WINE QUALITY CLASSIFICATION

와인 분류모델 예측

9조 김예지 류민경 박현영 양세정

ALC. 18% BY VOL. 750ML

CONTENTS

데이터셋 선정 배경

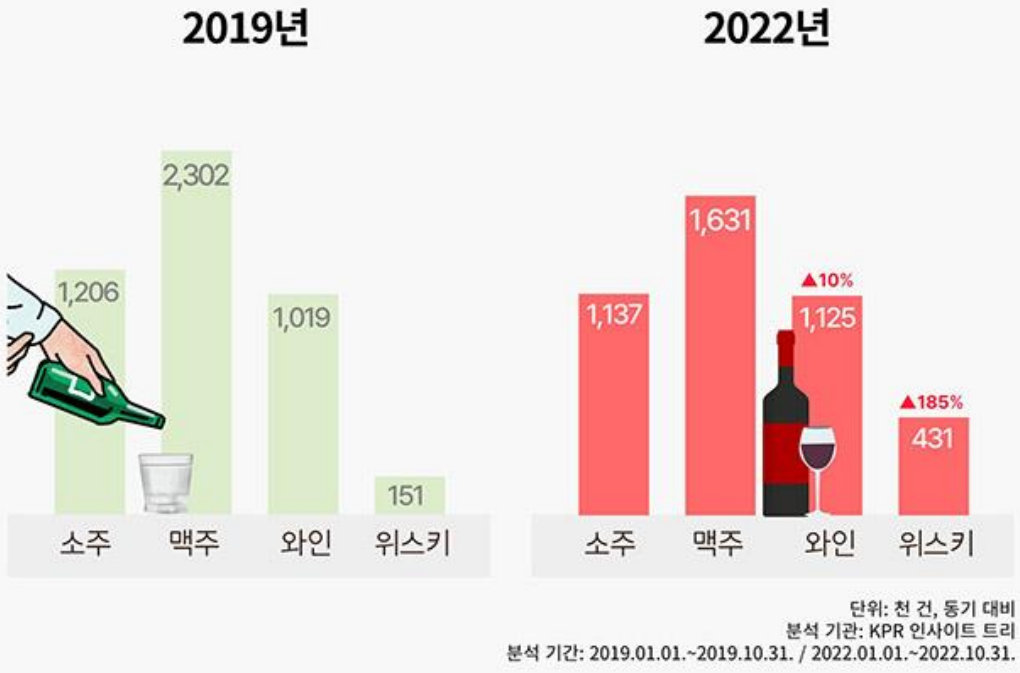
"소주 대신 와인"...MZ세대의 달라진 음주 문화

김경희 기자 lululala@chosun.com

기사입력 2022.11.29 16:31

최근 MZ세대는 소주보다 와인과 위스키를 선호하는 것으로 조사됐다. 이러한 트렌드는 대형 유통사를 중심으로 고급 주류 판매처가 확대되고 가격대가 낮아지면서 접근성이 높아졌기 때문이라는 분석이다.
또한, 색다른 경험을 중시하는 MZ세대의 취향을 반영한 주종 레시피가 공유되고 있는 것도 영향을 미치고 있다.

https://digitalchosun.dizzo.com/site/data/html_dir/2022/11/29/2022112980061.html



최근 와인에 대한 MZ세대의 관심

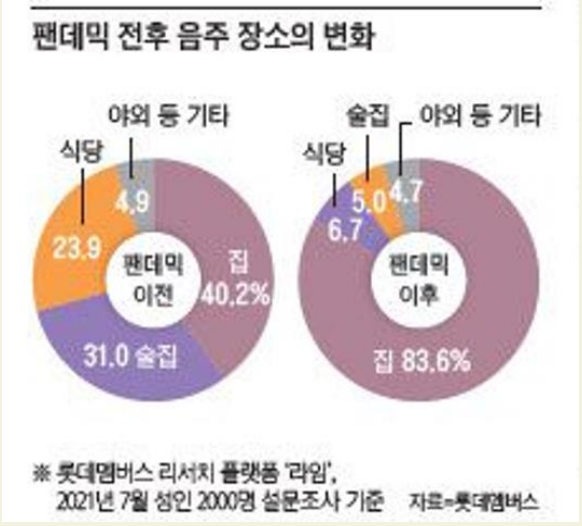
데이터셋 선정 배경

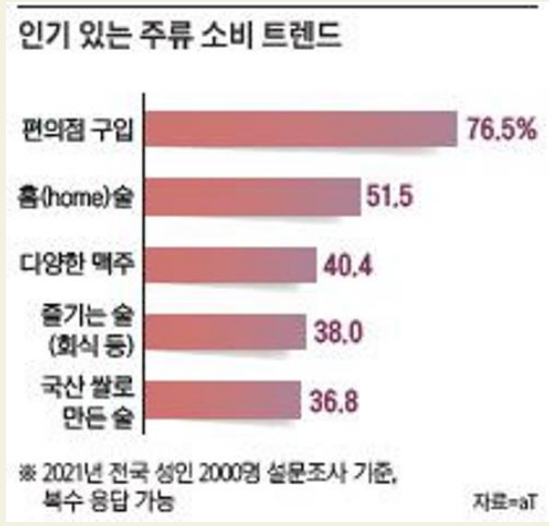
와인에 꽂힌 MZ세대... '취향 저격' 나서려면

김한나 기자 hanna7@kukinews.com
기사승인 2022-05-29 07:00:02

◆ 소주·맥주보다 '와인' 찾는 MZ세대
와인의 인기는 변화하고 있는 술 문화를 보여준다. '혼술'처럼 최근 몇 년간 MZ세대의 술 문화는 개인의 취향에 맞는 술 자체를 즐기는 방향으로 변화해 왔다. 이러한 트렌드는 소주와 맥주 등에 비해 종류가 다양하고 상대적으로 고급화된 이미지를 가진 와인에 대한 선호로 이어졌다.

