In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings(action='ignore')
```

In [2]:
```
train = pd.read_csv("./train.csv")
test = pd.read_csv("./test.csv")
```

1. 간단한 전처리 및 데이터 분포 확인

1.1 데이터 차원 및 중복행 확인

In [4]:
```
print(train.shape) #training set 약 10만행
print(test.shape) #test set 약 25000행
```

```
(103976, 13)
(25995, 12)
```

In [5]:
```
train.head(3)
```

Out[5]:

	country	description	designation	price	province	region_1	region_2	taster_name	taster_twitter_handle	title	variety	win
0	Australia	Possibly a little sweet, this is a soft, easyg...	NaN	5.0	Australia Other	South Eastern Australia	NaN	Joe Czerwinski	@JoeCz	Banrock Station 2006 Chardonnay (South Eastern...	Chardonnay	Banr Stat
1	France	A soft, almost off dry wine that is full in th...	Réserve	12.0	Rhône Valley	Côtes du Rhône	NaN	Roger Voss	@vossroger	Cellier des Dauphins 2015 Réserve Rosé (Côtes ...	Rosé	Cellier Dauph
2	Spain	Generic white-fruit aromas of peach and apple ...	Estate Grown & Bottled	9.0	Northern Spain	Rueda	NaN	Michael Schachner	@wineschach	Esperanza 2013 Estate Grown & Bottled Verdejo-...	Verdejo-Viura	Esperar

In [6]:
```
train.columns
```

Out[6]:
```
Index(['country', 'description', 'designation', 'price', 'province',
       'region_1', 'region_2', 'taster_name', 'taster_twitter_handle', 'title',
       'variety', 'winery', 'points'],
      dtype='object')
```

In [7]:
```
train.duplicated().sum() #모든 칼럼이 동일한 행이 약 6405행->의미가 없으므로 제거
```

Out[7]:
```
6405
```

In [11]:
```
#결측치 비율 확인
train.isnull().sum()/train.shape[0]*100
```

Out[11]:
```
country                  0.049195
description              0.000000
designation             28.854885
price                    6.976458
province                 0.049195
region_1                16.342971
region_2                61.042728
taster_name             20.624981
taster_twitter_handle   24.433489
title                    0.000000
variety                  0.001025
winery                   0.000000
points                   0.000000
dtype: float64
```

- 약 61%로 높은 결측치 비율을 보이는 region_2 column은 제외하는 것이 타당해보인다
- designation, taster_name, taster_twitter_handle 등은 각 칼럼의 특성을 확인한 이후 삭제 혹은 모델 사용 여부를 결정하기로 한다

In [16]:
```
#taster_name과 taster_twitter_handle 간 매칭 확인
train[(train['taster_name'].isna())&train['taster_twitter_handle'].notna()].shape

#taster_name은 결측치이지만, 트위터 아이디가 결측치가 아닌 경우는 존재하지 않음, 즉 최소한 taster_name이 기록된 항에만 트위터
```

Out[16]:
```
(0, 12)
```

In [66]:
```
train.drop("title", axis=1, inplace=True)
#train.drop("region_2", axis=1, inplace=True)
```

In [19]:
```
train.groupby("taster_name")['taster_twitter_handle'].nunique()
```

Out[19]:
```
taster_name
Alexander Peartree    0
Anna Lee C. Iijima    0
Anne Krebiehl MW      1
Carrie Dykes          0
Christina Pickard     1
Fiona Adams           1
Jeff Jenssen          1
Jim Gordon            1
Joe Czerwinski        1
Kerin O'Keefe         1
Lauren Buzzeo         1
Matt Kettmann         1
Michael Schachner     1
Mike DeSimone         1
Paul Gregutt          1
Roger Voss            1
Sean P. Sullivan      1
Susan Kostrzewa       1
Virginie Boone        1
Name: taster_twitter_handle, dtype: int64
```

- 모든 taster에 대해 고유한 트위터 아이디는 하나이므로 taster_twitter_handle은 삭제해도 무방

In [20]:
```
train.drop('taster_twitter_handle', axis=1, inplace=True)
```

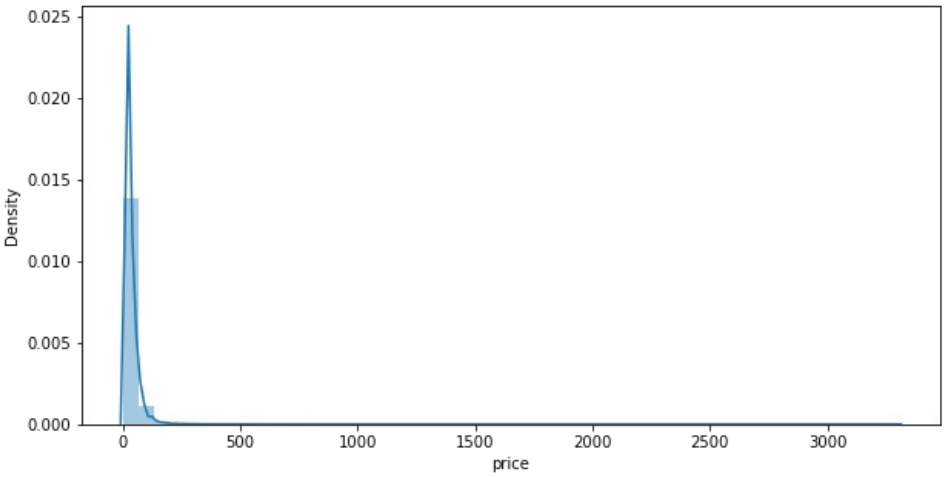
1.2 변수의 분포 확인

In [21]:
```
#price 변수 분포
train['price'].describe()
```

Out[21]:
```
count    90764.000000
mean        35.652021
std         43.356430
min          4.000000
25%         17.000000
50%         25.000000
75%         42.000000
max       3300.000000
Name: price, dtype: float64
```

- 최솟값이 3, 평균이 35, q3가 42이지만 max 값이 3300인점에서 이상치가 다수 존재한다는 점을 예측할 수 있다.

In [24]:
```
plt.figure(figsize=(10,5))
sns.distplot(train['price'])
plt.show()

print("Skewness : %.3f"%train['price'].skew())
print("Kurtosis : %.3f"%train['price'].kurtosis())
```

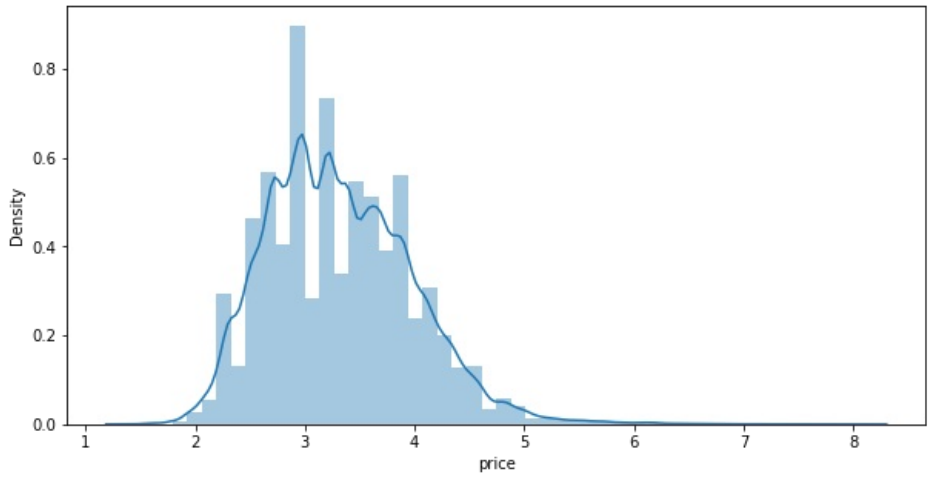



```
Skewness : 19.033
Kurtosis : 862.744
```

- price 변수의 distplot이 매우 왜곡되어 있고, 왜도 및 첨도도 정규분포의 가정을 위해 적합하지 않음을 알 수 있다.

In [26]:
```
plt.figure(figsize=(10,5))
```

```
sns.distplot(np.log(train['price']))
plt.show()

print("Skewness : %.3f"%np.log(train['price']).skew())
print("Kurtosis : %.3f"%np.log(train['price']).kurtosis())
```




```
Skewness : 0.614
Kurtosis : 0.817
```

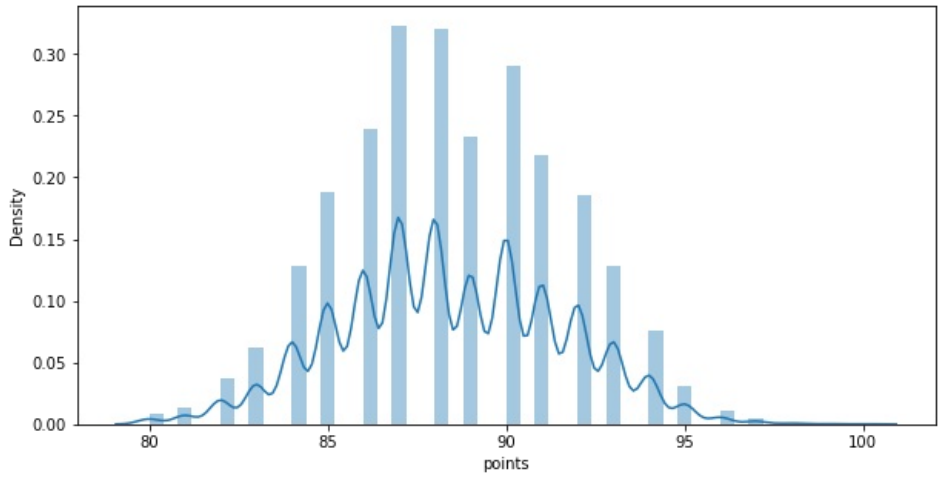
- 과하게 높은 와인의 가격을 이상치로 생각하기보다, 와인 산업의 특성상 높은 가격이 책정된 이유가 있을 것이라 판단해 이상치 제거가 아닌 log 변환을 하게 되는 경우 distplot과 왜도 및 첨도가 모두 정상적인 분포를 보임을 확인할 수 있다. 이에 모델 학습에서는 원본 데이터에서 이상치를 제거한 데이터와 로그 변환을 한 데이터를 모두 사용하여 성능을 비교하는 것이 가능할 것이다

In [27]:
```
#target(=points변수) 분포 확인
train['points'].describe()
```

Out[27]:
```
count    97571.000000
mean        88.444497
std          3.082897
min         80.000000
25%         86.000000
50%         88.000000
75%         91.000000
max        100.000000
Name: points, dtype: float64
```

- 타겟값인 포인트 변수의 경우 표준편차가 상당히 적고, 최솟값이 80점 이상이므로 모든 평점이 80~100점 사이에 위치함을 알 수 있다.

In [30]:
```
plt.figure(figsize=(10,5))
sns.distplot(train['points'])
plt.show()

print("Skewness : %.3f"%train['points'].skew())
print("Kurtosis : %.3f"%train['points'].kurtosis())
```



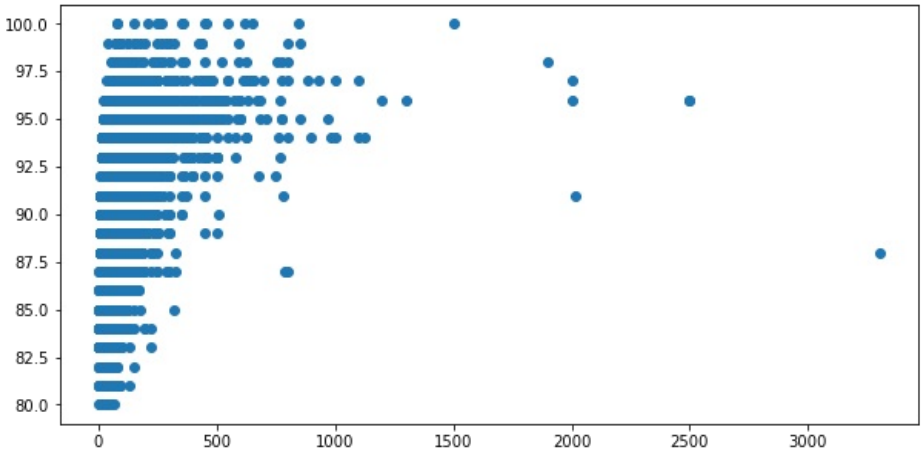

```
Skewness : 0.047
Kurtosis : -0.334
```

- points 변수의 displot과 왜도, 첨도 모두 정상적 범주의 분포를 보임을 확인할 수 있다.
- points 변수의 distplot에서 확인할 수 있는 또 다른 점은, points 변수가 연속형 자료이긴 하지만 이산적인 분포를 보이고 있다는 점이다. 즉 소수점 자리가 존재하거나 80~100점 사이 모든 점수에 대해 점수가 존재하는게 아니라 몇몇의 정수 점수만 사용되고 있으며 이는 다른 변수와 평점 간 상관성을 살펴볼 때 points 변수에 대한 boxplot을 그려볼 여지가 존재함을 의미한다

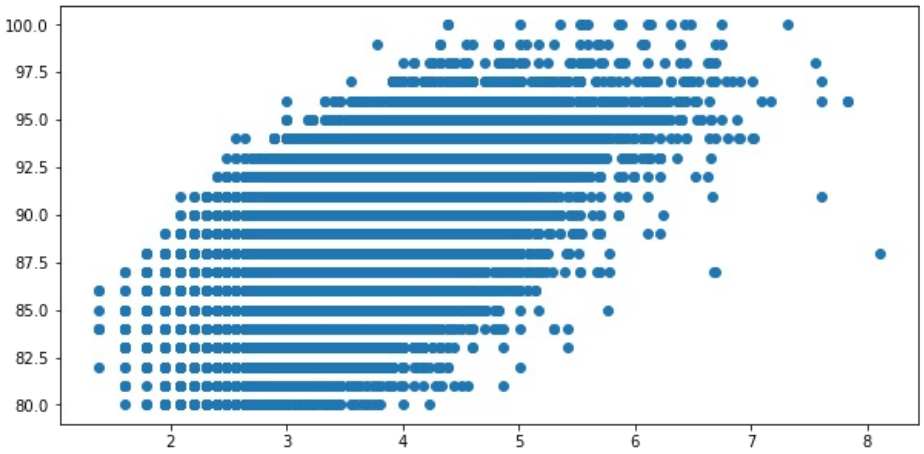
2. EDA

2.1 price와 points 변수의 관계성

In [32]:
```
plt.figure(figsize=(10,5))
plt.scatter(train['price'], train['points'])
plt.show()
```



In [33]:
```
plt.figure(figsize=(10,5))
plt.scatter(np.log(train['price']), train['points'])
plt.show()
```



- price 변수에 로그 변환을 진행한 후에 이를 points 변수와의 산점도로 표시해보면 훨씬 강한 양의 상관관계가 관찰됨을 확인할 수 있다

2.2 description의 길이와 points 간의 상관성

In [38]:
```
#전체 데이터 중 같은 taster가 같은 description을 내린 비율
print(train.duplicated(subset=['description', 'taster_name'], keep=False).sum()/train.shape[0]*100)
```

