

# **Bigdata Analytics**

Final Report

INE5015 – 22057

**빅데이터 애널리틱스 8조**

| 최준희 | 이강산 | 장혜연 | 황태영 |

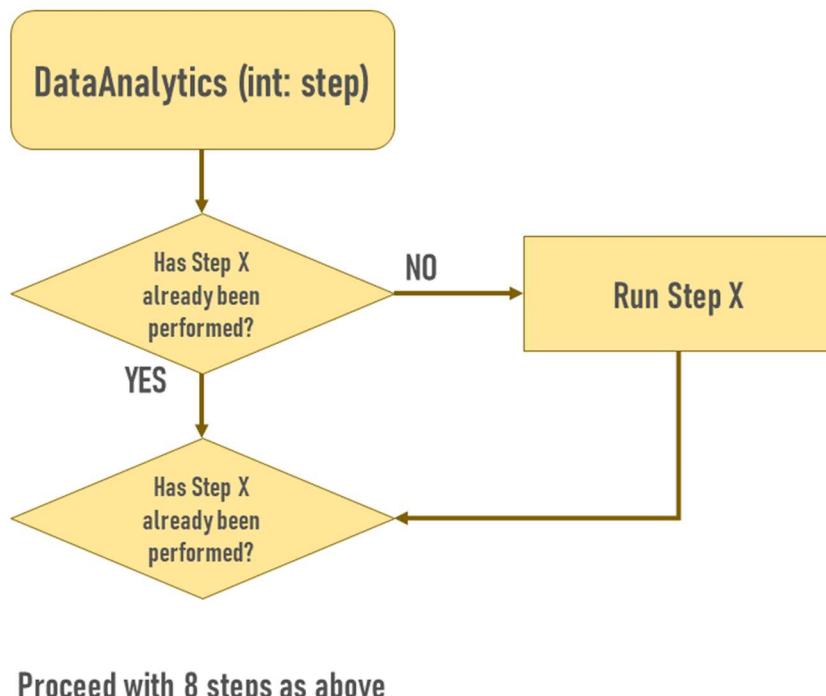
# List of Contents

- Logic of Data Preprocessing
- step-by-step process
  - Step 0 ~ Step 2 : raw file refining
  - Data Cleaning Overview (3 Steps)
    - ◆ Step 3 : correlation check and correction
    - ◆ Step 4 : Missing Value Imputation
    - ◆ Step 5 : Outlier corrections
  - Step 6 : Data Scaling
- Feature Selection
- Data Sampling overview
- Performance
  - The performance of two samplings
  - Prediction accuracy according to each algorithm
    - ◆ Logistic Regression
    - ◆ Decision Tree
    - ◆ Random Forest
    - ◆ Boosting

GITHUB : [https://github.com/ChoiJunhee/INE5015\\_bigdata\\_analytics](https://github.com/ChoiJunhee/INE5015_bigdata_analytics)

# Logic of our Data Processing

전반적인 Preprocessing 과정은 기능, 목적에 따라 Step으로 구별했고, 매 Step마다 시각적, 통계적 데이터를 확인하면서 진행하였다. 아래는 Step에 대한 구성 Logic이다.



DataAnalytics 함수에서 필요한 함수를 호출하여 사용하는 구조이며, 각 단계마다 csv 파일을 저장하도록 하여 변화를 직관적으로 확인하기 용이하도록 디자인하였다.

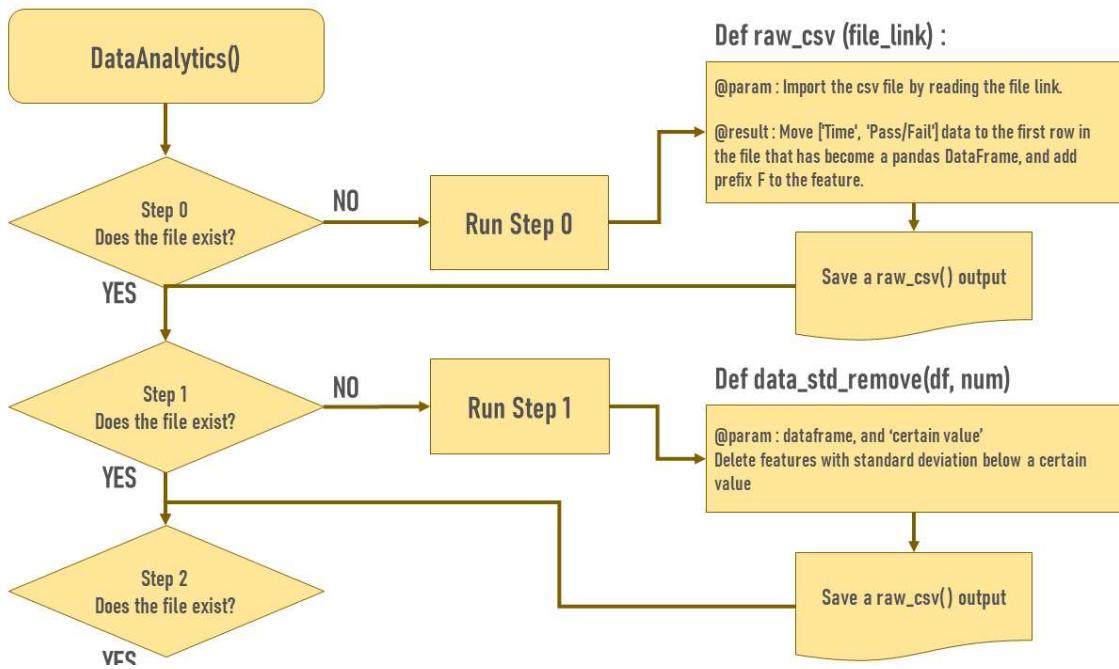
각 Step을 크게 나누어 보면 Raw data를 가공에 용이하도록 조정하는 부분, Pass/Fail 데이터를 분리해 총 3가지 데이터셋으로 나누어 진행할 수 있도록 하는 부분이 있다.

이후 각 Feature (독립변수)간 종속성을 확인하고, 제거하기 위해 진행하는 Correlation과, 결측치 처리/보정 과정, 이상치 처리/보정 과정. 즉, 데이터 클리닝 과정이 있다.

마지막으로 Feature Selection과 샘플링을 통해 최종 데이터 셋 후보를 선정하고, 퍼포먼스를 확인하는 과정으로 마무리한다.

# Step 0 ~ Step 2 : Ready to Preprocess

이 단계는 데이터 파일을 불러오고, 이후 보정에 있어 편의성을 높이기 위해 보정한다.



