2022년 제 2회 K-인공지능 제조데이터 분석 경진대회

원활한 양품 제조를 위한 용해 공정 규칙

LPK

서울과학기술대학교 데이터사이언스학과

22510112 최지현

22510107 이다인

CONTENT

문제 정의

- 공정 개요
  - 용해 공정이란?
    - 분말-액상 원재료 또는 액상-액상 원재료를 탱크에서 혼합하는 공정이다.
    - 식품가공, 제약, 화학, 화장품 등의 업종에서 다양하게 운용된다.
  - 본 과제의 용해 공정은 식품제조업의 용해 공정에 속한다.

  - 용해 공정의 품질
    - 용질과 용매의 화학적 특징, 용질과 용매의 상대적 용량, 용해 온도, 교반 속도 등에 영향을 받는다.

  - 용해 공정은 완제품의 품질을 결정할 뿐만 아니라, 완제품의 단가에도 큰 영향을 미친다.

    ⇒ 효율적으로 제어하고 최적화 해야 한다.

문제 정의

- 이슈 사항
  - 용해 공정은 원재료 전처리 작업의 첫 단계
    - 본 공정의 결과물이 이후 공정의 품질에 영향을 많이 미친다.
  - 용해 공정에서 다양한 원료가 균일하게 혼합되는 것이 매우 중요하다.

  - 용해 품질은 다양한 요인들에 의해 영향을 받는다.
    - 현장 작업자들은 경험이나 암묵지에 의존하여 문제를 해결할 수 밖에 없다.

  - 용해 공정이 진행되면서 현장 작업자는 중간에 직접 용해 상태를 확인하게 된다.
    - 현장 상황에 따라 확인하는 것이 불가능할 때도 있다.
    - 따라서 공정이 완료된 후 에야 품질을 확인하게 되는 경우가 많다.

문제 정의

- 문제 해결
  - 용해 공정이 진행중 연속적으로 변화하는 요인들 즉, 용해 온도나 교반 속도, 내용량 등의 데이터를 이용한다.
    - 사람에 의한 주관적인 판단이 아닌 데이터에 기반한 예측
      - 훨씬 편리하고 불량의 생산품을 만들어내는 확률도 줄어들 것이다.

  - **본 문제에 대해 우리는 양품과 불량의 규칙에 대한 가이드라인을 제시할 것이다.**

  - 용해 탱크 품질에 영향을 미치는 요인들이 어떤 조건을 가져야 공정 품질이 양품인지 불량인지에 대한 명확한 가이드라인이 있다면 효율적인 생산량을 기반으로 설비 운영비 및 인건비에서 비용 절감 효과를 얻을 수 있을 것이다.

  - 양품이 불량으로 분류된다면 용해 공정 이후로 넘어가지 않아 자원 낭비의 문제가 될 수 있다.
  - 더 큰 문제는 **불량을 양품으로 분류하는 것**이다.
    - 불량을 양품으로 분류하게 된다면 최종 생산품의 품질에 좋지 않은 영향을 미칠 것이다.
    - **불량을 양품으로 분류하는 비율이 가장 작아지도록 하는 것을 모델의 성능으로 판단했다.**

제조 데이터

- 총 835,200개의 rows × 7개의 columns
- 2020/03/04 ~ 2020/04/30 까지 수집한 시계열 데이터

STD_DT	NUM	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	INSP	TAG
2020/03/04 0:00	0	489	116	631	3.19	OK
2020/03/04 0:00	1	433	78	609	3.19	OK
2020/03/04 0:00	2	464	154	608	3.19	OK
2020/03/04 0:00	3	379	212	606	3.19	OK
…	…	…	…	…	…	…