```
0.04099578768281559
```

In [39]:
```
#taster와 description이 같은 데이터에 대해, 평점도 같은 비율
subset=train[train.duplicated(subset=['description', 'taster_name'], keep=False)]
subset.groupby(['taster_name', 'description'])['points'].nunique().sum()/train.shape[0]*100
```

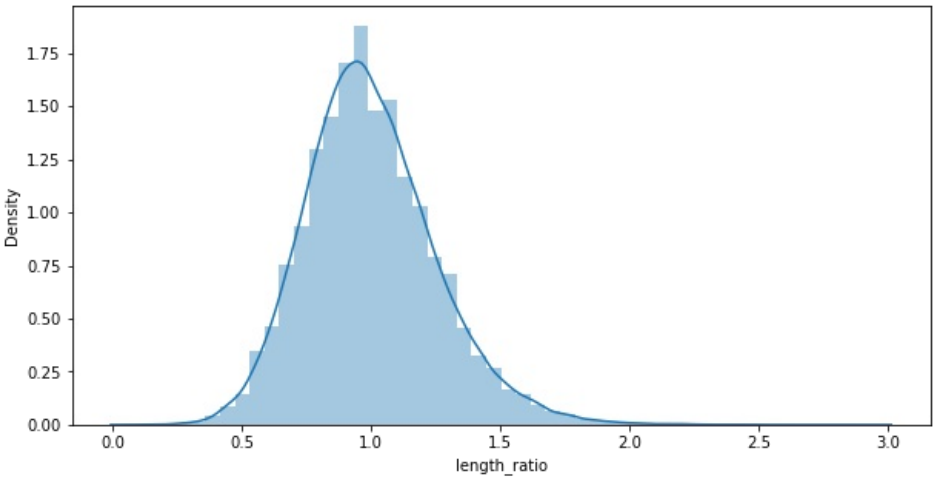
Out[39]:
```
0.03279663014625247
```

- 두 가지 자료의 비율을 고려할 때, taster가 해당 와인에 남기는 description은 각 와인에 대한 고유적 특성이 충분히 반영되었다고 판단할 수 있다. 이에 descrition을 points와 연관지어 생각해보기 위해 좋은 와인에는 더 긴 리뷰를 남겼을 것이라 가정하고 EDA를 진행

In [49]:
```
train['length'] = train['description'].apply(lambda x : len(str(x).split(" "))) #description 길이 추가
train = pd.merge(train, train.groupby('taster_name')['length'].mean(), how='left', on='taster_name') #train 데이

train.rename(columns={'length_x':'length', 'length_y':'length_mean'}, inplace=True ) #병합 과정에서 생긴 열 이름 수
train['length_ratio'] = train['length']/train['length_mean']


train['length_ratio'].describe()
```

Out[49]:
```
count    77447.000000
mean         1.000000
std          0.254319
min          0.073093
25%          0.825597
50%          0.976644
75%          1.147569
max          2.933435
Name: length_ratio, dtype: float64
```

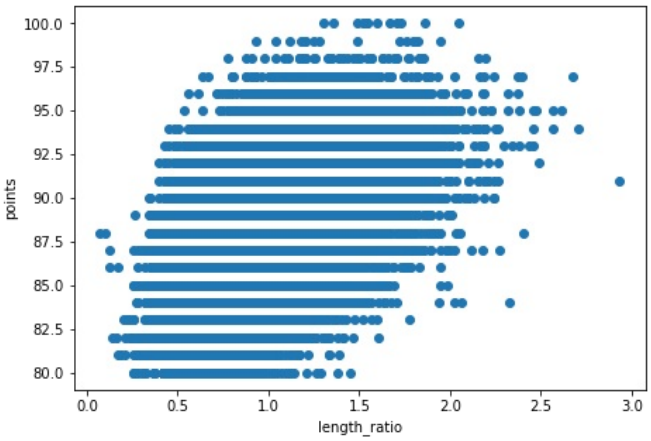
In [50]:
```
plt.figure(figsize=(10,5))
sns.distplot(train['length_ratio'])
plt.show()

print("Skewness : %.3f"%train['length_ratio'].skew())
print("Kurtosis : %.3f"%train['length_ratio'].kurtosis())
```

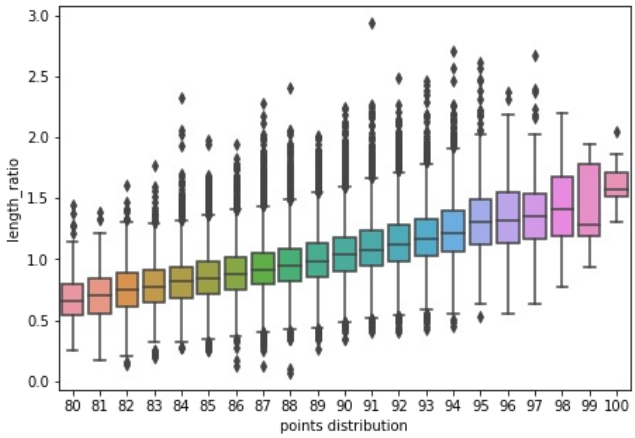



```
Skewness : 0.575
Kurtosis : 1.022
```

- 각 행에 남겨진 description의 길이를 새로운 칼럼으로 추가해줬다.
- '좋은 와인에는 상대적으로 긴 리뷰를 남길것이다'를 가설로 채택했지만, taster마다 description의 길이에 대한 경향성이 다를 것이므로 각 taster 당 평균 리뷰 길이를 계산한 후 각 행의 descrition에 대해 나누어줬다.
- 그렇게 계산된 length_ratio는 최솟값 0.07부터 2.93의 최댓값을 가지며 distplot과 왜도 첨도 모두 정상적 범주 내에 존재한다.

In [52]:
```
fig, axes = plt.subplots(figsize=(16,5), ncols=2)
axes[0].scatter(train['length_ratio'], train['points'])
axes[0].set_xlabel("length_ratio")
axes[0].set_ylabel("points")
sns.boxplot(train['points'], train['length_ratio'], ax=axes[1])
axes[1].set_xlabel("points distribution")
axes[1].set_ylabel("length_ratio")
plt.show()
```





- 리뷰를 남긴 리뷰어의 평균 리뷰 길이 대비 해당 와인의 리뷰 길이와 평점간 관계를 확인해보기 위해 length_ratio를 x축, points를 y축으로 하는 산점도와 points를 x축, length_ratio를 y축으로 하는 boxplot을 그려보았다.
- 좌측 산점도에서는 약간의 양의 상관성이 있는 것처럼 보이지만 유의미한 차이를 확인하기가 힘들다
- 하지만 우측의 boxplot을 살펴보면 평점이 높아질수록 length_ratio의 중간값이 점점 상승하고 있는 것을 확인할 수 있으며 각 평점 별 quantile 역시 동시에 상승하고 있는 것을 확인할 수 있다.
- 즉, 높은 평점이 매겨진 와인은 상대적으로 긴 리뷰 글자수 비율을 보인다는 점을 확인할 수 있다.

In [57]:
```
train[['points', 'length_ratio']].corr() #상관관계 역시 약 0.5 정도의 값을 보이고 있다.
```

Out[57]:

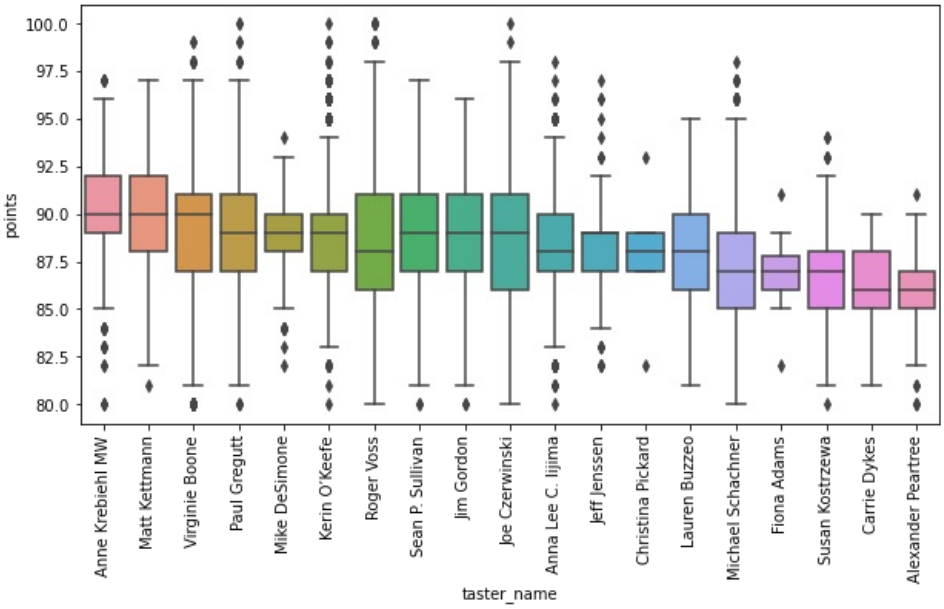
	points	length_ratio
points	1.000000	0.489781
length_ratio	0.489781	1.000000

2.3 taster_name 별 평점 분포

In [59]:
```
train.taster_name.value_counts()

#총 19명의 taster가 존재. 22, 6개를 기록한 두 명의 리뷰어를 제외하면 대부분 100건 이상을 진행
```

Out[59]:
```
Roger Voss            19290
Michael Schachner     11335
Kerin O'Keefe          7960
Paul Gregutt           7236
Virginie Boone         7053
Matt Kettmann          4702
Joe Czerwinski         3864
Sean P. Sullivan       3637
Anna Lee C. Iijima     3312
Jim Gordon             3033
Anne Krebiehl MW       2680
Lauren Buzzeo          1351
Susan Kostrzewa         817
Mike DeSimone           383
Jeff Jenssen            362
Alexander Peartree      299
Carrie Dykes            105
Fiona Adams              22
Christina Pickard         6
Name: taster_name, dtype: int64
```

In [61]:
```
index = train.groupby('taster_name').mean()['points'].sort_values(ascending=False).index

plt.figure(figsize=(10,5))
sns.boxplot(train.taster_name, train.points, order=index)
plt.xticks(rotation=90)
plt.show()
```



- quantile 및 이상치를 고려했을 때 어느 정도의 변동성은 존재하지만 리뷰어별로 median 값이 유의미한 경향성을 보이고 있음을 확인할 수 있다.

2. 4country와 평점 분포

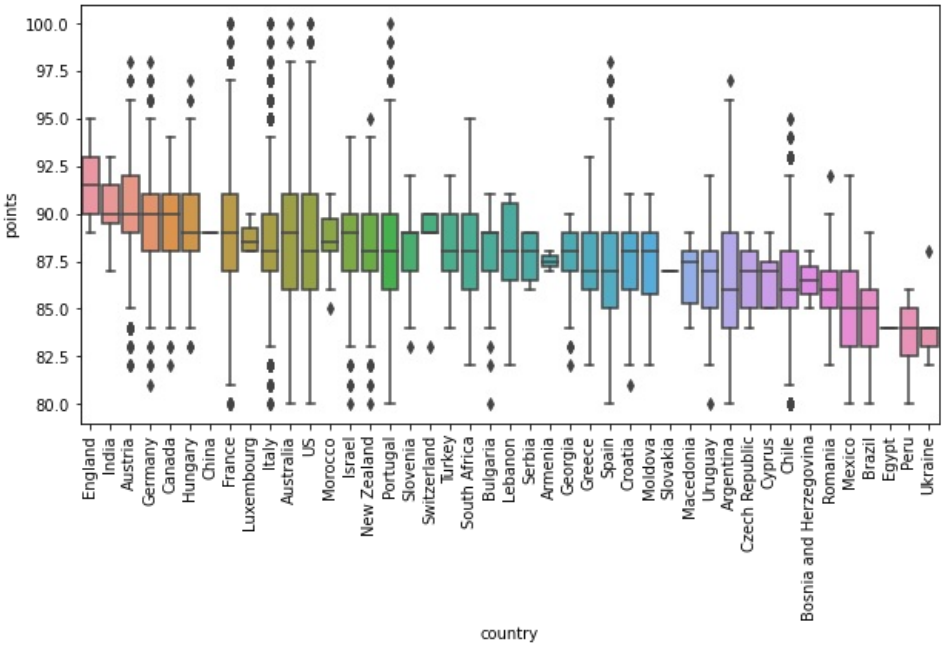
In [103…
```
country=pd.DataFrame(train.value_counts('country'), columns = ['count'])
country['ratio'] = country['count']/country.values.sum()*100

country[:20]
```

Out[103]:

country	count	ratio
US	40984	42.024958
France	16642	17.064692
Italy	14604	14.974929
Spain	4890	5.014202
Portugal	4323	4.432800
Chile	3401	3.487382
Argentina	2875	2.948023
Austria	2447	2.509152
Australia	1777	1.822134
Germany	1638	1.679604
New Zealand	1044	1.070517
South Africa	1040	1.066415
Israel	376	0.385550
Greece	352	0.360940
Canada	185	0.189699
Bulgaria	106	0.108692
Hungary	103	0.105616
Romania	86	0.088184
Uruguay	77	0.078956
Turkey	69	0.070753

- 설명 : 소수 국가가 많은 비율을 차지한다, 이는 province 피쳐의 존재가 필요할 수 있다는 반증

In [64]:
```
index = train.groupby('country').mean()['points'].sort_values(ascending=False).index

plt.figure(figsize=(10,5))
sns.boxplot(train.country, train.points, order=index)
plt.xticks(rotation=90)
plt.show()
```



- country와 평점 간 boxplot 역시 median 값이 유의미한 경향성을 띄고 있음을 확인할 수 있음

variety 분포

In [70]:
```
train['variety'].value_counts()#variety 변수는 총 676개의 column, 원핫인코딩 하기에 과하게 많지는 않은 칼럼수
```

Out[70]:
```
Pinot Noir                  10036
Chardonnay                   8825
Cabernet Sauvignon           7139
Red Blend                    6685
Bordeaux-style Red Blend     5222
                            ...
Chardonnay-Riesling             1
Biancale                        1
Thrapsathiri                    1
Aidani                          1
Gragnano                        1
Name: variety, Length: 676, dtype: int64
```

In [79]:
```
min_index = train['variety'].value_counts()[train['variety'].value_counts().values > 100].index
```

In [87]:
```
top_list = train[train['variety'].isin(min_index)].groupby("variety").mean()['points'].sort_values(ascending=Fa
bottom_list = train[train['variety'].isin(min_index)].groupby("variety").mean()['points'].sort_values(ascending
```

In [105…
```
top_list
```

Out[105]:
```
variety
Sangiovese Grosso    90.566901
Nebbiolo             90.323460
Blaufränkisch        90.120000
Grüner Veltliner     90.047521
Port                 89.857700
Tinta de Toro        89.718750
Champagne Blend      89.661366
Riesling             89.447804
Pinot Noir           89.418095
Syrah                89.318211
Name: points, dtype: float64
```

In [100…
```
bottom_list
```

Out[100]:
```
variety
Torrontés           85.494845
Garnacha            85.747863
Moscato             86.140741
Pinot Grigio        86.224592
Verdejo             86.336493
Prosecco            86.402174
Carmenère           86.590164
Rosé                86.790440
Montepulciano       86.850299
Portuguese White    86.928324
Name: points, dtype: float64
```

In [110…
```
province=pd.DataFrame(train.value_counts('province'), columns = ['count'])
province['ratio'] = province['count']/province.values.sum()*100