## Explanation of Steps

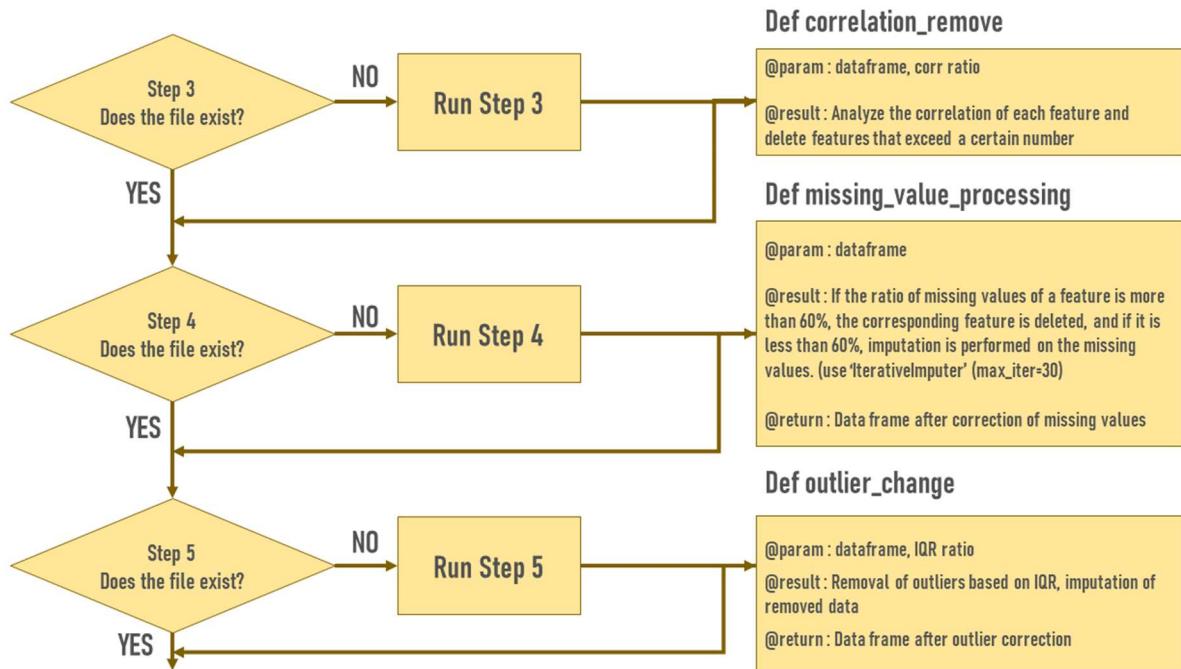
STEP	STEP DESCRIPTION
STEP 0	raw_csv 함수를 통해, Pandas Read_CSV를 실행하여 데이터 셋을 받아온다. 이후 작업의 편의를 위해 행 위치 변경, Prefix 추가 등의 과정을 거친 Dataframe 파일을 받고, 저장한다.
STEP 1	일정 계수 미만의 표준편차를 가진, 특정 선택의 필요도가 낮은 Feature들을 제거하고, Dataframe 파일을 저장한다. 그리고 데이터 셋을 3개로 나누는 과정을 거친다.
STEP 2	데이터 셋은 Pass Data, Fail Data, Both Data로 나누며, 이렇게 데이터 셋을 나누는 것은 프로젝트 초기의 아이디어 중 유용한, 유의미한 결과를 가져오는 계기가 된다.

Step 0 부터 Step 2까지의 3 과정은 데이터 전 처리를 위한 준비 단계에 불과하다.

> 위 과정에서 1566개 (1463+104)의 테스트 케이스, 247개의 Feature Data가 남았다.

## Step 3 ~ Step 5 : Data Cleaning

이 단계에서는 데이터셋의 노이즈를 제거하기 위해 데이터 클리닝을 진행하는 과정이다. 크게 독립변수 간 종속성 (비율), 결측치 제거 및 보정, 이상치 제거 및 보정으로 나뉜다.



### Explanation of Steps

STEP	STEP DESCRIPTION
STEP 3	각 독립변수 (Feature)간 상관성이 높다는 것은 다중공선성 발생 소지가 있고, 정확한 예측을 방해하는 요인으로 작용할 수 있다. 상관관계가 높은 Feature들을 제거하는 과정이다.
STEP 4	데이터에 결측치가 많아도 정확한 예측을 방해한다. 독립변수의 수를 최대한 유지하기 위해, 결측치가 60% 이상인 Feature를 제거하고, 60% 미만은 IterativeImputer를 통해 보정한다.
STEP 5	각 Feature의 사분위 값을 확인하고, IQR 방식을 통해 양 극단에 있는 이상치들을 제거한다. 이 때, Fail 데이터셋은 오버샘플링을 진행하지 않았기 때문에 가중치를 달리 설정한다.

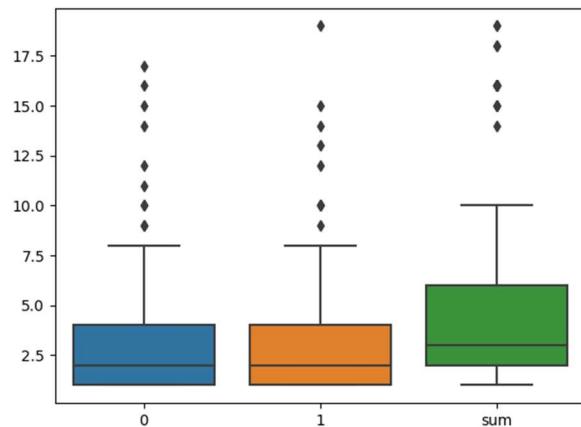
다음 페이지에는 Step3, 4, 5의 구체적인 내용과 편차 제거의 효용성을 비교한다.

## Data Cleaning - Overview

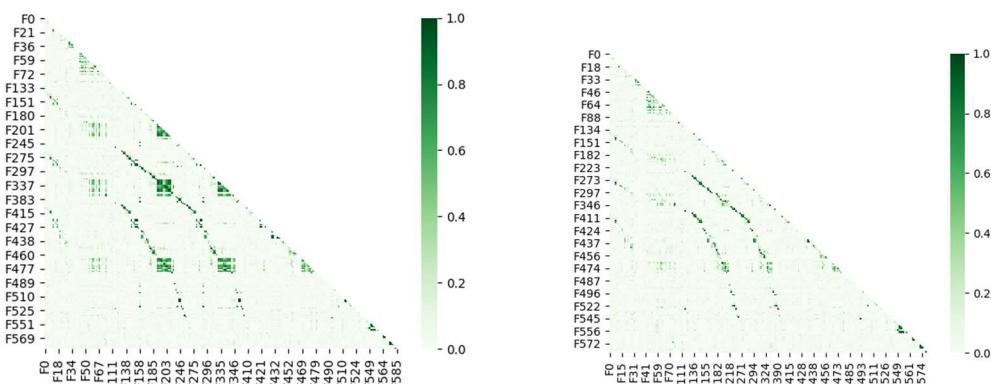
데이터 전처리 과정에 있어, 데이터 클리닝 과정은 매우 중요하다. 특히 결측치와 이상치에 대해 처리하는 것은 데이터 분석 및 모델링 결과를 크게 변화할 수 있기 때문에, 데이터의 불안전성과 잡음, 불일치 등을 최대한 효과적으로 처리해야 한다.

### Step 3 – Correlation

상관관계를 분석 (Correlation Analysis)을 한다는 것은 통계학적으로 두 변수간 선형적 관계를 분석하는 것이다. 독립 변수 (=Feature)간 상관관계가 높다면 두 변수의 연관성이 높다는 것이고, 필요치 않은 연산을 할 뿐만 아니라 Clustering 에도 방해가 된다.



우리는 이 프로젝트 과정에서 상관관계가 높은 Feature들을 계산했고 아래 예시처럼 선택된 Feature들을 제거했다. (여기서, Fail 데이터와 All/Pass 데이터간 상관관계 분석 내용이 달라 약 20여개의 Feature가 Fail 데이터에는 남아있게 되었다.



