아보카도 가격 예측

2021208024 김지헌

Contents


PART
01
“
개요 및 필요성
”

PART 01

개요 및 필요성

개요

아보카도 재배 방식, 지역별, 시기별에 따른 가격 파악

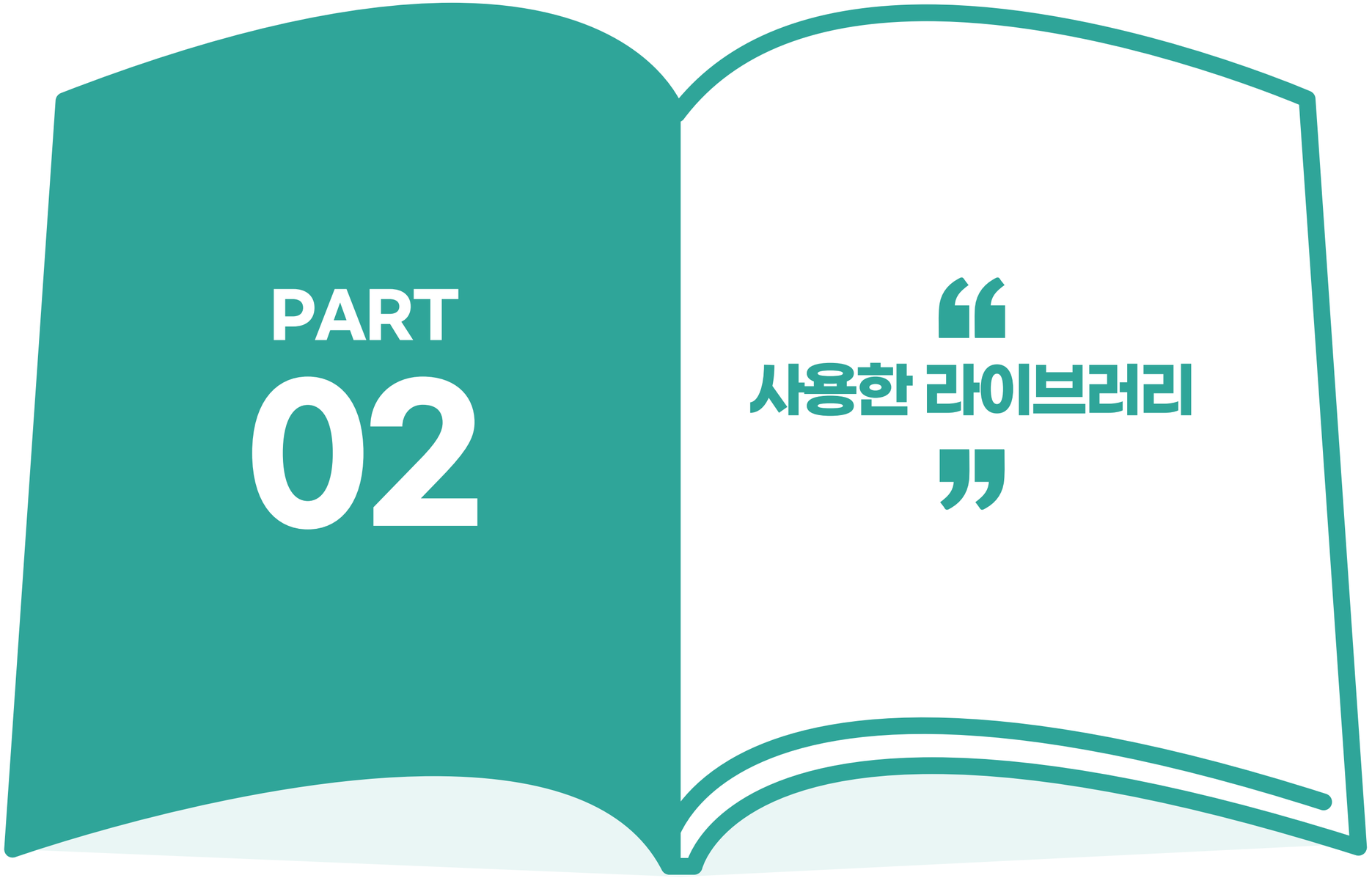
전처리를 통한 데이터셋 정리

알고리즘을 통해 가격 예측

필요성

합리적인 가격에 아보카드를 구매할 수 있음

아보카도의 가격 변동 기준을 파악할 수 있음


PART
02
“
사용한 라이브러리
”

PART 02

사용한 라이브러리

고성능의 다차원 배열 객체와 이를 다룰 도구를 제공하는 라이브러리

데이터를 전처리하거나 통계 처리, 분석을 위한 라이브러리.

Matplotlib을 기반dp 데이터 시각화 라이브러리

데이터를 다양한 그래프로 만들어주는 파이썬 라이브러리

사용한 라이브러리

그 외 추가적인 라이브러리

```
# 데이터 분석 라이브러리
import numpy as np
import pandas as pd
import datetime as dt

# 시각화 라이브러리
import matplotlib.pyplot as plt
from matplotlib import pyplot
import seaborn as sns
%matplotlib inline

# 시계열 분석 라이브러리
from fbprophet import Prophet

# 파이프라인 라이브러리
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
```


PART
03
“Data”