province[:20]
```

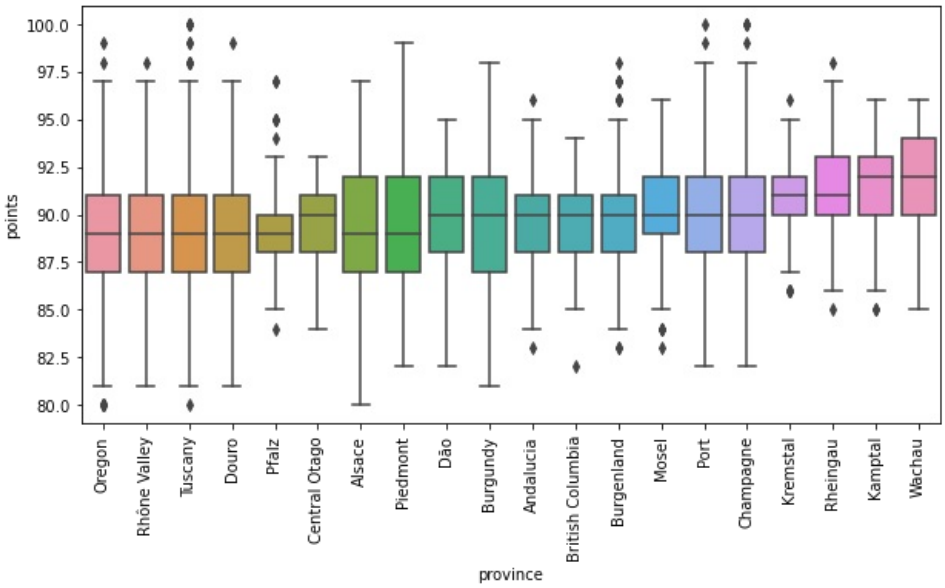
Out[110]:

province	count	ratio
California	27287	27.980066
Washington	6487	6.651764
Bordeaux	4525	4.639931
Tuscany	4380	4.491248
Oregon	4028	4.130308
Burgundy	3040	3.117213
Northern Spain	2825	2.896753
Piedmont	2814	2.885473
Mendoza Province	2475	2.537863
Veneto	2038	2.089763
New York	2019	2.070281
Alsace	1819	1.865201
Northeastern Italy	1596	1.636537
Loire Valley	1388	1.423254
Sicily & Sardinia	1333	1.366857
Champagne	1226	1.257139
Southwest France	1127	1.155625
South Australia	1013	1.038729
Southern Italy	1007	1.032577
Douro	982	1.006942

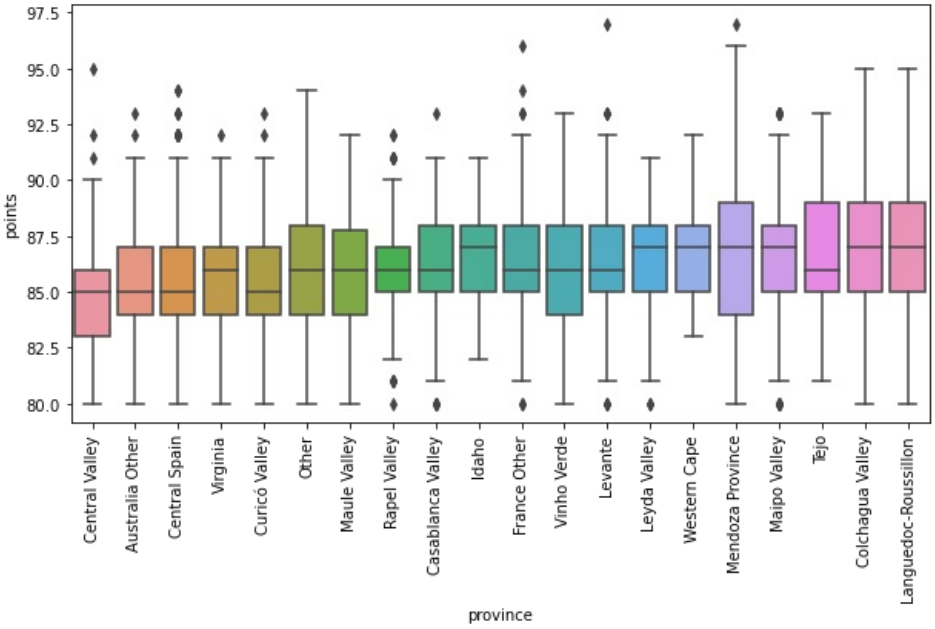
province와 평점의 관계성

In [131…
```
#최소 와인 생산 갯수가 100개 이상인 province에 대해 분석
min_index_province = train['province'].value_counts()[train['province'].value_counts().values > 100].index

top_list = train[train['province'].isin(min_index_province)].groupby("province").mean()['points'].sort_values(a
#최소 와인 생산 갯수가 100개 이상인 province 중 평점 상위 20개 지역
bottom_list = train[train['province'].isin(min_index_province)].groupby("province").mean()['points'].sort_value
#최소 와인 생산 갯수가 100개 이상인 province 중 평점 하위 20개 지역
```

In [133…
```
plt.figure(figsize=(10,5))
sns.boxplot(train[train['province'].isin(top_list.index)]['province'], train[train['province'].isin(top_list.in
            order = top_list.sort_values(ascending=True).index)
plt.xticks(rotation=90)
plt.show()
```



In [134…
```
plt.figure(figsize=(10,5))
sns.boxplot(train[train['province'].isin(bottom_list.index)]['province'], train[train['province'].isin(bottom_l
            order = bottom_list.sort_values(ascending=True).index)
plt.xticks(rotation=90)
plt.show()
```



- 평균 평점 상, 하위 20개 province에 대해 각각 평점의 boxplot을 plotting. x축의 province 나열 순서는 평균 평점 순이지만 상,하위 그래프 모두에서 province 별 유의미한 차이는 발견되지 않음

*country, province 별 평점 분포 시각화

1) 기본적인 전처리

- 중복된 데이터가 존재하고, description과 같은 변수만 중복된 경우 추가적인 확인 필요
- taster_name 칼럼과 taster_twitter_handle 칼럼이 1대1 대응
- 결측치를 포함하고 있는 행들이 있고, region_2의 결측치 비율이 큼

```
In [9]: # twitter id와 taster name이 일대일 대응하는지 확인
        df.groupby(['taster_name', 'taster_twitter_handle']).size()

Out[9]: taster_name        taster_twitter_handle
        Anne Krebiehl MW   @AnneInVino              2937
        Christina Pickard  @winewchristina             6
        Fiona Adams        @bkfiona                   23
        Jeff Jenssen       @worldwineguys            394
        Jim Gordon         @gordone_cellars         3279
        Joe Czerwinski     @JoeCz                   4089
        Kerin O' Keefe      @kerinokeefe            8673
        Lauren Buzzeo      @laurbuzz                1428
        Matt Kettmann      @mattkettmann            5109
        Michael Schachner  @wineschach             12026
        Mike DeSimone      @worldwineguys            425
        Paul Gregutt       @paulgwine               7673
        Roger Voss         @vossroger              20536
        Sean P. Sullivan   @wawinereport            3950
        Susan Kostrzewa    @suskostrzewa             855
        Virginie Boone     @vboone                  7591
        dtype: int64
```

```
In [12]: df[df['description'].duplicated()].index

Out[12]: Int64Index([ 24798,  31067,  33270,  38558,  47705,  48829,  49512,  55976,
                      63279,  68952,  70422,  79643,  85389,  87262,  90848,  95434,
                      95441,  97075,  98038,  98695, 101386],
                    dtype='int64')
```

1) 기본적인 전처리 – 중복 데이터 제거

- 전체가 중복된 데이터는 제거
- 만약 description 행만이 중복되었다면, 해당하는 데이터는 행을 각각 확인하여 오기 여부 판단

```
In [12]: df[df['description'].duplicated()].index
Out[12]: Int64Index([ 24798,  310
                      63279,  689
                      95441,  970
                     dtype='int64'

In [13]: # 중복된 데이터 각각 확
         df[df['description'] ==
```

Out[13]:

	country	descripti						
33843	Italy	Aromas wh flow green pe and flinty						
101386	Italy	Aromas white flower, green pear and flinty ...	Le Cigliate	29.0	Southern Italy	Campi Flegrei	NaN	Kerin O'Keefe

```
In [14]: # 오기로 판단되는 데이터 제거
         df.drop(index=[24798, 24231], inplace=True)
```

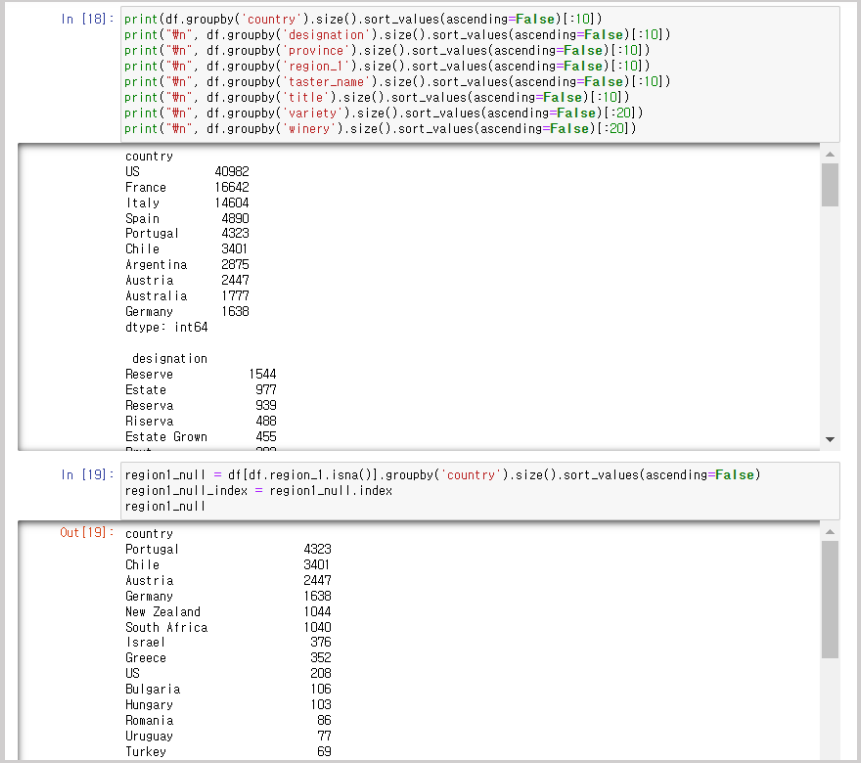
```
In [81]: # 중복된 데이터 각각 확인
         df[df['description'] == df.loc[24798, 'description']]
```

Out[81]:

	country	description	designation	price	province	region_1	region_2	taster_name	taster_twitter_handle	title	variety	winery	points
24258	US	Cigar box, café au lait, and dried tobacco aro...	Estate Grown	30.0	Washington	Red Mountain	Columbia Valley	Sean P. Sullivan	@wawinereport	Ambassador Vineyard 2013 Estate Grown Syrah (R...	Syrah	Ambassador Vineyard	88
24798	US	Cigar box, café au lait, and dried tobacco aro...	Estate	30.0	Washington	Red Mountain	Columbia Valley	Sean P. Sullivan	@wawinereport	Ambassador Vineyard 2013 Estate Syrah (Red Mou...	Syrah	Ambassador Vineyard	88

1) 기본적인 전처리 – twitter 칼럼 제거, 결측치 추정

- twitter id 칼럼과 region_2 칼럼은 제거
- 결측치는 확인을 통해 바람직하게 추정할 수 있는 경우 대체 – region_1 칼럼의 일부 정보, winery, taster 등



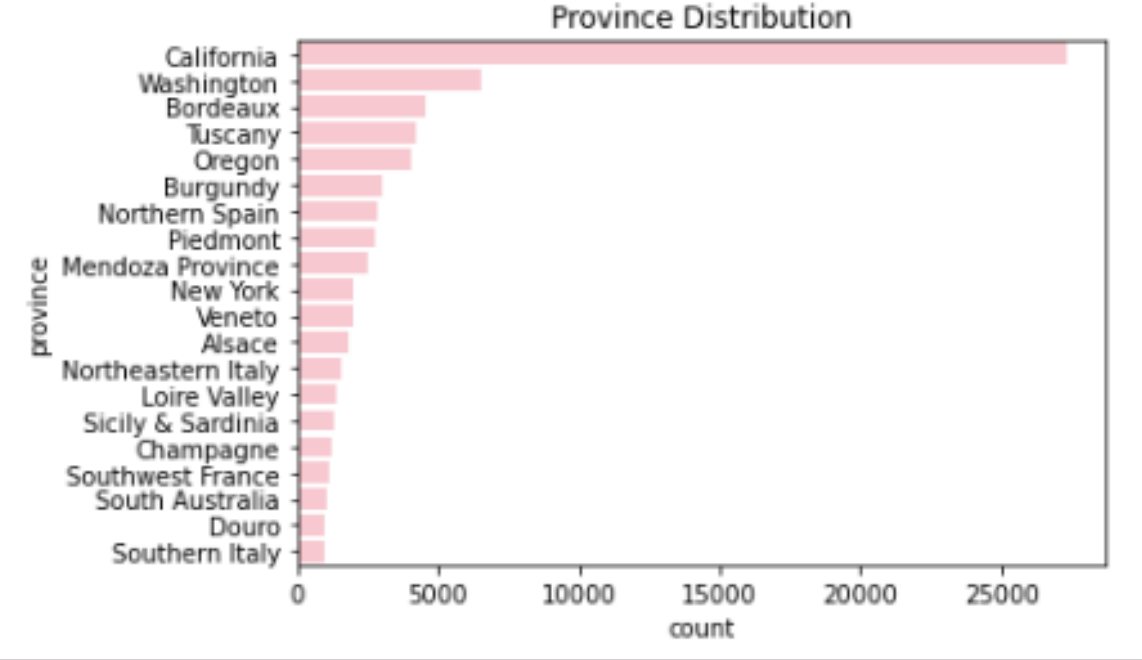
```
In [18]: print(df.groupby('country').size().sort_values(ascending=False)[:10])
         print("₩n", df.groupby('designation').size().sort_values(ascending=False)[:10])
         print("₩n", df.groupby('province').size().sort_values(ascending=False)[:10])
         print("₩n", df.groupby('region_1').size().sort_values(ascending=False)[:10])
         print("₩n", df.groupby('taster_name').size().sort_values(ascending=False)[:10])
         print("₩n", df.groupby('title').size().sort_values(ascending=False)[:10])
         print("₩n", df.groupby('variety').size().sort_values(ascending=False)[:20])
         print("₩n", df.groupby('winery').size().sort_values(ascending=False)[:20])

country
US           40982
France       16642
Italy        14604
Spain         4890
Portugal      4323
Chile         3401
Argentina     2875
Austria       2447
Australia     1777
Germany       1638
dtype: int64

 designation
Reserve            1544
Estate              977
Reserva             939
Riserva             488
Estate Grown        455

In [19]: region1_null = df[df.region_1.isna()].groupby('country').size().sort_values(ascending=False)
         region1_null_index = region1_null.index
         region1_null