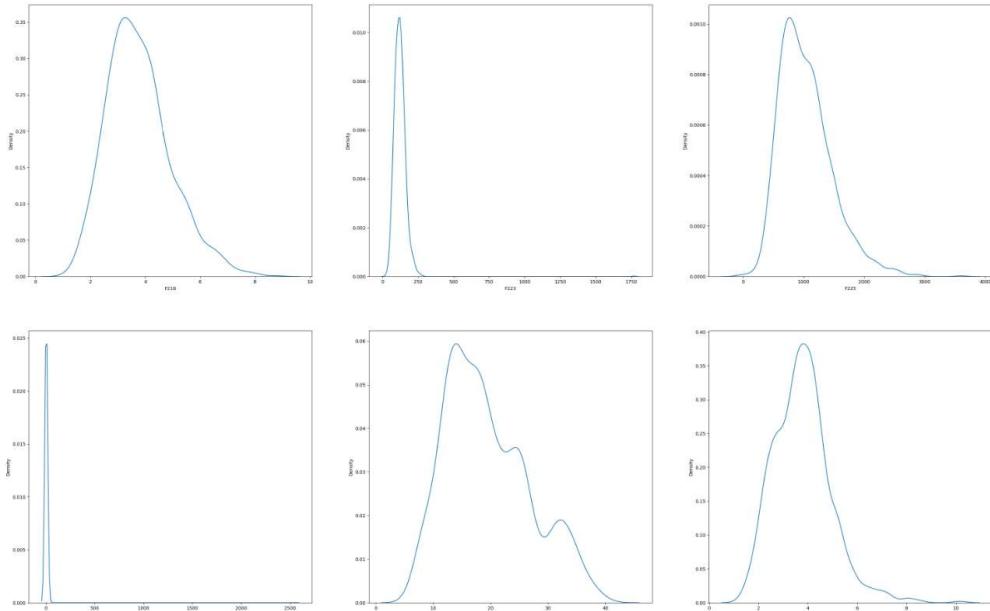
위의 사진은 상관관계가 높은 Feature들을 제거하기 전, 후의 모습이다.

## Step 4 – Missing Value

이 프로젝트에 있는 결측치는 패턴이 없는, Random Missing Feature 라 가정하고 진행하였다. 결측치를 처리하는 방법에는 삭제, 대치, 예측이 있는데, 결측치들의 특성이 패턴을 가지고 있다는 가정 하에 예측 모델을 구현해야 하기 때문에 이 프로젝트에서는 Deletion, Imputation 두 가지 방법을 사용하였다.

Scikit Learn에서 제공하고 있는 impute 중 Iterative Imputer를 사용하였다. 다른 모든 특성에서 개별 특성을 추정하는 다변량 대치 방식이며, Round Robin 알고리즘으로 각 Feature를 모델링 하여 결측값을 대치하는 기능을 한다.

또한 Iterative Imputer는 KNN 알고리즘으로 결측 치를 예측하여 채워 넣는 방식인데, Max-Iter를 30으로 조정함으로써 최종 라운드 동안 계산된 결과를 반환하기전에, 수행할 Round의 수를 늘림으로써 데이터의 완전성이 높아지기를 기대하였다.

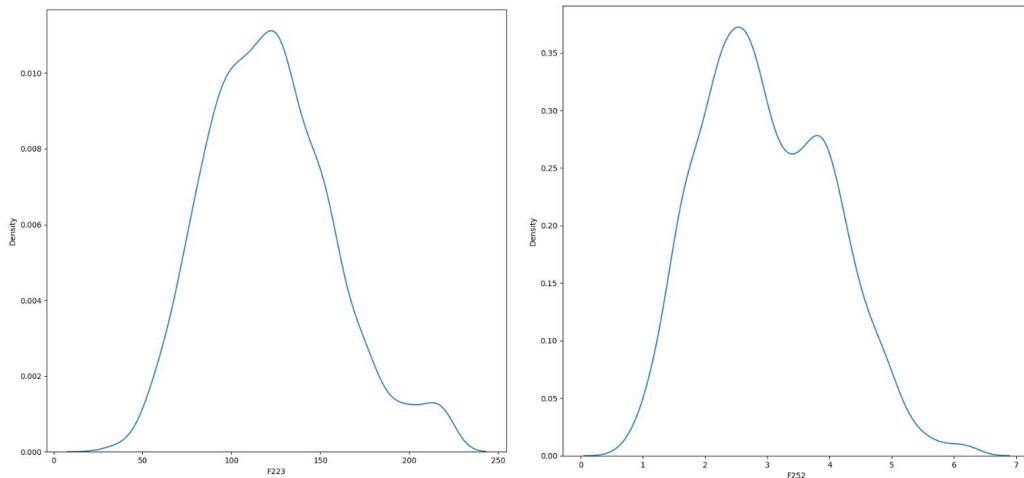


위 사진은 결측치 보정이 완료된 Feature들의 일부 사진이다. 위 사진에서 있는 F223, F252의 경우 이상치 처리와 보정이 이루어지지 않아 편차의 정도를 알 수 없으며, 다음 단계인 이상치 보정 (삭제, 보정, 편차제거) 단계를 통해 결과를 확인할 수 있다.

기준에는 Fail 데이터에 IQR 가중치를 높여 20여개의 Feature 개수 차이가 났으나, 데이터 왜곡이 우려되어 동일하게 적용하였다.

## Step 5 – Outlier Value

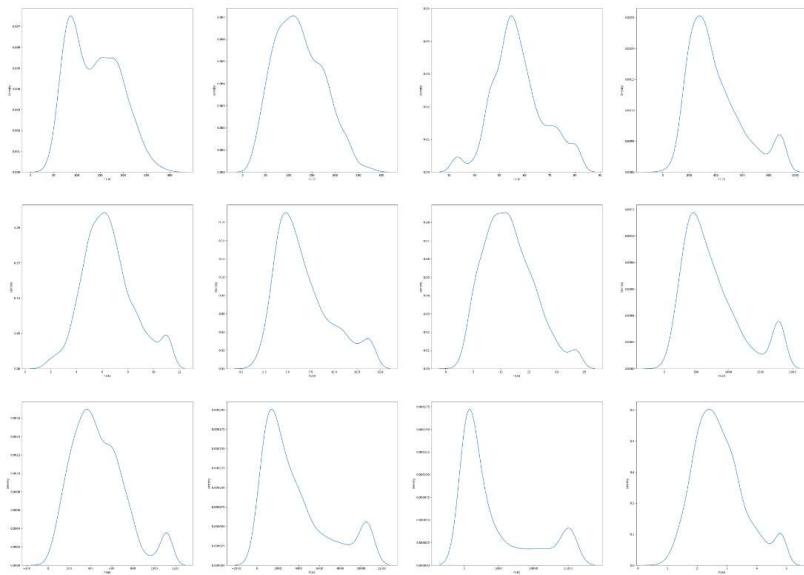
이상치의 기준은 IQR Weight 1.5 를 적용하였으며, 일반적인 이상치들을 보정할 수 있었다. 기준 이상인 데이터는 기준치의 최대값으로 대체 하였고, 기준 이하인 데이터는 기준치의 최솟값으로 대체 하였다. 이후 편차 제거를 통해 대체 방법의 단점을 상쇄하였다.



Step 4에서는 편차가 작아 제거되어야 할 대상으로 보였던 Feature 223, 252의 이상치 처리 이후의 그래프다. 이상치를 제거하면서 생긴 편차의 변동을 반영하기 위해 편차 제거를 약한 단계 (표준편차 기준 0.1 0.5 1.0)부터 높은 단계로 여러 번 시각화를 통해 확인한 결과 중간 단계가 이 단계에서의 최적의 값이라고 판단 되었다.

