**기계학습 와인 데이터 예측**



164487 이정석

진행방법

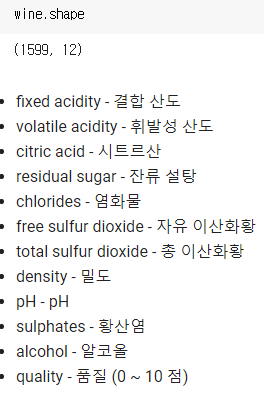
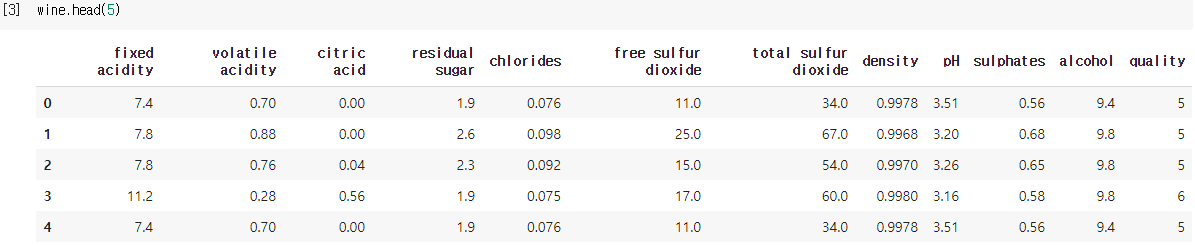
Git을 이용한 데이터 업로드

Google colab을 이용한 주피터 노트북 사용

1. 데이터 읽기

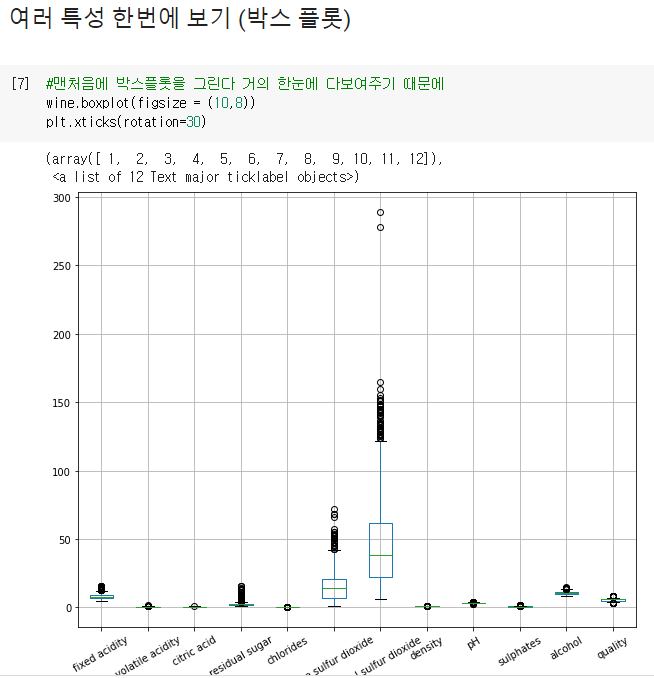


Git의 clone을 사용하여 데이터를 업로드하고 불러왔다.