Out[19]: country
Portugal                4323
Chile                   3401
Austria                 2447
Germany                 1638
New Zealand             1044
South Africa            1040
Israel                   376
Greece                   352
US                       208
Bulgaria                 106
Hungary                  103
Romania                   86
Uruguay                   77
Turkey                    69
```

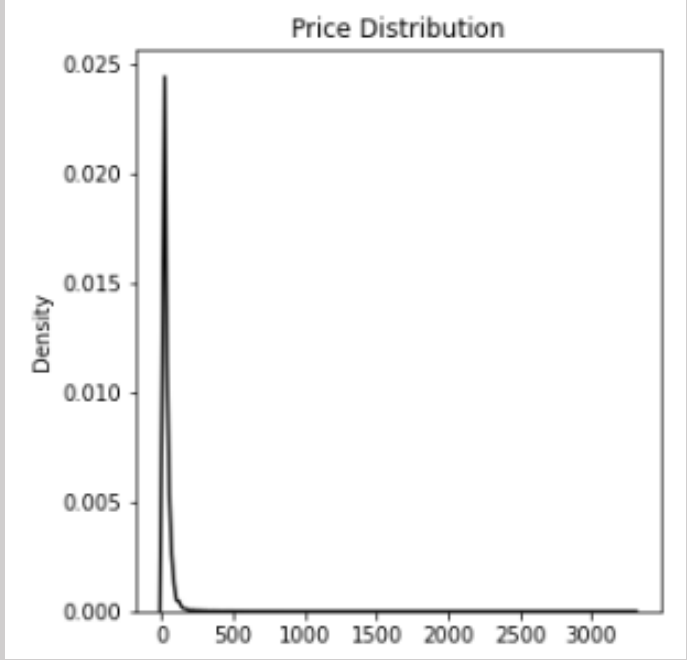
Portugal	4323
Chile	3401
Argentina	2875
Austria	2447
Australia	1777
Germany	1638

Portugal	4323
Chile	3401
Austria	2447
Germany	1638

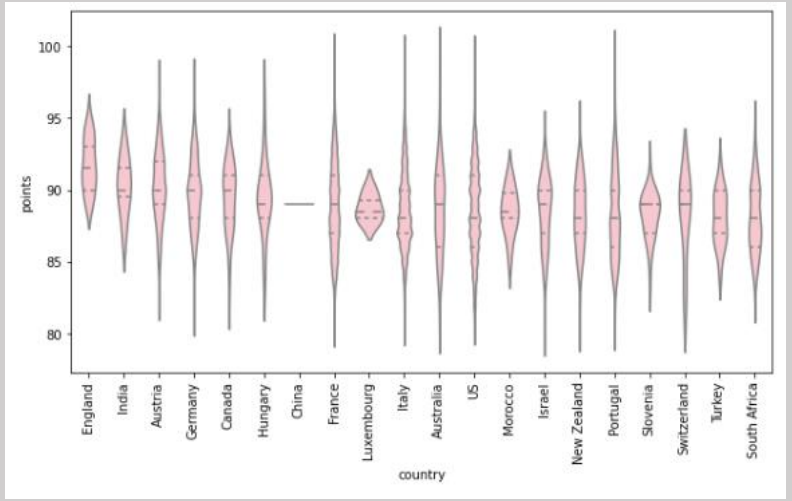
2) EDA – Univariate Visualization

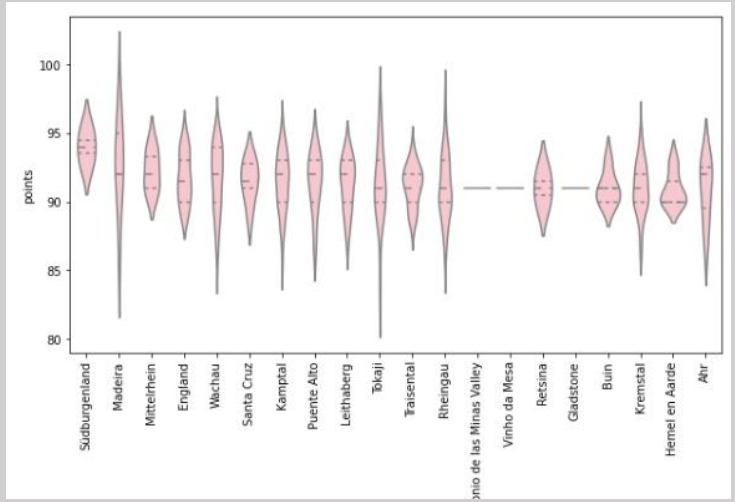
- 변수들 각각에 대하여 분포를 확인하기 위한 작업
- province의 집중도와 price의 right-skewed된 형태 등 특징적인 지점 확인

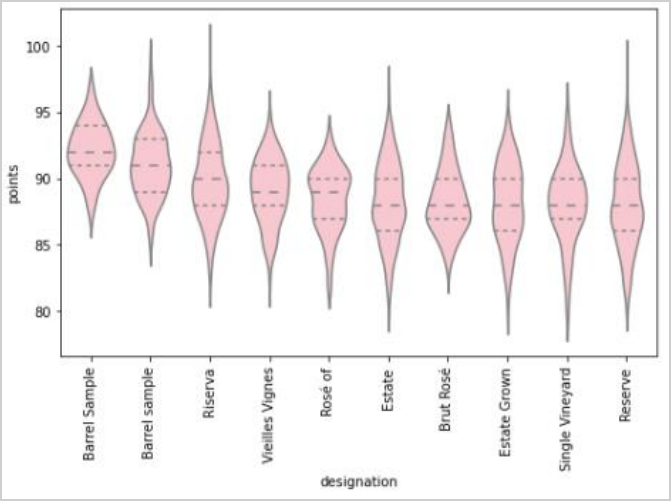


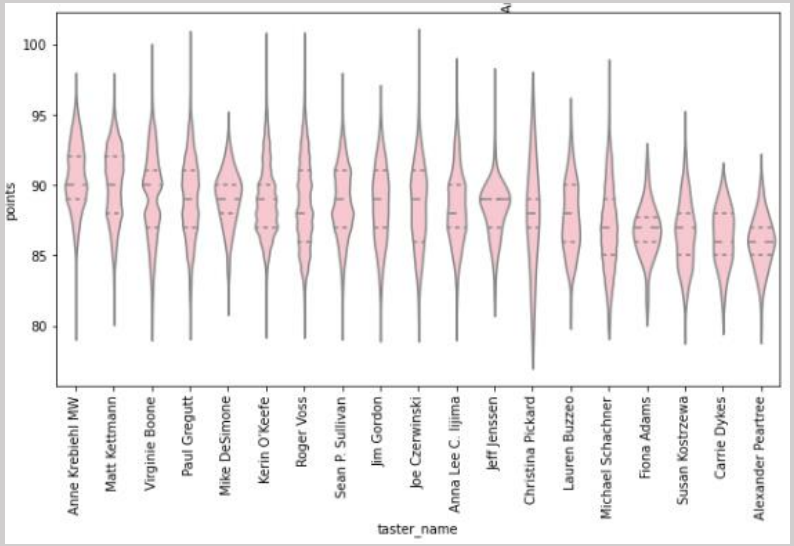


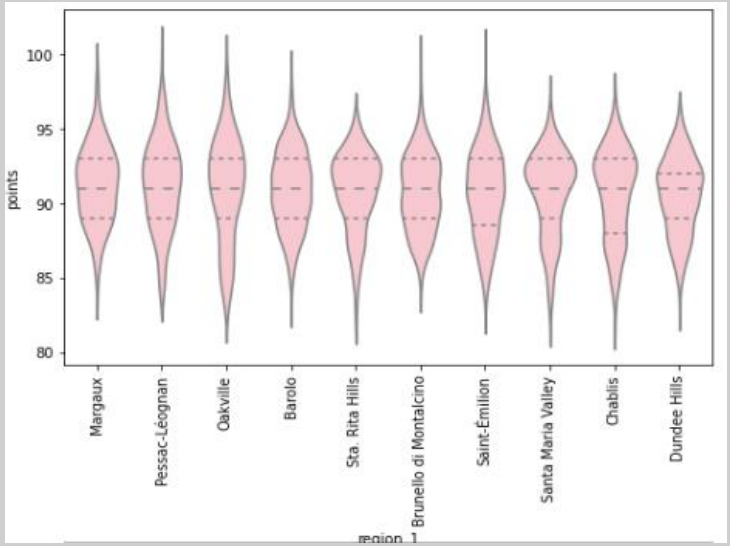
2) EDA – Vibariate Visualization

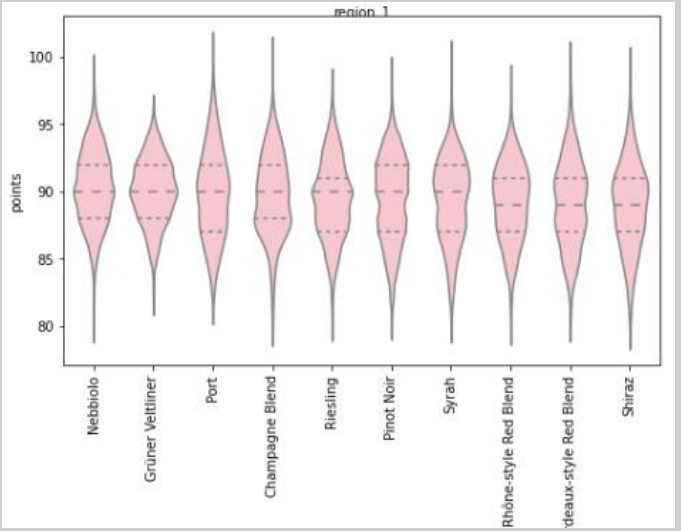






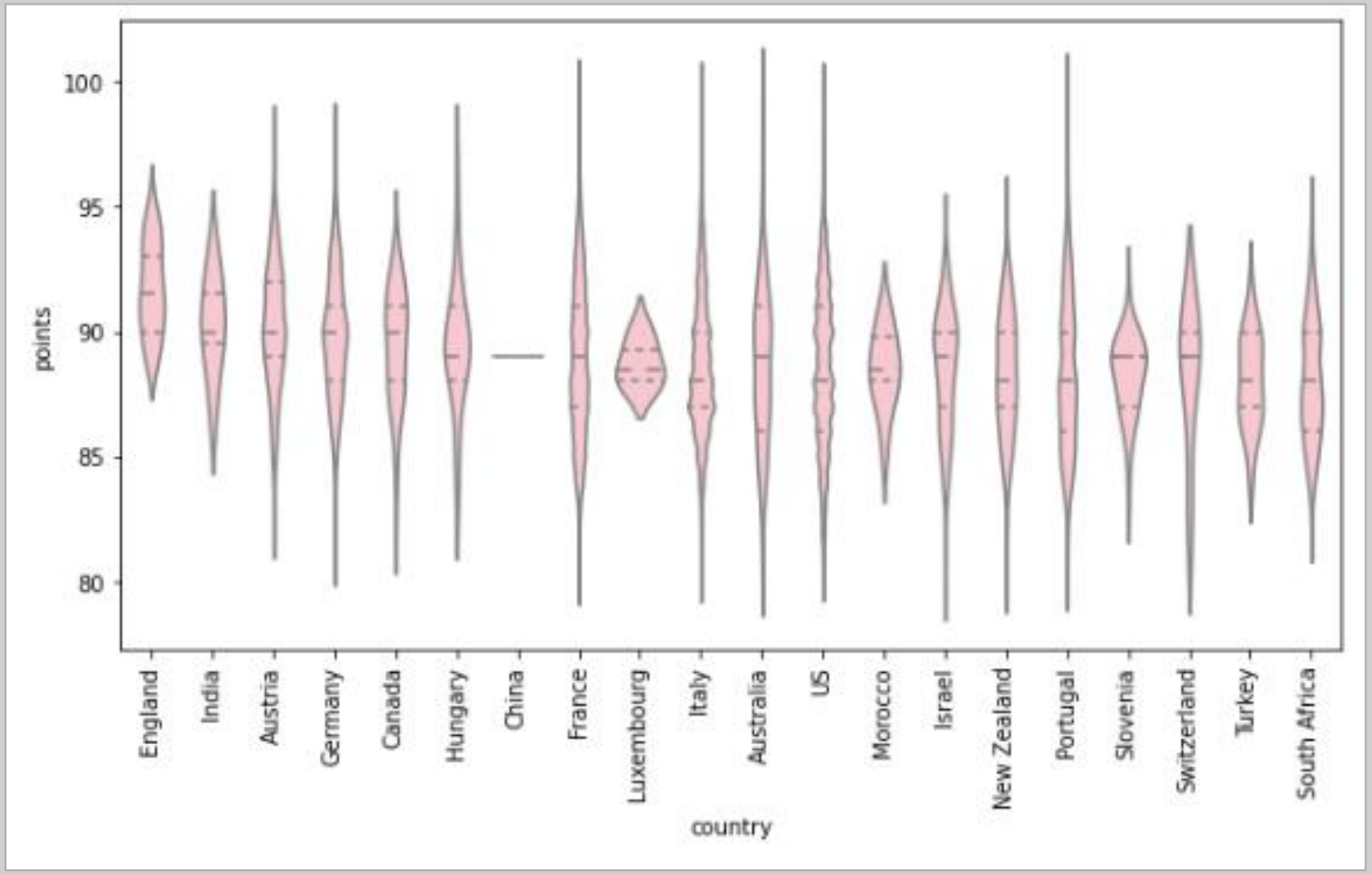






2) EDA – Vibariate Visualization

- EDA 목적: 점수의 기댓값 최대화, 분산(리스크) 최소화 항목 및 조합을 찾아보자



2) EDA – 데이터 점검

```
In [52]: df[df.country == 'England']
```

Out[52]:

	country	designation	price	province	region_1	taster_name	title	variety	winery	points
1116	England	Rosé Bella	85.0	England	England	Anne Krebiehl MW	Bride Valley Vineyard 2014 Rosé Bella Sparklin...	Sparkling Blend	Bride Valley Vineyard	94
1446	England	Classic Cuvée	50.0	England	England	Anne Krebiehl MW	Hoffmann & Rathbone 2013 Classic Cuvée Sparkli...	Sparkling Blend	Hoffmann & Rathbone	91
4442	England	Brut	50.0	England	England	Anne Krebiehl MW	Camel Valley 2012 Brut Sparkling (England)	Sparkling Blend	Camel Valley	89
7308	England	Cuvée Brut	40.0	England	England	Anne Krebiehl MW	Wiston Estate Winery 2010 Cuvée Brut Sparkling...	Sparkling Blend	Wiston Estate Winery	91
7948	England	Fitzrovia Rosé	42.0	England	England	Anne Krebiehl MW	Ridgeview Estate 2013 Fitzrovia Rosé	Sparkling Blend	Ridgeview Estate	89

```
In [53]: df[df.province == "Südburgenland"]
```

Out[53]:

	country	designation	price	province	region_1	taster_name	title	variety	winery	points
3447	Austria	Perwolff	79.0	Südburgenland	Austria	Anne Krebiehl MW	Krutzler 2012 Perwolff Blaufränkisch (Südburge...	Blaufränkisch	Krutzler	95
45793	Austria	Steinberg	35.0	Südburgenland	Austria	Anne Krebiehl MW	Jalits 2012 Steinberg Red (Südburgenland)	Red Blend	Jalits	93