https://www.kukinews.com/newsView/kuk202205280007

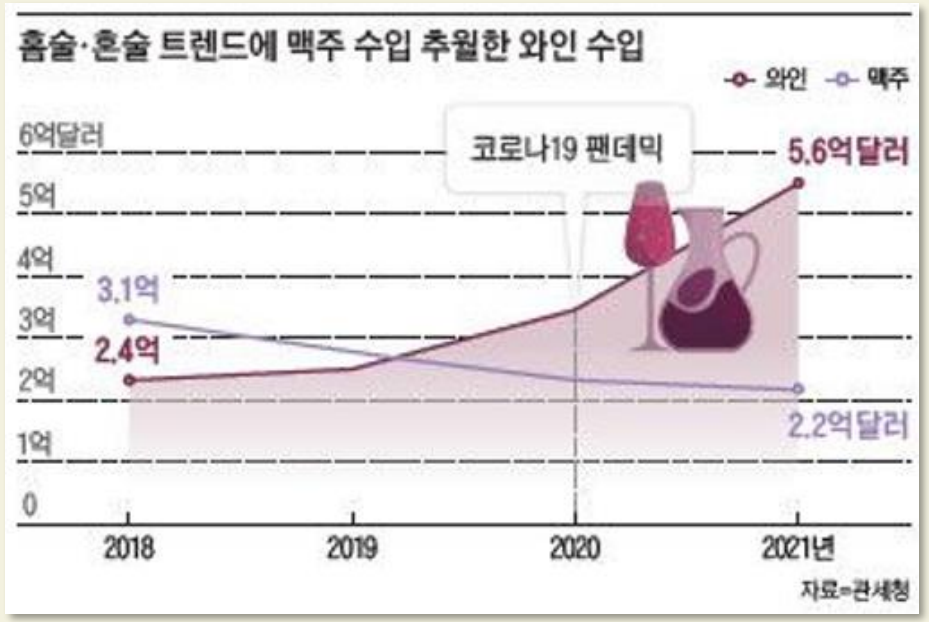




▶코로나로 인한 혼술/홈술 트렌드

▶ 와인의 고급스러운 이미지

데이터셋 선정 배경



와인에 대한 관심 및 수요 증가

→ **와인 품질 분류 & 활용 방안**

01

데이터 탐색, 전처리

Part 1. 데이터 탐색, 전처리

데이터 불러오기 & 탐색

```
[ ]  # 필요한 package 불러오기
     %matplotlib inline
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     import numpy as np
```

```
[ ]  # 데이터 불러오기
     wine = pd.read_csv('https://raw.githubusercontent.com/fbalsrud/ITB2022/main/wine_train.csv', index_col='index') #'index' 변수 index로 불러오기

     print(wine.columns)
     display(wine.info())  # IV 1개('quality'), DV 총 12개('index' 제외)
     display(wine)
```

index	quality	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	type
0	5	5.6	0.695	0.06	6.8	0.042	9.0	84.0	0.99432	3.44	0.44	10.2	white
1	5	8.8	0.610	0.14	2.4	0.067	10.0	42.0	0.99690	3.19	0.59	9.5	red
2	5	7.9	0.210	0.39	2.0	0.057	21.0	138.0	0.99176	3.05	0.52	10.9	white
3	6	7.0	0.210	0.31	6.0	0.046	29.0	108.0	0.99390	3.26	0.50	10.8	white
4	6	7.8	0.400	0.26	9.5	0.059	32.0	178.0	0.99550	3.04	0.43	10.9	white
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5492	5	7.7	0.150	0.29	1.3	0.029	10.0	64.0	0.99320	3.35	0.39	10.1	white
5493	6	6.3	0.180	0.36	1.2	0.034	26.0	111.0	0.99074	3.16	0.51	11.0	white
5494	7	7.8	0.150	0.34	1.1	0.035	31.0	93.0	0.99096	3.07	0.72	11.3	white
5495	5	6.6	0.410	0.31	1.6	0.042	18.0	101.0	0.99195	3.13	0.41	10.5	white
5496	6	7.0	0.350	0.17	1.1	0.049	7.0	119.0	0.99297	3.13	0.36	9.7	white

5497 rows × 13 columns

데이터 전처리

```
[ ]  # object 변수 수치형 변환
     from sklearn.preprocessing import LabelEncoder

     # LabelEncoder를 객체로 생성한 후 , fit( ) 과 transform( ) 으로 label 인코딩 수행.
     encoder = LabelEncoder()
     wine['type'] = encoder.fit_transform(wine['type']) # red = 0, white = 1

     display(wine.head(5))
```

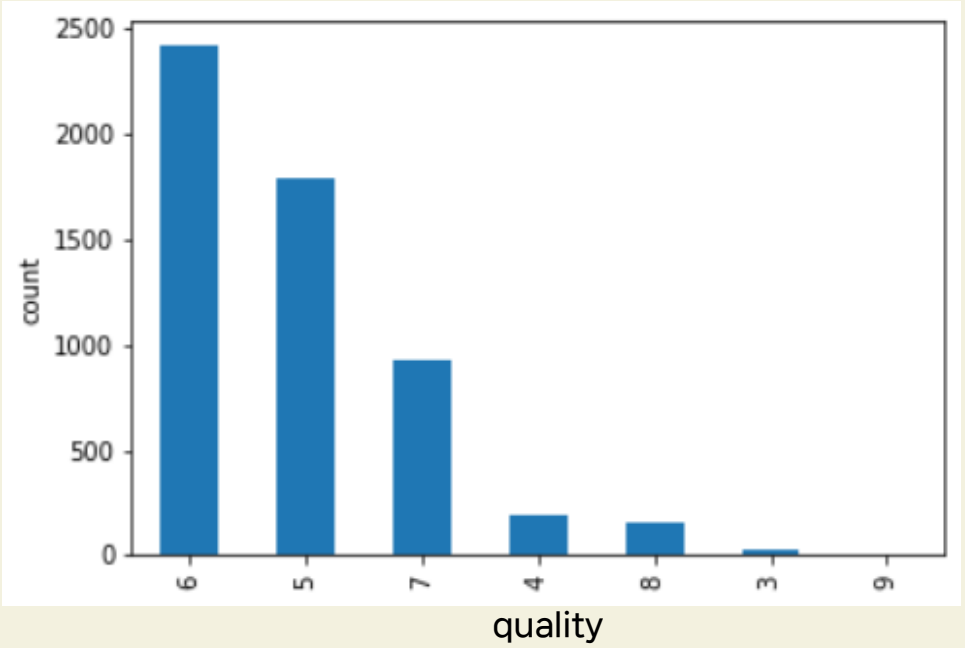
index	quality	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	type
0	5	5.6	0.695	0.06	6.8	0.042	9.0	84.0	0.99432	3.44	0.44	10.2	1
1	5	8.8	0.610	0.14	2.4	0.067	10.0	42.0	0.99690	3.19	0.59	9.5	0
2	5	7.9	0.210	0.39	2.0	0.057	21.0	138.0	0.99176	3.05	0.52	10.9	1
3	6	7.0	0.210	0.31	6.0	0.046	29.0	108.0	0.99390	3.26	0.50	10.8	1
4	6	7.8	0.400	0.26	9.5	0.059	32.0	178.0	0.99550	3.04	0.43	10.9	1

