

물리 화학적 측정 데이터로 예측하는 와인의 품질



4조 양용
2021.07.07

목차

문제 정의

소믈리에가 평가하는 와인의 품질을 단지 물리 화학적 데이터로 예측할 수 있을까?

분석 개발 환경

데이터셋 분석

데이터의 특성 분석

데이터 전처리

데이터 가공 및 전처리

모델링과 학습

- 머신러닝 모델
- 케라스 모델

모델 성능 평가

시사점 / 한계점 / 느낀점



문제 정의

와인의 물리 화학적 측정 데이터로 부터 기존에는 미각, 후각으로 측정하던 와인의 품질을 예측해본다.



소믈리에

Sommelier. '와인 감별사'를 의미하는 프랑스어 단어.

소믈리에의 특징

와인의 맛과 향, 산지 등 와인의 정보란 정보는 모조리 꿰차고 있는 전문가이며 와인의 맛, 상태 등도 감별 가능하다. 매우 섬세하고 민감한 미각이 필요하기 때문에 담배 따위 건드렸다가는 바로 퇴출당한다.
한국의 소믈리에에는 심지어 김치도 잘 못 먹는다고 한다.

소믈리에에는 환경적 영향을 많이 받는다.

소믈리에의 몸의 건강상태 및 주변 환경에 따라 미각과 후각은 영향을 받아 와인의 퀄리티를 정확하게 측정할 수 없을 수도 있다.
그러므로, 와인을 생산하는 기업에서는 항상 일정한 품질의 와인을 생산해 낼 수 있도록 물리화학적 특성 데이터를 이용하여

소믈리에를 대신해서 와인의 퀄리티를 측정!

분석 개발 환경



코딩 프로그램

- Google Colab

라이브러리

- numpy

- pandas

- seaborn

- sklearn

- keras

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

P5														
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	index	quality	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	type
0	0	5	5.6	0.695	0.06	6.8	0.042	9	84	0.99432	3.44	0.44	10.2	white
1	1	5	8.8	0.61	0.14	2.4	0.067	10	42	0.9969	3.19	0.59	9.5	red
2	2	5	7.9	0.21	0.39	2	0.057	21	138	0.99176	3.05	0.52	10.9	white
3	3	6	7	0.21	0.31	6	0.046	29	108	0.9939	3.26	0.5	10.8	white
4	4	6	7.8	0.4	0.26	9.5	0.059	32	178	0.9955	3.04	0.43	10.9	white
5	5	6	6	0.19	0.37	9.7	0.032	17	50	0.9932	3.08	0.66	12	white
6	6	5	6.1	0.22	0.49	1.5	0.051	18	87	0.9928	3.3	0.46	9.6	white
7	7	6	7.1	0.38	0.42	11.8	0.041	32	193	0.99624	3.04	0.49	10	white
8	8	5	6.8	0.24	0.31	18.3	0.046	40	142	1	3.3	0.41	8.7	white
9	9	5	6.8	0.39	0.35	11.6	0.044	57	220	0.99775	3.07	0.53	9.3	white
10	10	6	8	0.18	0.37	0.9	0.049	36	109	0.99007	2.89	0.44	12.7	red
11	11	7	6.2	0.16	0.33	1.1	0.057	21	82	0.991	3.32	0.46	10.9	white
12	12	7	7.3	0.33	0.4	6.85	0.038	32	138	0.992	3.03	0.3	11.9	white

데이터 총 건수: 5496 건

독립변수 (x): 11가지 (white, red 제외)

종속변수 (y): 1가지 (quality)

null: 없음

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

P5														
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	index	quality	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	type
0	0	5	5.6	0.695	0.06	6.8	0.042	9	84	0.99432	3.44	0.44	10.2	white
1	1	5	8.8	0.61	0.14	2.4	0.067	10	42	0.9969	3.19	0.59	9.5	red
2	2	5	7.9	0.21	0.39	2	0.057	21	138	0.99176	3.05	0.52	10.9	white
3	3	6	7	0.21	0.31	6	0.046	29	108	0.9939	3.26	0.5	10.8	white
4	4	6	7.8	0.4	0.26	9.5	0.059	32	178	0.9955	3.04	0.43	10.9	white
5	5	6	6	0.19	0.37	9.7	0.032	17	50	0.9932	3.08	0.66	12	white
6	6	5	6.1	0.22	0.49	1.5	0.051	18	87	0.9928	3.3	0.46	9.6	white
7	7	6	7.1	0.38	0.42	11.8	0.041	32	193	0.99624	3.04	0.49	10	white
8	8	5	6.8	0.24	0.31	18.3	0.046	40	142	1	3.3	0.41	8.7	white
9	9	5	6.8	0.39	0.35	11.6	0.044	57	220	0.99775	3.07	0.53	9.3	white
10	10	6	8	0.18	0.37	0.9	0.049	36	109	0.99007	2.89	0.44	12.7	red
11	11	7	6.2	0.16	0.33	1.1	0.057	21	82	0.991	3.32	0.46	10.9	white
12	12	7	7.3	0.33	0.4	6.85	0.038	32	138	0.992	3.03	0.3	11.9	white

종속변수 (y): 1가지 (quality) 이지만,

값은 3~9 까지 7가지 이므로

다중 분류로 결정

데이터셋 분석

와인 품질 데이터셋(출처: UCI)



```
df.info()
```

```
# 총 5497개의 데이터
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5497 entries, 0 to 5496
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype
---  -
0   index                 5497 non-null  int64
1   quality               5497 non-null  int64
2   fixed acidity         5497 non-null  float64
3   volatile acidity     5497 non-null  float64
4   citric acid           5497 non-null  float64
5   residual sugar        5497 non-null  float64
6   chlorides             5497 non-null  float64
7   free sulfur dioxide   5497 non-null  float64
8   total sulfur dioxide  5497 non-null  float64
9   density               5497 non-null  float64
10  pH                   5497 non-null  float64
11  sulphates            5497 non-null  float64
12  alcohol              5497 non-null  float64
13  type                 5497 non-null  object
dtypes: float64(11), int64(2), object(1)
memory usage: 601.4+ KB
```