```
In [57]: df[df.winery == "Cayuse"]
```

Out[57]:

	country	designation	price	province	region_1	taster_name	title	variety	winery	points
670	US	Armada Vineyard	95.0	Washington	Walla Walla Valley (WA)	Sean P. Sullivan	Cayuse 2013 Armada Vineyard Syrah (Walla Walla...	Syrah	Cayuse	94
878	US	Flying Pig	85.0	Washington	Walla Walla Valley (WA)	Paul Gregutt	Cayuse 2010 Flying Pig Red (Walla Walla Valley...	Bordeaux-style Red Blend	Cayuse	95
947	US	En Cerise Vineyard	75.0	Oregon	Walla Walla Valley (OR)	Paul Gregutt	Cayuse 2011 En Cerise Vineyard Syrah (Walla Wa...	Syrah	Cayuse	98
1306	US	Cailloux Vineyard	70.0	Washington	Walla Walla Valley (WA)	Sean P. Sullivan	Cayuse 2013 Cailloux Vineyard Viognier (Walla ...	Viognier	Cayuse	90
2276	US	God Only Knows	97.0	Oregon	Walla Walla Valley (OR)	Sean P. Sullivan	Cayuse 2014 God Only Knows Red (Walla Walla Va...	Red Blend	Cayuse	93
...	...	...	...	...	...	...	...	...	...	...
92837	US	Edith Rosé Armada Vineyard	50.0	Washington	Walla Walla Valley (WA)	Sean P. Sullivan	Cayuse 2014 Edith Rosé Armada Vineyard Grenach...	Grenache	Cayuse	91
98235	US	En Chamberlain Vineyard	75.0	Washington	Walla Walla Valley (WA)	Sean P. Sullivan	Cayuse 2012 En Chamberlain Vineyard Syrah (Wal...	Syrah	Cayuse	95
99513	US	Armada Vineyard	97.0	Oregon	Walla Walla Valley (OR)	Sean P. Sullivan	Cayuse 2014 Armada Vineyard Syrah (Walla Walla...	Syrah	Cayuse	93
100539	US	Wallah Wallah Special #4	165.0	Washington	Walla Walla Valley (WA)	Sean P. Sullivan	Cayuse 2012 Wallah Wallah Special #4 Syrah (Wa...	Syrah	Cayuse	94
101666	US	Cailloux Vineyard	75.0	Washington	Walla Walla Valley (WA)	Sean P. Sullivan	Cayuse 2014 Cailloux Vineyard Viognier (Walla ...	Viognier	Cayuse	91

65 rows × 10 columns

2) EDA – 데이터 점검

- 데이터 확인 작업은 시각화를 통해 확인했던 points와 의미가 있어보이는 데이터를 직접 확인해보았다.
  - 지정한 칼럼 이외의 칼럼에 공통적인 값이 있는지 확인
  - 정보의 신뢰도를 위해 해당하는 칼럼의 수 확인
- 이를 통해 체크해둔 의미 있는 변수들 중 일부를 제거하고, 추가적인 확인 사항을 체크
  - country = England: taster가 모두 동일하고, variety가 Sparkling Blend에 집중
  - designation = Barrel Sample: taster가 모두 동일하고, variety 동일, 모두 France산
  - winery = Cayuse/Betz Family/Gramercy: country가 모두 US로 동일하고, region_1 유사, taster는 두 명으로 축소, variety는 'Syrah'에 집중

2) EDA – 데이터 점검

- **country = England**: taster가 모두 동일하고, variety가 Sparkling Blend에 집중

```
In [60]: df.points.describe()

Out[60]: count    97162.000000
         mean        88.447860
         std          3.084275
         min         80.000000
         25%         86.000000
         50%         88.000000
         75%         91.000000
         max        100.000000
         Name: points, dtype: float64
```

```
In [61]: taster = ['Anne Krebiehl\xa0MW', 'Roger Voss', 'Sean P. Sullivan', 'Paul Gregutt']

         taster_points_mean = df.query(f'taster_name in {taster}').groupby('taster_name').mean().points
         taster_points_mean

Out[61]: taster_name
         Anne Krebiehl MW    90.597981
         Paul Gregutt        89.084312
         Roger Voss          88.737077
         Sean P. Sullivan    88.724422
         Name: points, dtype: float64
```

2) EDA – 데이터 점검

- **country = England**: taster가 모두 동일하고, variety가 Sparkling Blend에 집중

```
In [62]: variety = ['Sparkling Blend', 'Syrah', 'Bordeaux-style White Blend', 'Bordeaux-style Red Blend']

         variety_points_mean = df.query(f'variety in {variety}').groupby('variety').mean().points
         variety_points_mean

Out[62]: variety
         Bordeaux-style Red Blend      89.140176
         Bordeaux-style White Blend    88.702864
         Sparkling Blend               87.993129
         Syrah                         89.318502
         Name: points, dtype: float64
```

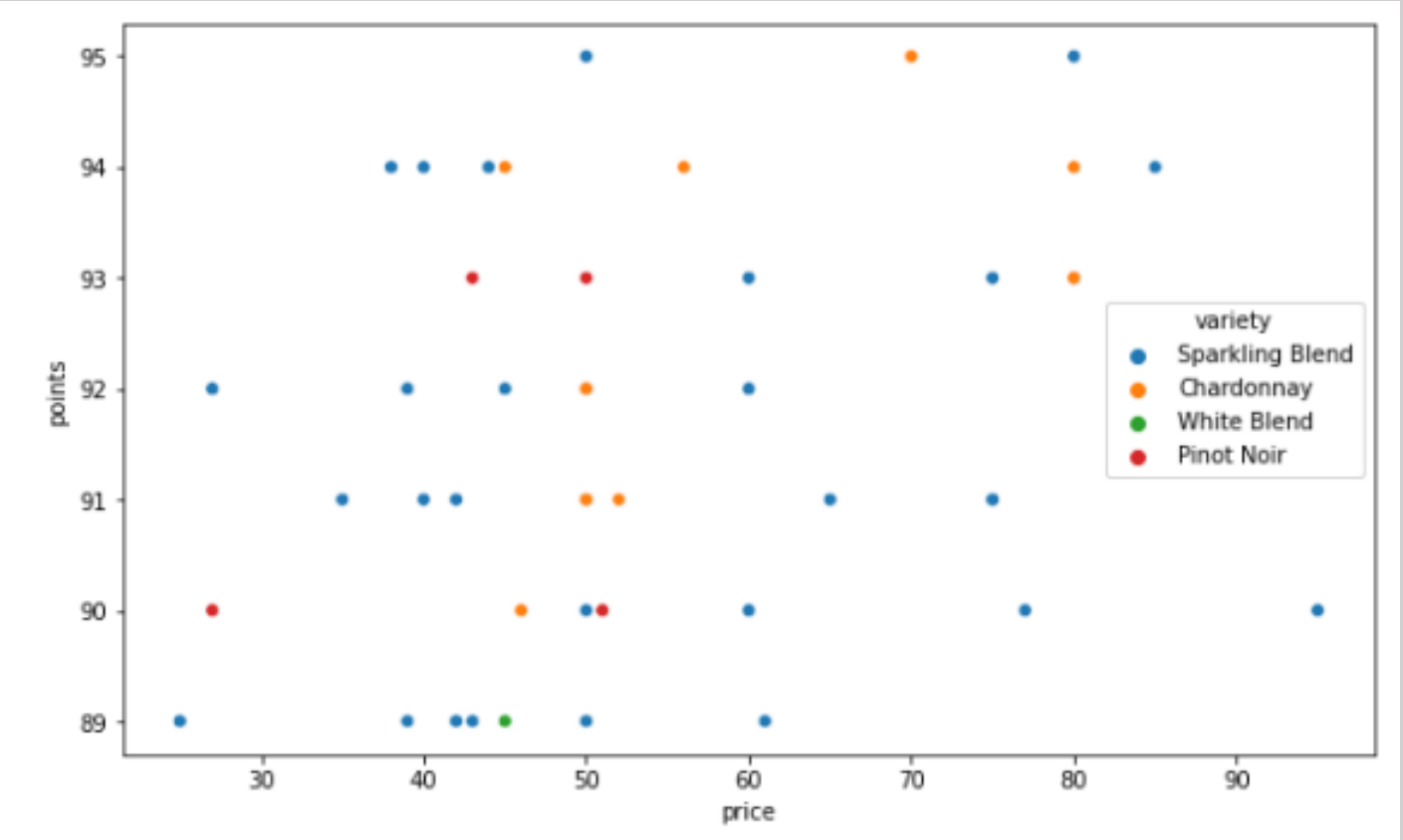
- taster가 전체를 기준으로 보았을 때에도 상당히 점수를 잘 주는 taster에 해당함. 하지만 해당 taster가 세 개 국가를 평가하는데, 그 중 영국에 특히 높은 점수를 부여하는 것을 알 수 있음.
- Sparkling Blend는 점수의 기댓값이 평균 이하에 해당하는 variet임. 하지만, 이 variety가 특히 영국에서 높은 퍼포먼스를 보이고 있음.

3) EDA – 인사이트 정리

- winery: Cayuse, variety: Syrah
    - US라는 국가 제품의 리스크가 크고, 두 명의 taster 중 한 명은 다소 편차가 있는 평가를 부여하는 사람임. 하지만 variety Syrah의 경우 variety라는 변수 자체가 카테고리별 차이가 큰 변수는 아니지만, Syrah의 분포가 긍정적인 선택지일 것으로 보임. 특히 Cayuse winer와 결합되었을 때 퍼포먼스가 좋음.
- country: England, variety: Sparkling Blend
    - 평가하는 taster가 가장 점수를 잘 주는 taster이자 영국 제품에 호의적인 taste임(Austria, France, England와 비교했을 때). 또한, Sparkling Blend라는 variety는 평균 이하의 기댓값을 받고 있지만 영국에서 좋은 퍼포먼스를 내고 있음.
- designation: Barrel Sample
    - taste가 점수의 편차가 다소 크기 때문에 리스크를 감수할 필요가 있고, variet나 countr에 대해서도 인상적인 특징이 보이지 않아서 3순위로 고려할 항목일 것으로 생각된다.

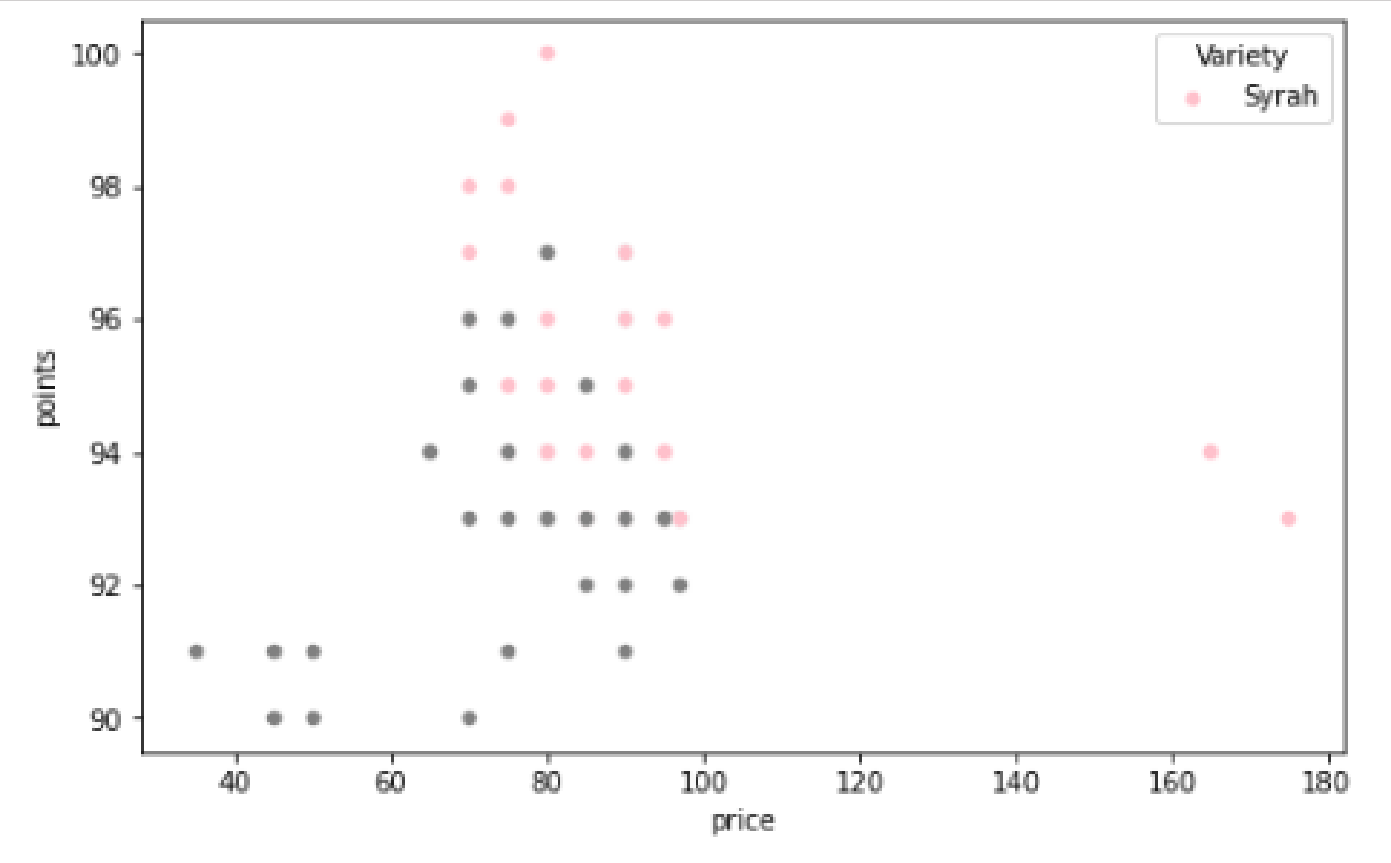
3) EDA – 인사이트 보완

- 영국과 Sparkling Blend variety의 결합은 효과가 있는가?



3) EDA – 인사이트 보완

- Cayuse winery와 Syrah의 결합이 효과가 있는가?



3) EDA – 인사이트 정리 2

- winery: Cayuse, variety: Syrah
    - Syrah라는 variety, Cayuse라는 winery 각각이 좋은 퍼포먼스를 내고 있을 뿐만 아니라, Cayuse 내에서 보더라도 Syrah가 가격 측면에서도, point 측면에서도 와인 생산자 입장에서는 좋은 선택지가 될 것으로 보임.
- country: England, variety: Sparkling Blend
    - England라는 국가의 점수대가 높고, Sparkling Blend라는 variety가 지배적이며, 해당 variety는 England에서 유독 높은 점수를 받고 있음. 하지만, England 내부적으로 보았을 때에는 Sparkling Blend를 선택하는 것이 최고의 선택지는 아닐 것으로 판단됨.
- designation: Barrel Sample
    - 모든 변수에 있어서 무난한 리스크와 점수를 받을 수 있는 선택지.

1. Linear Regression

목차

1. 데이터 전처리

```
train_df.drop(columns = ['taster_twitter_handle', 'title', 'designation', 'region_1', 'region_2'], inplace=True)
```

```
train_df['description_length'] = train_df.description.str.len()
```

```
mean_length = train_df.groupby('taster_name').mean()
mean_length.rename(columns={'description_length':'mean_length'}, inplace=True)
mean_length = mean_length.loc[:, 'mean_length']
```

```
train_df = pd.merge(train_df, mean_length, how='left', on='taster_name')
```

```
train_df['length_ratio'] = train_df['description_length']/train_df['mean_length']
```

```
train_df.drop(columns=['description', 'winery', 'description_length', 'mean_length'], inplace=True)
```

```
drop_index = train_df[train_df.isna().sum(axis=1) > 3].index
train_df.drop(index=drop_index, inplace=True)
```