LabelEncorder를 이용해
type 변수의 red, white 값을 각각 0과 1로 변경

데이터 탐색

```
[ ]  # quality별 빈도수 확인
     print(wine['quality'].value_counts())

     wine['quality'].value_counts().plot.bar()
     plt.ylabel('count')
     plt.show()
```




```
6    2416
5    1788
7     924
4     186
8     152
3      26
9       5

Name: quality, dtype: int64
```

quality 변수의 분포 확인

02

분류모델 만들기

Part 2. 분류모델 만들기

의사결정나무 Decision tree

```
[ ]  from sklearn.tree import DecisionTreeClassifier
     from sklearn.model_selection import train_test_split

     X = wine.iloc[:, 1:]  #12개 변수
     y = wine.iloc[:, 0]   # target 변수 quality

     # 데이터셋을 학습(train)과 테스트(test) 세트로 분할
     # random_state : random 으로 분할시 사용되는 난수 seed 숫자
     # X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, stratify=y, random_state=42)

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=42)

     tree = DecisionTreeClassifier(random_state=0)
     tree.fit(X_train, y_train)
     print("학습용 데이터 정확도: {:.3f}".format(tree.score(X_train, y_train)))
     print("시험용 데이터 정확도: {:.3f}".format(tree.score(X_test, y_test)))
```

```
학습용 데이터 정확도: 1.000
시험용 데이터 정확도: 0.591
```

```
[ ]  tree = DecisionTreeClassifier(max_depth=20, random_state=0)
     tree.fit(X_train, y_train)

     print("학습용 데이터 정확도: {:.3f}".format(tree.score(X_train, y_train)))
     print("시험용 데이터 정확도: {:.3f}".format(tree.score(X_test, y_test))) # 임의로 파라미터 적용시 0.591이상 정확도 도출하기 쉽지 않음.
```

```
학습용 데이터 정확도: 0.998
시험용 데이터 정확도: 0.587
```

의사결정나무의 시험용 데이터 분류 정확도: 0.591

Part 2. 분류모델 만들기

의사결정나무 & 그리드 서치 Decision tree & Grid search

```
[ ]  # 최적파라미터 찾기 위해 그리드 서치 실행
     from sklearn.model_selection import GridSearchCV

     # depth 제어보다 최고 정확도 개선을 우선 목표로 삼음

     # 파라미터를 dictionary 형태로 설정
     parameters = {'max_depth':[20,21,22,23,24,25], 'random_state': [0, 42]}

     grid_tree = GridSearchCV(tree, param_grid=parameters, cv=3, refit=True)
     grid_tree.fit(X_train, y_train)

     scores_df = pd.DataFrame(grid_tree.cv_results_)
     scores_df[['params', 'mean_test_score', 'rank_test_score', 'split0_test_score', 'split1_test_score', 'split2_test_score']]
```

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'max_depth': 20, 'random_state': 0}	0.530568	7	0.519651	0.533479	0.538574
1	{'max_depth': 20, 'random_state': 42}	0.531053	4	0.522562	0.537846	0.532751
2	{'max_depth': 21, 'random_state': 0}	0.530810	6	0.516012	0.532023	0.544396
3	{'max_depth': 21, 'random_state': 42}	0.531053	4	0.524017	0.539301	0.529840
4	{'max_depth': 22, 'random_state': 0}	0.531295	1	0.518195	0.529840	0.545852
5	{'max_depth': 22, 'random_state': 42}	0.530082	8	0.525473	0.541485	0.523290
6	{'max_depth': 23, 'random_state': 0}	0.531053	3	0.512373	0.534934	0.545852
7	{'max_depth': 23, 'random_state': 42}	0.530082	8	0.522562	0.544396	0.523290
8	{'max_depth': 24, 'random_state': 0}	0.531295	1	0.512373	0.535662	0.545852
9	{'max_depth': 24, 'random_state': 42}	0.528869	11	0.522562	0.540757	0.523290
10	{'max_depth': 25, 'random_state': 0}	0.529112	10	0.512373	0.529112	0.545852
11	{'max_depth': 25, 'random_state': 42}	0.527414	12	0.522562	0.536390	0.523290

```
[ ]  print('DecisionTreeCV 최적 파라미터:', grid_tree.best_params_)
     print('DecisionTreeCV 최고 정확도: {0:.4f}'.format(grid_tree.best_score_))
```

```
DecisionTreeCV 최적 파라미터: {'max_depth': 22, 'random_state': 0}
DecisionTreeCV 최고 정확도: 0.5313
```

```
[ ]  # 최적 파라미터로 학습이 된 모델을 이용하여 시험용 데이터 세트 분류
     print("시험용 데이터 세트 정확도: {:.3f}".format(grid_tree.score(X_test, y_test))) # 처음 만든 의사결정나무와(과적합) 동일 수치.

     # depth 조절 시 늘 해당 수치 이하의 정확도 도출됨. decision tree의 정확도 개선은 무의미.
```

```
시험용 데이터 세트 정확도: 0.591
```

최적 파라미터(깊이 22)를 적용한 의사결정나무 정확도 또한 0.591

랜덤 포레스트 Random Forest

```
[ ]  from sklearn.ensemble import RandomForestClassifier

     forest = RandomForestClassifier(n_estimators=100, random_state=0)
     forest.fit(X_train, y_train)

     print("학습용 데이터 세트 정확도: {:.3f}".format(forest.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(forest.score(X_test, y_test)))  # decision tree 보다 개선된 정확도
```

```
학습용 데이터 세트 정확도: 1.000
시험용 데이터 세트 정확도: 0.668
```

랜덤 포레스트로 시험용 데이터 분류 시 정확도: 0.668

그래디언트 부스팅 Gradient Boosting

```
[ ]  from sklearn.ensemble import GradientBoostingClassifier
     # 기본값은 max_depth= 3, n_estimators=100, learning_rate = 0.1

     gbrt = GradientBoostingClassifier(random_state=0)
     gbrt.fit(X_train, y_train)
```

```
[ ]  print("학습용 데이터 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
학습용 데이터 세트 정확도: 0.729
시험용 데이터 세트 정확도: 0.568
```

그래디언트 부스팅으로 시험용 데이터 분류 시 정확도: 0.568

그래디언트 부스팅 Gradient Boosting

```
[ ]  # 나무 깊이 2로 제한
     gbrt = GradientBoostingClassifier(random_state=0, max_depth=2)
     gbrt.fit(X_train, y_train)

     print("학습 데이터 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))  # 정확도 오히려 낮아짐
```

```
학습 데이터 세트 정확도: 0.624
시험용 데이터 세트 정확도: 0.545
```

```
[ ]  # 나무 깊이 4로 증가
     gbrt = GradientBoostingClassifier(random_state=0, max_depth=4)
     gbrt.fit(X_train, y_train)

     print("학습 데이터 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test))) # 정확도 증가함. random forest 보다는 낮은 수치.
```

```
학습 데이터 세트 정확도: 0.823
시험용 데이터 세트 정확도: 0.597
```

나무 깊이를 각각 2, 4로 제한해보니 제한 전(0.568)보다는 높아졌으나
랜덤 포레스트(0.668) 보다는 정확도가 낮음

Part 2. 분류모델 만들기

그래디언트 부스팅 & 그리드 서치

```
[ ]  # 최적 파라미터 찾기 위한 그리드 서치
     from sklearn.model_selection import GridSearchCV

     # 파라미터를 dictionary 형태로 설정
     parameters = {'max_depth':[1,2,3,4], 'learning_rate':[0.1,0.08, 0.06, 0.04, 0.02]}

     # param_grid의 파라미터들을 3개의 fold 로 나누어 검증
     # refit=True (기본값): 가장 좋은 파라미터 설정으로 재학습
     init_gbrt = GradientBoostingClassifier(random_state=0)
     grid_gbrt = GridSearchCV(init_gbrt, param_grid=parameters, cv=3, refit=True)
     print(grid_gbrt)
     # 파라미터들을 순차적으로 학습, 검증
     grid_gbrt.fit(X_train, y_train)

     # GridSearchCV 결과 추출하여 DataFrame으로 변환
     scores_df = pd.DataFrame(grid_gbrt.cv_results_)
     scores_df[['params', 'mean_test_score', 'rank_test_score', 'split0_test_score', 'split1_test_score', 'split2_test_score']]
```