테스트 해본 결과, 편차의 변동이 생기는 STEP5에서만 Feature 수의 변화가 있었다.

STEP3	STEP4	STEP5																																																																																																																																																															
<pre> 119 ## corr 0.3 / 0.6 큰 의미 없음 / std3 진행 120 s3_c30_all = correlation_remove(std3_all) 121 print("----- 필터링 -----") 122 print(s3_c30_all) 123 124 s3_c30_all_2 = data_std_remove(s3_c30_all) 125 print("----- 필터링 -----") 126 print(s3_c30_all_2) 127 print("----- 필터링 -----") 128 return 129 df.to_csv('./[step 0] - rawfile_low_refine' 130 131 s3_c30_all.to_csv('./[step 3] - correlation </pre> <p>1567 rows x 202 columns]</p> <p>----- 필터링 -----</p> <table border="1"> <thead> <tr> <th>Time</th><th>Pass/Fail</th><th>F0</th><th>...</th></tr> </thead> <tbody> <tr><td>2008-07-19 11:55</td><td>-1</td><td>3030.93</td><td>...</td></tr> <tr><td>2008-07-19 12:32</td><td>-1</td><td>3095.78</td><td>...</td></tr> <tr><td>2008-07-19 13:17</td><td>1</td><td>2932.61</td><td>...</td></tr> <tr><td>2008-07-19 14:43</td><td>-1</td><td>2988.72</td><td>...</td></tr> <tr><td>2008-07-19 15:22</td><td>-1</td><td>3032.24</td><td>...</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>562</td><td>2008-10-16 15:13</td><td>-1</td><td>2899.41</td><td>...</td></tr> <tr><td>563</td><td>2008-10-16 20:49</td><td>-1</td><td>3052.31</td><td>...</td></tr> <tr><td>564</td><td>2008-10-17 5:26</td><td>-1</td><td>2978.81</td><td>...</td></tr> <tr><td>565</td><td>2008-10-17 6:01</td><td>-1</td><td>2894.92</td><td>...</td></tr> <tr><td>566</td><td>2008-10-17 6:07</td><td>-1</td><td>2944.92</td><td>...</td></tr> </tbody> </table> <p>1567 rows x 202 columns]</p> <p>----- 필터링 -----</p>	Time	Pass/Fail	F0	...	2008-07-19 11:55	-1	3030.93	...	2008-07-19 12:32	-1	3095.78	...	2008-07-19 13:17	1	2932.61	...	2008-07-19 14:43	-1	2988.72	...	2008-07-19 15:22	-1	3032.24	...	...	...	...	...	562	2008-10-16 15:13	-1	2899.41	...	563	2008-10-16 20:49	-1	3052.31	...	564	2008-10-17 5:26	-1	2978.81	...	565	2008-10-17 6:01	-1	2894.92	...	566	2008-10-17 6:07	-1	2944.92	...	<pre> 158 s4_all = pd.read_csv('./[step 4] - Data' 159 s4_pass = pd.read_csv('./[step 4] - Data' 160 s4_fail = pd.read_csv('./[step 4] - Data' 161 162 print("----- 필터링 -----") 163 print(s4_all) 164 165 s4_all_2 = data_std_remove(s4_all, 1) 166 print("----- 필터링 -----") 167 print(s4_all_2) 168 print("----- 필터링 -----") 169 return 170 171 #이상치 처리-&gt; fail 데이터가 너무 작아 172 173 s5_w15_p50_pass = outlier_change(s4_all, </pre> <p>1567 rows x 181 columns]</p> <p>----- 필터링 -----</p> <table border="1"> <thead> <tr> <th>Time</th><th>Pass/Fail</th><th>F0</th><th>...</th></tr> </thead> <tbody> <tr><td>2008-07-19 11:55</td><td>-1</td><td>3030.93</td><td>...</td></tr> <tr><td>2008-07-19 12:32</td><td>-1</td><td>3095.78</td><td>...</td></tr> <tr><td>2008-07-19 13:17</td><td>1</td><td>2932.61</td><td>...</td></tr> <tr><td>2008-07-19 14:43</td><td>-1</td><td>2988.72</td><td>...</td></tr> <tr><td>2008-07-19 15:22</td><td>-1</td><td>3032.24</td><td>...</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>562</td><td>2008-10-16 15:13</td><td>-1</td><td>2899.41</td><td>...</td></tr> <tr><td>563</td><td>2008-10-16 20:49</td><td>-1</td><td>3052.31</td><td>...</td></tr> <tr><td>564</td><td>2008-10-17 5:26</td><td>-1</td><td>2978.81</td><td>...</td></tr> <tr><td>565</td><td>2008-10-17 6:01</td><td>-1</td><td>2894.92</td><td>...</td></tr> <tr><td>566</td><td>2008-10-17 6:07</td><td>-1</td><td>2944.92</td><td>...</td></tr> </tbody> </table> <p>1567 rows x 181 columns]</p> <p>----- 필터링 -----</p>	Time	Pass/Fail	F0	...	2008-07-19 11:55	-1	3030.93	...	2008-07-19 12:32	-1	3095.78	...	2008-07-19 13:17	1	2932.61	...	2008-07-19 14:43	-1	2988.72	...	2008-07-19 15:22	-1	3032.24	...	...	...	...	...	562	2008-10-16 15:13	-1	2899.41	...	563	2008-10-16 20:49	-1	3052.31	...	564	2008-10-17 5:26	-1	2978.81	...	565	2008-10-17 6:01	-1	2894.92	...	566	2008-10-17 6:07	-1	2944.92	...	<pre> 108 print("----- 필터링 -----") 109 print(s5_w15_p50_all) 110 111 s4_all_2 = data_std_remove(s5_w15_p50_all, 0.5) 112 print("----- 필터링 -----") 113 print(s4_all_2) 114 115 return 116 117 #이상치 처리-에서 처리에서 허용한 pun 부수 허용 개수, 118 #부수 양의 차이로 세기 초기에는 천부 허용이 있는 119 120 s5_w15_p50_pass = pd.read_csv('./[step 5] - Data' 121 s5_w15_p50_pass.to_csv('./[step 5] - Data' 122 s5_w15_p50_fail = pd.read_csv('./[step 5] - Data' 123 s5_w15_p50_fail.to_csv('./[step 5] - Data' 124 125 print("[*] Step 5 - Done!(*") 126 step = 6 127 128 (step == 0): 129 130 </pre> <p>1567 rows x 181 columns]</p> <p>----- 필터링 -----</p> <table border="1"> <thead> <tr> <th>Time</th><th>Pass/Fail</th><th>F0</th><th>...</th></tr> </thead> <tbody> <tr><td>2008-07-19 11:55</td><td>1</td><td>3030.98</td><td>...</td></tr> <tr><td>2008-07-19 12:32</td><td>1</td><td>3095.78</td><td>...</td></tr> <tr><td>2008-07-19 13:17</td><td>1</td><td>2932.63</td><td>...</td></tr> <tr><td>2008-07-19 14:43</td><td>1</td><td>2988.79</td><td>...</td></tr> <tr><td>2008-07-19 15:22</td><td>-1</td><td>3032.24</td><td>...</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>562</td><td>2008-10-16 15:13</td><td>-1</td><td>2899.41</td><td>...</td></tr> <tr><td>563</td><td>2008-10-16 20:49</td><td>-1</td><td>3052.31</td><td>...</td></tr> <tr><td>564</td><td>2008-10-17 5:26</td><td>1</td><td>2978.84</td><td>...</td></tr> <tr><td>565</td><td>2008-10-17 6:01</td><td>1</td><td>2894.99</td><td>...</td></tr> <tr><td>566</td><td>2008-10-17 6:07</td><td>1</td><td>2944.92</td><td>...</td></tr> </tbody> </table> <p>1567 rows x 181 columns]</p> <p>----- 필터링 -----</p>	Time	Pass/Fail	F0	...	2008-07-19 11:55	1	3030.98	...	2008-07-19 12:32	1	3095.78	...	2008-07-19 13:17	1	2932.63	...	2008-07-19 14:43	1	2988.79	...	2008-07-19 15:22	-1	3032.24	...	...	...	...	...	562	2008-10-16 15:13	-1	2899.41	...	563	2008-10-16 20:49	-1	3052.31	...	564	2008-10-17 5:26	1	2978.84	...	565	2008-10-17 6:01	1	2894.99	...	566	2008-10-17 6:07	1	2944.92	...
Time	Pass/Fail	F0	...																																																																																																																																																														
2008-07-19 11:55	-1	3030.93	...																																																																																																																																																														
2008-07-19 12:32	-1	3095.78	...																																																																																																																																																														
2008-07-19 13:17	1	2932.61	...																																																																																																																																																														
2008-07-19 14:43	-1	2988.72	...																																																																																																																																																														
2008-07-19 15:22	-1	3032.24	...																																																																																																																																																														
...	...	...	...																																																																																																																																																														
562	2008-10-16 15:13	-1	2899.41	...																																																																																																																																																													
563	2008-10-16 20:49	-1	3052.31	...																																																																																																																																																													
564	2008-10-17 5:26	-1	2978.81	...																																																																																																																																																													
565	2008-10-17 6:01	-1	2894.92	...																																																																																																																																																													
566	2008-10-17 6:07	-1	2944.92	...																																																																																																																																																													
Time	Pass/Fail	F0	...																																																																																																																																																														
2008-07-19 11:55	-1	3030.93	...																																																																																																																																																														
2008-07-19 12:32	-1	3095.78	...																																																																																																																																																														
2008-07-19 13:17	1	2932.61	...																																																																																																																																																														
2008-07-19 14:43	-1	2988.72	...																																																																																																																																																														
2008-07-19 15:22	-1	3032.24	...																																																																																																																																																														
...	...	...	...																																																																																																																																																														
562	2008-10-16 15:13	-1	2899.41	...																																																																																																																																																													
563	2008-10-16 20:49	-1	3052.31	...																																																																																																																																																													
564	2008-10-17 5:26	-1	2978.81	...																																																																																																																																																													
565	2008-10-17 6:01	-1	2894.92	...																																																																																																																																																													
566	2008-10-17 6:07	-1	2944.92	...																																																																																																																																																													
Time	Pass/Fail	F0	...																																																																																																																																																														
2008-07-19 11:55	1	3030.98	...																																																																																																																																																														
2008-07-19 12:32	1	3095.78	...																																																																																																																																																														
2008-07-19 13:17	1	2932.63	...																																																																																																																																																														
2008-07-19 14:43	1	2988.79	...																																																																																																																																																														
2008-07-19 15:22	-1	3032.24	...																																																																																																																																																														
...	...	...	...																																																																																																																																																														
562	2008-10-16 15:13	-1	2899.41	...																																																																																																																																																													
563	2008-10-16 20:49	-1	3052.31	...																																																																																																																																																													
564	2008-10-17 5:26	1	2978.84	...																																																																																																																																																													
565	2008-10-17 6:01	1	2894.99	...																																																																																																																																																													
566	2008-10-17 6:07	1	2944.92	...																																																																																																																																																													
변화 없음	변화 없음	변화 있음																																																																																																																																																															