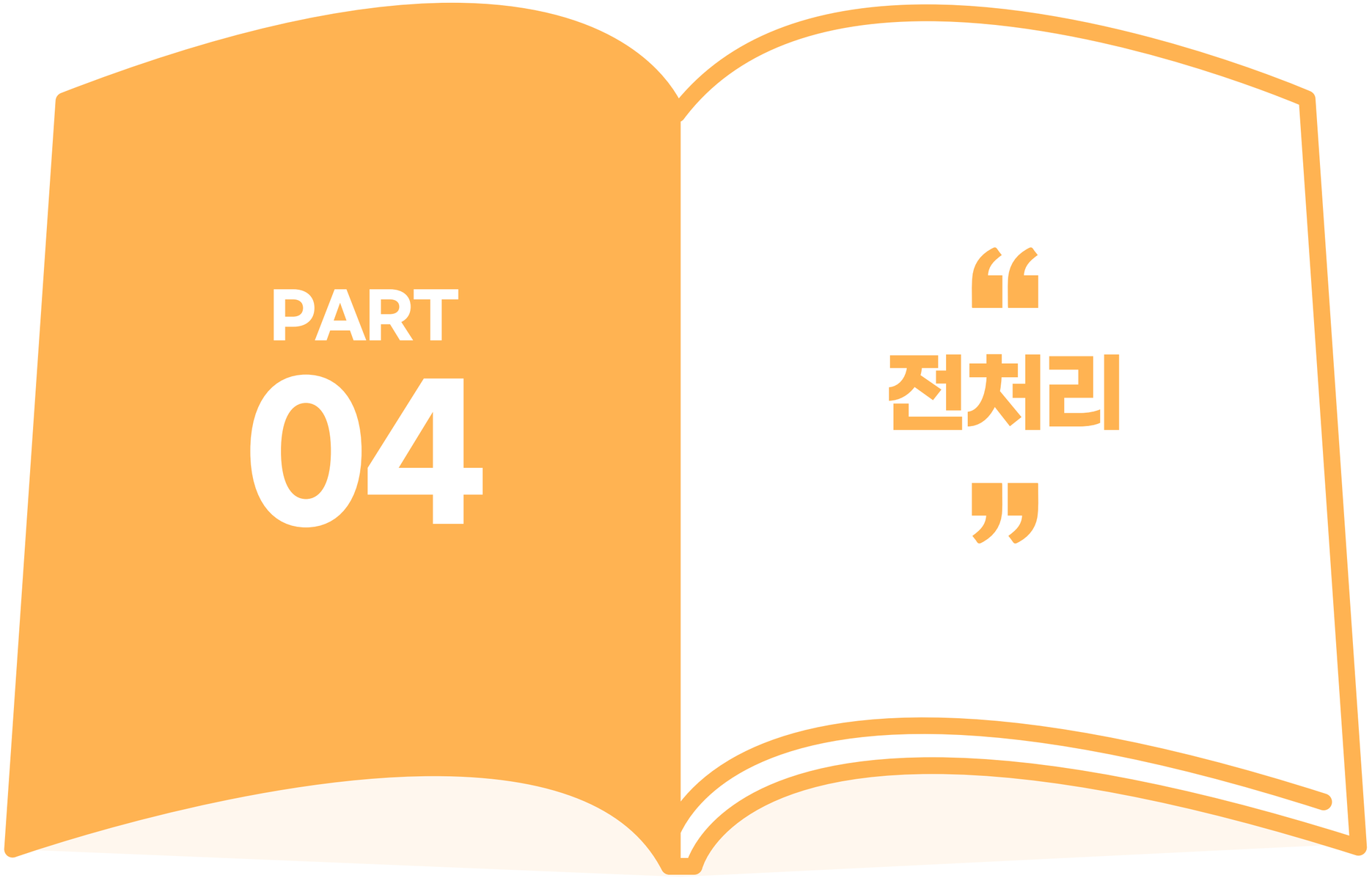
PART 03

Data

```
df = pd.read_csv('/kaggle/input/avocado-prices/avocado.csv')

#상위 5개 컬럼
df.head(5)
```

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	2015	Albany
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	2015	Albany
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	2015	Albany
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	2015	Albany
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	2015	Albany

- Date: 판매된 날짜
- AveragePrice : 아보카도 한개의 평균 가격
-Total Volume : 판매된 아보카도의 전체 수량
- 4046 : 판매된 PLU 4046 아보카도의 갯수 (작은 Hass 아보카도)
- 4225 : 판매된 PLU 4225 아보카도의 갯수 (큰 Hass 아보카도)
- 4770 : 판매된 PLU 4770 아보카도의 갯수 (매우 큰 Hass 아보카도)
- Total Bags : 포장되어 팔린 갯수
- Smal Bags : 작은 사이즈가 포장되어 팔린 갯수
- Large Bags : 큰 사이즈가 포장되어 팔린 갯수
- XLage Bags : 매우 큰 사이즈가 포장되어 팔린 갯수
- tpye : 일반(conventional) 또는 유기농(organic)
- year : 년도
- region : 판매된 도시


PART
04
“
전처리
”

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Unnamed: 0    18249 non-null  int64
 1   Date          18249 non-null  object
 2   AveragePrice  18249 non-null  float64
 3   Total Volume  18249 non-null  float64
 4   4046          18249 non-null  float64
 5   4225          18249 non-null  float64
 6   4770          18249 non-null  float64
 7   Total Bags    18249 non-null  float64
 8   Small Bags    18249 non-null  float64
 9   Large Bags    18249 non-null  float64
 10  XLarge Bags   18249 non-null  float64
 11  type          18249 non-null  object
 12  year          18249 non-null  int64
 13  region        18249 non-null  object
dtypes: float64(9), int64(2), object(3)
memory usage: 1.9+ MB
```

DataFrame.info() 함수를 사용하여 각 컬럼 명과 데이터 타입을 확인 할 수 있으면, 전체 데이터는 18,249개와 column은 14개로 구성되어 있고 결측치는 없다.

```
df = data[:]

#해당 분석에서 가격 예측시 필요 없어서 삭제
for col in ["Small Bags", "Large Bags", "XLarge Bags"]:
    del df[col]
```

아보카도 가격 예측 시에 필요없는 컬럼은 del 예약어를 사용하여 제거

PART 04

전처리

```
names = ["Unnamed: 0","Date", "AveragePrice", "Total Volume", "small", "large", "xlarge", "Total Bags", "t
df = df.rename(columns=dict(zip(df.columns, names)))


df.head()
```

	Unnamed: 0	Date	AveragePrice	Total Volume	small	large	xlarge	Total Bags	type	year	region
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	conventional	2015	Albany
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	conventional	2015	Albany
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	conventional	2015	Albany
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	conventional	2015	Albany
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	conventional	2015	Albany

데이터 사용을 용이하기 위해 일부 컬럼명 변경

전처리

```
# object 타입 데이터 살펴보기
df["type"].value_counts()
```

```
conventional    9126
organic         9123
Name: type, dtype: int64
```

중복 값 갯수 카운트(일반, 유기농)

```
df["region"].value_counts()
```

```
Pittsburgh              338
Orlando                 338
SanDiego                338
LosAngeles              338
HartfordSpringfield     338
Southeast               338
Sacramento              338
Detroit                 338
MiamiFtLauderdale       338
Boston                  338
TotalUS                 338
Nashville               338
Charlotte               338
Midsouth                338
Boise                   338
HarrisburgScranton      338
Jacksonville            338
Seattle                 338
West                    338
SouthCarolina           338
California              338
Indianapolis            338
Northeast               338
```

중복 값 갯수 카운트(지역)