이후 크기와 칼럼등을 확인하여 데이터의 분포 상태를 체크해볼 수 있었다. 이후

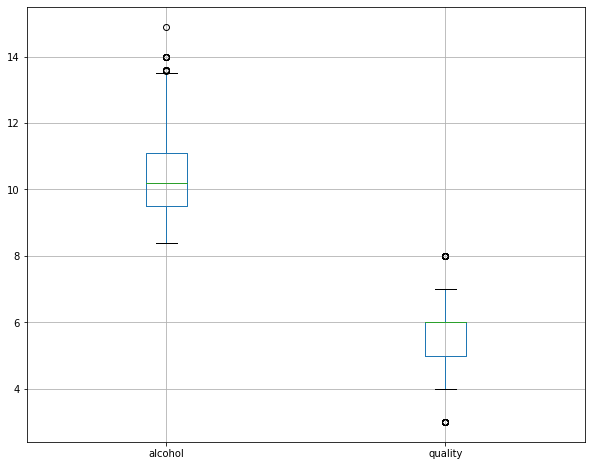
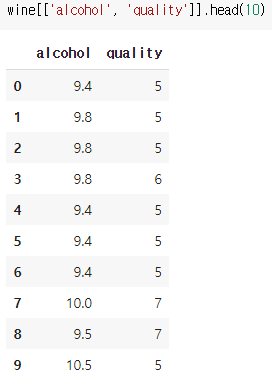
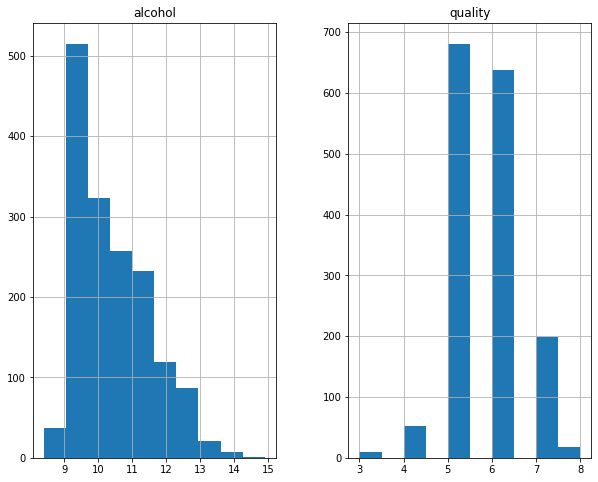
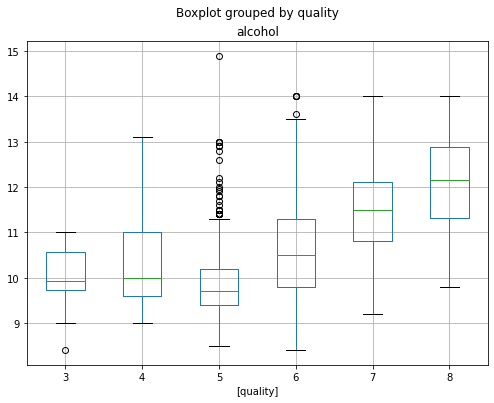
Describe()를 통해 와인의 데이터의 분포, 범위등을 확인 하였다.

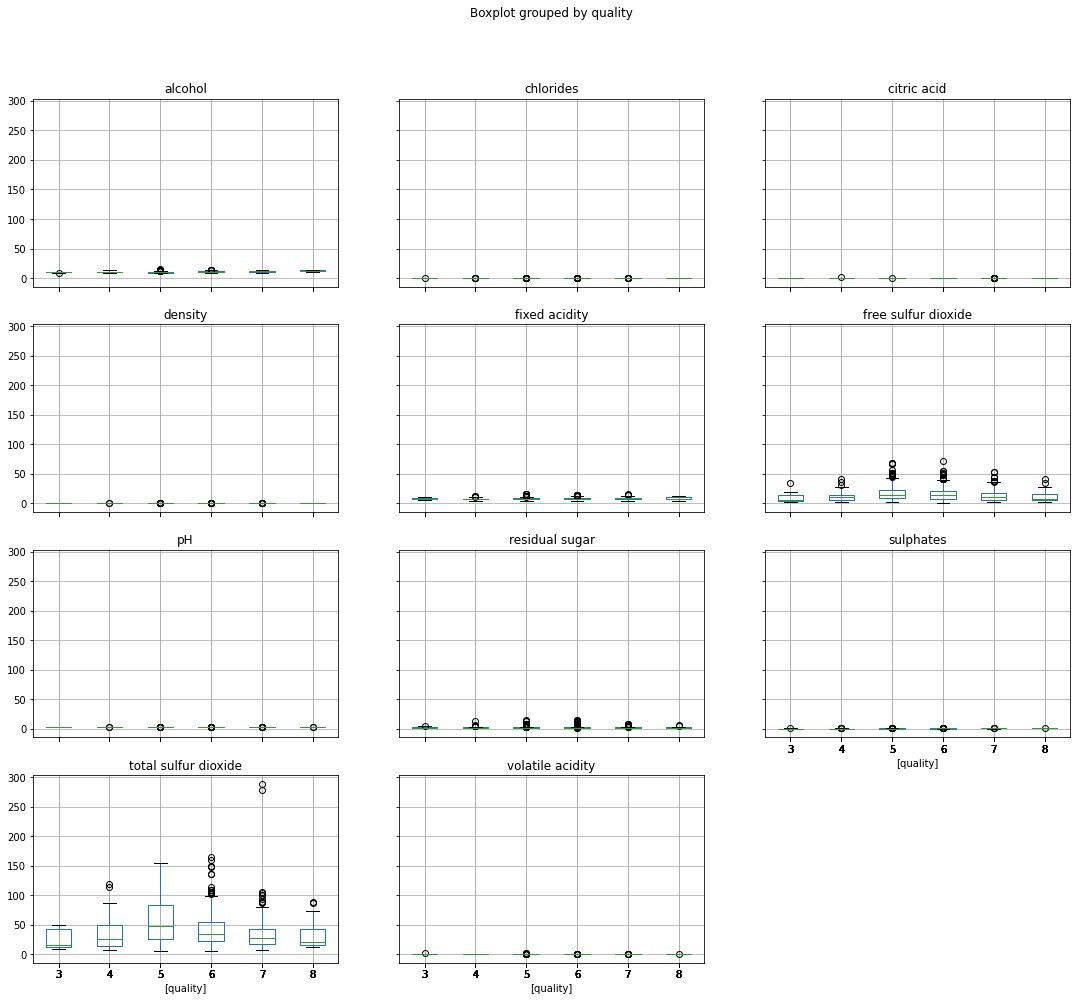
이러한 특성을 박스 플롯을 이용하여 볼 수 있었다.