데이터타입은 대부분 int, float 형태
type만 object

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

와인의 구성
(11가지 독립 변수)

Alcohol

Acids

Sugar

Sulfur Compounds

Other Compounds

1) Volatile Acidity

2) Fixed Acidity

3) Citric Acid

1) Total Sulfur Dioxide

2) Free Sulfur Dioxide

3) Sulphates

1) pH 2) Chlorides 3) density

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

1. Alcohol

- 도수와 단맛에 영향
- 바디감에 영향
- 알코올 농도가 낮으면 ($< 8\%$): 초산균 등에 의해 오염될 가능성
- 알코올 농도 높으면 ($> 12\%$): 유통, 보관 중에 품질이 안정적임.
- 와인의 평균 알코올 농도 5~15%

(너무 높아도 너무 낮아도 좋지 않음.)

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

2. Acids

1) Volatile Acidity (휘발성 산도)

- 효모 / 박테리아의 발효 과정에서 발생한 산
- 휘발성 산은 기화가 잘 일어나는 산
- 와인의 향에 연관
- 너무 높으면, 불쾌한 식초 맛 (Negative 요소 일 것으로 예측)
- 와인의 pH에 영향을 미침

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

2. Acids

2) Fixed Acidity (고정 산도)

- 포도에서 자연적으로 발생하는 산
- 와인의 pH에 영향을 미침

(너무 높아도 너무 낮아도 좋지 않음.)

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

2. Acids

3) Citric Acid (시트르산)

- 구연산
- 와인의 신선함을 올려주는 역할. (Positive요소 일 것으로 예측)
- pH에 영향을 미침.

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

3. Sugar

1) Residual sugar (잔류 설탕)

- 와인 발효 후, 잔류하는 설탕의 양
- 와인이 달도록 느끼게 함
- 드라이 10g / 1L 이하
- 미디엄 10 ~ 18g / 1L
- 스위트 18g / 1L 이상

(보통 스위트 와인은 가격이 저렴한 편에 속함. Negative 요소 일 것으로 예측)

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

4. Sulfur Compounds

1) Total Sulfur Dioxide (총 이산화황)

- 아황산가스
- 유리 SO₂는 와인의 미생물에 강하게 결합 -> 고정 SO₂를 생성
- 고정 SO₂ 함량이 높으면? 산화 발생 및 미생물을 손상 시킨다.
- 미생물에 대해 강한 독성
- 즉, 품질을 유지하기 위한 방부제 역할을 한다.

(품질을 유지시켜주므로 Positive 요소 일 것으로 예측)

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

4. Sulfur Compounds

2) Free Sulfur Dioxide (자유 이산화황)

- 아황산가스

- 미생물에 대해 강한 독성 -> 인간에게도 좋지 않은 영향을 줄 수 있음.

- 즉, 품질을 유지하기 위한 방부제 역할을 한다.

(품질을 유지시켜주므로 Positive 요소 일 것으로 예측)

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

4. Sulfur Compounds

3) Sulphates (황산염)

- 와인 첨가제로 항균과 항산화 작용

(품질을 유지시켜주므로 Positive 요소 일 것으로 예측)

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

5. Other Compounds

1) pH (수소이온농도지수)

- 낮을수록 산성도 강함
- pH 3.0 ~ 3.3: 신맛 강함, 품질 낮음
- pH 3.3 ~ 3.6: 적정범위 (Best)
- pH 3.6 이상: 신맛 약함, 미생물 오염 가능성 높아짐

(너무 높아도 너무 낮아도 좋지 않음.)

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

5. Other Compounds

2) Chlorides (염화물)

- 와인에 들어 있는 소금의 양 (와인의 짠맛 좌우)

(너무 높아도 너무 낮아도 좋지 않을 것으로 예상.)

데이터셋 분석

와인 품질 데이터셋(출처: UCI)

5. Sulfur Compounds

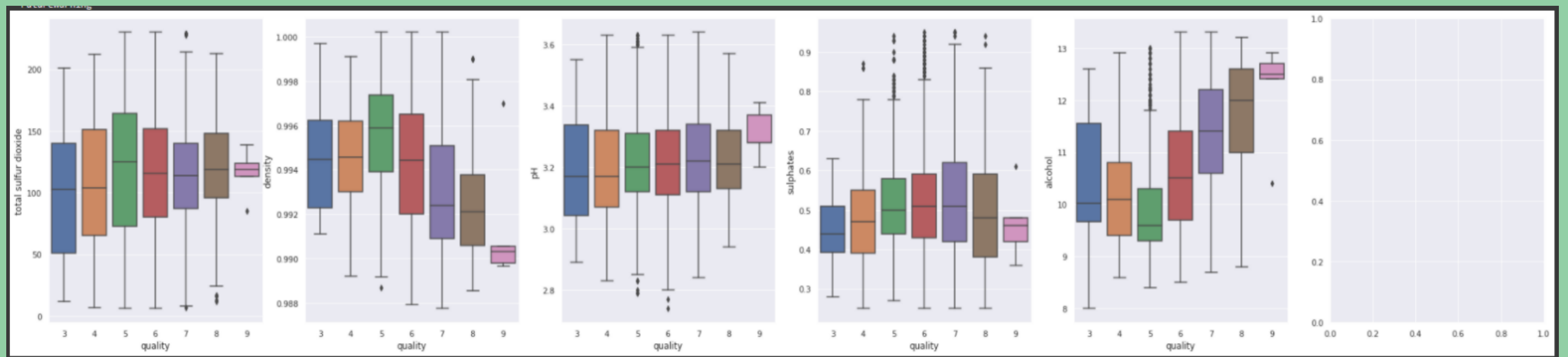
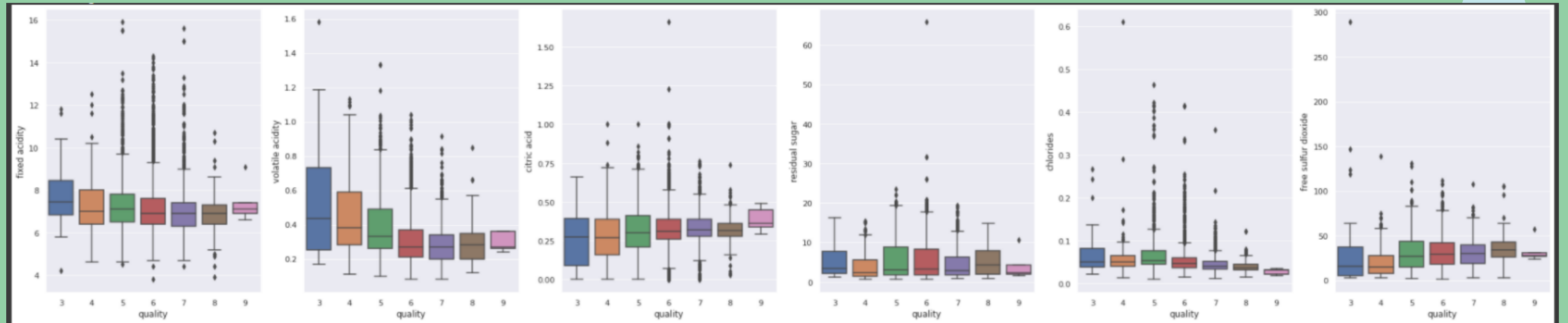
3) density (밀도)