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'learning_rate': 0.1, 'max_depth': 1}	0.547792	15	0.549491	0.538574	0.555313
1	{'learning_rate': 0.1, 'max_depth': 2}	0.568171	10	0.564047	0.562591	0.577875
2	{'learning_rate': 0.1, 'max_depth': 3}	0.574236	6	0.558224	0.569141	0.595342
3	{'learning_rate': 0.1, 'max_depth': 4}	0.587821	1	0.580786	0.589520	0.593159

```
[ ]  print('GridSearchCV 최적 파라미터:', grid_gbrt.best_params_)
     print('GridSearchCV 최고 정확도: {0:.4f}'.format(grid_gbrt.best_score_))
```

```
GridSearchCV 최적 파라미터: {'learning_rate': 0.1, 'max_depth': 4}
GridSearchCV 최고 정확도: 0.5878
```

그리드 서치로 찾은 최적 파라미터는 깊이 4

```
[ ]  # 최적 파라미터로 학습이 된 모델을 이용하여 시험용 데이터 세트 분류
     print("시험용 데이터 세트 정확도: {:.3f}".format(grid_gbrt.score(X_test, y_test))) # 그리드 서치하기 전과 동일한 정확도.
```

```
시험용 데이터 세트 정확도: 0.597
```

이를 적용한 모델 정확도: 0.597

모델 별 정확도 비교

```
[ ]  # 최적 파라미터로 학습이 된 모델을 이용하여 시험용 데이터 세트 분류
     print("시험용 데이터 세트 정확도: {:.3f}".format(grid_tree.score(X_test, y_test))) # 처음 만든 의사결정나무와(과적합) 동일 수치.

     # depth 조절 시 늘 해당 수치 이하의 정확도 도출됨. decision tree의 정확도 개선은 무의미.
```

```
시험용 데이터 세트 정확도: 0.591
```

의사결정나무: 0.591

```
[ ]  from sklearn.ensemble import RandomForestClassifier

     forest = RandomForestClassifier(n_estimators=100, random_state=0)
     forest.fit(X_train, y_train)

     print("학습용 데이터 세트 정확도: {:.3f}".format(forest.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(forest.score(X_test, y_test)))  # decision tree 보다 개선된 정확도
```

```
학습용 데이터 세트 정확도: 1.000
시험용 데이터 세트 정확도: 0.668
```

랜덤 포레스트: 0.668

```
[ ]  # 최적 파라미터로 학습이 된 모델을 이용하여 시험용 데이터 세트 분류
     print("시험용 데이터 세트 정확도: {:.3f}".format(grid_gbrt.score(X_test, y_test))) # 그리드 서치하기 전과 동일한 정확도.
```

```
시험용 데이터 세트 정확도: 0.597
```

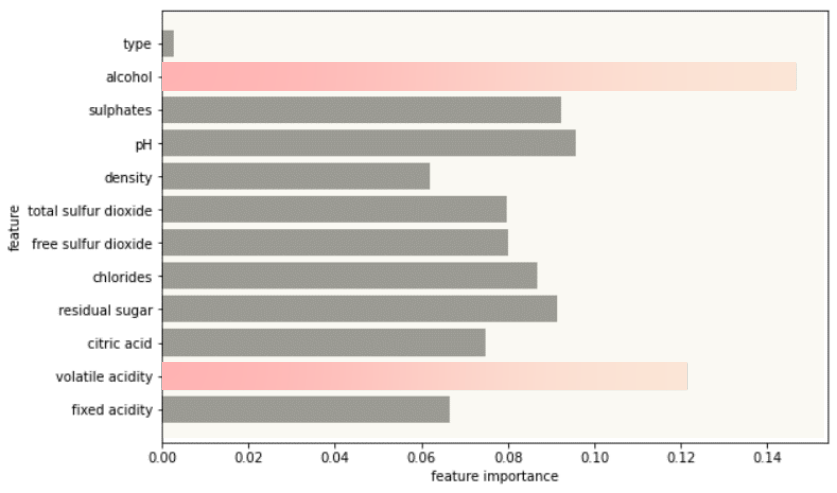
그래디언트 부스팅: 0.597

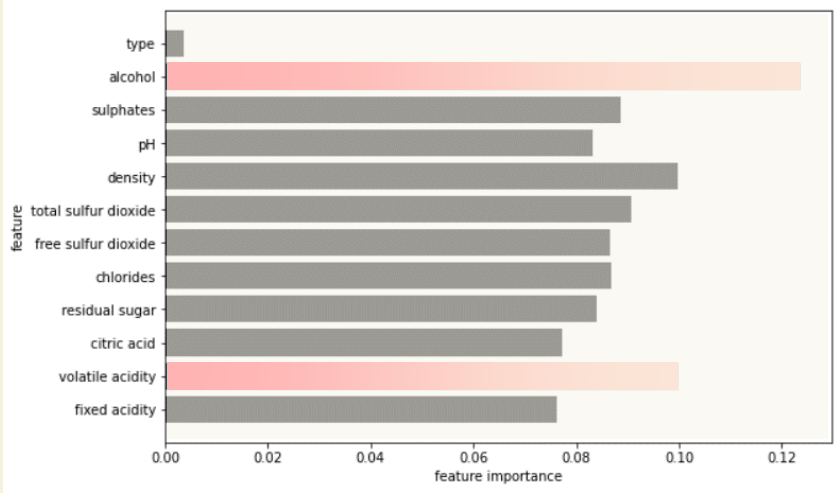
03

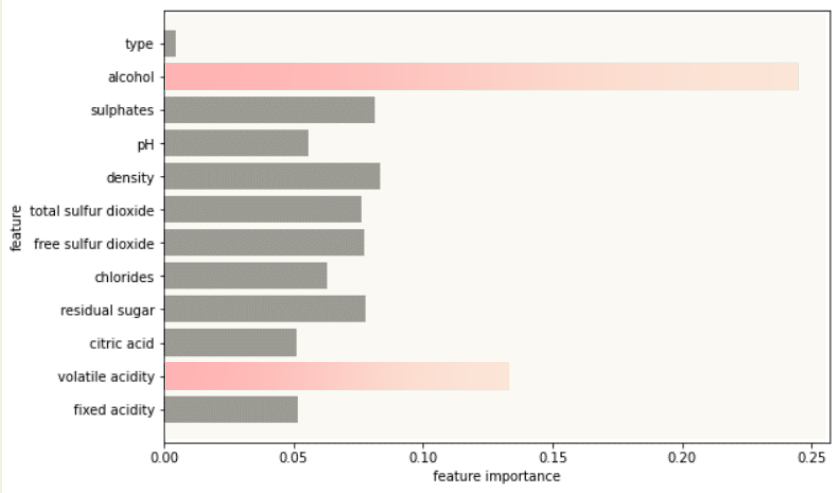
분류모델 평가

대안1 : 독립변수 제한

각 모델에서 공통적으로 나타난 중요변수 2개만을 기준으로 재분류







각 모델 별 변수의 중요도 그래프
(의사결정나무, 랜덤포레스트, 그래디언트부스팅)