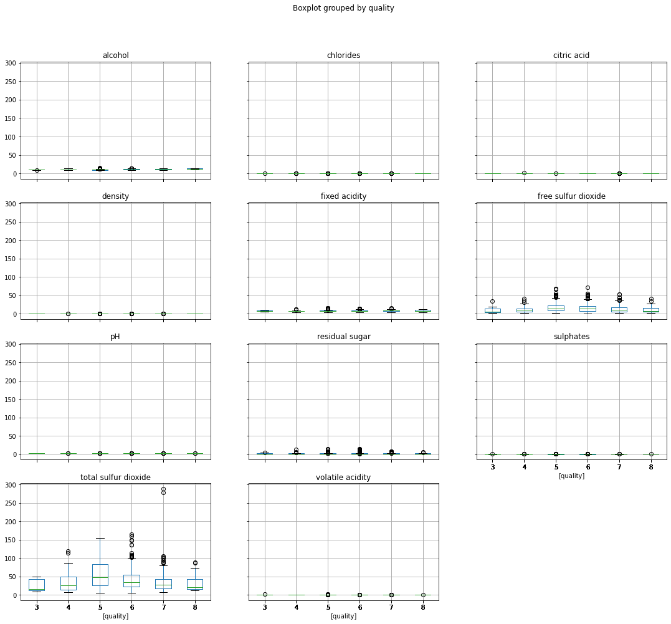
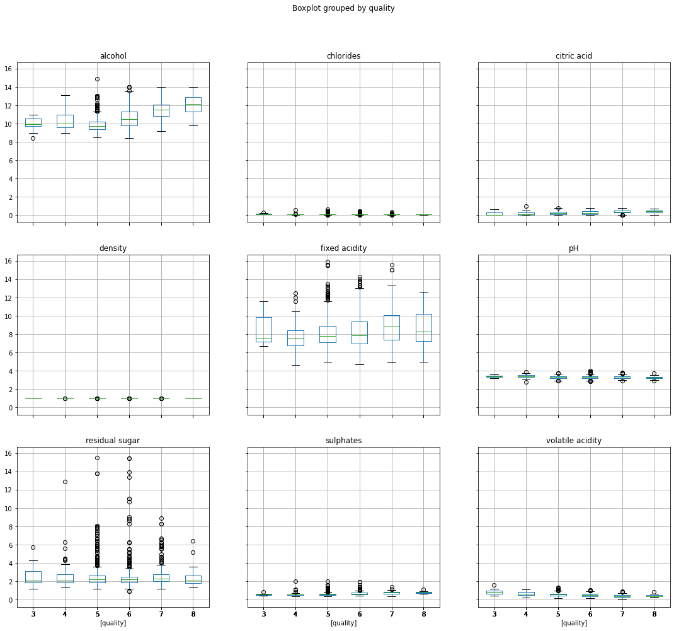
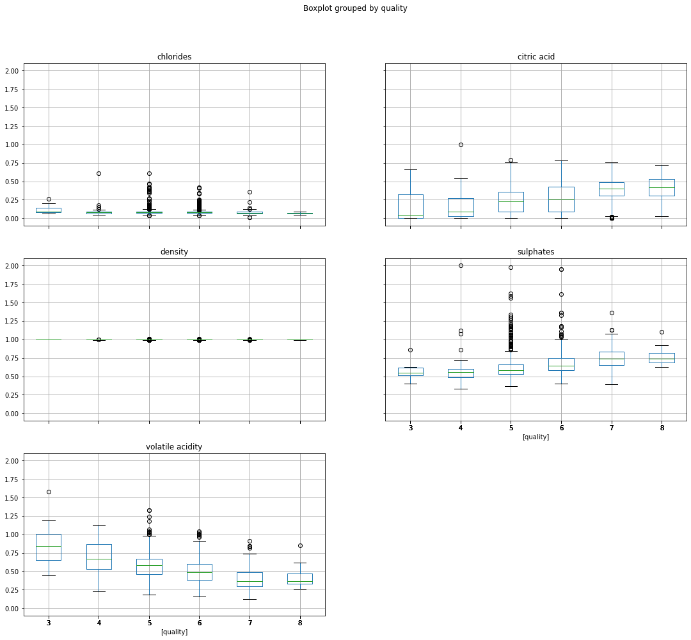
describe()에서도 확인 할 수 있듯이 **total sulfur dioxide 의 분포가 넓게 나타났다.**