-바디의 높고 낮음, 와인의 무게감

(밀도의 높고 낮음에 따라 quality 예측 안됨.)

데이터 전처리

이상치 제거



데이터 전처리

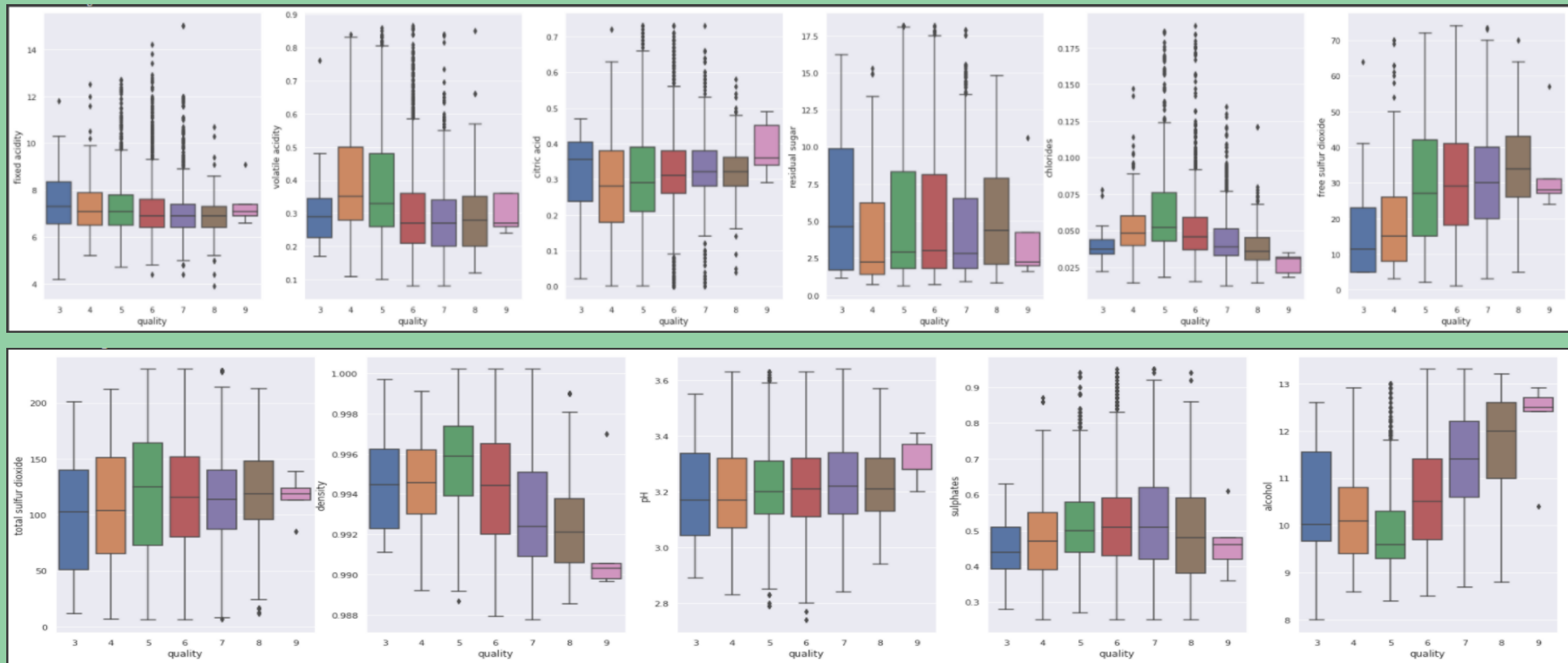
이상치 제거

```
df = df[df['volatile acidity'] < df['volatile acidity'].quantile(0.99)]
df = df[df['citric acid'] < df['citric acid'].quantile(0.99)]
df = df[df['residual sugar'] < df['residual sugar'].quantile(0.99)]
df = df[df['chlorides'] < df['chlorides'].quantile(0.99)]
df = df[df['free sulfur dioxide'] < df['free sulfur dioxide'].quantile(0.99)]
df = df[df['total sulfur dioxide'] < df['total sulfur dioxide'].quantile(0.99)]
df = df[df['density'] < df['density'].quantile(0.99)]
df = df[df['pH'] < df['pH'].quantile(0.99)]
df = df[df['sulphates'] < df['sulphates'].quantile(0.99)]
df = df[df['alcohol'] < df['alcohol'].quantile(0.99)]
```

상위 1% 이상치 제거

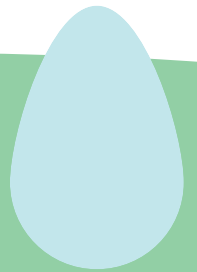
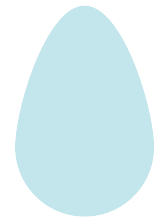
데이터 전처리