제조 데이터

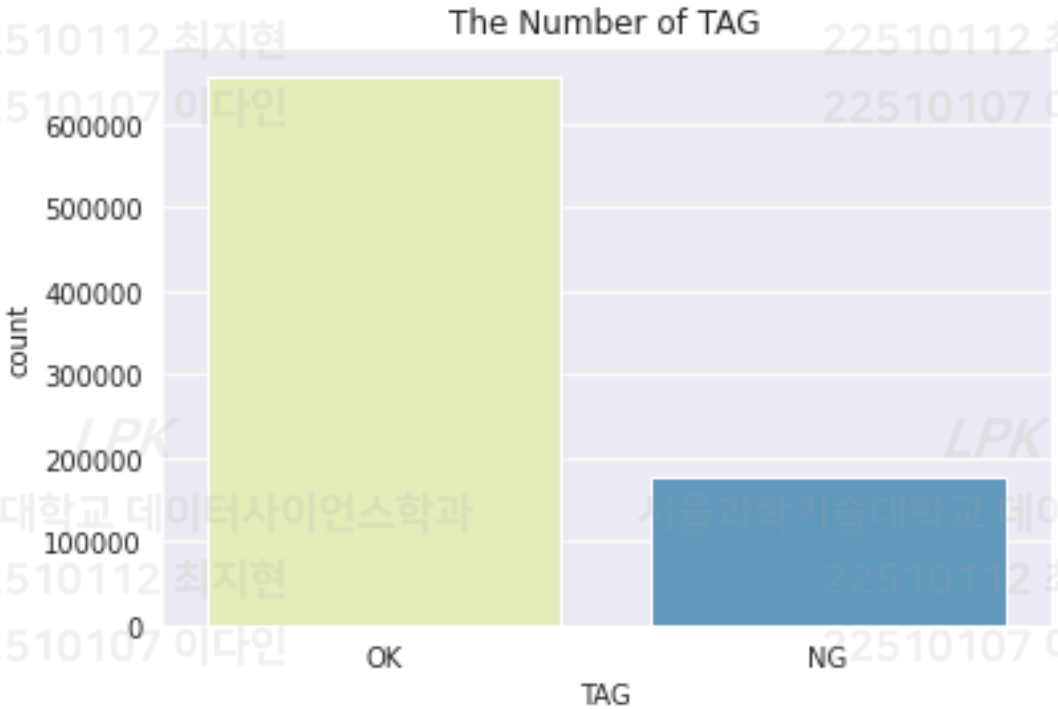
- 7개의 columns

변수 명	설명	타입
STD_DT	날짜와 시간 (YYYY-MM-DD HH:MM:SS)	Object
NUM	인덱스	Int
MELT_TEMP	용해 온도	Int
MOTORSPEED	용해 교반 속도	Int
MELT_WEIGHT	용해 탱크 내용량	Int
INSP	생산품의 수분 함유량	Float
TAG	불량(NG), 양품(OK)	object

제조 데이터

- 설명변수 : MELT_TEMP, MOTORSPEED, MELT_WEIGHT, INSP
- 목표변수 : TAG

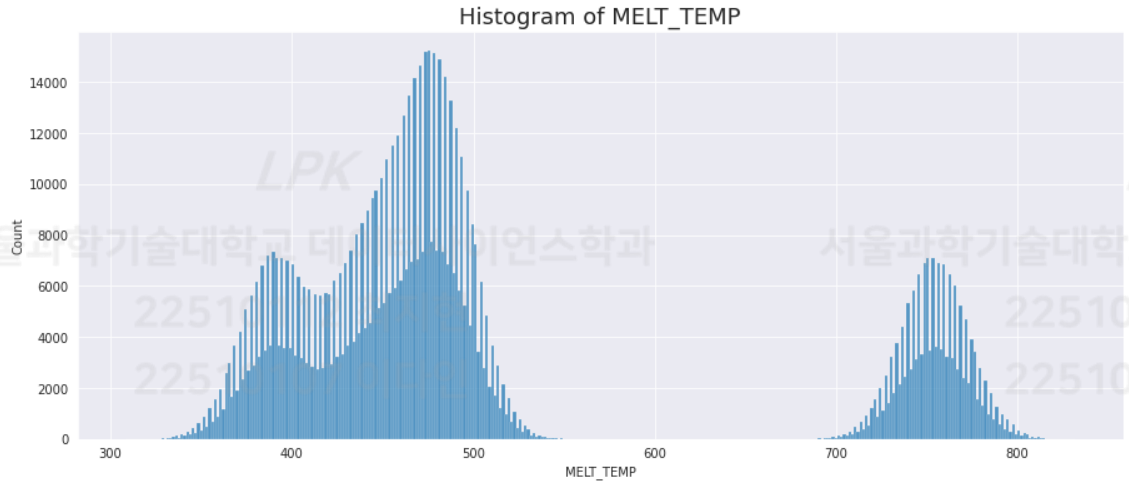
- 1(OK)과 0(NG)의 개수 확인
  - 양품(OK) : 658,133개, 불량(NG) : 177,067개 → 불균형 데이터
    - Accuracy만 성능지표로 쓰는 것은 바람직하지 않다.