Step 5를 진행한 Feature들의 모습이다.

(STEP3 ~ 5의 히스토그램 등 변화를 알 수 있는 시각화 데이터)

## Data Cleaning Abstract

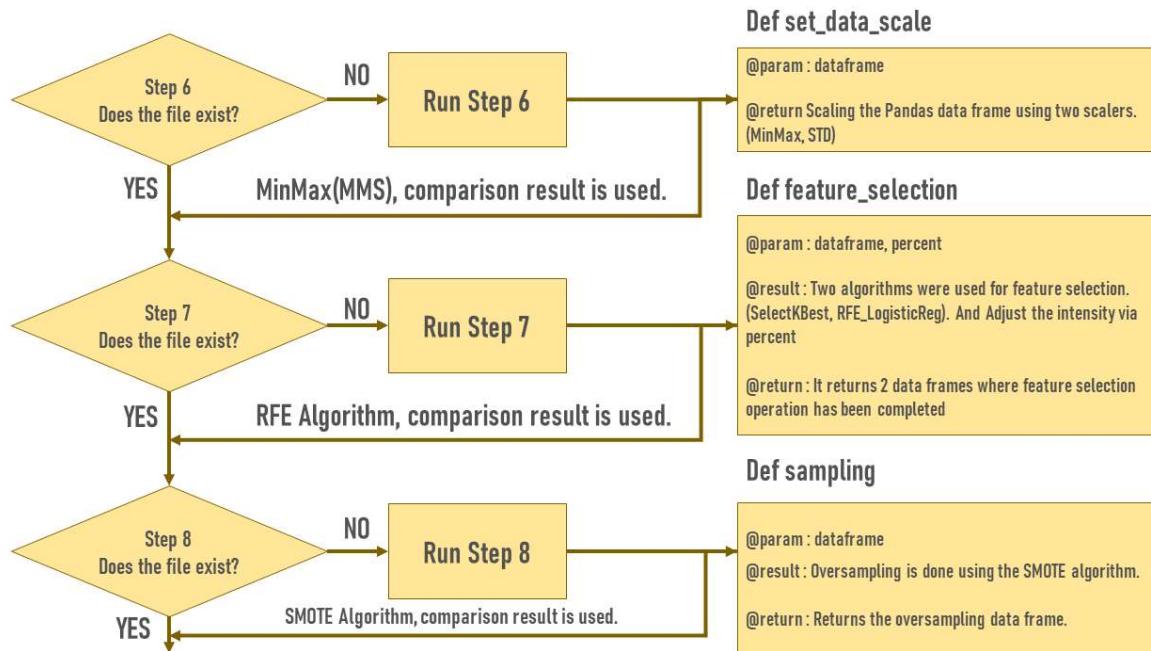
데이터 과학의 목표인 모집단에 대한 결론 도출을 정상적으로 진행하기 위해 필요한 작업이다. 표본 평균과 실제 평균의 차이를 표본 오차, 표본 데이터들의 평균값을 산출하여 표본 평균의 표준 편차를 구할 수 있다.

우리는 이 프로젝트에서 신뢰 구간을  $IQR * 1.5$  으로 정했고, 그 외의 값은 결측치로 간주하여 값을 대체하도록 하였다. Skewed Data를 다루다 보니 데이터 샘플링 이전까지는 Feature 제거에 있어 색다른 시도 보다는, 일반적이고 보수적으로 진행하였다.