중요도 상위 2개 변수:
Alcohol과 Volatile acidity

대안1 : 독립변수 제한

```
[ ]  # 기존 분석결과 정확도가 가장 높았던 random forest 대상으로 재분류 실시
     from sklearn.ensemble import RandomForestClassifier

     # 중요도 높았던 alcohol, volatile acidity 변수만 이용하여 5개의 나무를 만드는 예
     forest = RandomForestClassifier(n_estimators=5, random_state=2)
     forest.fit(X_train[['alcohol','volatile acidity']], y_train)

     # 5개 나무 각각의 분류 경계를 그림
     fig, axes = plt.subplots(2, 3, figsize=(20, 10))
     for i, (ax, tree) in enumerate(zip(axes.ravel(), forest.estimators_)):
         ax.set_title("Tree {}".format(i))
         mglearn.plots.plot_tree_partition(X[['alcohol','volatile acidity']].values, y, tree, ax=ax)

     mglearn.plots.plot_2d_separator(forest, X[['alcohol','volatile acidity']].values, fill=True, ax=axes[-1, -1], alpha=.4)
     axes[-1, -1].set_title("Random forest")
     mglearn.discrete_scatter(X.iloc[:, 0], X.iloc[:, 1], y)
     plt.show()
```

```
[ ]  # 중요변수만 뽑아 생성하자 정확도 오히려 낮아짐. (기존 분석 최대 정확도: 0.668)
     print("학습용 데이터 세트 정확도: {:.3f}".format(forest.score(X_train[['alcohol','volatile acidity']], y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(forest.score(X_test[['alcohol','volatile acidity']], y_test)))
```

```
학습용 데이터 세트 정확도: 0.740
시험용 데이터 세트 정확도: 0.538
```

독립변수 제한 시 오히려 정확도 감소

대안2 : 독립변수 수정

대안2: 독립변수 수정

2-1. 2개의 독립변수를 임의로 제거

2-2. 중요도 하위 3개 독립변수를 제거

대안2: 독립변수 수정

2-1. 2개의 독립변수를 임의로 제거
2-2. 중요도 하위 3개 독립변수를 제거

```
[ ]  # 'pH', 'chlorides' 칼럼 제외한 wine_1 데이터프레임 신규 생성
     wine_1 = wine.drop(columns=["pH", "chlorides"])
     display(wine_1.head(5))
```

```
[ ]  # 신규 데이터 프레임으로(wine_1) 의사결정나무(tree_1) 만들기
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.model_selection import train_test_split

     X = wine_1.iloc[:, 1:]
     y = wine_1.iloc[:, 0]  #target 변수: 퀄리티(위에서 index 칼럼 제거해서 quality 칼럼이 맨 앞)

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=42)

     tree_1 = DecisionTreeClassifier(random_state=0)
     tree_1.fit(X_train, y_train)

     print("학습용 데이터 정확도: {:.3f}".format(tree_1.score(X_train, y_train)))
     print("시험용 데이터 정확도: {:.3f}".format(tree_1.score(X_test, y_test)))  #기존 데이터프레임보다(0.591) 정확도 소폭 상승.
```

```
학습용 데이터 정확도: 1.000
시험용 데이터 정확도: 0.596
```

```
[ ]  # 나무 depth 4로 제한
     tree_1 = DecisionTreeClassifier(max_depth=4, random_state=0)
     tree_1.fit(X_train, y_train)

     print("학습용 데이터 정확도: {:.3f}".format(tree_1.score(X_train, y_train)))
     print("시험용 데이터 정확도: {:.3f}".format(tree_1.score(X_test, y_test)))  #나무 깊이 제한하자 정확도 다시 감소.
```

```
학습용 데이터 정확도: 0.566
시험용 데이터 정확도: 0.532
```

독립변수 수정 1안

: 2개의 독립변수('pH', 'chlorides')
임의로 제거 후
분류 모델 생성

1안 의사결정나무 실행 결과:

기존 데이터보다 정확도가 소폭
상승했으나 깊이를 제한하니
0.596 > 0.532 로 감소하였음

Part 3. 분류모델 평가

대안2: 독립변수 수정

2-1. 2개의 독립변수를 임의로 제거
2-2. 중요도 하위 3개 독립변수를 제거

```
[ ]  # 신규 데이터 프레임으로(wine_1) 랜덤포레스트(forest_1) 만들기
     from sklearn.ensemble import RandomForestClassifier

     # wine_1 데이터프레임('pH', 'chlorides' 제외한 10개 변수)으로 10개의 나무를 생성함
     ## 100개 생성 시 과적합 문제 발생
     forest_1 = RandomForestClassifier(n_estimators=10, random_state=0)
     forest_1.fit(X_train, y_train)

     print("학습용 데이터 세트 정확도: {:.3f}".format(forest_1.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(forest_1.score(X_test, y_test))) # 기존 데이터프레임보다(0.668) 정확도 낮아짐.
```

```
학습용 데이터 세트 정확도: 0.983
시험용 데이터 세트 정확도: 0.643
```

```
[ ]  # 정확도 향상 위한 파라미터 조정 (나무 깊이 4로 제한)
     gbrt_1 = GradientBoostingClassifier(random_state=0, max_depth=4)
     gbrt_1.fit(X_train, y_train)

     print("학습 데이터 세트 정확도: {:.3f}".format(gbrt_1.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(gbrt_1.score(X_test, y_test))) # 정확도 소폭 상승했으나 여전히 기존 df보다 정확도 낮음.

     # 1안 적용 결과, 랜덤포레스트와 그래디언트 부스팅에서는 정확도 감소했으나 의사결정나무에서는 정확도 향상하는 모습 보임.
     ## 랜덤포레스트의 정확도 향상 유의미하다고 판단해 2안으로 진행.
```

```
학습 데이터 세트 정확도: 0.811
시험용 데이터 세트 정확도: 0.591
```

1안 적용 결과, 랜덤 포레스트, 그래디언트 부스팅 모델의 정확도는 감소했으나 의사결정나무에서는 소폭 상승

→ 조금 더 구체적으로 변수 수정

대안2: 독립변수 수정

2-1. 2개의 독립변수를 임의로 제거
2-2. 중요도 하위 3개 독립변수를 제거

독립변수 수정 2안

의사결정나무 변수 중요도

10	0.146706	alcohol
1	0.121405	volatile acidity
8	0.095694	pH
9	0.092257	sulphates
3	0.091321	residual sugar
4	0.086672	chlorides
5	0.079898	free sulfur dioxide
6	0.079820	total sulfur dioxide
2	0.074939	citric acid
0	0.066479	fixed acidity
7	0.061976	density
11	0.002836	type