이상치 제거

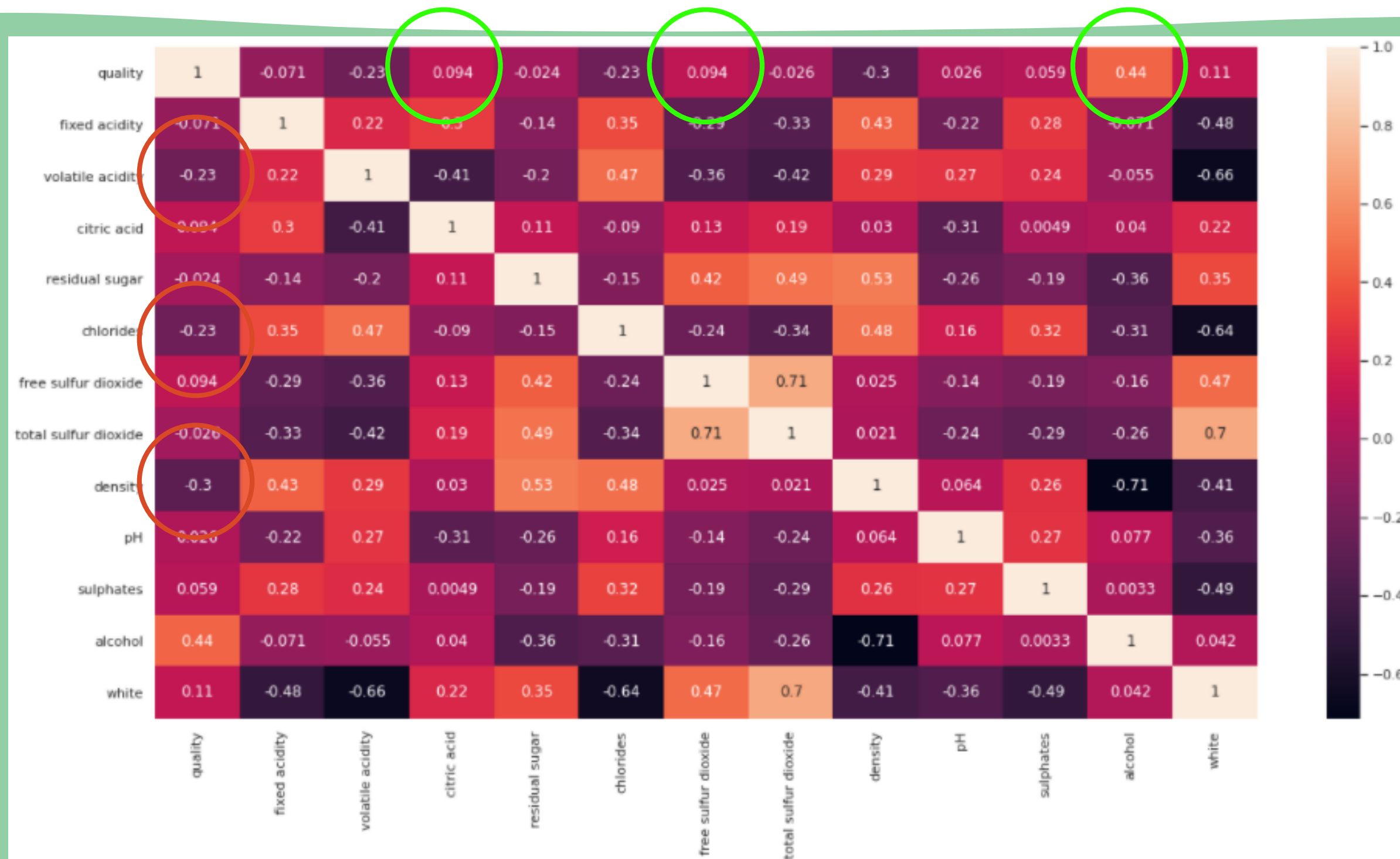


데이터 전처리

Heat map



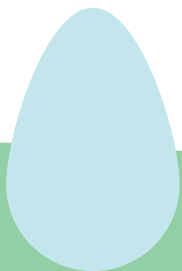
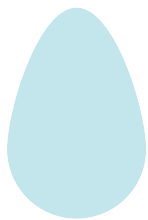
부정적인 요소
1순위 : density
2순위 : chloride,
volatile acidity



긍정적인 요소
1순위 : alcohol
2순위 : citric acid,
free sulfur dioxide

데이터 전처리

Heat map



	내 예측	컴퓨터 예측	비고
Alcohol	-	Positive	
Volatile Acidity	Negative	Negative	
Fixed Acidity	-	Negative	
Citric Acid	Positive	Positive	일치
Residual Sugar	Negative	Negative	일치
Total Sulfur Dioxide	Positive	Negative	
Free Sulfur Dioxide	Positive	Positive	일치
Sulphates	Positive	Positive	일치
pH	-	Positive	
Chloride	-	Negative	
Density	-	Negative	

데이터 전처리

train_test_split

Train-Test split

```
[25] X = df.loc[:,df.columns!='quality']  
      y = df['quality']  
  
      X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.80, random_state=2)
```

```
[26] print(X.shape)  
      print(y.shape)  
      #print(X_val.shape)  
      #print(y_val.shape)  
      print(X_train.shape)  
      print(y_train.shape)
```

```
(4942, 12)  
(4942,)  
(3953, 12)  
(3953,)
```

데이터 전처리

Standard Scaler (정규화)
: 단위x, 상대적인 영향력을 표준화 시키기 위함.

```
numeric_var = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol']  
scaler = StandardScaler()  
X_train[numeric_var] = scaler.fit_transform(X_train[numeric_var])  
X_test[numeric_var] = scaler.fit_transform(X_test[numeric_var])
```

X_train[numeric_var]