```
# date 열의 데이터가 datatime 타입이 아니기 때문에, 날짜를 년, 월, 일로 나누기
dates = [dt.datetime.strptime(ts, "%Y-%m-%d") for ts in df['Date']]
dates.sort()
sorted_dates = [dt.datetime.strftime(ts, "%Y-%m-%d") for ts in dates]
df['Date'] = pd.DataFrame({'date':sorted_dates})
df['year'], df['month'], df['day'] = df['Date'].str.split('-').str
df.head()
```

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	type	year	region
0	0	2015-01-04	1.33	64236.62	1036.74	54454.85	48.16	8696.87	conventional	2015	Albany
1	1	2015-01-04	1.35	54876.98	674.28	44638.81	58.33	9505.56	conventional	2015	Albany
2	2	2015-01-04	0.93	118220.22	794.70	109149.67	130.50	8145.35	conventional	2015	Albany
3	3	2015-01-04	1.08	78992.15	1132.00	71976.41	72.58	5811.16	conventional	2015	Albany
4	4	2015-01-04	1.28	51039.60	941.48	43838.39	75.78	6183.95	conventional	2015	Albany

datetime 라이브러리를 이용하여 Date 열의 데이터를 년, 월, 일로 변경해준다.

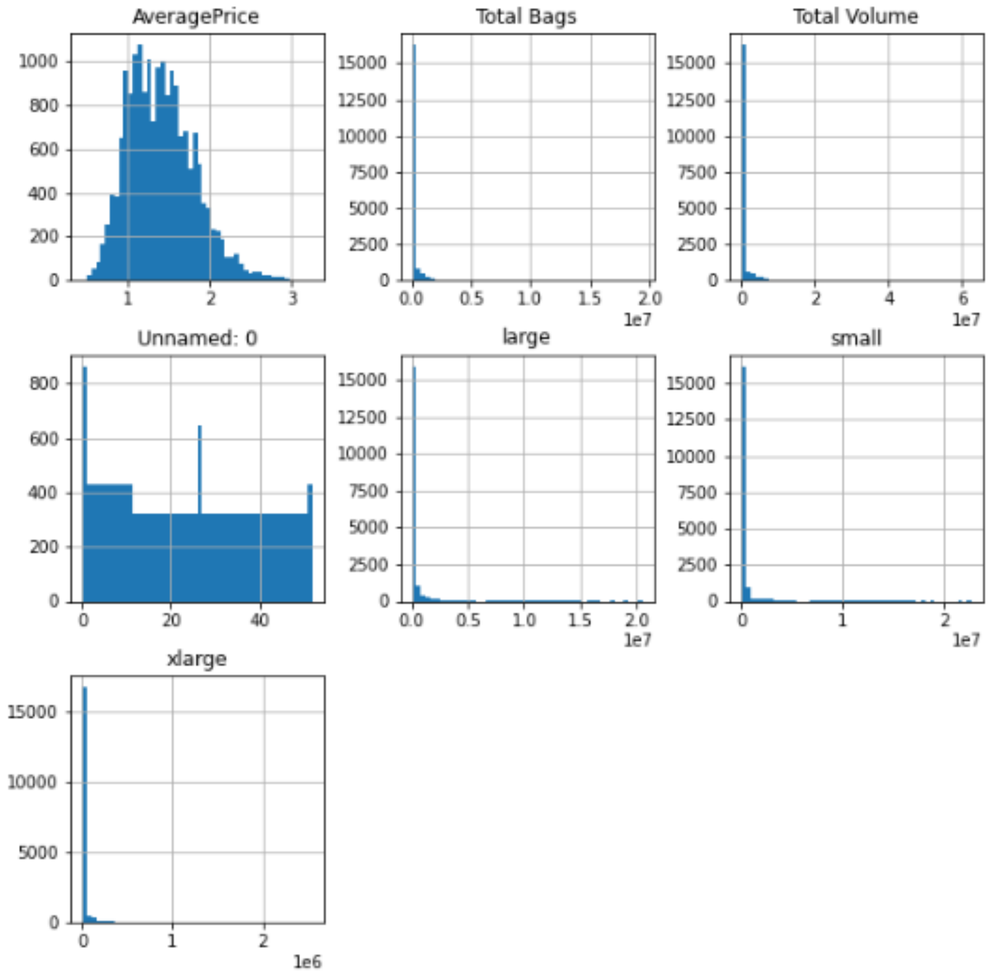
PART

05

“데이터 분석 및 시각화”

데이터 분석 및 시각화

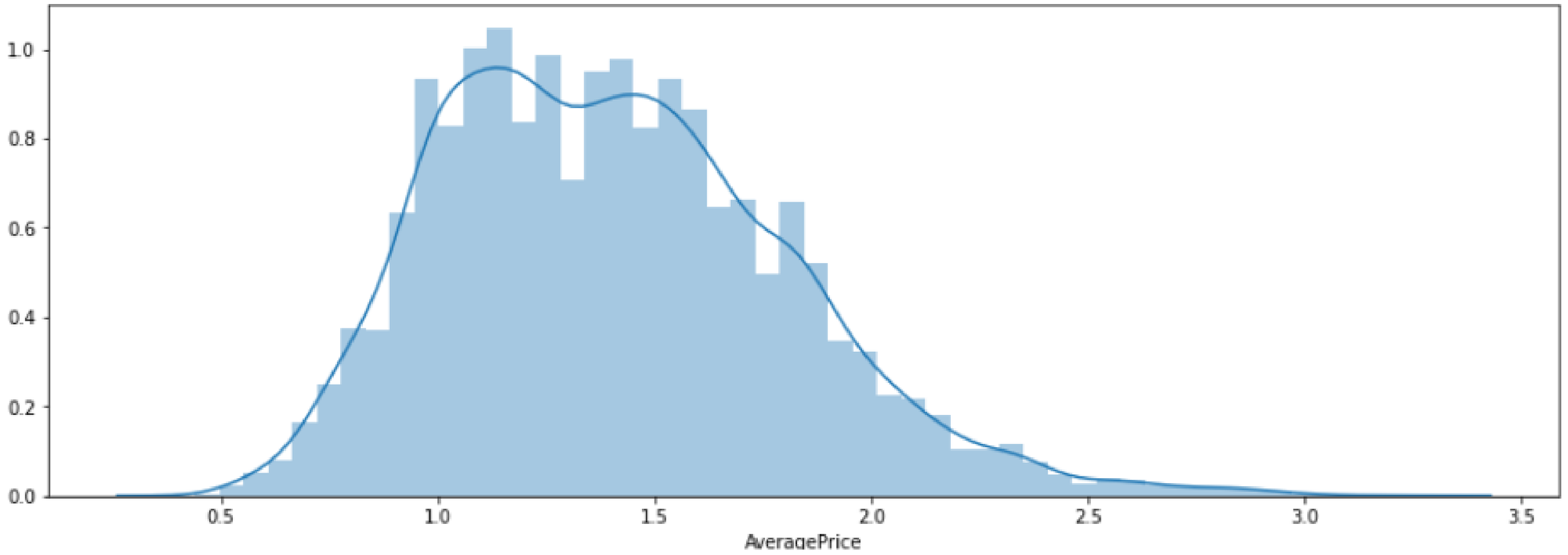
```
# 모든 float 타입 데이터 시각화하여 살펴보기
df.hist(bins=50, figsize=(10, 10))
plt.show()
```



수치형 변수의 분포는 전체적으로 오른쪽으로 꼬리가 긴 형태입니다.

데이터 분석 및 시각화

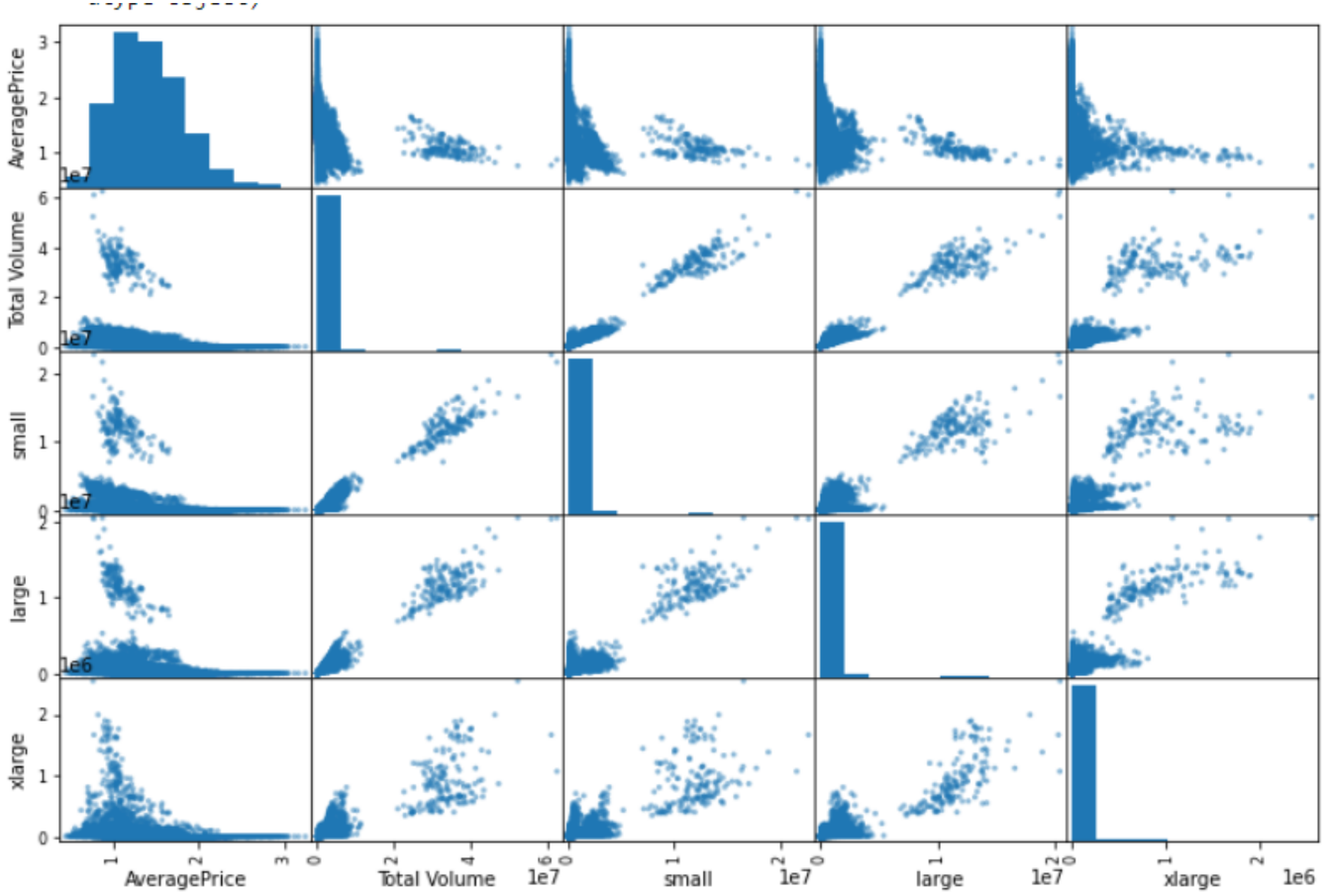
```
# 평균 가격 데이터만 시각화
plt.figure(figsize=(15,5))
ax = sns.distplot(df["AveragePrice"])
```



아보카도의 평균 가격이 대체적으로 1~1.5달러임을 알 수 있다.

데이터 분석 및 시각화