**우리는 퀄리티를 계산해야 하기 때문에 퀄리티와 가장 연관성이 높은 품목을 찾을 경우 알코올을 볼 수 있었다.**

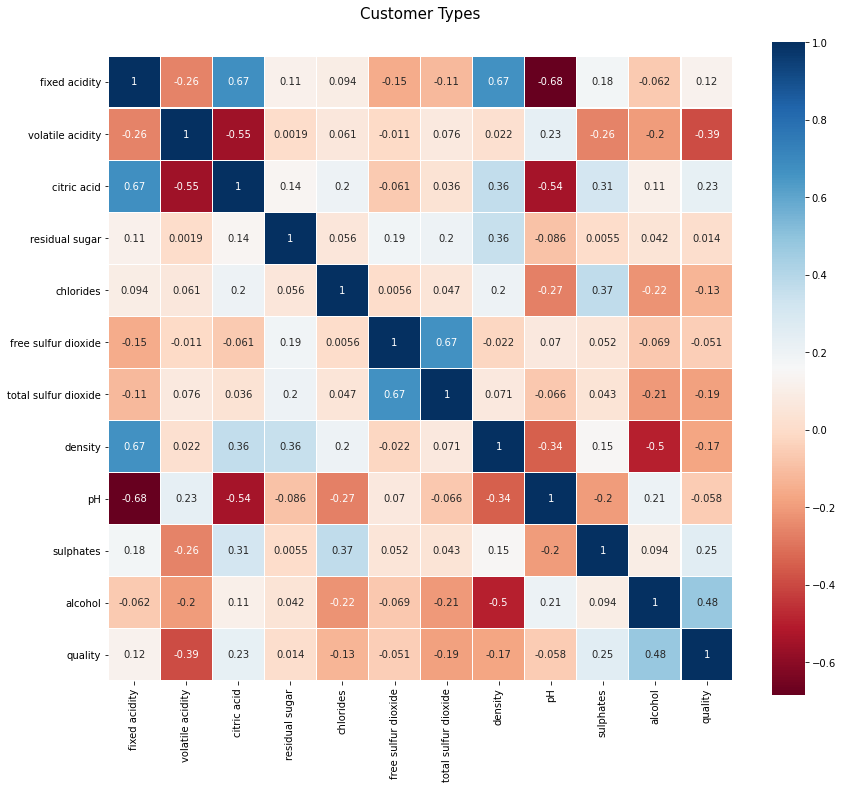
따라서 알코올과 퀄리티의 관계를 조사해 볼 수 있고 이를 히스토그램과 박스플롯으로 나타내면  

알코올은 퀄리티가 높은 것들은 품질이 좋은 것들이 많았다.