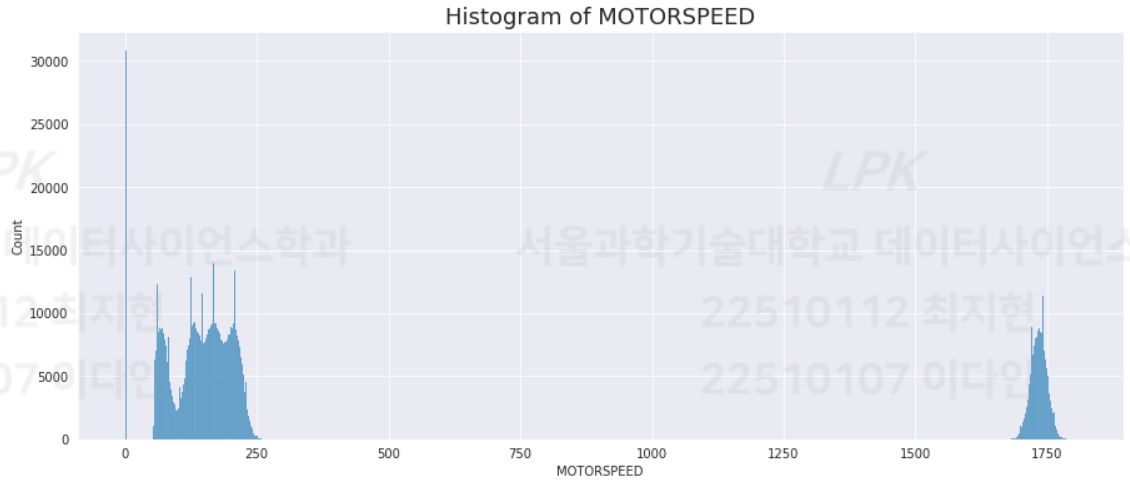


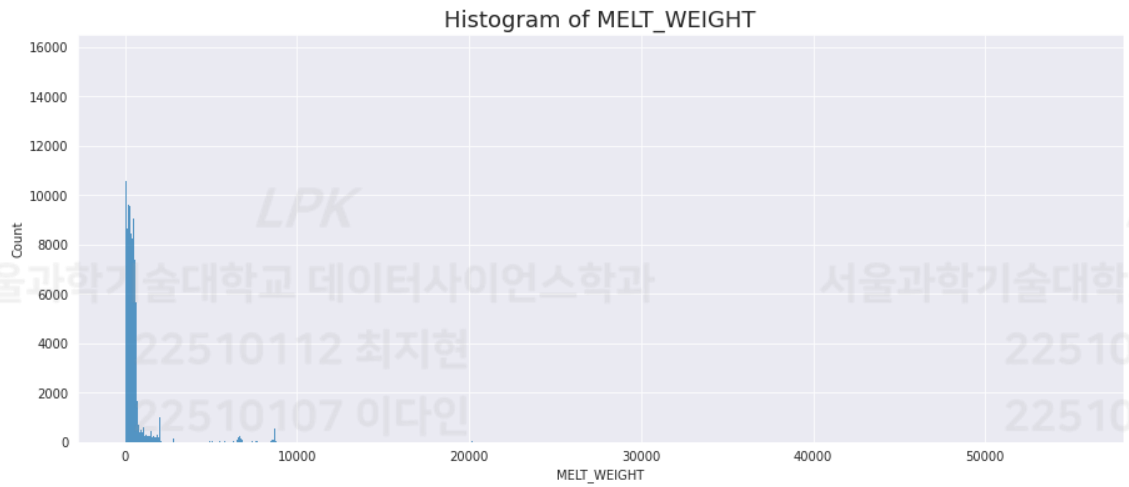
Raw data

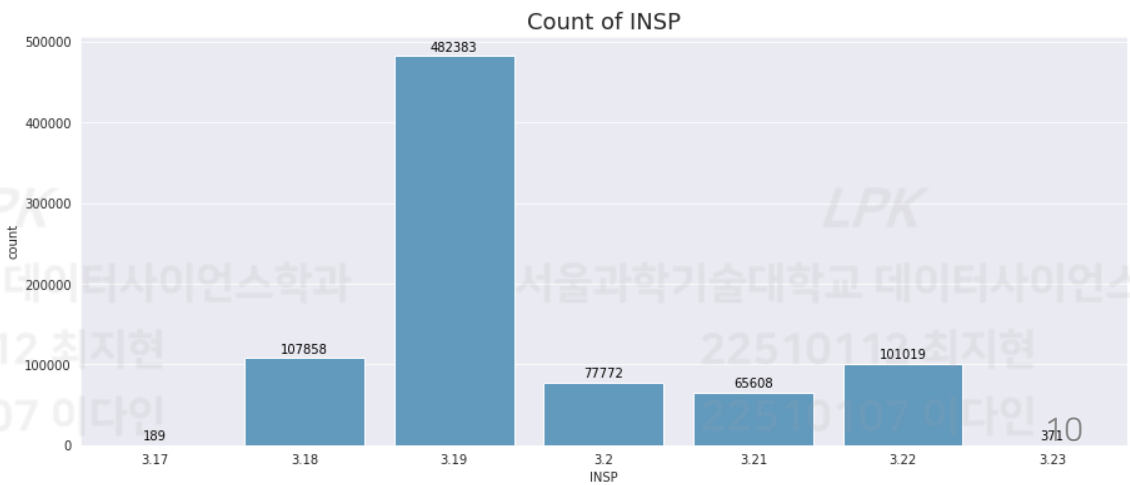
EDA

- Histogram
  - MELT_TEMP와 MOTOR SPEED는 유사한 분포를 보인다. → 주로 양 극단에 분포