```
# 특성 사이의 상관관계 확인
from pandas.plotting import scatter_matrix
attributes = ["AveragePrice", "Total Volume", "small", "large", "xlarge"]
scatter_matrix(df[attributes], figsize=(12, 8))
```

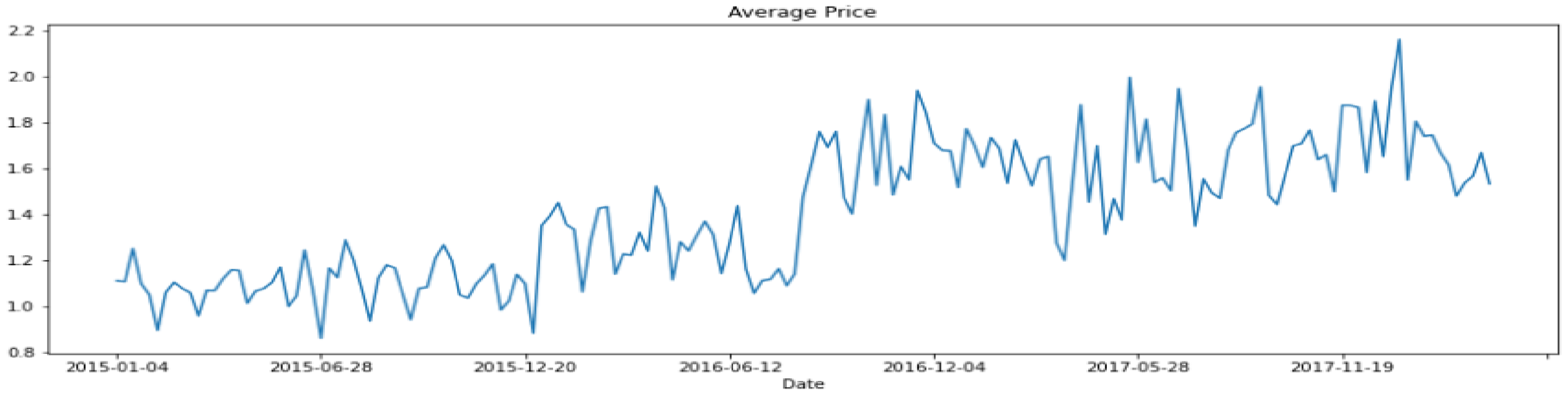


특성관 상관관계를 시각화한 자료이며 평균 가격이 1달러 언저리일 때, 모든 크기의 아보카도가 많이 팔리는 경향이 있습니다.
total volume, small, large, xlarge는 수량과 관련된 특성으로 1차 함수의 관계를 갖는다.

데이터 분석 및 시각화

```
# 날짜에 따른 가격 분포
price_date=df.groupby('Date').mean()
plt.figure(figsize=(15,5))
price_date['AveragePrice'].plot(x=df.Date)
plt.title('Average Price')
```

```
Text(0.5, 1.0, 'Average Price')
```

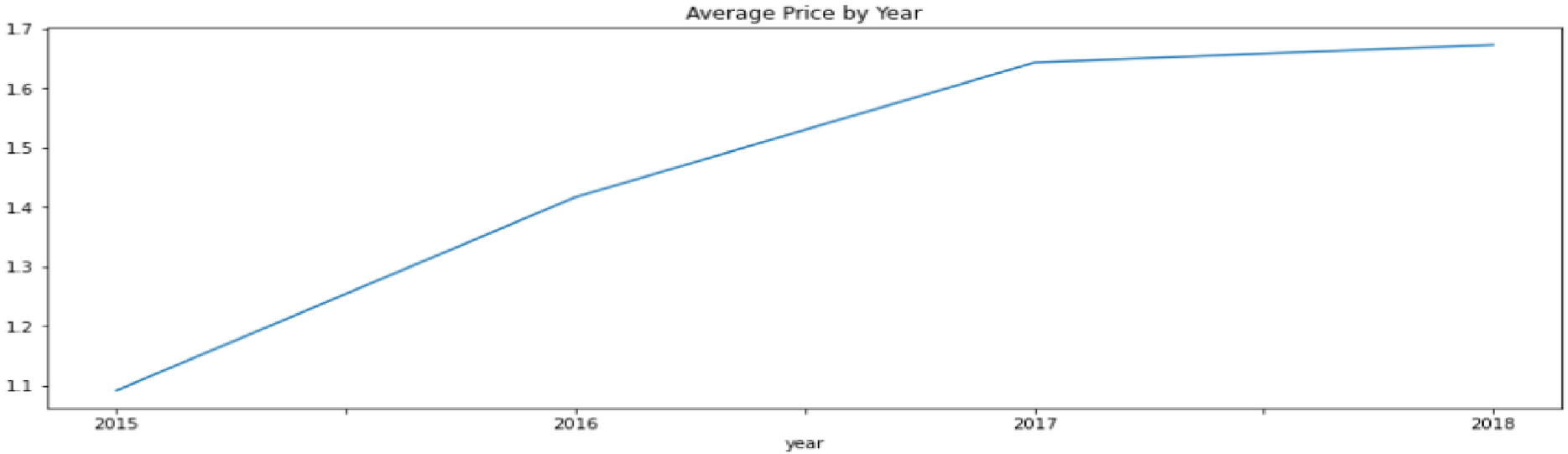


날씨에 다른 시각화 자료이며 아보카도 가격은 전체적으로 꾸준히 상승하는 경향을 보이나, 한달을 기준으로 증가했다 감소하는 반복적 추세를 보입니다.

데이터 분석 및 시각화