이와 같게 다른 데이터들에게도 같게 박스 플롯을 적용하여 관계를 살펴 보았다. 

컬럼을 드랍하여 여러 개를 비교하며 오차를 예측하고 이를 최소화한다.   

이때 피터슨 상관계수를 이용하면 보다 빠르게 파악 가능하다고 생각되어 찾아보게 되었다.

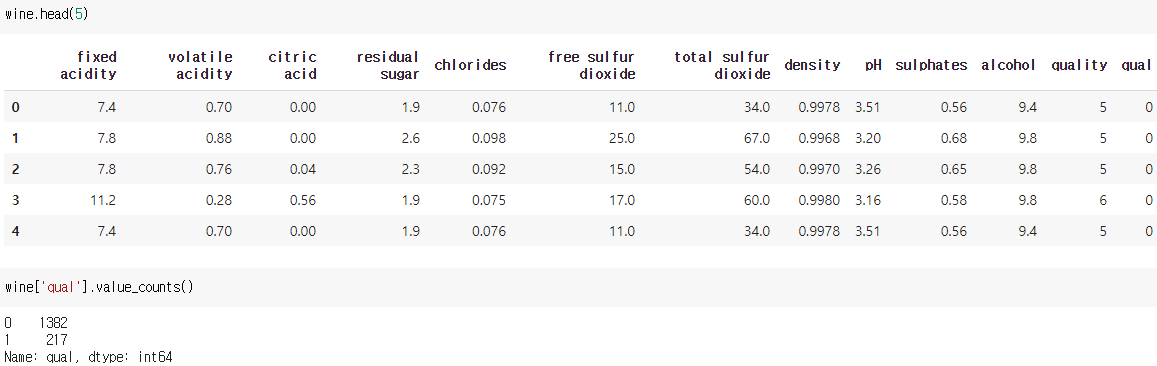


여기서도 보면 퀄리티와 알코올의 상관 계수가 가장 높음을 볼 수 있다.

레이블을 나누어 데이터를 분류하기 좋게 만들었다.



8만맛있다로하려하였으나8의분포가적어서7~8이맛있다로했다.



표준 스케일링을 하여 자료 분포를 평균 0 분포1이 되도록 만든다.

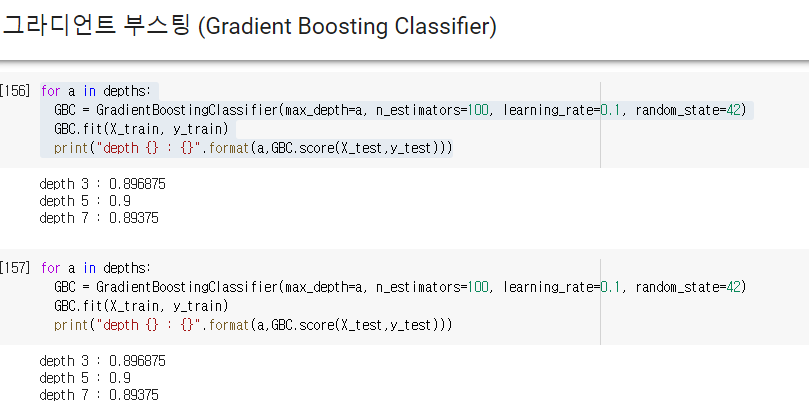


분류



Knn과 결정 트리는 그냥 진행 하였다.





랜덤 포레스트와 그래디언트 부스팅을 진행 하였는데

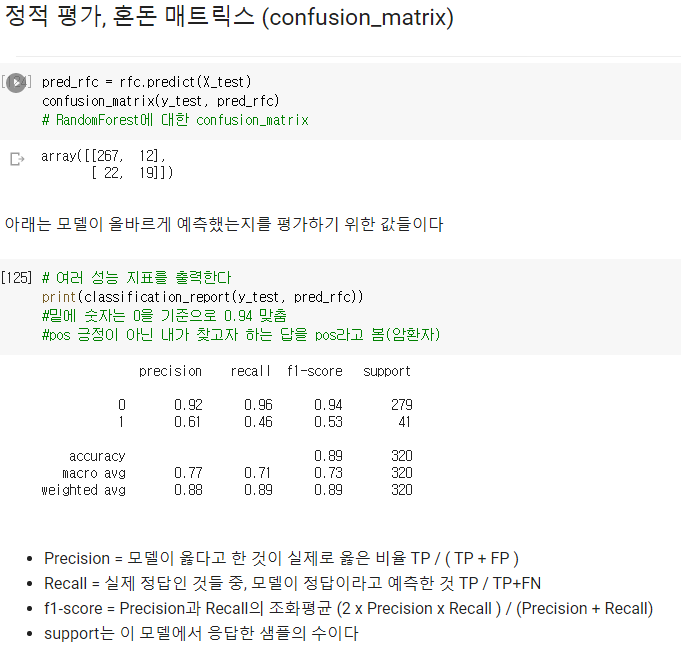
랜덤 포레스트는 돌릴때마다 값이 동일하지 않았으나 그라디언트는 계속 동일하였다.