  - INSP는 3.19 인 경우가 가장 많다.









EDA

- Boxplot
  - MELT_WEIGHT의 경우 histogram으로는 분포가 잘 보이지 않아 boxplot을 그려보았다.

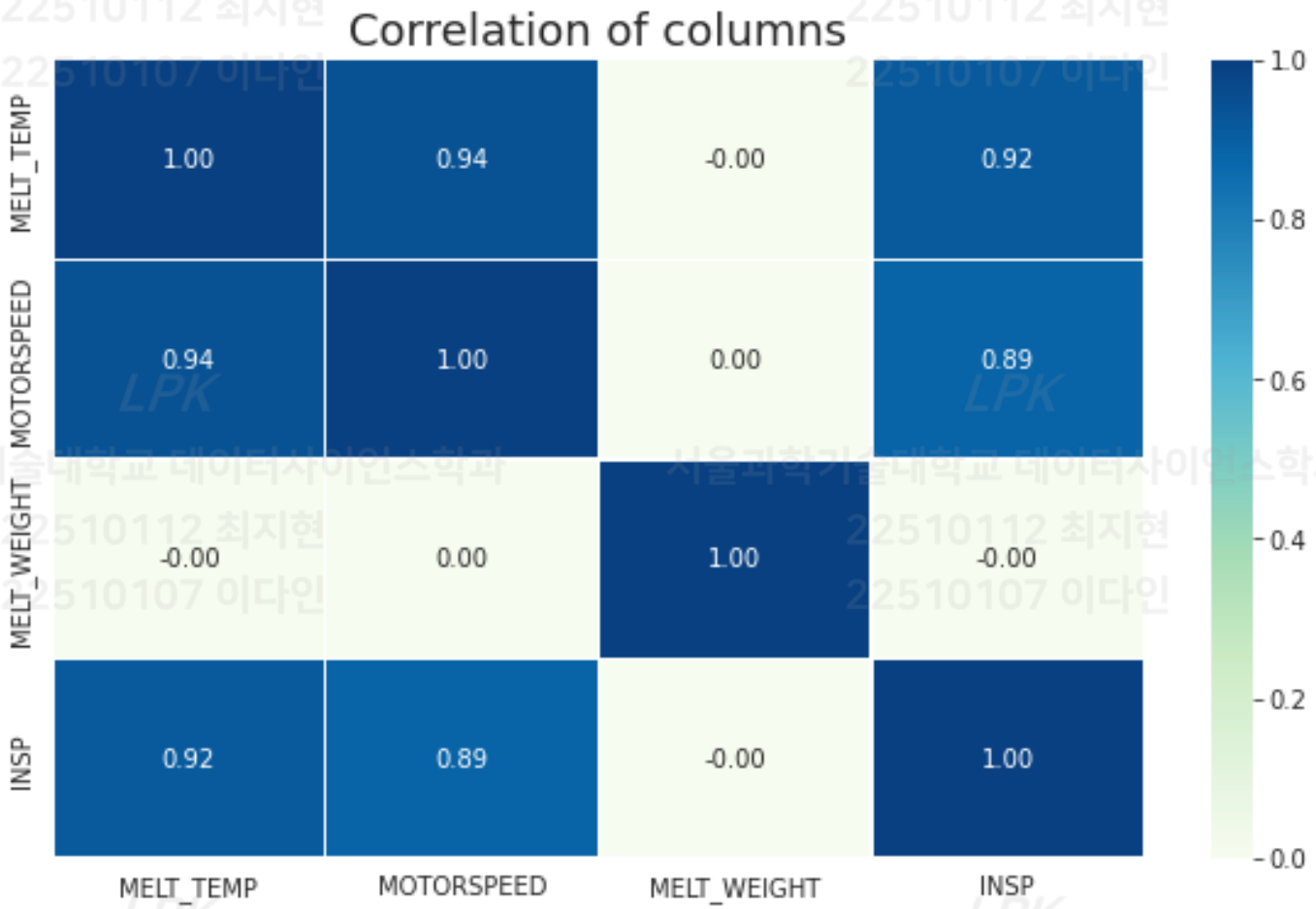
    → 분포가 넓게 퍼져 있지만 데이터는 주로 하단에 몰려 있다.