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
2487	-0.903925	0.144254	-0.564907	-0.672448	-0.809773	-0.300468	0.347813	-1.386135	0.824858	0.576421	1.283391
3328	0.443664	2.425867	1.356912	-0.785332	2.682122	-0.614614	-0.046463	0.828630	-0.038366	1.537920	-1.057067
3655	-0.903925	-0.325490	-0.334289	-0.130608	-0.982213	1.647232	0.479239	-1.450435	-0.370376	-0.785702	1.716809
2347	-0.398579	-0.593915	-0.411161	1.652949	0.009313	0.893283	1.267791	1.264438	1.356073	-0.785702	-0.710333
345	-0.903925	-0.929446	0.434439	-0.604718	-0.508004	-0.363297	-0.290538	-1.053921	-0.237572	-0.144703	0.329871
...
3710	-0.145906	-0.862340	-0.103670	0.185465	-0.680444	0.076506	0.103738	-0.207309	0.160839	-0.064578	0.243187
1243	-0.145906	2.157442	-2.256108	-0.717602	0.569741	-1.368562	-1.679892	0.971518	1.688083	1.217420	-0.883700
2803	-0.061682	-0.862340	-0.103670	-0.988522	-1.456421	-1.619878	-1.623566	-1.846949	-0.104768	-1.266452	1.283391
4027	-0.482804	-0.325490	0.818803	0.659575	-0.378675	1.898548	1.530641	0.407110	-0.503179	-0.304953	-0.970384
2872	-0.988150	-1.063659	-0.180543	0.998225	-0.378675	0.202164	-0.309313	0.314233	-0.635983	-1.266452	-1.143751

3953 rows x 11 columns

데이터 전처리

Standard Scaler (정규화)

```
numeric_var = ['fixed acidity','volatile acidity','citric acid','residual sugar','chlorides','free sulfur dioxide','total sulfur dioxide','density','pH','sulphates','alcohol']
scaler = StandardScaler()
X_train[numeric_var] = scaler.fit_transform(X_train[numeric_var])
X_test[numeric_var] = scaler.fit_transform(X_test[numeric_var])
```

x_train[numeric_var]											
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
2487	-0.903925	0.144254	-0.564907	-0.672448	-0.809773	-0.300468	0.347813	-1.386135	0.824858	0.576421	1.283391
3328	0.443664	2.425867	1.356912	-0.785332	2.682122	-0.614614	-0.046463	0.828630	-0.038366	1.537920	-1.057067
3655	-0.903925	-0.325490	-0.334289	-0.130608	-0.982213	1.647232	0.479239	-1.450435	-0.370376	-0.785702	1.716809
2347	-0.398579	-0.593915	-0.411161	1.652949	0.009313	0.893283	1.267791	1.264438	1.356073	-0.785702	-0.710333
345	-0.903925	-0.929446	0.434439	-0.604718	-0.508004	-0.363297	-0.290538	-1.053921	-0.237572	-0.144703	0.329871
...
3710	-0.145906	-0.862340	-0.103670	0.185465	-0.680444	0.076506	0.103738	-0.207309	0.160839	-0.064578	0.243187
1243	-0.145906	2.157442	-2.256108	-0.717602	0.569741	-1.368562	-1.679892	0.971518	1.688083	1.217420	-0.883700
2803	-0.061682	-0.862340	-0.103670	-0.988522	-1.456421	-1.619878	-1.623566	-1.846949	-0.104768	-1.266452	1.283391
4027	-0.482804	-0.325490	0.818803	0.659575	-0.378675	1.898548	1.530641	0.407110	-0.503179	-0.304953	-0.970384
2872	-0.988150	-1.063659	-0.180543	0.998225	-0.378675	0.202164	-0.309313	0.314233	-0.635983	-1.266452	-1.143751
3953 rows × 11 columns											

모델링과 학습

Scikit - Learn

Model Fitting - 1) Linear Regressor

```
[32] log_reg = LogisticRegression().fit(X_train,y_train) # X_train 시험지 / y_train 정답지 자 이제 공부해
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

```
[33] # 정확도 확인
```

```
print('Train data Score : ' , log_reg.score(X_train,y_train))
```

```
print('Test data score : ' , log_reg.score(X_test,y_test))
```

```
Train data Score : 0.5497090817100936
```

```
Test data score : 0.5419615773508595
```

54%

모델링과 학습

Scikit - Learn

▶ 머신 러닝 모델 구축 - 2) Decision Tree

```
[34] tree = DecisionTreeClassifier(max_depth=5,  
                                   min_samples_leaf=20,  
                                   min_samples_split=40).fit(X_train, y_train)
```

```
[35] # 정확도 확인  
print('Train data Score : ', tree.score(X_train, y_train))  
print('Test data Score : ', tree.score(X_test, y_test))
```

Train data Score : 0.5755122691626613

Test data Score : 0.5490394337714863 55%

모델링과 학습

Scikit - Learn

머신러닝 모델 구축 - 3) GradientBoostingClassifier

```
[36] boost = GradientBoostingClassifier(max_depth=3,  
                                         learning_rate=0.05).fit(X_train,y_train)
```

```
[37] # 정확도 확인  
print('Train data Score : ', boost.score(X_train, y_train))  
print('Test data Score : ', boost.score(X_test, y_test))
```

#근데 이렇게 나와도 정확도가 평가 지표는 아니다! ROC로 평가한다.

Train data Score : 0.6632937009865925

Test data Score : 0.5813953488372093

58%

모델링과 학습

Scikit - Learn

```
# Logistic Regression ROC
plot_auc_roc(log_reg)

[0. 0. 0. ... 0. 0. 0.]