랜덤포레스트 변수 중요도

10	0.123599	alcohol
1	0.099731	volatile acidity
7	0.099632	density
6	0.090595	total sulfur dioxide
9	0.088654	sulphates
4	0.086774	chlorides
5	0.086609	free sulfur dioxide
3	0.084055	residual sugar
8	0.083153	pH
2	0.077332	citric acid
0	0.076162	fixed acidity
11	0.003704	type

그래디언트 부스팅 변수 중요도

10	0.244839	alcohol
1	0.133047	volatile acidity
7	0.083346	density
9	0.081375	sulphates
3	0.077748	residual sugar
5	0.077039	free sulfur dioxide
6	0.076350	total sulfur dioxide
4	0.062957	chlorides
8	0.055841	pH
0	0.051435	fixed acidity
2	0.051233	citric acid
11	0.004790	type

3개 모델 공통 중요도 하위 3개 변수인 'type', 'citric acid', 'fixed acidity' 변수 제거

대안2: 독립변수 수정

2-1. 2개의 독립변수를 임의로 제거
2-2. 중요도 하위 3개 독립변수를 제거

```
[ ]  # 3개 모델 공통 중요도 하위 3개 변수 제거하고 모델 별 정확도 검토

     # 'type', 'citric acid', 'fixed acidity' 변수 제거한 새 데이터프레임(wine_2) 생성
     wine_2 = wine[['quality','alcohol','volatile acidity','density','sulphates','residual sugar','free sulfur dioxide','total sulfur dioxide', 'chlorides','pH']]

     display(wine_2.info())  # IV 1개('quality'), DV 총 9개
     display(wine_2.head(5))
```

index	quality	alcohol	volatile acidity	density	sulphates	residual sugar	free sulfur dioxide	total sulfur dioxide	chlorides	pH
0	5	10.2	0.695	0.99432	0.44	6.8	9.0	84.0	0.042	3.44
1	5	9.5	0.610	0.99690	0.59	2.4	10.0	42.0	0.067	3.19
2	5	10.9	0.210	0.99176	0.52	2.0	21.0	138.0	0.057	3.05
3	6	10.8	0.210	0.99390	0.50	6.0	29.0	108.0	0.046	3.26
4	6	10.9	0.400	0.99550	0.43	9.5	32.0	178.0	0.059	3.04

대안2: 독립변수 수정

2-1. 2개의 독립변수를 임의로 제거
2-2. 중요도 하위 3개 독립변수를 제거

```
[ ]  # 신규 데이터 프레임으로(wine_2) 의사결정나무(tree_2) 만들기
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.model_selection import train_test_split

     X = wine_2.iloc[:, 1:]  #9개 변수
     y = wine_2.iloc[:, 0]   # target 변수 quality

     # 데이터셋을 학습(train)과 테스트(test) 세트로 분할
     # random_state : random 으로 분할시 사용되는 난수 seed 숫자
     # X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, stratify=y, random_state=42)

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=42)

     tree_2 = DecisionTreeClassifier(random_state=0)
     tree_2.fit(X_train, y_train)
     print("학습용 데이터 정확도: {:.3f}".format(tree_2.score(X_train, y_train)))
     print("시험용 데이터 정확도: {:.3f}".format(tree_2.score(X_test, y_test))) #기존 데이터프레임보다(0.591) 정확도 소폭 상승.
```

```
학습용 데이터 정확도: 1.000
시험용 데이터 정확도: 0.593
```

3개 변수 제거 후 기존 정확도 (0.591)보다 소폭 상승

```
[ ]  # 과적합 해결하기 위해 나무 깊이 제한
     tree_2 = DecisionTreeClassifier(max_depth=4, random_state=0)
     tree_2.fit(X_train, y_train)

     print("학습용 데이터 정확도: {:.3f}".format(tree_2.score(X_train, y_train)))
     print("시험용 데이터 정확도: {:.3f}".format(tree_2.score(X_test, y_test))) # 파라미터 조절 결과 depth 줄일수록 정확도 급감. 기존 데이터프레임과 마찬가지 경향.
```

```
학습용 데이터 정확도: 0.564
시험용 데이터 정확도: 0.532
```

나무 깊이 제한하니 정확도 다시 감소

대안2: 독립변수 수정

2-1. 2개의 독립변수를 임의로 제거
2-2. 중요도 하위 3개 독립변수를 제거

```
[ ]  # 신규 데이터 프레임으로(wine_2) 그래디언트 부스팅(gbrt_2) 만들기
     from sklearn.ensemble import GradientBoostingClassifier

     gbrt_2 = GradientBoostingClassifier(random_state=0)
     gbrt_2.fit(X_train, y_train)
```

```
[ ]  print("학습용 데이터 세트 정확도: {:.3f}".format(gbrt_2.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(gbrt_2.score(X_test, y_test))) # 기존 데이터프레임보다(0.597) 정확도 낮지만 1안의 결과보다는 개선.
```

```
학습용 데이터 세트 정확도: 0.716
시험용 데이터 세트 정확도: 0.572
```

3개 변수 제거 후 그래디언트 부스팅 정확도는 0.572

```
[ ]  # 정확도 향상 위한 파라미터 조정 (나무 깊이 4로 제한)
     gbrt_2 = GradientBoostingClassifier(random_state=0, max_depth=4)
     gbrt_2.fit(X_train, y_train)

     print("학습 데이터 세트 정확도: {:.3f}".format(gbrt_2.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(gbrt_2.score(X_test, y_test))) # 1안의 개선결과와 동일한 시험용 데이터 세트 정확도 결과.

     #독립변수 수정 결과 의사결정나무의 경우 정확도가 향상되었지만 나머지에서는 유의미한 결과를 얻지는 못했음.
     #정확도와 변수의 중요도 간에는 상관관계 없음을 확인.
     #독립변수를 제거하는 것이 오히려 정확도를 떨어뜨림. 독립변수의 개수는 분류모델의 정확도를 떨어뜨리는 요인이 아님.
     #오히려 정확한 분류를 위해 필요함.
```

나무 깊이 제한하니 0.591로 정확도 향상됨

```
학습 데이터 세트 정확도: 0.807
시험용 데이터 세트 정확도: 0.591
```

대안1, 2 실행 결과

독립변수를 조작하여 모델을 생성해 본 결과...