선형 회귀 모델에서는 이 외에도 추가적인 조건이 필요하다. 노이즈 (outlier)가 없어야 되는 것은 기본이고 상관관계가 높은 변수가 있는 경우 overfitting의 여지가 있다. 또한, 정규 분포를 따를 때 더 신뢰할 수 있는 예측이 이루어진다.

그렇게 이 프로젝트에서는 이상치 제거와 상관관계 제거를 진행했으며, 오버샘플링 이후에도 추가적인 상관관계 제거를 실행하였다. (샘플링 과정에서 진행)

## Step 6~8 : Data Scaling, Feature Selection, Sampling



이전의 3 단계를 통해 데이터 클리닝을 진행했다면, 앞으로 할 3단계는 모델 성능을 높이기 위한 작업이다. Step 6에서는 스케일링을 진행, Scikit-Learn에서 제공하는 4가지 알고리즘 (Standard Scaler, Robust Scaler, MinMax Scaler, Normalizer)을 사용하였다.

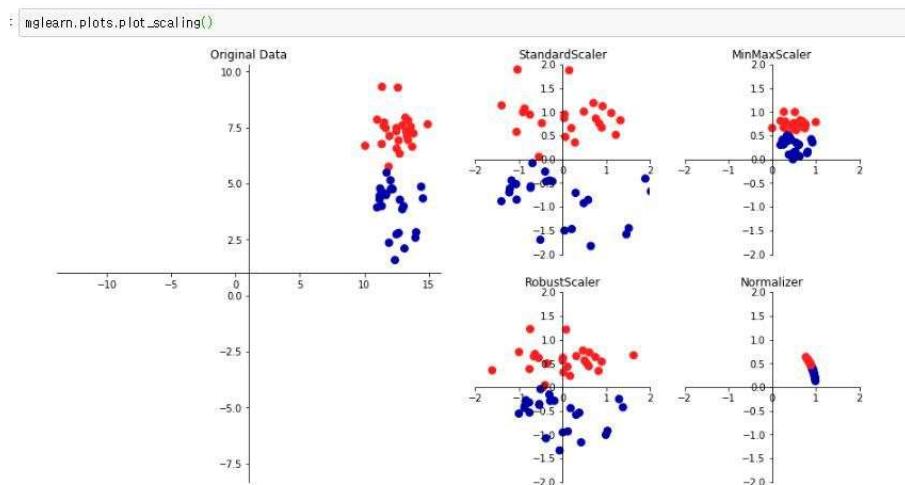


Image reference: <https://homeproject.tistory.com/entry/%EB%8D%B0%EC%9D%B4%ED%84%BO-%EC%8A%A4%EC%BC%80%EC%9D%BC%EB%A7%81-Data-Scaling>

4 가지 스케일러의 특징을 나타낸 이미지이다. 여기서 이상치의 영향을 받지 않는 Robust 는 추후에 탐지 못할 이상치를 데이터셋에 그대로 남길 여지가 있어 제외하였고, Standard Scaler, MinMax Scaler 를 선택하였다. Logistic Regression 테스트와 여러 테스트 결과 두 가지 방법에서 큰 차이는 없었다.

→ MinMax Scaler 를 사용하기로 결정하였다.

# Feature Selection

Feature Selection은 크게 3가지 방법으로 나눌 수 있다. Filtering, Wrapper, 그리고 둘을 합친 Embedded 가 있으며, 이 프로젝트에서 Filter Method와 Wrapper Method를 하나씩 사용하여 두 방법의 차이를 비교하였다. 그리고 두 과정 모두 모델 학습을 위한 작업으로, Feature들의 품질을 향상을 위해 특성을 선별하는 과정이라 할 수 있다.

## SelectKBest

단일 변수 선택법으로, 일종의 양상을 기법을 사용한다. Feature를 하나씩 사용했을 때의 예측 모델의 성능을 평가하여 정확도가 가장 좋은 Feature들을 선택하는 것이다.

## Recursive Feature Elimination

모든 특성의 조합을 시도하고, 가장 좋은 특성 셋을 구하는 Brute Force 방법을 사용한다. 또한 이 과정에서는 '데이터 전처리'에 초점을 맞추어 특성 개수를 자동으로 정해주는 RFECV가 아닌 RFE를 사용하였다.

## Result

```
RFE_MMS_ALL
0.8272 0.8061 0.8624 0.8332 0.8947

KBS_MMS_ALL
0.8112 0.7843 0.8595 0.8201 0.8804
```

Logistic Regression, SMOTE 알고리즘을 사용하여 샘플링 후 성능평가를 무작위로 100회 시도한 평균 값이다. (순서대로 Accuracy, Precision, Recall, F1-Score, ROC\_curve)

	SelectKBest	Recursive Feature Elimination
Accuracy	-	약 1%p 우세
Precision	-	약 2%p 우세
Recall		무의미한 차이
F1-Score	-	약 1%p 우세
ROC_curve	-	약 1.5%p 우세

위 결과에 따라, 두 가지 방법 중 RFE를 선정하여 오버 샘플링을 진행하였다.

## Feature Selection - Result



Recursive Feature Elimination 과정을 통해 아래 128개의 Feature를 선별하였다.

['F0', 'F1', 'F2', 'F6', 'F14', 'F15', 'F18', 'F21', 'F22', 'F24', 'F27', 'F28', 'F32', 'F39', 'F40', 'F41', 'F48', 'F51', 'F55', 'F59', 'F60', 'F63', 'F64', 'F65', 'F66', 'F68', 'F70', 'F71', 'F83', 'F90', 'F115', 'F117', 'F122', 'F129', 'F133', 'F134', 'F135', 'F137', 'F138', 'F139', 'F142', 'F151', 'F160', 'F161', 'F166', 'F180', 'F182', 'F183', 'F188', 'F200', 'F201', 'F208', 'F218', 'F250', 'F252', 'F268', 'F269', 'F270', 'F271', 'F272', 'F294', 'F295', 'F316', 'F319', 'F344', 'F361', 'F388', 'F406', 'F409', 'F410', 'F412', 'F413', 'F415', 'F416', 'F417', 'F418', 'F419', 'F423', 'F424', 'F428', 'F429', 'F432', 'F433', 'F434', 'F435', 'F437', 'F438', 'F439', 'F452', 'F453', 'F454', 'F455', 'F460', 'F467', 'F468', 'F470', 'F471', 'F472', 'F474', 'F476', 'F480', 'F482', 'F483', 'F484', 'F485', 'F486', 'F487', 'F488', 'F489', 'F490', 'F491', 'F493', 'F497', 'F499', 'F500', 'F510', 'F511', 'F520', 'F525', 'F527', 'F539', 'F540', 'F541', 'F545', 'F570', 'F572', 'F577', 'F585']

# Sampling

데이터 샘플링의 목표는 데이터 셋의 값들을 정리, 조정하여 최적의 입력 데이터 (학습 데이터)를 만드는 것이다. 데이터 샘플링의 종류에는 크게 확률적 샘플링, 비확률적 샘플링으로 나누는데, 이 프로젝트에서는 SMOTE, 즉 확률적 샘플링을 사용하였다.