-----
ValueError                                Traceback (most recent call last)
<ipython-input-106-780ee873ad4c> in <module>()
      1 # Logistic Regression ROC
----> 2 plot_auc_roc(log_reg)

----- 2 frames -----
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_ranking.py in _binary_clf_curve(y_true, y_score, pos_label, sample_weight)
    534     if not (y_type == "binary" or
    535           (y_type == "multiclass" and pos_label is not None)):
--> 536         raise ValueError("{0} format is not supported".format(y_type))
    537
    538     check_consistent_length(y_true, y_score, sample_weight)

ValueError: multilabel-indicator format is not supported
```

다중 분류에서는 ROC Curve가 그려지지 않음.

모델링과 학습

Scikit - Learn

머신 러닝 모델 구축 - 4) Random Forest Classifier

```
[38] from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
print(rfc.get_params())

{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini',
  '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': None, 'oob_score': False, 'random_state': None,
  'verbose': 0, 'warm_start': False}

[39] # Fit the model
rfc.fit(X_train,y_train)

RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', 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=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)

[40] y_pred=rfc.predict(X_test)
accuracy_score(y_test,y_pred)

0.6622851365015167
```

Classification Report:					
	precision	recall	f1-score	support	
4	0.70	0.23	0.34	31	
5	0.72	0.67	0.69	332	
6	0.60	0.79	0.69	420	
7	0.78	0.48	0.59	182	
8	0.86	0.27	0.41	22	
9	0.00	0.00	0.00	2	
accuracy			0.66	989	
macro avg			0.61	0.41	0.45 45%
weighted avg			0.68	0.66	0.65

모델링과 학습

keras

▼ 모델 구성하기

```
[93] model = Sequential()  
  
model.add(Dense(64, activation='relu', input_shape=(12,)))  
model.add(Dense(32, activation='relu'))  
model.add(Dense(7, activation='softmax'))
```

다중분류

y값 7개

▼ 모델 설정하기

```
[94] model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

다중분류

모델링과 학습

keras

모델 학습하기

```
history = model.fit(X_train,
                    y_train,
                    epochs = 1000,
                    batch_size=128,
                    validation_data=(X_val, y_val))
```

Epoch 1/1000
36/36 [=====] - 1s 11ms/step - loss: 8.1802 - accuracy: 0.2560 - val_loss: 1.7436 - val_accuracy: 0.3801
Epoch 2/1000
36/36 [=====] - 0s 2ms/step - loss: 1.7701 - accuracy: 0.3582 - val_loss: 1.3276 - val_accuracy: 0.3801
Epoch 3/1000
36/36 [=====] - 0s 3ms/step - loss: 1.3175 - accuracy: 0.3977 - val_loss: 1.2444 - val_accuracy: 0.4457
Epoch 4/1000
36/36 [=====] - 0s 3ms/step - loss: 1.2667 - accuracy: 0.4496 - val_loss: 1.2280 - val_accuracy: 0.4389
Epoch 5/1000
36/36 [=====] - 0s 3ms/step - loss: 1.2544 - accuracy: 0.4526 - val_loss: 1.2152 - val_accuracy: 0.4434
Epoch 6/1000
36/36 [=====] - 0s 3ms/step - loss: 1.2743 - accuracy: 0.4465 - val_loss: 1.2187 - val_accuracy: 0.4706
Epoch 7/1000
36/36 [=====] - 0s 2ms/step - loss: 1.2410 - accuracy: 0.4608 - val_loss: 1.2169 - val_accuracy: 0.4661
Epoch 8/1000
36/36 [=====] - 0s 2ms/step - loss: 1.2458 - accuracy: 0.4590 - val_loss: 1.2218 - val_accuracy: 0.4683
Epoch 9/1000
36/36 [=====] - 0s 2ms/step - loss: 1.2501 - accuracy: 0.4511 - val_loss: 1.2410 - val_accuracy: 0.4683
Epoch 10/1000
36/36 [=====] - 0s 2ms/step - loss: 1.2460 - accuracy: 0.4528 - val_loss: 1.2071 - val_accuracy: 0.4819
Epoch 11/1000
36/36 [=====] - 0s 3ms/step - loss: 1.2332 - accuracy: 0.4667 - val_loss: 1.2031 - val_accuracy: 0.4910
Epoch 12/1000
36/36 [=====] - 0s 3ms/step - loss: 1.2326 - accuracy: 0.4585 - val_loss: 1.1913 - val_accuracy: 0.4955
Epoch 13/1000
36/36 [=====] - 0s 3ms/step - loss: 1.2037 - accuracy: 0.4754 - val_loss: 1.2039 - val_accuracy: 0.4706
Epoch 14/1000
36/36 [=====] - 0s 3ms/step - loss: 1.2304 - accuracy: 0.4686 - val_loss: 1.1765 - val_accuracy: 0.4955
Epoch 15/1000
36/36 [=====] - 0s 3ms/step - loss: 1.2120 - accuracy: 0.4746 - val_loss: 1.1607 - val_accuracy: 0.5007