- 모델 정확도와 독립변수의 종류 간에는 상관 관계 없음
- 독립변수 제거 시 오히려 모델 정확도는 감소
- 독립변수의 개수는 정확도 상승에 영향을 미치지 않음

→ 독립변수가 아닌 종속변수를 조작하는 방향

대안3: 종속변수 수정

대안3: 종속변수 수정

기존의 종속변수 'quality'를
multiple classes가 아닌
분류모델에 적합한 **이진분류**로 수정 후 재 분류

대안3: 종속변수 수정

```
#wine의 quality 변수를 5를 기준으로 5 초과는 G(좋음), 5 이하는 B(나쁨)으로 수정
wine['quality'] = np.where(wine['quality'] > 5, 'G', 'B')
wine
```

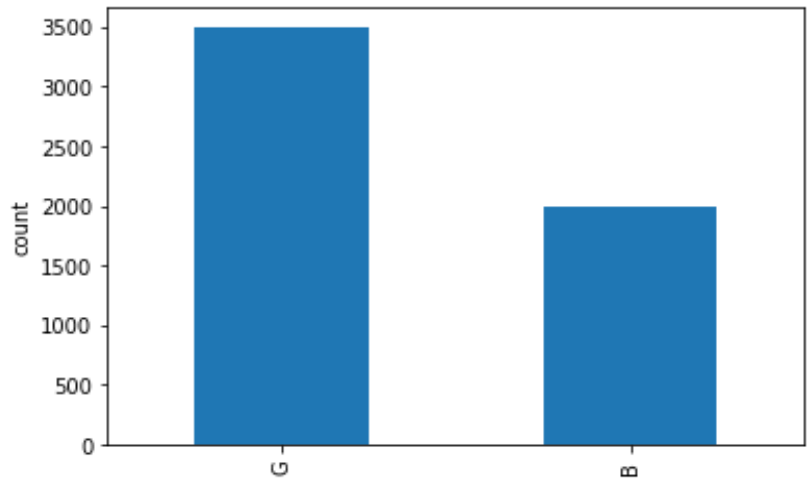
index	quality	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	type
0	B	5.6	0.695	0.06	6.8								
1	B	8.8	0.610	0.14	2.4								
2	B	7.9	0.210	0.39	2.0								
3	G	7.0	0.210	0.31	6.0								
4	G	7.8	0.400	0.26	9.5								
...	...	...	...	...	...								
5492	B	7.7	0.150	0.29	1.3								
5493	G	6.3	0.180	0.36	1.2								
5494	G	7.8	0.150	0.34	1.1								
5495	B	6.6	0.410	0.31	1.6								
5496	G	7.0	0.350	0.17	1.1								

5497 rows × 13 columns

```
[ ] # quality별 빈도수 확인
    print(wine['quality'].value_counts())

    wine['quality'].value_counts().plot.bar()
    plt.ylabel('count')
    plt.show()
```

```
G    3497
B    2000
Name: quality, dtype: int64
```



quality > 5 = 좋음(G)

quality ≤ 5 = 나쁨(B)

G : B

3497 : 2000

대안3 : 의사결정나무

```
[ ]  # 의사결정나무 생성 과정

     X = wine.drop('quality', axis = 1)
     y = wine['quality']

     # 데이터셋을 학습(train)과 테스트(test) 세트로 분할
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
     tree = DecisionTreeClassifier(random_state=0)
     tree.fit(X_train, y_train)
     print("학습용 데이터 정확도: {:.3f}".format(tree.score(X_train, y_train)))
     print("시험용 데이터 정확도: {:.3f}".format(tree.score(X_test, y_test)))   # 기존 데이터프레임보다도 정확도 훨씬 향상됨.
```

```
학습용 데이터 정확도: 1.000
시험용 데이터 정확도: 0.771
```

0.771

```
[ ]  # max_depth 설정으로 과적합 방지
     tree = DecisionTreeClassifier(max_depth=15, random_state=0)
     tree.fit(X_train, y_train)

     print("학습용 데이터 정확도: {:.3f}".format(tree.score(X_train, y_train)))
     print("시험용 데이터 정확도: {:.3f}".format(tree.score(X_test, y_test)))
```

```
학습용 데이터 정확도: 0.982
시험용 데이터 정확도: 0.777
```

깊이 조정 후 0.777

대안3 : 랜덤 포레스트

```
[ ]  # 랜덤포레스트 생성 과정

     forest = RandomForestClassifier(n_estimators=100, random_state=0)
     forest.fit(X_train, y_train)

     print("학습용 데이터 세트 정확도: {:.3f}".format(forest.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(forest.score(X_test, y_test)))
```

```
학습용 데이터 세트 정확도: 1.000
시험용 데이터 세트 정확도: 0.842
```

0. 842

```
[ ]  # max_depth 설정으로 과적합 방지
     forest = RandomForestClassifier(n_estimators=100, random_state=0, max_depth=10)
     forest.fit(X_train, y_train)

     print("학습용 데이터 세트 정확도: {:.3f}".format(forest.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(forest.score(X_test, y_test)))
```

```
학습용 데이터 세트 정확도: 0.918
시험용 데이터 세트 정확도: 0.805
```

깊이 조정 후 0.805

대안3 : 그래디언트 부스팅

```
[ ]  # 그래디언트 부스팅 생성 과정

     gbrt = GradientBoostingClassifier(random_state=0)
     gbrt.fit(X_train, y_train)
```

```
     GradientBoostingClassifier(random_state=0)
```

```
[ ]  print("학습용 데이터 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
     print("시험용 데이터 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))

     # 세 분류모델 모두에서, DV의 class를 multiple에서 binary로 수정하자, 정확도 대폭 향상됨.
```

```
     학습용 데이터 세트 정확도: 0.812
     시험용 데이터 세트 정확도: 0.768
```

0.768

“ Binaray classes로 수정 후 분류모델 정확도 대폭 향상 ”

Part 3. 분류모델 평가

대안3 : 그리드 서치

의사결정나무

```
[ ] # 의사결정나무 그리드서치
    from sklearn.model_selection import GridSearchCV

    # 파라미터를 dictionary 형태로 설정
    parameters = {'max_depth':[1,3,5,10,15],'min_samples_leaf':[1,2,3]}

    init_dt = DecisionTreeClassifier(random_state=0)
    grid_dt = GridSearchCV(init_dt, param_grid=parameters, cv=3, refit=True)
    print(grid_dt)

    grid_dt.fit(X_train, y_train)

    # GridSearchCV 결과 추출하여 DataFrame으로 변환
    scores_df = pd.DataFrame(grid_dt.cv_results_)
    scores_df[['params', 'mean_test_score', 'rank_test_score', 'split0_test_score', 'split1

GridSearchCV(cv=3, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [1, 3, 5, 10, 15],
                         'min_samples_leaf': [1, 2, 3]})
```

랜덤 포레스트

```
[ ] # 랜덤포레스트 그리드서치
    from sklearn.model_selection import GridSearchCV

    # 파라미터를 dictionary 형태로 설정
    parameters = {'max_depth':[1,3,5,10,15],'n_estimators':[50,100,150,200]}

    init_rf = GradientBoostingClassifier(random_state=0)
    grid_rf = GridSearchCV(init_rf, param_grid=parameters, cv=3, refit=True)
    print(grid_rf)


    grid_rf.fit(X_train, y_train)

    # GridSearchCV 결과 추출하여 DataFrame으로 변환
    scores_rf = pd.DataFrame(grid_rf.cv_results_)
    scores_rf[['params', 'mean_test_score', 'rank_test_score', 'split0_test_score', 'split1_

GridSearchCV(cv=3, estimator=GradientBoostingClassifier(random_state=0),
             param_grid={'max_depth': [1, 3, 5, 10, 15],
                         'n_estimators': [50, 100, 150, 200]})
```