SECOM 데이터는 Pass와 Fail간 비율이 약 14:1 수준으로 매우 심한 불균형을 가지고 있다. 이번 샘플링에서는 재현율(Recall)의 성능 향상을 기준으로 SMOTE와 ADASYN 알고리즘을 비교, 사용하였다. (오버샘플링과 언더샘플링의 결과도 첨부함)

## OVER SAMPLING

SMOTE 알고리즘은, 클래스 데이터와 그 데이터에서 가장 가까운 k개의 데이터 중 무작위로 선택된 데이터 사이의 직선 상에 가상의 클래스 데이터를 만드는 방법이다. (KNN)

ADASYN 알고리즘은 SMOTE와 전반적으로 동일하지만 ADASYN은 생성된 데이터를 무조건 소수 클래스라 하는 반면, SMOTE는 생성된 데이터를 무조건 소수 클래스라 하지 않고, 분류 모형에 따라 분류한다.

[SMOTE] RFE_MMS - try_1000	[SMOTEEENN] RFE_MMS - Try 1000
0.8374 0.8124 0.8786 0.844 0.895	0.8373 0.8121 0.8789 0.844 0.8948

성능 테스트를 통해 확인한 결과, SMOTE와 ADASYN중 SMOTE가 근소한 차이로 좋은 성능을 보였으며, SMOTE 알고리즘과 다른 알고리즘을 동시에 이용하는 SMOTE+ENN 중에는 별다른 차이가 있지 않았다. (성능 테스트는 Logistic Regression을 사용했으며, 무작위 오버 샘플링, 무작위 테스트 셋 분리를 적용했다.)

## UNDER SAMPLING

이번에는 다른 관점으로, Fail의 특징이 잘 드러나는 Feature들이 선별되었다 가정하언더 샘플링을 시도하려 했다. 언더 샘플링의 알고리즘 중 SVM과 원리가 비슷한 Condensed Nearest Neighbor 알고리즘을 이용하려 했으나…, 데이터가 너무 적어 중단했다.

		Time	Pass/Fail	Pass/Fail	...	F572	F577	F585
0	2008-07-19	11:55	-1	-1.0	...	0.570827	0.675213	0.145414
1	2008-07-19	12:32	-1	-1.0	...	0.512833	0.496882	0.325547
2	2008-07-19	13:17	1	-1.0	...	0.501426	0.306106	1.000000
3	2008-07-19	14:43	-1	-1.0	...	0.336027	0.465173	0.627939
4	2008-07-19	15:22	-1	-1.0	...	0.175380	0.657286	0.128990
..		...	...	...	...	...	...	...
123	2008-05-08	21:07	-1	1.0	...	0.518536	0.446422	0.531183
124	2008-05-08	21:22	-1	1.0	...	0.199144	0.412053	0.534367
125	2008-05-08	23:02	-1	1.0	...	0.529933	0.388721	0.329542
126	2008-05-08	23:06	-1	1.0	...	0.350285	0.365120	0.735002
127	2008-05-08	23:38	-1	1.0	...	0.615494	0.566798	0.650117

[128 rows x 131 columns]Traceback (most recent call last):

# Tried Additional Preprocessing

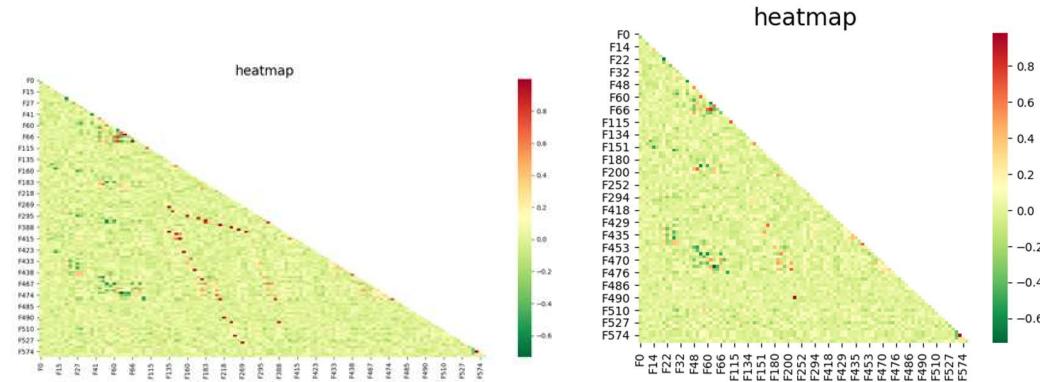
추가적으로 데이터셋의 완전성을 높이기 위해 여러가지 시도를 하였고, 실험 조건은 SMOTE(test\_size 0.2), Performance(LogisticRegression)으로 무작위 100회 평균이다.

## 스케일링에 따른 퍼포먼스 비교

RFE_MMS_ALL	0.8272	0.8061	0.8624	0.8332	0.8947
RFE_STD_ALL	0.8264	0.8065	0.8596	0.8321	0.8941
KBS_MMS_ALL	0.8112	0.7843	0.8595	0.8201	0.8804
KBS_STD_ALL	0.8112	0.783	0.862	0.8205	0.8817

MinMax와 Standardzation 스케일러 간 유의미한 차이가 없어 MinMax를 사용한다.

## 상관관계 제거



오버 샘플링을 진행한 상관관계 heatmap과, 상관관계와 편차에 대한 보정을 실행한 후의 heatmap이다. 비교적 높아 보이는 일부 Feature들이 제거된 모습을 보여주고 있고 균일한 상관관계에 가까워 보여 더 좋은 성능을 기대했었다.

RFE_MMS	0.8169	0.7958	0.8535	0.8235	0.8907

그러나 성능 평가 결과 오히려 성능이 감소하는 역효과가 발생하였다. Feature Selection과 Scaling을 거친 데이터에서 임의로 Feature를 제거하는 것의 문제로 추정된다.

# Prediction accuracy

## Logistic Regression

Binary Classification 모델로 사용되는 로지스틱 회귀는 회귀를 사용하여 데이터가 어느 범주 (cluster)에 해당하는 지 분류해주는 지도 학습 알고리즘이다. 정확하게는 어느 범주에 해당할 확률을 0과 1사이의 값으로 분류한다.

## Decision Tree

Decision Tree는 여러 특성을 순차적으로 적용하면서 독립 변수 공간을 분할하는 모형이다. 전체 Train 데이터를 해당 독립 변수 (Feature)의 값이 기준 값보다 낮은 데이터 그룹과 기준 값보다 높은 데이터 그룹으로 나눈다. (자식 노드 2개 생성) 각각의 노드에 대해 이 과정을 반복하여 완전 이진 트리를 형성한다. Leaf node에는 확률 정보가 있다.

## Random Forest

Decision Tree와 유사한 알고리즘이나, 여러 개의 Tree를 생성하여 각 Tree들의 예측을 종합하여 결론을 내리는, 앙상블 느낌의 알고리즘이다. 각 Tree들이 서로 모두 다른 구조를 가지고 있고, 데이터를 복원 샘플링을 통해 모델링을 하는 Bagging을 사용하여 최종 모델을 생성하기 때문에 예측 성능 향상에 도움을 준다.

