveragePrice.index, y=groupBy_linear.AveragePrice,name='LINEAR')

avo2['AveragePrice'] = np.expm1(forest_final.predict(X_test))
forest_avocado = pd.concat([avo1] + [avo2], axis = 0, ignore_index = True)
groupBy_forest = forest_avocado.groupby('Date').mean()
RandomForest = go.Scatter(x=groupBy_forest.AveragePrice.index, y=groupBy_forest.AveragePrice,name='RandomForest')

avo2['AveragePrice'] = np.expm1(xgb_model_best.predict(X_test))
xgb_avocado = pd.concat([avo1] + [avo2], axis = 0, ignore_index = True)
groupBy_xgb = xgb_avocado.groupby('Date').mean()
xgBoosting = go.Scatter(x=groupBy_xgb.AveragePrice.index, y=groupBy_xgb.AveragePrice,name='xgboost')

avo2['AveragePrice'] = np.expm1(lgb_final.predict(X_test))
lgb_avocado = pd.concat([avo1] + [avo2], axis = 0, ignore_index = True)
groupBy_lgb = lgb_avocado.groupby('Date').mean()
LGB = go.Scatter(x=groupBy_lgb.AveragePrice.index, y=groupBy_lgb.AveragePrice,name='LGB')

data = [avocado_scatter, LGB, LINEAR, RandomForest,xgBoosting]
layout=go.Layout(title="predict averageprice vs averageprice", xaxis={'title':'Date'}, yaxis={'title':'AveragePrices'})
figure=go.Figure(data=data,layout=layout)
iplot(figure)
```

5-9. 가격 예측하기

2) 시계열 그래프로 실제 가격과 예측모델 그래프 비교



5-9. 가격 예측하기

3) 예측모델의 정확도

```
print("Linear의 score : ",linear_final.score(X_test_rb,y2),"₩n"
      "Randomforest의 score : ",forest_final.score(X_test,y2),"₩n"
      "xgb의 score : ",xgb_model_best.score(X_test,y2),"₩n"
      "lgb의 score : ",lgb_final.score(X_test,y2))
```

```
Linear의 score :  0.5101262031854356
Randomforest의 score :  0.5046802937835921
xgb의 score :  0.5992162262162954
lgb의 score :  0.6271528884350397
```

감사합니다