```
# 년도에 따른 가격 분포
price_year=df.groupby('year').mean()
fig, ax = plt.subplots(figsize=(15,5))
price_year['AveragePrice'].plot(x=df.year)
plt.title('Average Price by Year')
```

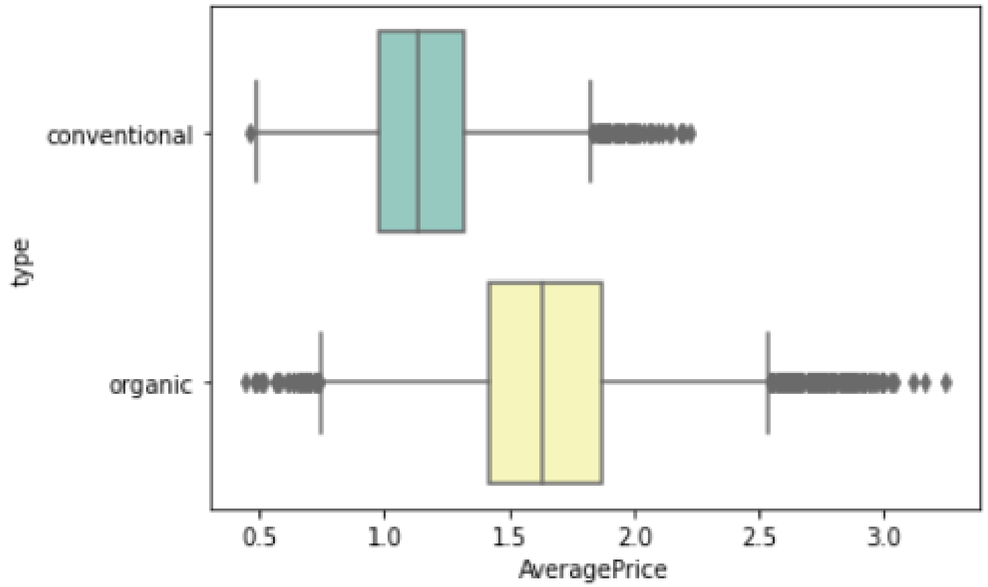
```
Text(0.5, 1.0, 'Average Price by Year')
```



연도에 따른 아보카도 퍙균 가격 변동은 2015년부터 아보카도 가격은 꾸준히 상승했으며 그 상승폭은 점점 작아지고 있음을 확인 할 수 있다.

```
sns.boxplot(y="type", x = "AveragePrice", data=df, palette = 'Set3')
```

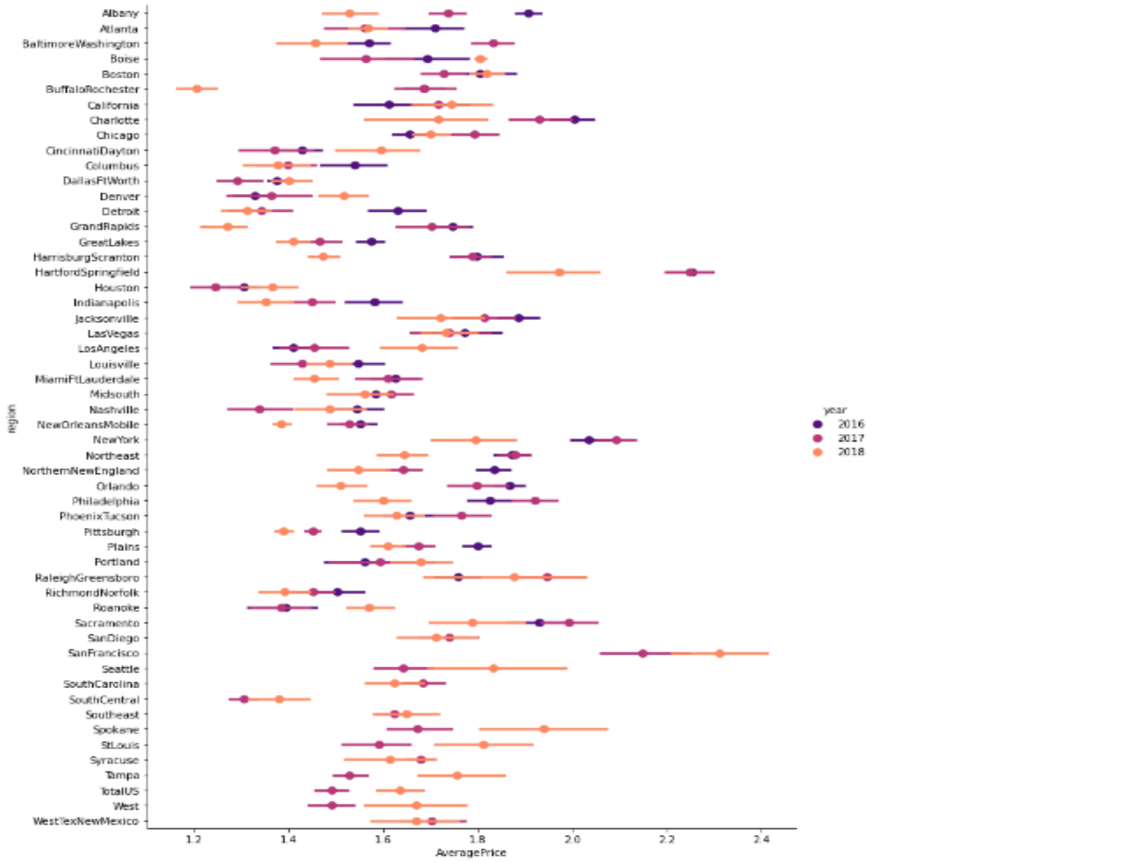
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa8f2ae4b90>
```