## Boosting – Gradient Boosting Algorithm (GBM)

앙상블에는 Bagging과 Boosting이 존재하는데, Gradient를 이용하여 Boosting을 하는 알고리즘이다. Gradient Boost는 tree의 root이 아닌, 하나의 leaf에서부터 시작하여 그 Single leaf 모델이 예측하는 확률의 평균을 구한다.

```
def Confuse_Matrix_Performance(FINAL_DF, ORIGIN_DF, n):
    #https://injo.tistory.com/13
    #http://blog.naver.com/PostView.nhn?blogId=siniphia&logNo=221396370872

    X_test = ORIGIN_DF.iloc[:, 1:]
    Y_test = ORIGIN_DF.iloc[:, 0]

    X_train = FINAL_DF.iloc[:, 1:]
    Y_train = FINAL_DF.iloc[:, 0]

    if(n==0):
        lr_clf = LogisticRegression()
    elif(n==1):
        lr_clf = DecisionTreeClassifier(max_depth=10)
    elif(n==2):
        lr_clf = RandomForestClassifier(max_depth=10, n_estimators=100, n_jobs=-1)
    else:
        lr_clf = xgboost.XGBClassifier(max_depth=10, n_estimators=100, eta=0.1)
```

자세한 내용은 코드를 참조.

## Performance Result (hyperparameters : Default)

```

LogisticRegression
[0.5591, 0.6171, 0.3114, 0.4139, 0.5591]

DecisionTree
[0.6059, 0.712, 0.3548, 0.4732, 0.6059]

RandomForest
[0.6174, 0.889, 0.2684, 0.4122, 0.6174]

XGBOOST
[0.642, 0.9371, 0.3045, 0.4597, 0.642]

```

\* 순서대로 Accuracy, Precision, Recall, F1-score, ROC\_AUC Curve

Depth나 estimator같은 하이퍼 파라미터를 임의로 변경하여 얻은 퍼포먼스 테스트 결과이다. (랜덤 100회 평균) Recall과 F1 score가 전반적으로 낮은 점은 아쉬운 점이다.

## Hyper Parameter Tuning with GridSearchCV

```

LogisticRegression
[0.6455, 0.727, 0.4659, 0.5679, 0.6455]

DecisionTree
[0.5725, 0.6561, 0.2909, 0.4008, 0.5725]

RandomForest
[0.6528, 0.8758, 0.3559, 0.5056, 0.6528]

XGBOOST
[0.6886, 0.9611, 0.3932, 0.5581, 0.6886]

```

```

def Confuse_Matrix_Performance(X_train, X_test, Y_train, Y_test, n):
    warnings.filterwarnings('ignore')

    if(n==0):
        clf = LogisticRegression(max_iter=200, penalty='l2')
    elif(n==1):
        clf = DecisionTreeClassifier(max_depth=8, criterion='entropy', min_samples_leaf=0.01, min_samples_split=6 )
    elif(n==2):
        clf = RandomForestClassifier(max_depth= 8, max_features='log2', min_samples_leaf=1, min_samples_split=2, n_estimators=100)
    else:
        clf = xgboost.XGBClassifier(colsample_bytree=0.5, gamma=0.2, learning_rate=0.15, max_depth=8, n_estimators=100, reg_alpha= 0.05, reg_lambda=0.1, subsample=0.4)

```

GridSearchCV를 통해 약 4시간동안 하이퍼파라미터 튜닝을 진행하였다. 방식은 값을 조금씩 바꾸어 가면서 score 목표인 f1-score가 높은 쪽으로 유도했으며, 오버 피팅을 방지하기 위해 max\_depth나 estimator등의 일부 하이퍼파라미터는 특정 수치 이하로 제한했다.

결과적으로 Decision Tree의 튜닝은 성공적이지 못했고, f1-score 는 전반적으로 상승한 결과를 얻을 수 있었다.

# Abstract

<< 업데이트 0.2.3 >>

이번 프로젝트를 통해서 데이터 전처리와 모델 학습, 그리고 모델 튜닝까지 전반적인 머신 러닝 개발에 대해 심도 있는 경험을 할 수 있었다. 비록, 좋은 성능의 모델을 만들지는 못했지만 그 과정속에서 얻은 경험들을 토대로 종강 이후 성능 향상을 위한 버전업 업데이트를 계획하고 있다.

프로젝트의 마지막 부분인 모델 학습, 하이퍼 파라미터 튜닝에 있어서 미숙함을 보였다. 모델 학습에 있어 데이터를 제대로 분리하지 못하기도 하고, 오버샘플링에 있어서 SMOTE 와 SMOTENNN 을 반대로 적용하기도 하였다. 또한 GridSearchCV 라는 좋은 Scikit-Learn 의 함수를 하이퍼 파라미터 개념 부족 등의 이유로 제대로 사용하지 못한 점이 매우 아쉬웠다. 아쉬워서 다시 도전한 결과 **약간의 성능 향상을 얻을 수 있었다.**

그렇지만 Pass, Fail 데이터를 나누어 진행하며 시각화를 통해 데이터 전처리에 있어 좋은 방향성을 얻을 수 있었고 Feature Selection 에 있어 좋은 결과가 있었던 것 같다. 그리고 100 회 내지 1000 회의 연산을 통해 CPU 를 혹사 시켜 가며 랜덤한 테스트 셋 분리, 랜덤한 오버샘플링, 랜덤한 결과를 평균 내어 결과값과 알고리즘 비교를 위한 계산을 하였으나, 과한 반복 연산이었던 것 같다.

마지막으로, 우리의 Git Repository 와 혹사당한 CPU 로 보고서를 마무리한다.



# References

(Special Thanks to Prof. and <https://scikit-learn.org/stable/modules>)

## [Seaborn]

<https://www.python-graph-gallery.com/92-control-color-in-seaborn-heatmaps>

## [Sampling on imbalanced-data]

<https://datascienceschool.net/03%20machine%20learning/14.02%20%EB%B9%84%EB%8C%80%EC%B9%AD%20%EB%8D%B0%EC%9D%B4%ED%84%B0%20%EB%AC%B8%EC%A0%9C.html>

[https://mkjjo.github.io/python/2019/01/04/smote\\_duplicate.html](https://mkjjo.github.io/python/2019/01/04/smote_duplicate.html)

<https://dining-developer.tistory.com/18?category=936702>

<https://muzukphysics.tistory.com/124>

<https://hwidoc.tistory.com/entry/%EC%96%B8%EB%8D%94-%EC%83%98%ED%94%8C%EB%A7%81Undersampling%EA%B3%BC-%EC%98%A4%EB%B2%84-%EC%83%98%ED%94%8C%EB%A7%81Oversampling>

## [Modeling and performance]

[https://dschloe.github.io/python/python\\_edu/04\\_machinelearning/chapter\\_4\\_3\\_regression\\_tree/](https://dschloe.github.io/python/python_edu/04_machinelearning/chapter_4_3_regression_tree/)

<https://velog.io/@fiifa92/%EC%95%99%EC%83%81%EB%B8%94Ensemble-%EA%B8%B0%EB%B2%95>

<https://riverzayden.tistory.com/14>

<https://junklee.tistory.com/8, https://wikidocs.net/89514, https://injo.tistory.com/44>

<https://programmers.co.kr/learn/courses/21/lessons/944>

<https://teddylee777.github.io/scikit-learn/grid-search-%EB%A1%9C-hyperparameter%EC%B5%9C%EC%A0%81%ED%99%94>

<https://3months.tistory.com/368>

<https://bkshin.tistory.com/entry/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-15-Gradient-Boost>

## [GIT\_LFS]

<https://artiper.tistory.com/132>

## [ETC]

<https://nittaku.tistory.com/category/%ED%95%9C%EC%9D%98%EB%8C%80%20%EC%83%9D%ED%99%9C/%E2%94%94%20%ED%86%B5%EA%B3%84%EC%97%90%20%EB%8C%80%ED%95%9C%20%EB%82%98%EC%9D%98%20%EC%A0%95%EB%A6%AC3>