모델 그려보기

```
import matplotlib.pyplot as plt

his_dict = history.history
loss = his_dict['loss']
val_loss = his_dict['val_loss'] # 검증 데이터가 있는 경우 'val_' 수식어가 붙습니다.

epochs = range(1, len(loss) + 1)
fig = plt.figure(figsize = (10, 5))

# 훈련 및 검증 손실 그리기
ax1 = fig.add_subplot(1, 2, 1)
ax1.plot(epochs, loss, color = 'blue', label = 'train_loss')
ax1.plot(epochs, val_loss, color = 'orange', label = 'val_loss')
ax1.set_title('train and val loss')
ax1.set_xlabel('epochs')
ax1.set_ylabel('loss')
ax1.legend()

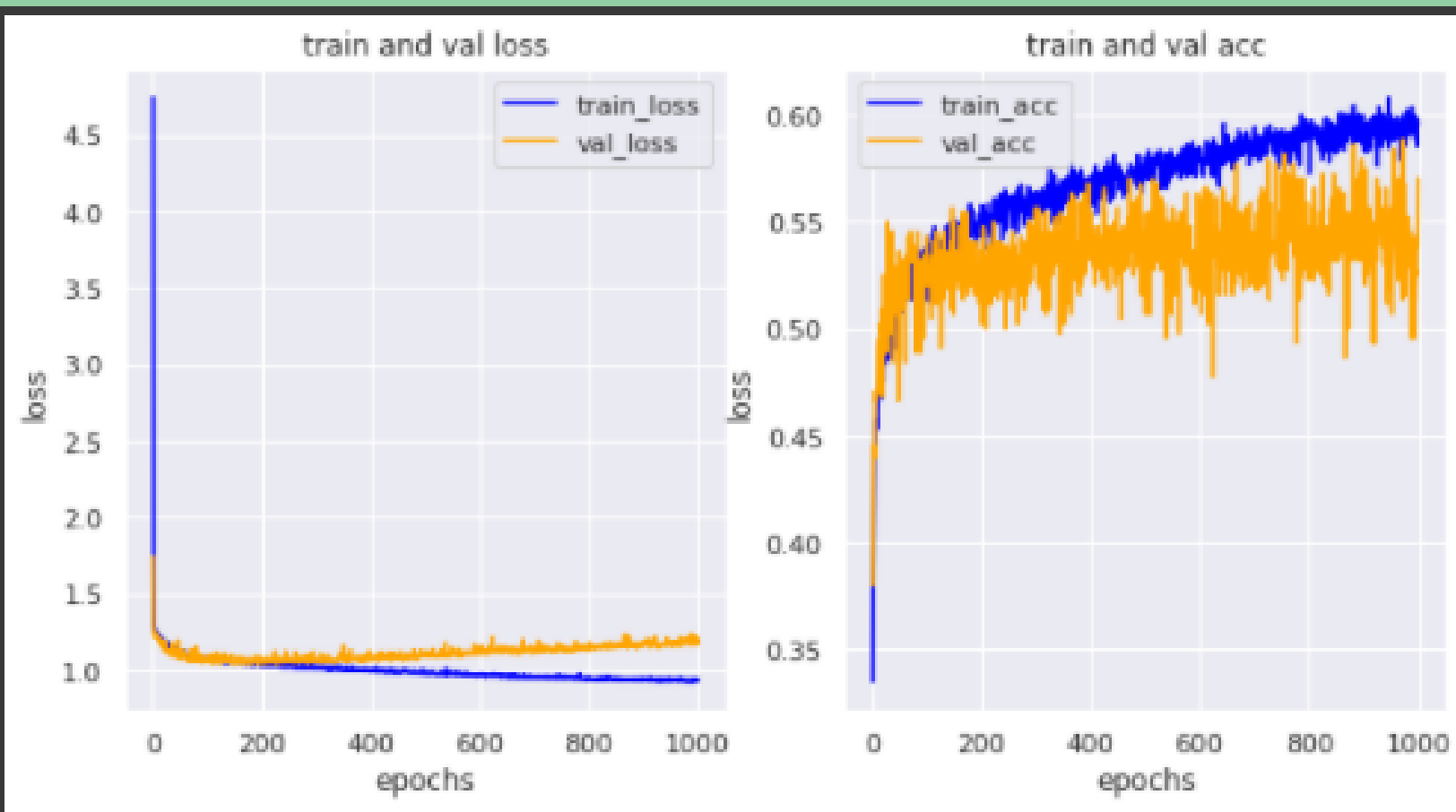
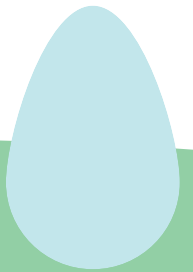
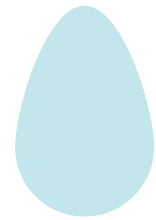
acc = his_dict['accuracy']
val_acc = his_dict['val_accuracy']

# 훈련 및 검증 정확도 그리기
ax2 = fig.add_subplot(1, 2, 2)
ax2.plot(epochs, acc, color = 'blue', label = 'train_acc')
ax2.plot(epochs, val_acc, color = 'orange', label = 'val_acc')
ax2.set_title('train and val acc')
ax2.set_xlabel('epochs')
ax2.set_ylabel('loss')
ax2.legend()

plt.show()
```

모델 성능 평가

keras



모델 평가하기



```
scores = model.evaluate(X_test, y_test)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

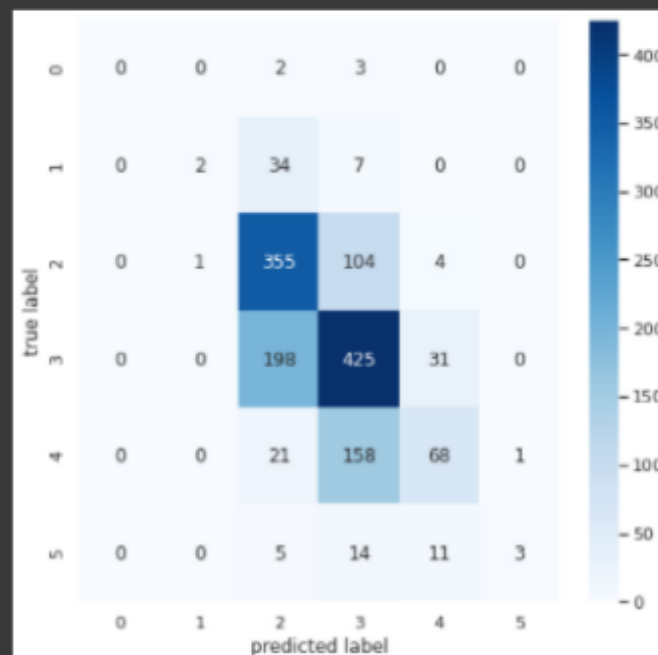
```
46/46 [=====] - 0s 1ms/step - loss: 1.0029 - accuracy: 0.5895
accuracy: 58.95%
```

모델 성능 평가

keras

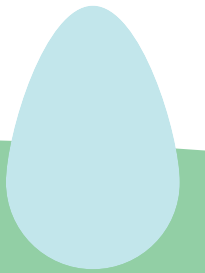
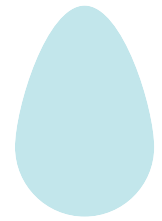
모델 평가 방법 - 혼동행렬