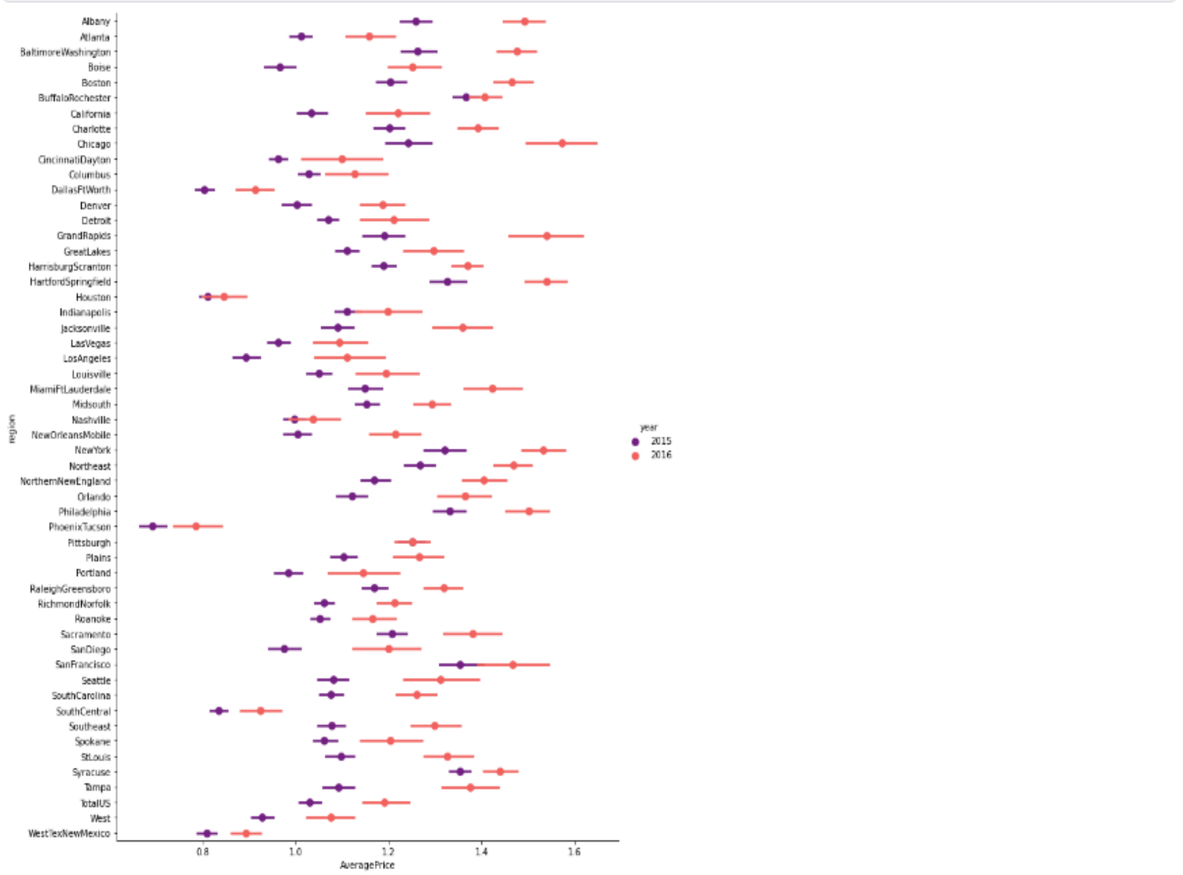
시기에 따른 가격 변동이 아닌 아보카도 품질에 따른 가격 차이를 확인할 수 있다. 이 자료는 wjsxhd(conventional) 과 유기농(organic) 최소 최대 이상치를 나타내며 일반 아보카드는 1.0달러부터 1.4정도 유기농 아보카드는 1.4부터 1.8의 범위를 가진다. 여기서 확인 할 수 있는 정보는 유기농 아보카도가 전통식 아보카도보다 비싸다는 것이다.

데이터 분석 및 시각화

```
#유기농
mask = df['type']=='organic'
g = sns.factorplot('AveragePrice', 'region', data=df[mask], hue='year',size=13,aspect=0.8,palette='magma',join=False)
```




```
#유기농
mask = df['type']=='conventional'
g = sns.factorplot('AveragePrice', 'region', data=df[mask], hue='year',size=13,aspect=0.8,palette='magma',join=False)
```

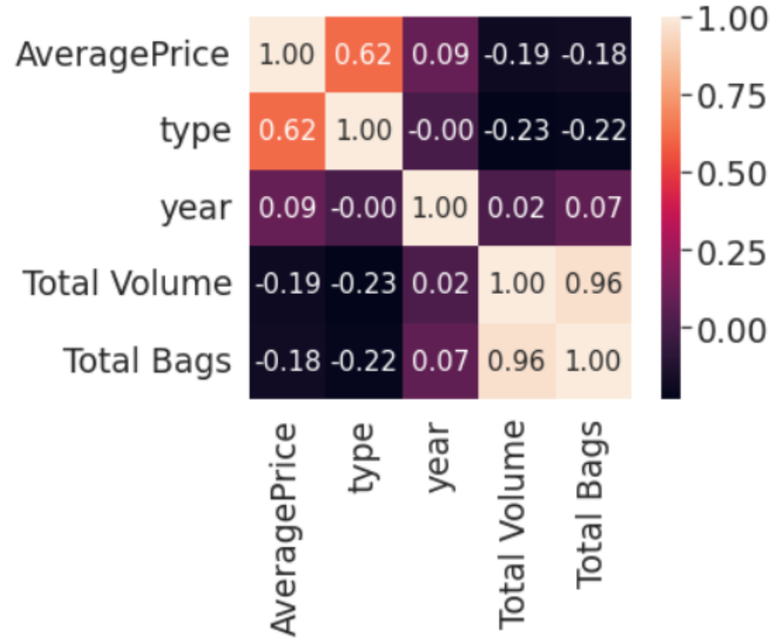


지역별로 아보카도 가격 확인할 수 있으며 2017년에 샌프란시스코의 가격이 높아지는 이유는 아보카도가 부족했기 때문이라는 기사가 있다.

PART 05

데이터 분석 및 시각화

```
cols = ['AveragePrice', 'type', 'year', 'Total Volume', 'Total Bags']
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale = 1.7)
hm = sns.heatmap(cm,cbar =True, annot = True, square = True, fmt = '.2f', annot_kws = {'
```



아보카도의 가격은 그 종류에 영향을 받는것을 확인이 가능합니다. Total volume(판매된 아보카도 갯수)와 Total bags에 강한 상관관계 발견 가능

PART

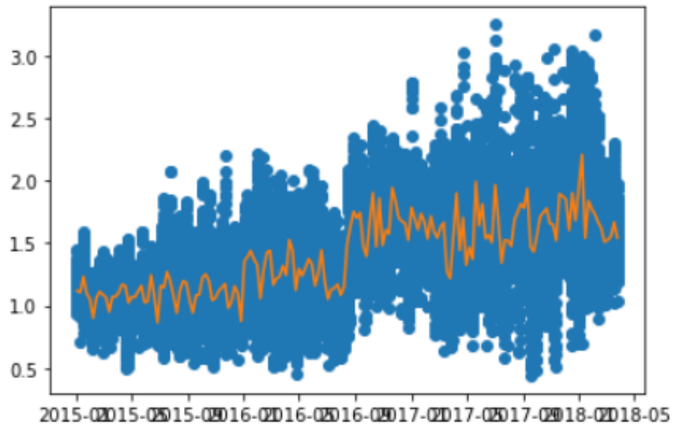
06

“데이터 분석 및 시각화”