```
normal_train_df = train_df.copy()
log_train_df = train_df.copy()
log_train_df['price'] = np.log(train_df['price'])
```

1. 데이터 전처리 – 범주형:get_dummies()

```
normal_train = pd.DataFrame()
for col in ['country', 'province', 'taster_name']:
    encoding_table = pd.get_dummies(normal_train_df[col], prefix=col)
    normal_train = pd.merge(normal_train, encoding_table, left_index=True, right_index=True, how='outer')

normal_train.head(3)
```

country_Argentina	country_Armenia	country_Australia	country_Austria	country_Bosnia and Herzegovina	country_Brazil	country_Bulgaria	co

```
normal_train = pd.merge(normal_train, normal_train_df[['price', 'length_ratio', 'points']],
                        left_index=True, right_index=True, how='outer')

normal_train.head(3)
```

ıael ınner	taster_name_Mike DeSimone	taster_name_Paul Gregutt	taster_name_Roger Voss	taster_name_Sean P. Sullivan	taster_name_Susan Kostrzewa	taster_name_Virginie Boone	price	length_ratio	points
				0	0	0	5.0	0.438881	83
				0	0	0	12.0	0.751700	85
				0	0	0	9.0	0.808542	86

```
log_train = normal_train.copy()
log_train['price'] = log_train_df['price']
log_train.head(3)
```

2. 선형회귀모델 적합

- 1) price normal scale & null value drop

```
drop_index = normal_train[normal_train.isna().sum(axis=1)>0].index
normal_drop_df = normal_train.drop(index=drop_index)
```

```
normal_drop_x = normal_drop_df.loc[:, :'length_ratio']
normal_drop_y = normal_drop_df.loc[:, 'points']
```

```
x_train, x_test, y_train, y_test = train_test_split(normal_drop_x, normal_drop_y, train_size=0.8, test_size=0.2, random_state = 77)

# 모델 적합
mlr = LinearRegression()
mlr.fit(x_train, y_train)
```

```
LinearRegression()
```

```
print('회귀계수 값:', mlr.coef_)
print('상수 값:', mlr.intercept_)
```

```
 -6.74725734e-01  1.48778915e-01  8.04939106e-01  4.18548376e+11
  1.56452276e-01 -1.36738205e+00 -5.87050303e+00 -5.18917531e-01
  2.57018086e-01 -1.62502261e+00  2.08121239e+00  2.90661907e+00
```

2. 선형회귀모델 적합

- 1) price normal scale & null value drop

```
# summary
model = sm.OLS(y_train, sm.add_constant(x_train)).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	points	**R-squared:**	0.488
Model:	OLS	**Adj. R-squared:**	0.484
Method:	Least Squares	**F-statistic:**	142.1
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	01:40:00	**Log-Likelihood:**	-1.3384e+05
No. Observations:	61756	**AIC:**	2.685e+05
Df Residuals:	61344	**BIC:**	2.722e+05
Df Model:	411		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	78.2329	0.342	228.546	0.000	77.562	78.904
country_Argentina	-0.1492	0.266	-0.560	0.575	-0.671	0.373
country_Armenia	-0.0074	1.079	-0.007	0.995	-2.122	2.108
country_Australia	1.2666	0.328	3.864	0.000	0.624	1.909
country_Austria	0.5650	0.383	1.474	0.140	-0.186	1.316

Train set 적합

```
y_train_predict = mlr.predict(x_train)
r_squared = r2_score(y_train, y_train_predict)
print('Variance score(R-square): {0:.3f}'.format(r_squared))
```

```
Variance score(R-square): 0.482
```

test set 적합

```
y_test_predict = mlr.predict(x_test)
test_r_squared = r2_score(y_test, y_test_predict)
print('Variance score(R-square): {0:.3f}'.format(test_r_squared))
```

```
Variance score(R-square): 0.467
```

2. 선형회귀모델 적합

- 2) normal scale & fill null value by mean

```
normal_mean_df = normal_train.copy()
normal_mean_df.price = normal_train.price.fillna(normal_train.price.mean())
```

```
normal_mean_df = normal_train.fillna(0)
```

```
normal_mean_x = normal_mean_df.loc[:, :'length_ratio']
normal_mean_y = normal_mean_df.loc[:, 'points']
```

```
x_train, x_test, y_train, y_test = train_test_split(normal_mean_x, normal_mean_y, train_size=0.8, test_size=0.2, random_state = 77)

# 모델 적합
mlr = LinearRegression()
mlr.fit(x_train, y_train)
```

```
LinearRegression()
```

2. 선형회귀모델 적합

- 2) normal scale & fill null value by mean

```
# summary
model = sm.OLS(y_train, sm.add_constant(x_train)).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	points	**R-squared:**	0.401
Model:	OLS	**Adj. R-squared:**	0.398
Method:	Least Squares	**F-statistic:**	132.5
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	01:40:06	**Log-Likelihood:**	-1.8922e+05
No. Observations:	83180	**AIC:**	3.793e+05
Df Residuals:	82761	**BIC:**	3.832e+05
Df Model:	418		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	85.9303	0.440	195.458	0.000	85.069	86.792
country_Argentina	0.0820	0.309	0.265	0.791	-0.524	0.688
country_Armenia	-0.4334	0.866	-0.501	0.617	-2.131	1.264
country_Australia	1.0868	0.390	2.788	0.005	0.323	1.851
country_Austria	1.0314	0.446	2.310	0.021	0.156	1.907

```
y_train_predict = mlr.predict(x_train)
r_squared = r2_score(y_train, y_train_predict)
print('Variance score(R-square): {0:.3f}'.format(r_squared))
```

```
Variance score(R-square): 0.401
```

```
y_test_predict = mlr.predict(x_test)
r_squared = r2_score(y_test, y_test_predict)
print('Variance score(R-square): {0:.3f}'.format(r_squared))
```

```
Variance score(R-square): 0.367
```

2. 선형회귀모델 적합

- 3) price log scale & drop null value

```
drop_index = log_train[log_train.isna().sum(axis=1)>0].index
log_drop_df = log_train.drop(index=drop_index)
```

```
log_drop_x = log_drop_df.loc[:, :'length_ratio']
log_drop_y = log_drop_df.loc[:, 'points']
```

```
x_train, x_test, y_train, y_test = train_test_split(log_drop_x, log_drop_y, train_size=0.8, test_size=0.2, random_state = 77)

# 모델 적합
mlr = LinearRegression()
mlr.fit(x_train, y_train)
```

```
LinearRegression()
```

```
print('회귀계수 값:', mlr.coef_)
print('상수 값:', mlr.intercept_)
```

2. 선형회귀모델 적합

- 3) price: log scale & drop null value

```
# summary
model = sm.OLS(y_train, sm.add_constant(x_train)).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	points	**R-squared:**	0.572
Model:	OLS	**Adj. R-squared:**	0.569
Method:	Least Squares	**F-statistic:**	199.7
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	01:40:11	**Log-Likelihood:**	-1.2827e+05
No. Observations:	61756	**AIC:**	2.574e+05
Df Residuals:	61344	**BIC:**	2.611e+05
Df Model:	411		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	73.4623	0.315	233.487	0.000	72.846	74.079
country_Argentina	0.1340	0.243	0.550	0.582	-0.343	0.611
country_Armenia	0.2414	0.986	0.245	0.807	-1.691	2.174
country_Australia	1.2179	0.300	4.066	0.000	0.631	1.805
country_Austria	0.4541	0.350	1.297	0.195	-0.232	1.141
country_Bosnia and Herzegovina	-0.3127	0.704	-0.444	0.657	-1.692	1.066

```
y_train_predict = mlr.predict(x_train)
r_squared = r2_score(y_train, y_train_predict)
print('Variance score(R-square): {0:.3f}'.format(r_squared))
```

```
Variance score(R-square): 0.572
```

```
y_test_predict = mlr.predict(x_test)
test_r_squared = r2_score(y_test, y_test_predict)
print('Variance score(R-square): {0:.3f}'.format(test_r_squared))
```

```
Variance score(R-square): -5931847847196464128.000
```

2. 선형회귀모델 적합

- 4) price: log scale & fill null value by mean

```
log_mean_df = log_train.copy()
log_mean_df.price = log_train.price.fillna(log_train.price.mean())

log_mean_df = log_mean_df.fillna(0)

log_mean_x = log_mean_df.loc[:, :'length_ratio']
log_mean_y = log_mean_df.loc[:, 'points']
```

```
x_train, x_test, y_train, y_test = train_test_split(log_mean_x, log_mean_y, train_size=0.8, test_size=0.2, random_state = 77)

# 모델 적합
mlr = LinearRegression()
mlr.fit(x_train, y_train)
```

```
LinearRegression()
```

2. 선형회귀모델 적합

- 4) price: log scale & fill null value by mean

```
# summary
model = sm.OLS(y_train, sm.add_constant(x_train)).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	points	**R-squared:**	0.502
Model:	OLS	**Adj. R-squared:**	0.500
Method:	Least Squares	**F-statistic:**	199.6
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	01:40:17	**Log-Likelihood:**	-1.8153e+05
No. Observations:	83180	**AIC:**	3.639e+05
Df Residuals:	82761	**BIC:**	3.678e+05
Df Model:	418		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	79.2839	0.403	196.673	0.000	78.494	80.074
country_Argentina	0.3865	0.282	1.370	0.171	-0.166	0.939
country_Armenia	-0.0781	0.789	-0.099	0.921	-1.626	1.469
country_Australia	1.0837	0.355	3.049	0.002	0.387	1.780
country_Austria	0.7014	0.407	1.723	0.085	-0.096	1.499
country_Bosnia and Herzegovina	-0.2924	0.785	-0.373	0.709	-1.830	1.245
country_Brazil	-1.1711	0.580	-2.018	0.044	-2.308	-0.034
country_Bulgaria	-0.0787	0.572	-0.137	0.891	-1.201	1.043

```
y_train_predict = mlr.predict(x_train)
r_squared = r2_score(y_train, y_train_predict)
print('Variance score(R-square): {0:.3f}'.format(r_squared))
```

```
Variance score(R-square): 0.502
```

```
y_test_predict = mlr.predict(x_test)
test_r_squared = r2_score(y_test, y_test_predict)
print('Variance score(R-square): {0:.3f}'.format(test_r_squared))
```

```
Variance score(R-square): -12933647256853471232O.000
```

3. Feature Selection

- 1) 값이 10개 미만인 변수 None으로 대체

```
drop_list = []
for value in train_df.taster_name.unique():
    try:
        if train_df.taster_name.value_counts()[value] < 10:
            drop_list.append(value)
    except:
        print(value)
```

```
train_df.drop(columns=['variety'], inplace=True)
```

```
country_drop = ['Luxembourg', 'Armenia', 'Egypt', 'Bosnia and Herzegovina', 'Switzerland', 'China', 'Slovakia', 'Cyprus', 'Czech Republic', 'India']
province_drop = ['Cederberg', 'Attica', 'Korčula', 'Santa Cruz', 'Alenquer', 'Samos', 'Algarve', 'Kentucky', 'Waitaki Valley', 'Tarnave', 'Krania Olympus', '
                 'Progreso', 'Neusiedlersee', 'Cauquenes Valley', 'Cappadocia', 'Dealurile Munteniei', 'Waipara Valley', 'Waiheke Island', 'Primorska', 'Mont

for country in country_drop:
    train_df.country = train_df.country.str.replace(country, "None")
for province in province_drop:
    train_df.province = train_df.province.str.replace(province, "None")
train_df.taster_name = train_df.taster_name.str.replace('Christina Pickard', "None")
```

3. Feature Selection

- 1) 값이 10개 미만인 변수 None으로 대체

```
x_train, x_test, y_train, y_test = train_test_split(log_drop_x, log_drop_y, train_size=0.8, test_size=0.2, random_state = 77)

# 모델 적합
mlr = LinearRegression()
mlr.fit(x_train, y_train)
```

```
# summary
model = sm.OLS(y_train, sm.add_constant(x_train)).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	points	**R-squared:**	0.571
Model:	OLS	**Adj. R-squared:**	0.569
Method:	Least Squares	**F-statistic:**	324.3
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	01:40:32	**Log-Likelihood:**	-1.2840e+05
No. Observations:	61756	**AIC:**	2.573e+05
Df Residuals:	61503	**BIC:**	2.596e+05
Df Model:	252		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	73.4694	0.250	293.693	0.000	72.979	73.960
country_Argentina	0.1132	0.208	0.544	0.587	-0.295	0.521
country_Australia	1.2310	0.245	5.030	0.000	0.751	1.711
country_Austria	-1.4174	0.676	-2.097	0.036	-2.742	-0.093

```
y_train_predict = mlr.predict(x_train)
r_squared = r2_score(y_train, y_train_predict)
print('Variance score(R-square): {0:.3f}'.format(r_squared))
```

```
Variance score(R-square): 0.571
```

```
y_test_predict = mlr.predict(x_test)
test_r_squared = r2_score(y_test, y_test_predict)
print('Variance score(R-square): {0:.3f}'.format(test_r_squared))
```

```
Variance score(R-square): 0.559
```

3. Feature Selection

- 2) 값이 10개 미만인 변수 None으로 대체+VIF Factor가 100 이상인 변수 삭제

```
log_drop_train = pd.DataFrame()
for col in ['country', 'province', 'taster_name']:
    encoding_table = pd.get_dummies(train_df[col], prefix=col)
    log_drop_train = pd.merge(log_drop_train, encoding_table,
                              left_index=True, right_index=True, how='outer')

log_drop_train = pd.merge(log_drop_train, train_df[['price', 'length_ratio', 'points']],
                          left_index=True, right_index=True, how='outer')

drop_index = log_drop_train[log_drop_train.isna().sum(axis=1)>0].index
log_drop_df = log_drop_train.drop(index=drop_index)

log_drop_x = log_drop_df.loc[:, :'length_ratio']
log_drop_y = log_drop_df.loc[:, 'points']

vif = pd.DataFrame()
vif['VIF Factor'] = [variance_inflation_factor(log_drop_x.values, i) for i in range(log_drop_x.shape[1])]
vif['features'] = log_drop_x.columns
```

```
drop_cols = vif[vif['VIF Factor'] > 100].features.values
log_train_df.drop(columns=drop_cols)
```

3. Feature Selection

- 2) 값이 10개 미만인 변수 None으로 대체+VIF Factor가 100 이상인 변수 삭제

```
x_train, x_test, y_train, y_test = train_test_split(log_drop_x, log_drop_y, train_size=0.8, test_size=0.2, random_state = 77)

# 모델 적합
mlr = LinearRegression()
mlr.fit(x_train, y_train)
```

```
y_train_predict = mlr.predict(x_train)
r_squared = r2_score(y_train, y_train_predict)
print('Variance score(R-square): {0:.3f}'.format(r_squared))
```

```
Variance score(R-square): 0.543
```

```
y_test_predict = mlr.predict(x_test)
test_r_squared = r2_score(y_test, y_test_predict)
print('Variance score(R-square): {0:.3f}'.format(test_r_squared))
```

```
Variance score(R-square): 0.536
```

```
# summary
model = sm.OLS(y_train, sm.add_constant(x_train)).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	points	**R-squared:**	0.543
Model:	OLS	**Adj. R-squared:**	0.542
Method:	Least Squares	**F-statistic:**	381.1
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	01:59:48	**Log-Likelihood:**	-1.3031e+05
No. Observations:	61756	**AIC:**	2.610e+05
Df Residuals:	61563	**BIC:**	2.628e+05
Df Model:	192		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	77.1845	0.048	1602.197	0.000	77.090	77.279
country_Brazil	-3.2910	0.471	-6.983	0.000	-4.215	-2.367

3. Feature Selection

- 3) VIF Factor가 100 이상인 변수 삭제