이는 배깅과 부스팅의 차이점으로보인다.

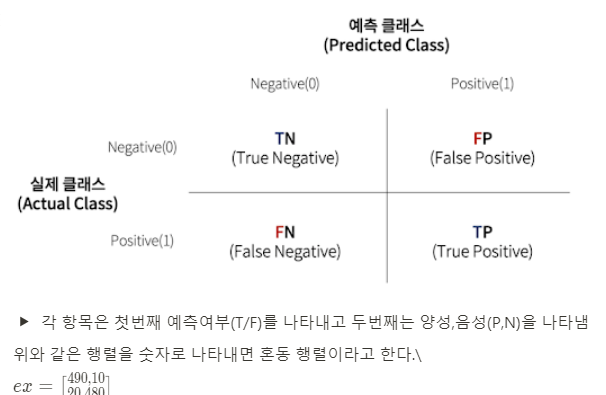
배깅은 부트스트랩을 통해 여러개의 학습 데이터 집합을 만들고 각 학습 데이터 집합별로 분류기를 만들어 이들이 **투표나 가중치 투표**를 하여 최종 판정을 하는 기법이고,

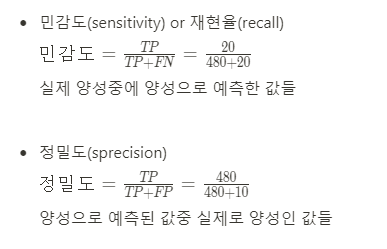
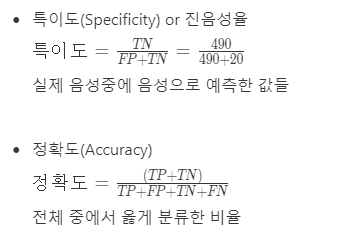
부스팅은 k개의 분류기를 순차적으로 만들어가는 앙상블 분류기 생성기법이므로 베깅에서 가중치가 달라짐에 따라 돌릴때마다 결과값이 같음을 보장해주지 않는 것 같다.

성능 측정의 방법으로 혼돈 매트릭스를 보였다.

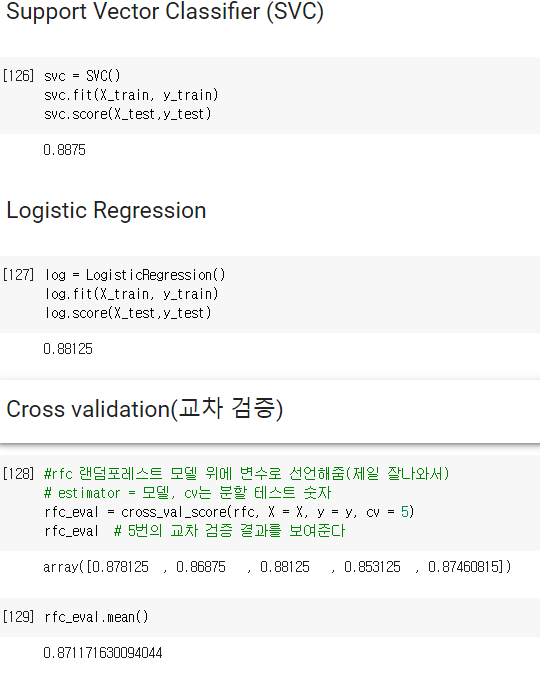


참고 자료로 아래와 같이 들 수 있다.