DecisionTreeRegressor회귀 학습(결정 나무)

```
DTR = DecisionTreeRegressor(random_state = 0)
DTR.fit(train_X, train_y) #fit은 DTR에 이 데이터를 몸에 맞게 학습 시켜줌

#테스트 계산하기, test_score
score = DTR.score(test_X, test_y) #스스로 자기가 문제와 정답 체크
print('Score:', format(score, '.3f'))
# score의 의미: 정확하게 예측하면 1, 평균으로 예측하면 0, 더 못 예측하면 음수
#score 리턴된 값


X = df_avocado["ds"]
y = df_avocado["y"]
plt.plot(X, y, 'o')
plt.plot(X,RFR.predict(X.values.reshape(-1,1)))
plt.show()
```

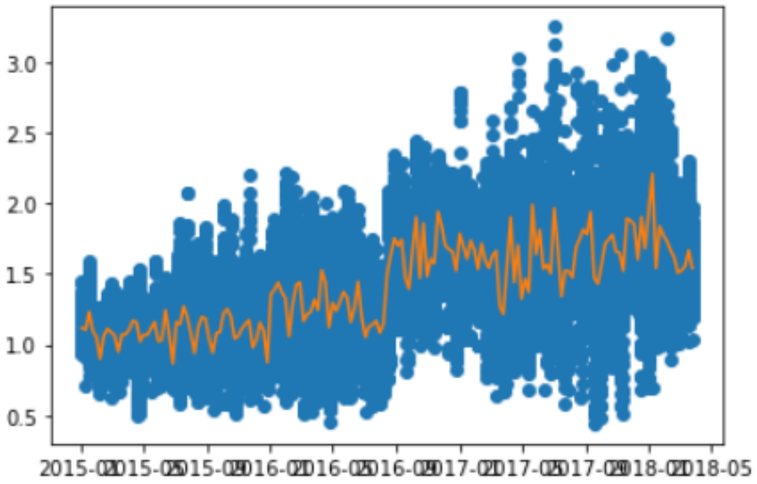
```
Score: 0.503
```



RandomForestRegressor

```
RFR = RandomForestRegressor(n_estimators=28,random_state=0)
RFR.fit(train_X, train_y)

score = RFR.score(test_X, test_y)
print('Score:', format(score,'.3f'))

X = df_avocado["ds"]
y = df_avocado["y"]
plt.plot(X, y, 'o')
plt.plot(X,RFR.predict(X.values.reshape(-1,1)))
plt.show()
```

```
Score: 0.503
```



로지스틱 회귀분석

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
import warnings
warnings.filterwarnings('ignore')

x = df.drop(['type', 'region', 'Date'], axis = 1)
y = df.type

x_train,x_test,y_train,y_test = train_test_split(x, y, random_state = 0)
```

+ Code + Markdown

```
# 로지스틱 회귀분석
logreg = LogisticRegression(penalty='l1', tol=0.0001, solver='liblinear').fit(x_train,y_train)
print("LogisticRegression train data score:{:.3f}".format(logreg.score(x_train,y_train)))
print("LogisticRegression train data score:{:.3f}".format(logreg.score(x_test,y_test)))
```

```
LogisticRegression train data score:0.933
LogisticRegression train data score:0.933
```

시계열 예측(Prophet)

훈련 데이터와 테스트 데이터 나누기

```
n_data = 1296
train_avocado = df_avocado[:-n_data]
test_avocado = df_avocado[-n_data:]
print(len(train_avocado), "train +", len(test_avocado), "test")
```

```
16953 train + 1296 test
```

시계열 예측

```
# 계절성 고려
train_avocado['cap'] = train_avocado.y.max()
train_avocado['floor'] = train_avocado.y.min()

time_model = Prophet(growth='logistic', interval_width=0.95)
time_model.add_seasonality(name='monthly', period=30.5, fourier_order=1)
time_model.add_seasonality(name='quarterly', period=91.25, fourier_order=5)
time_model.add_seasonality(name='yearly', period=365.25, fourier_order=10)

time_model.fit(train_avocado)
```

예측값의 상한과 하한을 제어해야 할 때 cap, floor 컬럼에 값을 지정해줍니다.
(상한과 하한을 설정하여 Prophet 객체를 생성할 때 growth='logistic' 추가)

시계열 예측

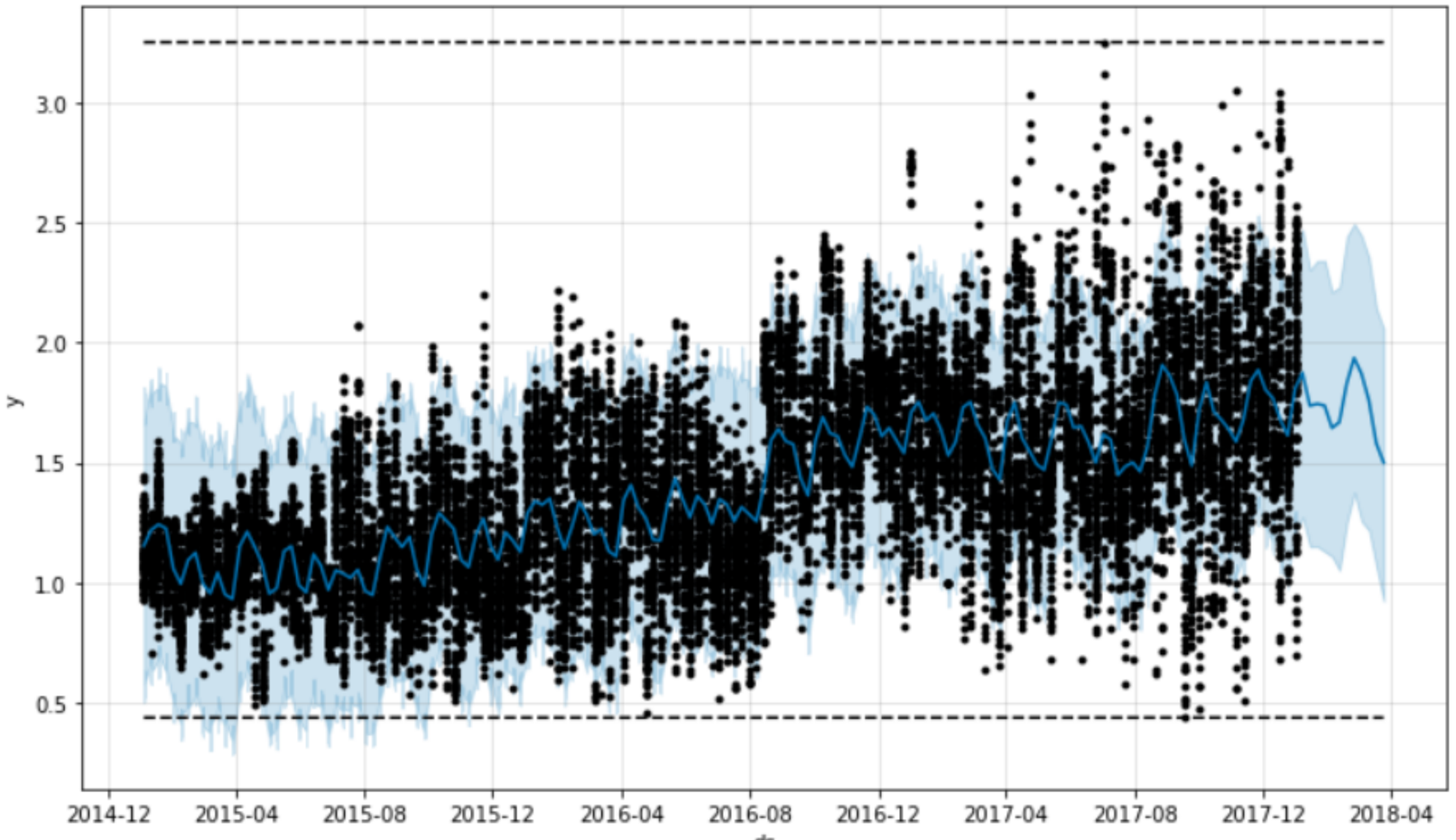
```
# 미래 가격 예측
future_dates = time_model.make_future_dataframe(periods=12, freq='w')
future_dates['cap'] = train_avocado.y.max()
future_dates['floor'] = train_avocado.y.min()

forecast = time_model.predict(future_dates)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()