```
# sklearn.metrics 모듈은 여러가지 평가 지표에 관한 기능을 제공합니다.  
from sklearn.metrics import classification_report, confusion_matrix  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
# 혼동 행렬을 만듭니다.  
plt.figure(figsize = (7, 7))  
cm = confusion_matrix(np.argmax(y_test, axis = -1), np.argmax(results, axis = -1))  
sns.heatmap(cm, annot = True, fmt = 'd', cmap = 'Blues')  
plt.xlabel('predicted label')  
plt.ylabel('true label')  
plt.show()  
  
# 색이 진할수록 정확도가 높아진다.  
# 예측에 많이 틀리는 것을 찾을 수 있다.
```



모델 성능 평가

keras



scikit-learn

- 1) Linear Regressior : 54%
- 2) Decision Tree : 55%
- 3) Gradient Boosting Classifier : 58%

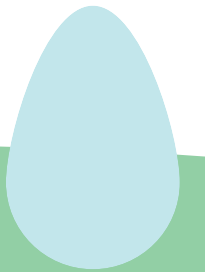
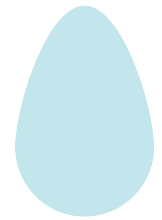
4) Random Forest : Accuracy 66% & F1 Score 45%

keras

- 1) Sequential : 59%

결론

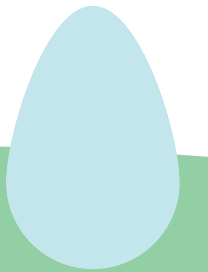
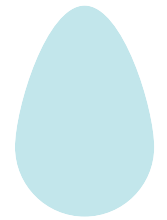
keras



- 물리화학적 데이터를 통해 현재까지 배운 머신러닝 모델과 스킴로는 정확도를 따라갈 수가 없었다.
- 위 모델로는 3개중에 2개를 맞추는 수준. (정확도 66%)
- 가장 중요한 변수는 취향. 입맛은 사람마다 다름. (개취)

느낀점 / 한계점

keras



- 자료를 조사하다보니, PDP 라는 라이브러리에 pdp_plot 이라는 그래프를 그리면, y값에 대한 각 특성들의 기여도가 나온다.
예를들어 quality 7 이라는 y값을 가진 와인이 있다고하면, 7이라는 퀄리티를 만드는데 누가 가장 많이 기여하고 누가 가장 저해하는지를 알 수 있는 알 수 있다. 미리 학습했더라면, 더 와인 품질 예측하는데에 도움이 되었을 것이다.
- 단위의 부재
- 그리고 다중분류는 이진분류에 비해 성능평가하기가 어렵다는 것을 알았다.
왜 다들 이진분류를 하는지를 몸소 깨달아버렸다...

지난 발표 준비과정에서, 공공데이터 포털 문의

6월26일 문의 -> 7월2일 답변 완료

오류상세내용

데이터유형	파일데이터
데이터명	제주특별자치도_주제10_중국인 관광객 타겟 마케팅을 위한 활용 데이터_매쉬업결과
기관명	제주특별자치도
오류내용	<p>안녕하세요. 담당자님.</p> <p>제주도에 관심이 많아서 제주도 중국인 소비현황 데이터분석을 위해 업로드해주신 데이터를 다운받아 사용하던 중, 의아한 부분이 생겨서 문의 드립니다.</p> <p>문의 드릴 사항은 다음과 같습니다.</p> <p>1. 이용금액에 단위가 없는데 혹시 '원' 단위가 맞습니까?</p> <p>2. 1732행 서귀포시 남원읍 데이터를 보면, 2018년 10월 한달 간 7440명이 숙박업에 4000억을 소비했다는데, 객단가로 보면 1인당 약 5400만원 수준이고 30일을 숙박했다고 하더라도 1인당 하루 180만원 수준입니다.</p> <p>위 데이터에 대해서 확인 할 수 있는 내용이 있으시면 답변 주시면 감사하겠습니다.</p>
등록일자	2021-06-26
처리상태	처리완료
오류등록 유형	데이터정확성오류(품질)
처리일자	2021-06-26
답변내용	<p>안녕하세요. 문의 주신 파일데이터에 오류가 있어서 수정 데이터를 메일로 우선 전송해 드렸고 공공데이터 포털에는 다음주 월요일에는 반영이 될 것 같습니다. 이용에 불편을 드려 죄송합니다. 감사합니다.</p>

지난 발표 준비과정에서, 공공데이터 포털 문의

6월26일 문의 -> 7월2일 답변 완료

[제주도청] 공공데이터 오류신고 건 받은편지함 x




"추현진" <chu0530@korea.kr>

나에게 ▾

안녕하세요. 제주도청 공공데이터 담당자 추현진입니다.
문의주신 파일데이터에 오류가 있어서 수정 데이터를 전송해 드립니다.
공공데이터 포털에는 다음주 월요일에는 반영이 될 것 같습니다.
이용에 불편을 드려 죄송합니다.
감사합니다.

X 제주특별자치도_주제10_중국인 관광객 타겟 마케팅을 ... 한 활용 데이터_매쉬업결과_20210628.xlsx 연결

A	B	C	D	E	F	G	H	I
년월	이용자구분	시도명	읍면동명	업종	방문인구	이용자수	이용금액	데이터기준일자
2018-01	중국인관광객	서귀포시	대륜동	소매	22743.846	1	600,000	2021-06-28
2018-01	중국인관광객	서귀포시	대륜동	쇼핑	22743.846	110	11,558,810	2021-06-28
2018-01	중국인관광객	서귀포시	대륜동	숙박	22743.846	4	418,600	2021-06-28
2018-01	중국인관광객	서귀포시	대륜동	식음료	22743.846	3	187,840	2021-06-28
2018-01	중국인관광객	서귀포시	대천동	쇼핑	7274.675	4	198,430	2021-06-28
2018-01	중국인관광객	서귀포시	대천동	숙박	7274.675	1	30,000	2021-06-28
2018-01	중국인관광객	서귀포시	동홍동	교통	17931.536	4	341,000	2021-06-28



감사합니다

질문이 있다면 말씀해주세요.