```
x_train, x_test, y_train, y_test = train_test_split(log_drop_x, log_drop_y, train_size=0.8, test_size=0.2, random_state = 77)

# 모델 적합
mlr = LinearRegression()
mlr.fit(x_train, y_train)
```

```
# summary
model = sm.OLS(y_train, sm.add_constant(x_train)).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	points	**R-squared:**	0.527
Model:	OLS	**Adj. R-squared:**	0.527
Method:	Least Squares	**F-statistic:**	2221.
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	01:59:49	**Log-Likelihood:**	-1.3136e+05
No. Observations:	61756	**AIC:**	2.628e+05
Df Residuals:	61724	**BIC:**	2.631e+05
Df Model:	31		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	77.0121	0.047	1648.431	0.000	76.921	77.104
country_Brazil	-2.8341	0.371	-7.641	0.000	-3.561	-2.107
country_Bulgaria	0.1075	0.267	0.402	0.687	-0.416	0.631

```
y_train_predict = mlr.predict(x_train)
r_squared = r2_score(y_train, y_train_predict)
print('Variance score(R-square): {0:.3f}'.format(r_squared))
```

```
Variance score(R-square): 0.527
```

```
y_test_predict = mlr.predict(x_test)
test_r_squared = r2_score(y_test, y_test_predict)
print('Variance score(R-square): {0:.3f}'.format(test_r_squared))
```

```
Variance score(R-square): 0.524
```

3. Feature Selection

- 3) VIF Factor가 100 이상인 변수 삭제

```
x_train, x_test, y_train, y_test = train_test_split(log_drop_x, log_drop_y, train_size=0.8, test_size=0.2, random_state = 77)

# 모델 적합
mlr = LinearRegression()
mlr.fit(x_train, y_train)
```

```
# summary
model = sm.OLS(y_train, sm.add_constant(x_train)).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	points	**R-squared:**	0.543
Model:	OLS	**Adj. R-squared:**	0.542
Method:	Least Squares	**F-statistic:**	381.1
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	01:59:48	**Log-Likelihood:**	-1.3031e+05
No. Observations:	61756	**AIC:**	2.610e+05
Df Residuals:	61563	**BIC:**	2.628e+05
Df Model:	192		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	77.1845	0.048	1602.197	0.000	77.090	77.279
country_Brazil	-3.2910	0.471	-6.983	0.000	-4.215	-2.367
country_Bulgaria	-1.7289	0.714	-2.422	0.015	-3.128	-0.330

3. Feature Selection

- 1) 값이 10개 미만인 변수 None으로 대체
- variance score(r^2) (Train) 0.571
- variance score(r^2) (Test) 0.559

- 2) 값이 10개 미만인 변수 None으로 대체+VIF Factor가 100 이상인 변수 삭제
- variance score(r^2) (Train) 0.543
- variance score(r^2) (Test) 0.536

- 3) VIF Factor가 100 이상인 변수 삭제
- variance score(r^2) (Train) 0.527
- variance score(r^2) (Test) 0.524

3. Feature Selection

- 4) province 칼럼 제거, 모델 적합

```
drop_cols = []
for col in log_drop_df.columns:
    if "province_" in col:
        drop_cols.append(col)

log_drop_df.drop(columns=drop_cols, inplace=True)
```

```
log_drop_x = log_drop_df.loc[:, :'length_ratio']
log_drop_y = log_drop_df.loc[:, 'points']
```

```
x_train, x_test, y_train, y_test = train_test_split(log_drop_x, log_drop_y, train_size=0.8, test_size=0.2, random_state = 77)

# 모델 적합
mlr = LinearRegression()
```

3. Feature Selection

- 4) province 칼럼 제거, 모델 적합

```
# summary
model = sm.OLS(y_train, sm.add_constant(x_train)).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	points	**R-squared:**	0.527
Model:	OLS	**Adj. R-squared:**	0.527
Method:	Least Squares	**F-statistic:**	2221.
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	01:59:49	**Log-Likelihood:**	-1.3136e+05
No. Observations:	61756	**AIC:**	2.628e+05
Df Residuals:	61724	**BIC:**	2.631e+05
Df Model:	31		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	77.0121	0.047	1648.431	0.000	76.921	77.104
country_Brazil	-2.8341	0.371	-7.641	0.000	-3.561	-2.107
country_Bulgaria	0.1075	0.267	0.402	0.687	-0.416	0.631
country_Croatia	-1.5671	0.316	-4.958	0.000	-2.187	-0.948
country_Greece	1.9098	0.167	11.409	0.000	1.582	2.238
country_Hungary	-0.3503	0.257	-1.362	0.173	-0.855	0.154
country_Israel	-0.3834	0.165	-2.318	0.020	-0.708	-0.059

2. Decision Tree

목차

0. 데이터 전처리

- train, test 데이터 통합 후 전처리

```
#데이터 불러오기
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

#데이터 통합(onehotencoding 위해 통합)
tgt = pd.concat([train, test], ignore_index=True)
```

```
#중복값 제거
tgt = tgt.drop_duplicates()
```

```
#description 통해 새로운 변수들 생성
tgt['length'] =tgt['description'].apply(lambda x : len(str(x).split(" ")))
tgt = pd.merge(tgt, tgt.groupby('taster_name')['length'].mean(), how='left', on='taster_name')
```

```
tgt.rename(columns={'length_x':'length', 'length_y':'length_mean'}, inplace=True )
tgt['length_ratio'] = tgt['length']/tgt['length_mean']
```

0. 데이터 전처리 – feature selection

```
#모델링에 사용할 feature 선택
features = ['price', 'length', 'country', 'province', 'taster_name', 'variety', 'length_ratio']

categorical_columns = ['country', 'province', 'taster_name', 'variety']
num_columns = ['price', 'length', 'length_ratio']
```

```
#feature 중에 categorical variable 결측치 처리 (None으로 대체)

# categorical_columns = ['country', 'province', 'taster_name', 'variety']

tgt[categorical_columns] = tgt[categorical_columns].fillna('None')
```

```
tgt[categorical_columns].head()
```

	country	province	taster_name	variety
0	Australia	Australia Other	Joe Czerwinski	Chardonnay
1	France	Rhône Valley	Roger Voss	Rosé
2	Spain	Northern Spain	Michael Schachner	Verdejo-Viura
3	US	California	None	Cabernet Sauvignon
4	US	California	None	Pinot Noir

0. 데이터 전처리 – One Hot encoding

```
#categorical variables - OneHotEncoding 진행

from sklearn.preprocessing import OneHotEncoder

OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse= False)
OH_cols_tgt = pd.DataFrame(OH_encoder.fit_transform(tgt[categorical_columns])) # 모든 categorical variable 대한 onehotencoding
OH_cols_tgt.index = tgt.index # 인덱스 복원
OH_cols_tgt.columns = OH_encoder.get_feature_names_out(categorical_columns) #column 명 복원
```

```
OH_cols_tgt
```

	country_Argentina	country_Armenia	country_Australia	country_Austria	country_Bosnia and Herzegovina	country_Brazil	country_Bulgaria	country_Canada	countr
0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
123165	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
123166	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
123167	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
123168	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

0. 데이터 전처리 – 연속형 변수 결측치

```
# feature 중에 numerical variable 처리

from sklearn.impute import SimpleImputer

num_tgt = tgt[num_columns]

my_imputer = SimpleImputer(strategy='mean') #평균으로 결측치 대체
# my_imputer = SimpleImputer(strategy='median')
imputed_num_tgt = pd.DataFrame(my_imputer.fit_transform(num_tgt), columns =num_tgt.columns)

imputed_num_tgt.index = num_tgt.index #기존의 index 복원
```

```
# 각각 결측치 처리한 데이터 하나로 통합
alltogether = pd.concat([imputed_num_tgt, OH_cols_tgt, tgt['points']], axis = 1)
```

0. 데이터 전처리 - train, test set 구분

```
# 각각 결측치 처리한 데이터 하나로 통합
alltogether = pd.concat([imputed_num_tgt, OH_cols_tgt, tgt['points']], axis = 1)
```

```
alltogether
```

	price	length	length_ratio	country_Argentina	country_Armenia	country_Australia	country_Austria	country_Bosnia and Herzegovina	country_Brazil	country_Bul
0	5.0	18.0	0.438726	0.0	0.0	1.0	0.0	0.0	0.0	
1	12.0	31.0	0.825427	0.0	0.0	0.0	0.0	0.0	0.0	
2	9.0	33.0	0.769747	0.0	0.0	0.0	0.0	0.0	0.0	
3	29.0	40.0	1.000000	0.0	0.0	0.0	0.0	0.0	0.0	
4	40.0	35.0	1.000000	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
123165	16.0	28.0	0.699911	0.0	0.0	0.0	0.0	0.0	0.0	

```
# points 점수의 유무를 기준으로 trainset, testset 구분

trainset = alltogether[alltogether.points.notnull()]
testset = alltogether[alltogether.points.isnull()]
```

1. 모델 적합

```
# train, valid, test 셋 구분

x = trainset.drop('points', axis = 1).copy()
y = trainset['points'].copy()

x_train, x_valid, y_train, y_valid = train_test_split(x, y,train_size = 0.8, test_size = 0.2, random_state = 1)
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

for n in [5, 10, 15]: #max_depth -> 10이 가장 결과 좋음

    #모델 적합
    model = DecisionTreeRegressor(random_state=10, max_depth=n)
    model.fit(x_train, y_train)

    #모델 예측
    valid_preds = model.predict(x_valid)

    #모델 평가
    print('features:{}'.format(features))
    print('Max Depth : {}'.format(n))
    print('MSE : {}'.format(mean_squared_error(y_valid, valid_preds)))
    print('RMSE: {}'.format(mean_squared_error(y_valid, valid_preds, squared=False)))
    print('R^2 : {}'.format(r2_score(y_valid, valid_preds)))
```

```
features:['price', 'length', 'country', 'province', 'taster_name', 'variety', 'length_ratio']
Max Depth : 5
MSE : 4.922829723963125
RMSE: 2.2187450786341194
R^2 : 0.48733372750003934
features:['price', 'length', 'country', 'province', 'taster_name', 'variety', 'length_ratio']
Max Depth : 10
MSE : 4.792795077928151
RMSE: 2.1196214468456747
R^2 : 0.5321177788263374
features:['price', 'length', 'country', 'province', 'taster_name', 'variety', 'length_ratio']
Max Depth : 15
MSE : 4.717407259967713
RMSE: 2.171959313607811
R^2 : 0.5087265391164247
```

2. 모델 성능 개선 – Grid search

```
#Grid_Search

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings("ignore")
```

```
params = {
    'max_depth': [10, 20, 50],
    'min_samples_split': [1, 5, 10, 50],
    'min_samples_leaf': [0.1, 0.03, 0.003],
    'max_leaf_nodes': [100, 200, 300],
}


grid = GridSearchCV(DecisionTreeRegressor(random_state=10),
                    params, scoring = 'r2', cv = 5,verbose = 3)
grid.fit(x_train, y_train)
```

```
grid.best_params_
```

```
{'max_depth': 20,
 'max_leaf_nodes': 300,
 'min_samples_leaf': 0.003,
 'min_samples_split': 5}
```

```
grid.best_score_
```

```
0.517257681472379
```

```
# 최적 모델 추천

y_pred = grid.predict(x_valid)

print("MSE: ", mean_squared_error(y_valid,y_pred))
print("RMSE: ", (mean_squared_error(y_valid,y_pred))**0.5)
print("MAE: ", mean_absolute_error(y_valid,y_pred))
print('R^2(Score) : {}'.format(r2_score(y_valid, valid_preds)))
```

```
MSE:  4.523538915640542
RMSE:  2.126861282651161
MAE:  1.682450094593127
R^2(Score) : 0.5087265391164247
```

2. 모델 성능 개선 – Random Search

```
#Random Search

from sklearn.model_selection import RandomizedSearchCV
params = {'max_depth': [10, 15, 20, 40],
          'min_samples_split': [1, 5, 10],
          'min_samples_leaf': [50, 25, 10, 1, 0.1, 0.003, 0.003],
          'max_leaf_nodes': [300, 400, 500],}


random_grid = RandomizedSearchCV(DecisionTreeRegressor(random_state=10),
                                 params, n_jobs=-1,
                                 scoring='r2',
                                 n_iter=100)
random_grid.fit(x_train, y_train)
```

```
random_grid.best_params_
```

```
{'min_samples_split': 10,
 'min_samples_leaf': 10,
 'max_leaf_nodes': 500,
 'max_depth': 40}
```

```
random_grid.best_score_
```

```
0.5343913679896175
```

```
#최적 모델 추천

y_pred = random_grid.predict(x_valid)

print("MSE: ", mean_squared_error(y_valid,y_pred))
print("RMSE: ", (mean_squared_error(y_valid,y_pred))**0.5)
print("MAE: ", mean_absolute_error(y_valid,y_pred))
```

```
MSE:  4.393722372024975
RMSE:  2.0961207913727145
MAE:  1.654189742179463
```

2. 모델 성능 개선 – 하이퍼파라미터 튜닝

- Grid Search 결과 추천 parameter 값 : {'max_depth': 20, 'max_leaf_nodes': 300, 'min_samples_leaf': 0.003, 'min_samples_split': 5}
  MSE: 4.523538915640542
  RMSE: 2.126861282651161
  MAE: 1.682450094593127
  R^2(Score) : 0.5087265391164247

- Random Search 결과 추천 parameter 값 : 'min_samples_split': 10, 'min_samples_leaf': 10, 'max_leaf_nodes': 500, 'max_depth': 40}
  MSE: 4.393722372024975
  RMSE: 2.0961207913727145
  MAE: 1.654189742179463
  R^2(Score) : 0.5343913679896175

-> 최종적으로 Random Search의 결과를 따르기로 결정

3. 최적 모델 적합

```
#최적 모델 적합
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

#모델 적합
model = DecisionTreeRegressor(random_state=10, min_samples_split=10, min_samples_leaf=10, max_leaf_nodes=500, max_depth=40)
model.fit(x_train, y_train)

#모델 예측
valid_preds = model.predict(x_valid)

#모델 평가
print('features:{}'.format(features))
print('MSE : {}'.format(mean_squared_error(y_valid, valid_preds)))
print('RMSE: {}'.format(mean_squared_error(y_valid, valid_preds, squared=False)))
print('R^2 : {}'.format(r2_score(y_valid, valid_preds)))
```

```
features:['price', 'length', 'country', 'province', 'taster_name', 'variety', 'length_ratio']
MSE : 4.393722372024975
RMSE: 2.0961207913727145
R^2 : 0.5424352664685824
```

4. 결과 예측

- Test set에 대한 데이터 전처리는 앞에서 완료
- 여기에서는 null 값으로 채워진 points column 만 삭제

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error


model = DecisionTreeRegressor(random_state=10, min_samples_split=10, min_samples_leaf=10, max_leaf_nodes=500, max_depth=40)
model.fit(x_train, y_train)