EDA

- Heatmap
  - 변수들 간의 correlation을 이용한 heatmap

  - 일반적으로 MOTOR SPEED가 빨라질 수록 MELT_TEMP가 높아진다는 생각을 할 수 있다.
    → 실제로 상관계수가 매우 높다. (0.94)

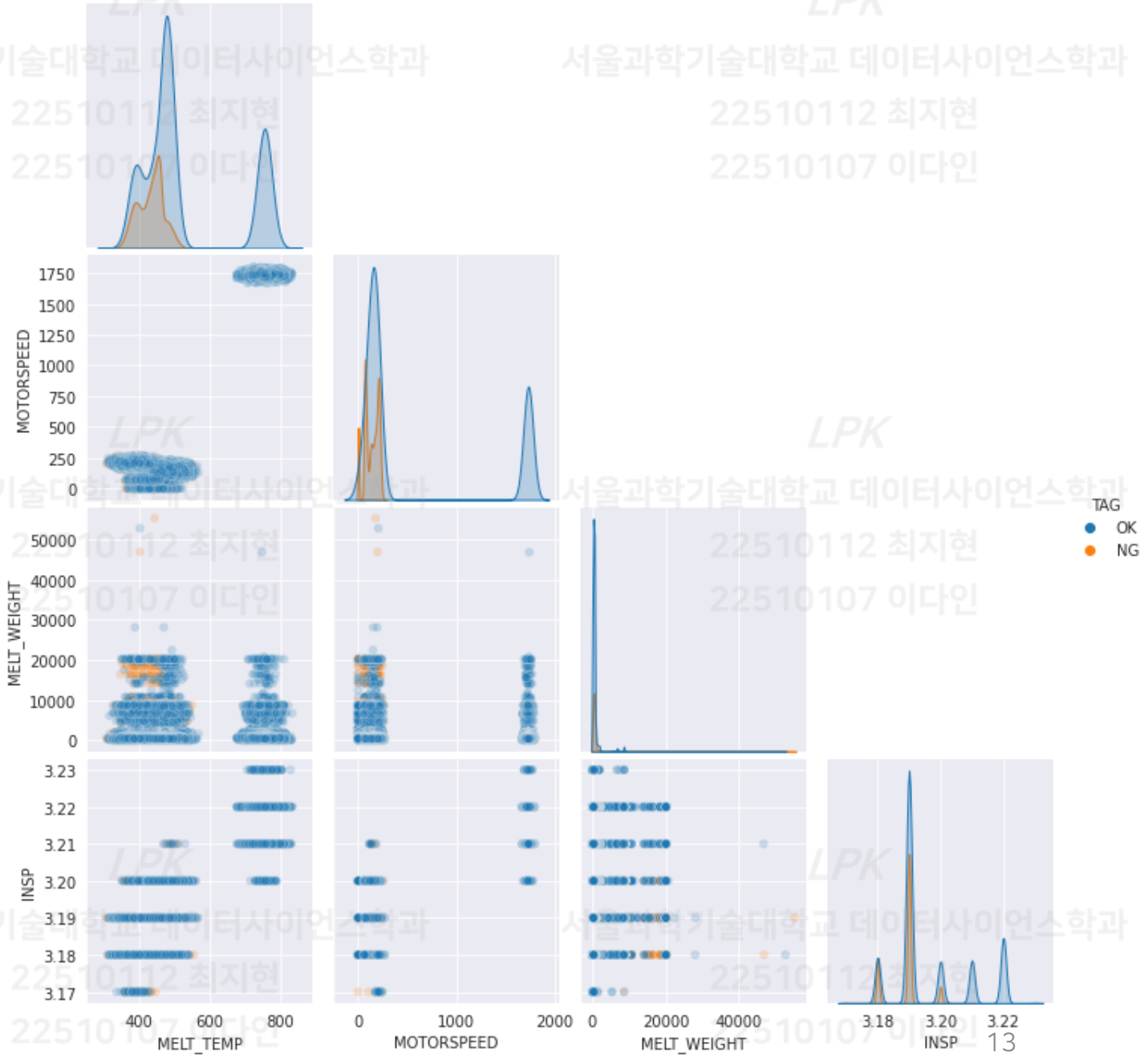
  - MELT_WEIGHT는 다른 변수들과 상관성이 없다. (0.00)