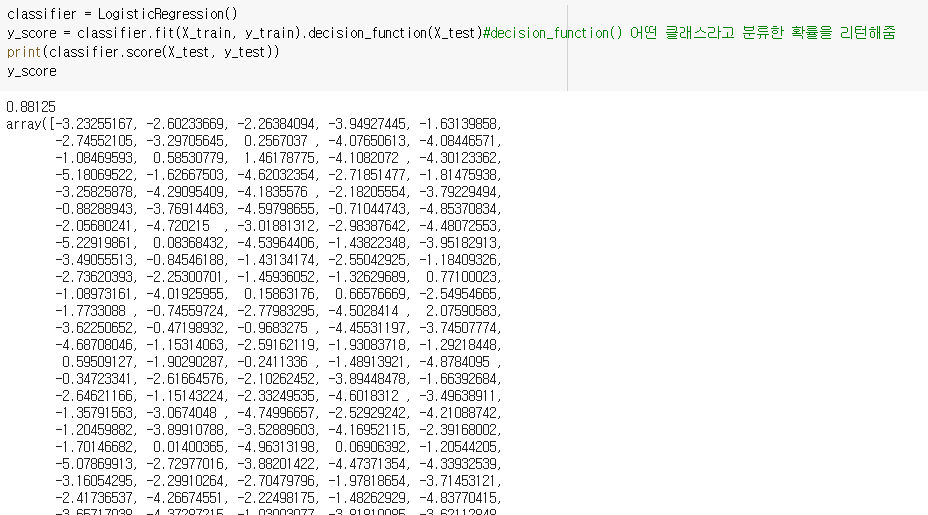
[https://www.notion.so/4-eae1fc91c14f4fe083ebb8232ebfa87c]



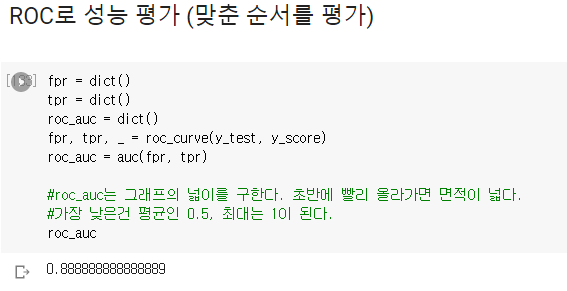
이후 순차적으로 SVC를 이용한 방법 Logistic회귀를 이용한 방법, 교차검증을 이용한 방법에 대해 나온다.

이후 test데이터를 각각의 모델에 맞게 학습하여 진행한 결과가 나온다.

로지스틱 회귀



ROC로 성능 평가



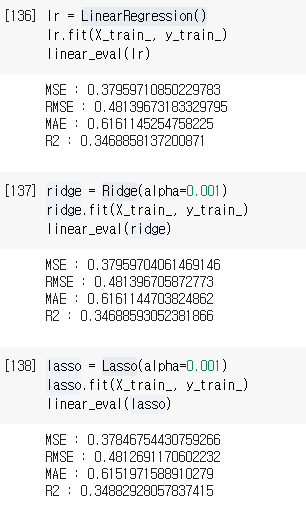
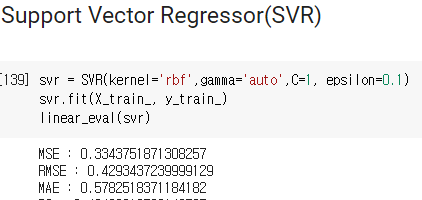
이후 회귀를 이용하여

LinearRegression

Ridge

Lasso

SVR을 이용한 학습이다

마지막으로 본인이 XGBoost를 이용한 학습을 하고 리포트를 마치겠다.

성능은 훈련 세트와 테스트 세트의 정확도, 그리고 실제 테스트의 정밀도와 재현율을 나타냈다.