#모델 예측
test_pred = model.predict(testset)
```

```
# 결과를 파일로 저장
wine_prediction = pd.DataFrame({'id': testset.index, 'points': test_pred})
wine_prediction

wine_prediction.to_csv('wine_point_prediction.csv', index=False)
```

3. 랜덤포레스트 회귀

목차

1. Pycaret을 활용하여 모델 간 성능비교
2. 모델 선택
3. 모델 적합
4. 하이퍼파라미터 튜닝
5. 시각화
6. 예측결과

0. input data

```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 119988 entries, 0 to 119987
Data columns (total 7 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   country        119929 non-null  object
 1   points         119988 non-null  int64
 2   price          119988 non-null  float64
 3   province       119929 non-null  object
 4   taster_name    95071 non-null   object
 5   variety        119987 non-null  object
 6   wordcnt_ratio  119988 non-null  float64
dtypes: float64(2), int64(1), object(4)
memory usage: 7.3+ MB
```

- (X) categorical: country, province, taster_name, variety
- (X) Numerical: price, wordcnt_ratio
- (Y) Target variable: points

Price: 결측치 평균 대체 후 로그스케일링 진행.

#1. train, test set 분리

```
[ ]  data = df.sample(frac=0.8, random_state=786)
     data_unseen = df.drop(data.index)

     data.reset_index(drop=True, inplace=True)
     data_unseen.reset_index(drop=True, inplace=True)

     print('Data for Modeling: ' + str(data.shape))
     print('Unseen Data For Predictions: ' + str(data_unseen.shape))
     ## 예측용 data 10%. 빼놓기

     Data for Modeling: (95990, 7)
     Unseen Data For Predictions: (23998, 7)
```

- Train, test set를 각각 0.8:0.2로 분리

1. Pycaret을 활용하여 모델 간 성능비교

```
# pycaret에서 사용 가능한 모델 목록
models()
```

ID		Reference	Turbo
lr	Linear Regression	sklearn.linear_model._base.LinearRegression	True
lasso	Lasso Regression	sklearn.linear_model._coordinate_descent.Lasso	True
ridge	Ridge Regression	sklearn.linear_model._ridge.Ridge	True
en	Elastic Net	sklearn.linear_model._coordinate_descent.Elast...	True
lar	Least Angle Regression	sklearn.linear_model._least_angle.Lars	True
llar	Lasso Least Angle Regression	sklearn.linear_model._least_angle.LassoLars	True
omp	Orthogonal Matching Pursuit	sklearn.linear_model._omp.OrthogonalMatchingPu...	True
br	Bayesian Ridge	sklearn.linear_model._bayes.BayesianRidge	True
ard	Automatic Relevance Determination	sklearn.linear_model._bayes.ARDRegression	False
par	Passive Aggressive Regressor	sklearn.linear_model._passive_aggressive.Passi...	True
ransac	Random Sample Consensus	sklearn.linear_model._ransac.RANSACRegressor	False
tr	TheilSen Regressor	sklearn.linear_model._theil_sen.TheilSenRegressor	False
huber	Huber Regressor	sklearn.linear_model._huber.HuberRegressor	True
kr	Kernel Ridge	sklearn.kernel_ridge.KernelRidge	False
svm	Support Vector Regression	sklearn.svm._classes.SVR	False
knn	K Neighbors Regressor	sklearn.neighbors._regression.KNeighborsRegressor	True
dt	Decision Tree Regressor	sklearn.tree._classes.DecisionTreeRegressor	True
rf	Random Forest Regressor	sklearn.ensemble._forest.RandomForestRegressor	True
et	Extra Trees Regressor	sklearn.ensemble._forest.ExtraTreesRegressor	True
ada	AdaBoost Regressor	sklearn.ensemble._weight_boosting.AdaBoostRegr...	True
gbr	Gradient Boosting Regressor	sklearn.ensemble._gb.GradientBoostingRegressor	True
mlp	MLP Regressor	sklearn.neural_network._multilayer_perceptron....	False
lightgbm	Light Gradient Boosting Machine	lightgbm.sklearn.LGBMRegressor	True
dummy	Dummy Regressor	sklearn.dummy.DummyRegressor	True

➔ 24개의 모델 간 비교 가능

```
models_used = ['lr', 'dt', 'rf', 'ada', 'gbr', 'lightgbm' ]
models_not_used = [i for i in list(models().index) if i not in models_used]
```

→ 6개 모델 간 비교:

- → Linear Regression
- → Decision tree Regressor
- → Random Forest Regressor
- → Adaboost
- → Gradient Boosting Regressor
- → Light GBM

2. 모델 선택

```
# Compare all models
best = compare_models(include = models_used) # 전부 돌리면 너무 오래걸려서(gpu 사용하면 다를려나..
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
rf	Random Forest Regressor	1.4730	4.1532	2.0379	0.5665	0.0228	0.0167	140.1630
lightgbm	Light Gradient Boosting Machine	1.5895	4.2579	2.0634	0.5556	0.0231	0.0180	1.6520
gbr	Gradient Boosting Regressor	1.8190	5.2740	2.2965	0.4496	0.0257	0.0206	42.3220
ada	AdaBoost Regressor	2.0056	6.1704	2.4839	0.3561	0.0278	0.0226	65.3010
dt	Decision Tree Regressor	1.7141	7.1714	2.6777	0.2513	0.0300	0.0194	3.2960
lr	Linear Regression	3.1368	12606.4593	57.4863	-1322.3892	0.0657	0.0357	8.1740

→ R squared, RMSE 등 모든 값에서 Random Forest Regressor가 가장 성능이 높음을 알 수 있음.

→ Random Forest Regressor 선택

3. 모델 적합

#3. 모델 선택: random forest regressor

```
# random forest 모델 생성, 4 fold cross validation.
rf = create_model('rf', fold=4, verbose=True)
```

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	1.5074	4.2585	2.0636	0.5539	0.0231	0.0171
1	1.4938	4.2214	2.0546	0.5616	0.0230	0.0169
2	1.5231	4.3170	2.0777	0.5525	0.0233	0.0173
3	1.5187	4.3586	2.0877	0.5416	0.0234	0.0172
Mean	1.5107	4.2889	2.0709	0.5524	0.0232	0.0171
Std	0.0113	0.0528	0.0127	0.0071	0.0001	0.0001

→ 시간상 cross validation fold 수를 4로 지정하고 진행(default=10)

4. 하이퍼파라미터 튜닝

#4. 모델 하이퍼파라미터 튜닝

- 소요시간이 너무 길어서 반복 수를 3으로(default=3) 지정하고 진행했는데 정확도가 기존 rf모델보다 낮게 나와서 실제로 쓰진 않음.

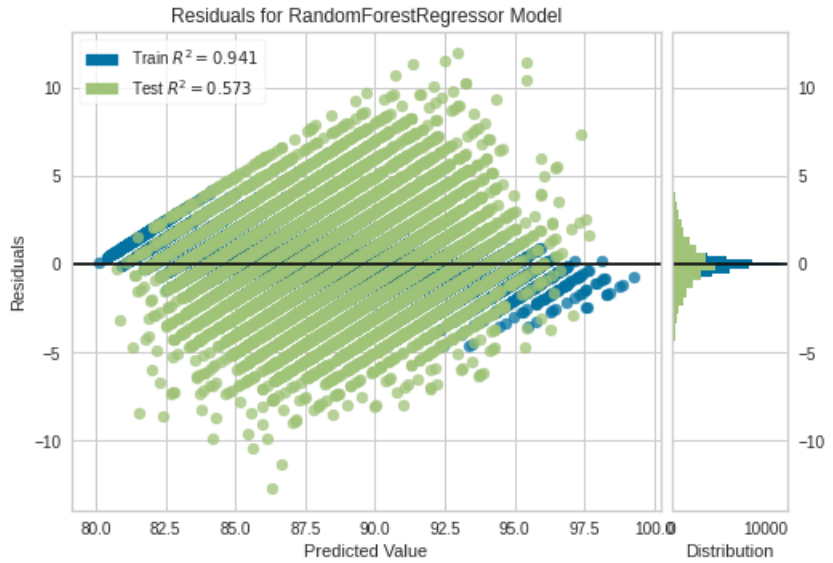
```
# Tune the model
tuned_rf = tune_model(rf, n_iter = 3)
```

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	1.9902	6.2073	2.4914	0.3532	0.0279	0.0226
1	1.9851	6.1811	2.4862	0.3495	0.0279	0.0225
2	2.0070	6.3964	2.5291	0.3306	0.0284	0.0228
3	1.9831	6.2009	2.4902	0.3578	0.0279	0.0225
4	1.9852	6.2472	2.4994	0.3508	0.0280	0.0225
5	2.0266	6.4425	2.5382	0.3452	0.0285	0.0230
6	1.9942	6.2738	2.5048	0.3404	0.0281	0.0227
7	1.9701	6.1211	2.4741	0.3431	0.0277	0.0223
8	1.9868	6.2475	2.4995	0.3507	0.0280	0.0225
9	1.9823	6.2537	2.5007	0.3483	0.0280	0.0225
Mean	1.9911	6.2572	2.5014	0.3470	0.0280	0.0226
Std	0.0148	0.0917	0.0183	0.0072	0.0002	0.0002

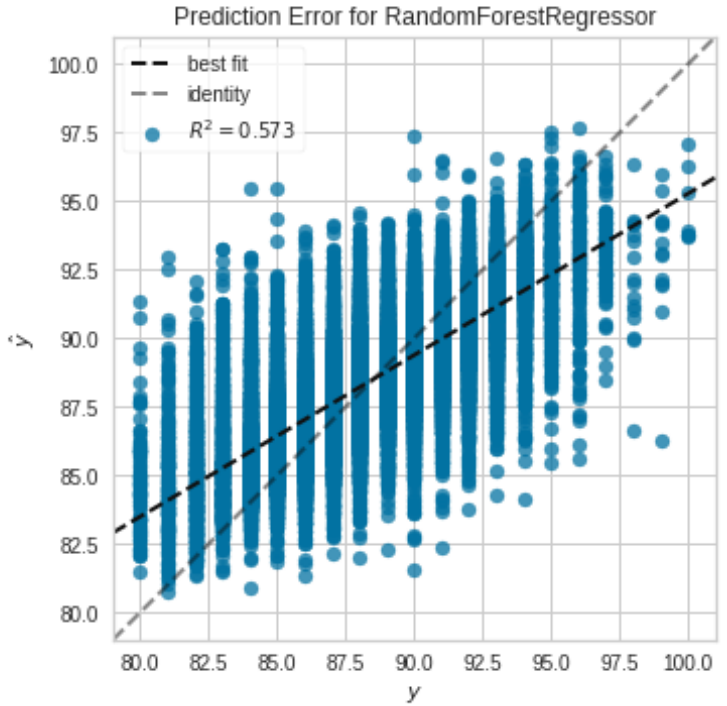
5. 결과 시각화 - 잔차

5.1. 잔차

```
plot_model(rf)
# Train set에서는 94.1% 적합, Test set에서는 57.3% 적합 -> 과적합 발생한 것으로 해석 가능.
```



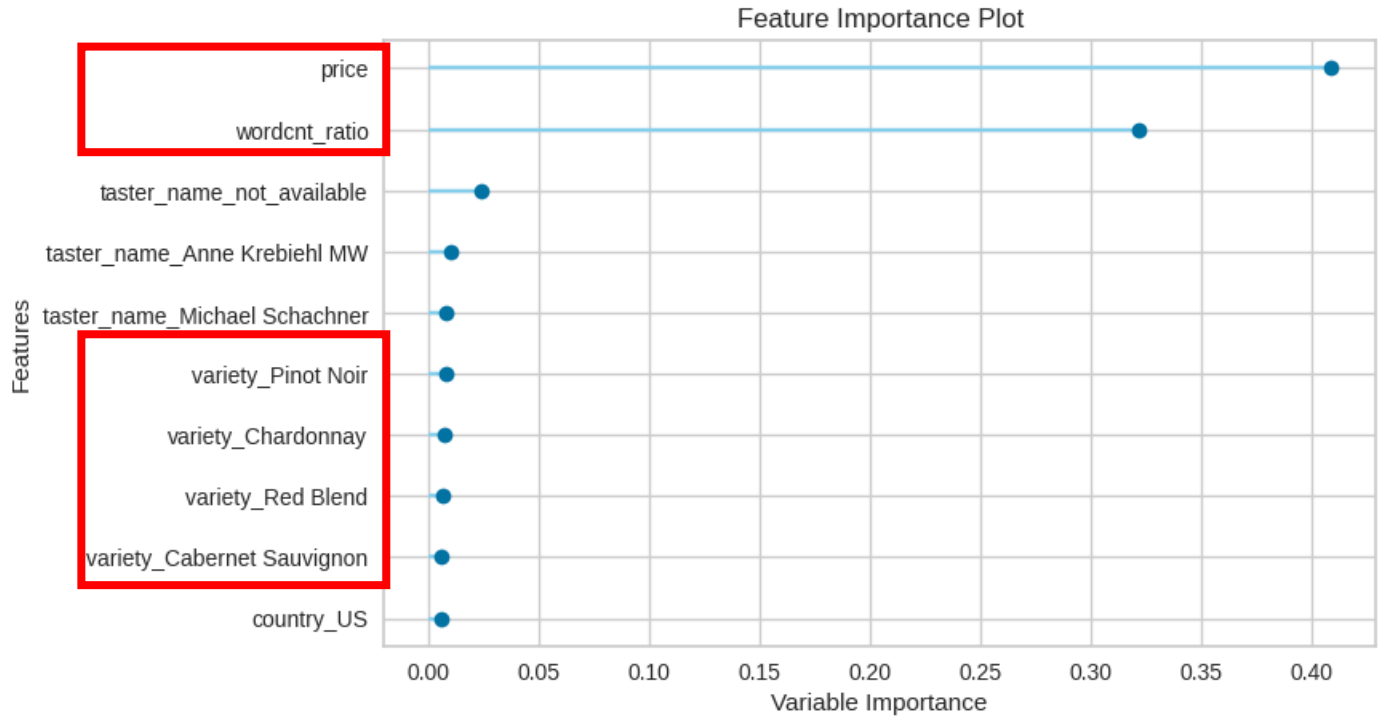

```
plot_model(rf, plot = 'error')
```



5. 결과 시각화 – Feature Importance

5.2. Feature Importance

```
# Feature Importance
plot_model(rf, plot='feature')
## Price, 리뷰 단어 수, 와인 종류(Pinot Noir, Chardonnay, Red Blend, Cabernet Sauvignon)이 기여도 높았음.
```



6. Points 예측

```
unseen_predictions = predict_model(rf, data=data_unseen)
unseen_predictions.head()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Random Forest Regressor	1.4335	4.0152	2.0038	0.5775	0.0225	0.0163

	country	points	price	province	taster_name	variety	wordcnt_ratio	Label
0	US	87	2.564949	Michigan	Alexander Peartree	Riesling	0.879686	85.53
1	Spain	87	2.708050	Northern Spain	Michael Schachner	Tempranillo-Merlot	1.119600	86.88
2	Italy	87	2.772589	Sicily & Sardinia	Kerin O'Keefe	Frappato	0.826372	87.61
3	Germany	87	2.484907	Rheinhessen	Anna Lee C. Iijima	Gewürztraminer	0.639772	87.33
4	US	87	3.526361	California	Virginie Boone	Cabernet Sauvignon	0.746400	88.07

관측값

예측값