그래디언 부스팅

```
[ ] # 그래디언트 부스팅 그리드서치
    from sklearn.model_selection import GridSearchCV

    # 파라미터를 dictionary 형태로 설정
    parameters = {'max_depth':[1,2,3,4,5], 'learning_rate':[0.1,0.08, 0.06, 0.04, 0.02]}

    init_gbrt = GradientBoostingClassifier(random_state=0)
    grid_gbrt = GridSearchCV(init_gbrt, param_grid=parameters, cv=3, refit=True)
    print(grid_gbrt)

    grid_gbrt.fit(X_train, y_train)

    # GridSearchCV 결과 추출하여 DataFrame으로 변환
    scores_gbrt = pd.DataFrame(grid_gbrt.cv_results_)
    scores_gbrt[['params', 'mean_test_score', 'rank_test_score', 'split0_test_score', 'split1_te

GridSearchCV(cv=3, estimator=GradientBoostingClassifier(random_state=0),
             param_grid={'learning_rate': [0.1, 0.08, 0.06, 0.04, 0.02],
                         'max_depth': [1, 2, 3, 4, 5]})
```

```
[ ] print('GridSearchCV 최적 파라미터:', grid_dt.best_params_)
    print('GridSearchCV 최고 정확도: {0:.4f}'.format(grid_dt.best_score_))  #그리드서치 이전이 더 높은 정확도(0.777)

    GridSearchCV 최적 파라미터: {'max_depth': 5, 'min_samples_leaf': 3}
    GridSearchCV 최고 정확도: 0.7455
```

0.777 → 0.7455

그리드 서치 이후 의사결정나무와 랜덤 포레스트와 달리

그래디언트 부스팅은 정확도 소폭 향상

```
[ ] print('GridSearchCV 최적 파라미터:', grid_rf.best_params_)
    print('GridSearchCV 최고 정확도: {0:.4f}'.format(grid_rf.best_score_))

    GridSearchCV 최적 파라미터: {'max_depth': 10, 'n_estimators': 200}
    GridSearchCV 최고 정확도: 0.8015
```

0.805 → 0.8015

```
[ ] print('GridSearchCV 최적 파라미터:', grid_gbrt.best_params_)
    print('GridSearchCV 최고 정확도: {0:.4f}'.format(grid_gbrt.best_score_))

    GridSearchCV 최적 파라미터: {'learning_rate': 0.1, 'max_depth': 5}
    GridSearchCV 최고 정확도: 0.7908
```

0.768 → 0.7908

대안3 : 교차 검증 정확도 비교

```
[ ]  from sklearn.model_selection import cross_val_score

     # 교차 검증을 10번 수행하여 10번의 교차 검증 평균 정확도를 비교 (10-fold cross validation)
     # default cv=5

     # 교차 검증을 10번 수행하여 10번의 교차 검증 평균 정확도를 비교 (10-fold cross validation)
     # default cv=5

     dt_scores = cross_val_score(tree, X_train, y_train, cv=10, scoring='accuracy')
     rf_scores = cross_val_score(forest, X_train, y_train, cv=10, scoring='accuracy')
     gbrt_scores = cross_val_score(gbrt, X_train, y_train, cv=10, scoring='accuracy')

     print("Accuracy")
     print("Decision tree: ", dt_scores)
     print("Random forest: ", rf_scores)
     print("Gradient boosting: ", gbrt_scores)

     print("Accuracy mean")
     print("Decision tree :{:.3f}".format(dt_scores.mean()))
     print("Random forest :{:.3f}".format(rf_scores.mean()))
     print("Gradient boosting :{:.3f}".format(gbrt_scores.mean()))
```

```
Accuracy
Decision tree:  [0.775      0.75       0.75454545 0.79772727 0.775      0.75454545
 0.74090909 0.76309795 0.79726651 0.77676538]
Random forest:  [0.80227273 0.79090909 0.79545455 0.79318182 0.79090909 0.78636364
 0.79772727 0.81093394 0.8428246  0.78
Gradient boosting:  [0.78863636 0.7636
 0.775      0.78132118 0.80865604 0.76
Accuracy mean
Decision tree :0.768
Random forest :0.800
Gradient boosting :0.772
```

의사결정나무: 0.768

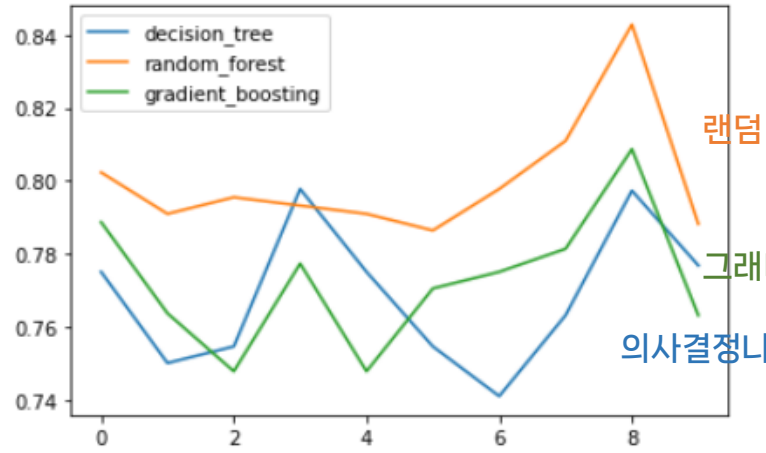
랜덤 포레스트: 0.800

그래디언트 부스팅: 0.772

```
[ ]  # 정확도 평가 결과 시각화
     cv_list = [
                ['decision_tree',dt_scores],
                ['random_forest',rf_scores],
                ['gradient_boosting',gbrt_scores],
               ]
     df = pd.DataFrame.from_dict(dict(cv_list))
     df.plot()

     # 세 모델 모두 종합했을 때 random forest가 가장 높은 정확도를 보임.
     # '대안3'의 그리드서치에서도 마찬가지의 결과 보임.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6057e8cc10>
```



04

인사이트 도출

INSIGHT

해당 데이터 분석에
가장 적합한 모델

랜덤 포레스트

와인품질에 영향
을 미친 변수

알코올

추가 분석 방향

Predict 함수로
외부 와인 데이터 품질 예측

기대효과 및 활용방안

B2B MODEL

1ST

등급순서	등급명	한국어 발음
최하위 등급	Vin de Table	벵 드 따블
지방와인	Vin de Pays	벵 드 뻬이
AOC 대기와인	V.D.Q.S.	
품질 인증 등급 (AOC)	A.O.C Appelation (d'Origine) Controlée	아뻴라시용 도리진 꽁트롤레
최상위 등급	Cru Bourgeois Grand Cru	크뤼 부르죠아 그랑 크뤼

품질 확인 서비스를 제공

생산한 와인이 목표 품질에 도달했는지 확인

MARKETING

2ND

제품 포지셔닝에 주요한 역할

브랜딩 및 마케팅 믹스 계획　　생산 활동과 마케팅에 도움

출시 목표인
와인의 특징 기입

→

와인의 품질
정보 제공

→

가격 책정

THANK YOU

It & Buisiness 002

2022

ALC. 18% BY VOL. 750ML