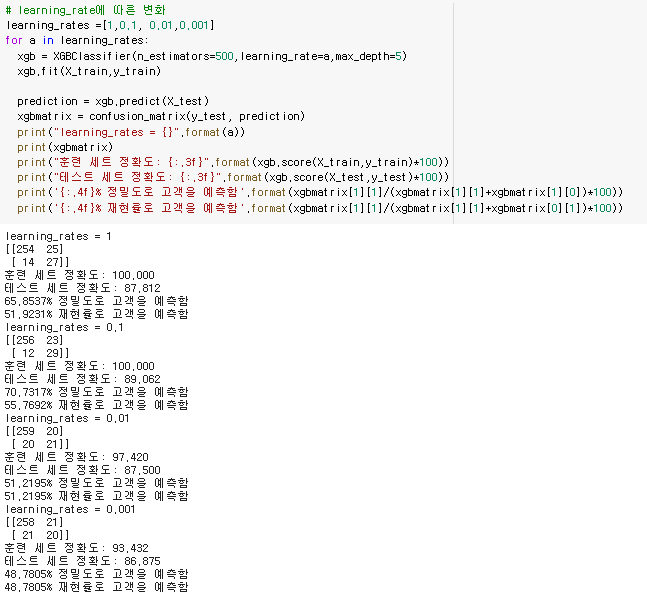
깊이에 따른 Xgboost



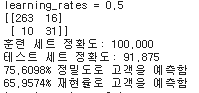
깊이가 깊어질수록 재현율이 올라감을 확인 할 수 있다.

정밀도는 3-> 5로갈때는 올라가나 5->7을 갈때는 올라가지 않았다. (12/258)

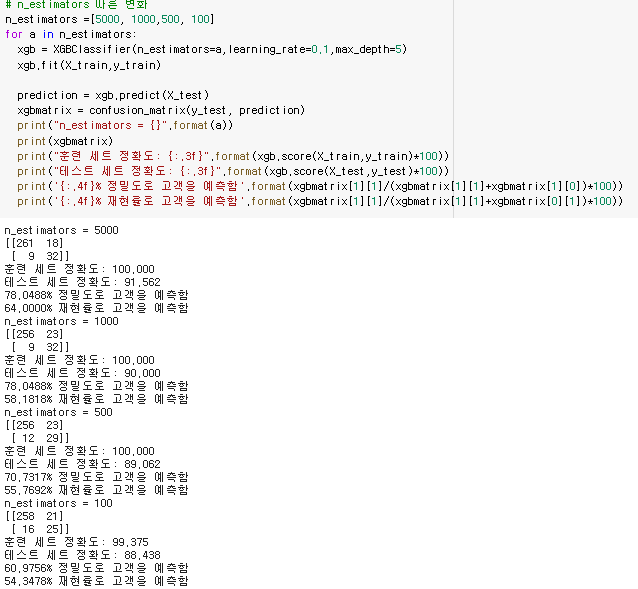
2. learning\_rate에따른 변화



Learning 가 너무 클경우(1)일때도 높은 수치로 나오지 않고 낮을 경우(0.001)일때도 높은 성능을 보이지 않았다. 위의 결과에서는 0.1이 가장 높게 나왔지만 차후에 0.5를 넣을 경우 0.5가 가장 높게 나왔다.

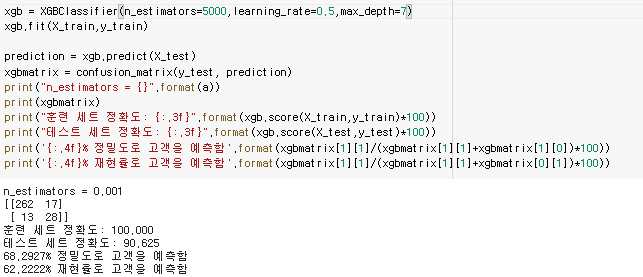


3. n\_estimators 따른 변화

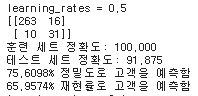
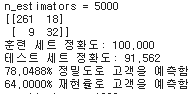
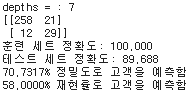


n\_estimators도 조절을 해보았으나 당연하게도 높을수록 점수가 잘 나왔음을 확인할 수 있었다.

위의 수치를 기반으로 모든 곳에서 높은 점수가 나온 계수를 합쳐서 모델을 생성하였다.



하지만 오히려 각 모델들보다 값이 좋지 않았는데 이는 각 모델들간의 계수가 독립적인 것이 아닌 긴밀한 관계가 있음을 알 수 있다.

이는 각 모델들의 변수 조절을 통해 산출한 최고의 결과물이다.

(defult = n\_estimators=500,learning\_rate=0.1,max\_depth=5)