# ds: the datestamp of the forecasted value
# yhat: the forecasted value of metric
# yhat_lower: the lower bound of forecasts
# yhat_upper: the upper bound of forecasts
```

	ds	yhat	yhat_lower	yhat_upper
16960	2018-02-25	1.936575	1.376434	2.495486
16961	2018-03-04	1.871003	1.256973	2.444780
16962	2018-03-11	1.760727	1.219345	2.353334
16963	2018-03-18	1.582004	1.068390	2.143691
16964	2018-03-25	1.501410	0.925482	2.058667

forecast = time_model.predict(future_dates)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
미래 데이터 예측하는 부분입니다.

시계열 예측

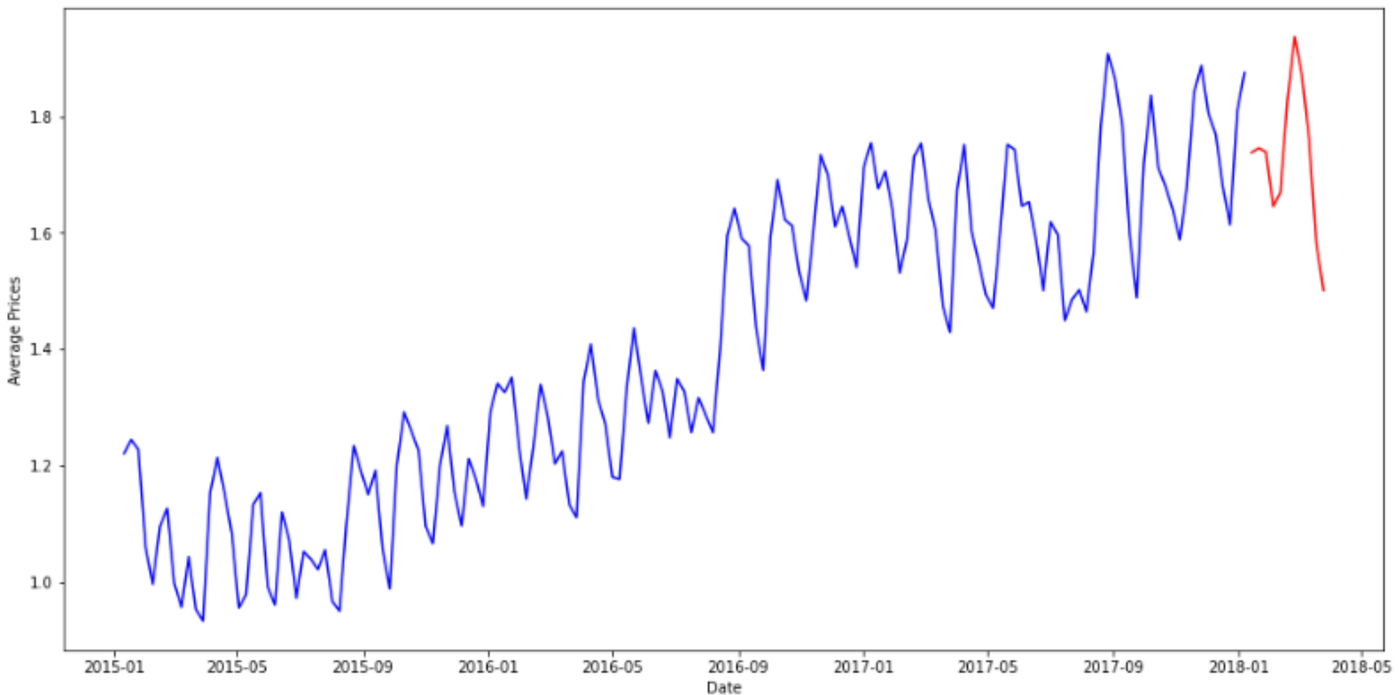
```
fig = time_model.plot(forecast)
```



시계열 예측

```
# 예측된 그래프 살펴보기
forecast_copy = forecast['ds']
forecast_copy2 = forecast['yhat']
forecast_copy = pd.concat([forecast_copy,forecast_copy2], axis=1)

mask = (forecast_copy['ds'] > "2018-01-07") & (forecast_copy['ds'] <= "2018-03-25")
forecasted_values = forecast_copy.loc[mask]
mask = (forecast_copy['ds'] > "2015-01-04") & (forecast_copy['ds'] <= "2018-01-07")
forecast_copy = forecast_copy.loc[mask]

fig, ax1 = plt.subplots(figsize=(16, 8))
ax1.plot(forecast_copy.set_index('ds'), color='b')
ax1.plot(forecasted_values.set_index('ds'), color='r')
ax1.set_ylabel('Average Prices')
ax1.set_xlabel('Date')
print("Red = Predicted Values, Blue = Base Values")
```

```
Red = Predicted Values, Blue = Base Values
```



모델 평가

```
# 정확성 측정
test_avocado['e'] = test_avocado.y - test_avocado.yhat
rmse = np.sqrt(np.mean(test_avocado.e**2)).round(2)
mape = np.round(np.mean(np.abs(100*test_avocado.e/test_avocado.y)), 0)
print('RMSE = $', rmse)
print('MAPE =', mape, '%')
```

```
RMSE = $ 0.4
MAPE = 20.0 %
```

에러값은 0.4달러이며 20퍼센트를 차지합니다.

PART 06

모델 평가

```
test_avocado = pd.concat([test_avocado.set_index('ds'),forecast.set_index('ds')], axis=1, join='

columns = ['y', 'yhat', 'yhat_lower', 'yhat_upper']
test_avocado = test_avocado[columns]

test_avocado.head()
```

ds	y	yhat	yhat_lower	yhat_upper
2018-01-07	1.88	1.874891	1.273218	2.470423
2018-01-07	2.06	1.874891	1.273218	2.470423
2018-01-07	1.59	1.874891	1.273218	2.470423
2018-01-07	1.72	1.874891	1.273218	2.470423
2018-01-07	1.42	1.874891	1.273218	2.470423

하지만 데이터 값이 부족해서 그런지 18년도 초반에는 계속해서 가격이 오르지만 뒤로 가면 갈수록 점점 떨어진다.

“소감”

PART
END

아보카도 가격 예측

머신러닝 알고리즘에 대해 이해 안되는 부분이 있었지만 이번 학습을 통해 어느정도 이해할 수 있는 시간을 가졌고, 추가적으로 선형회귀 알고리즘 등을 적용해보려 했지만 오류를 잡지 못해 아쉬웠습니다.

Thank You!