EDA

- Pair plot
  - TAG를 나눈 변수들 간의 pair plot
  - 대체적으로 0(NG, 주황색)이 각 변수들 에서 값이 적은 곳에 위치 한다.

→ 따라서 TEMP도 낮고, SPEED도 낮고, WEIGHT도 낮으면 불량으로 판단할 수 있지 않을까 예측



Machine Learning

- ML(머신러닝)과 DL(딥러닝)을 함께 분석하는 이유 : 일반적으로 성능이 좋은 DL은 설명력이 떨어지기 때문에 그 부분을 보완하기 위해 ML을 함께 분석했다.

- Decision Tree / Ada Boost / Gradient Boost / Random Forest

- 세팅
  - 4가지 기법 모두 train, test를 7:3으로 분리

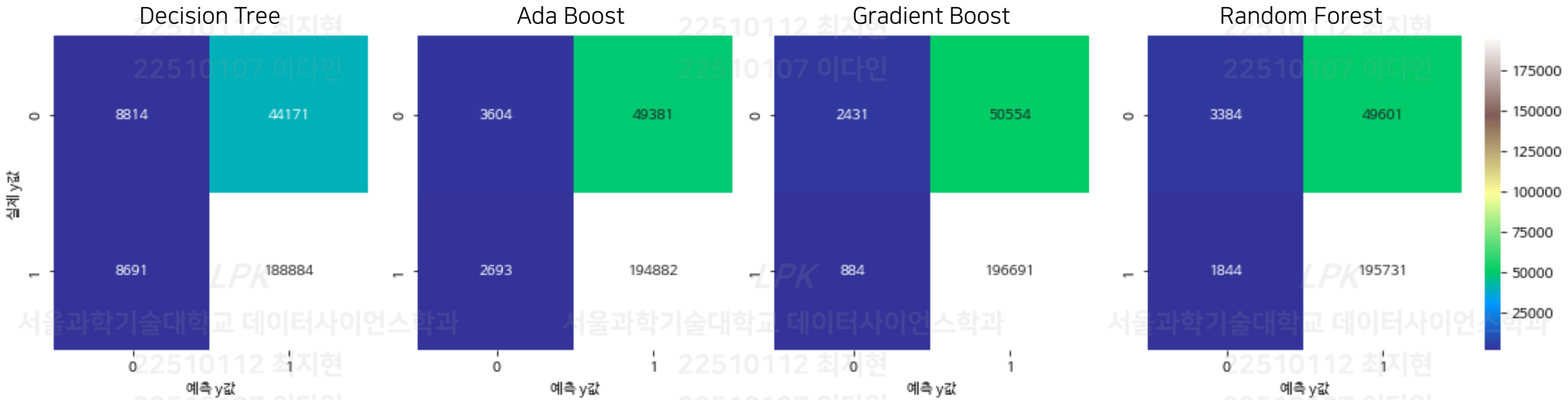
- 성능지표
  - Accuracy, F1-Score, Precision, Recall

불량을 양품으로 분류

  - 각 기법의 confusion matrix에서 **0(NG)을 1(OK)로 분류**하는 비율이 적은 모델을 찾으려고 했음

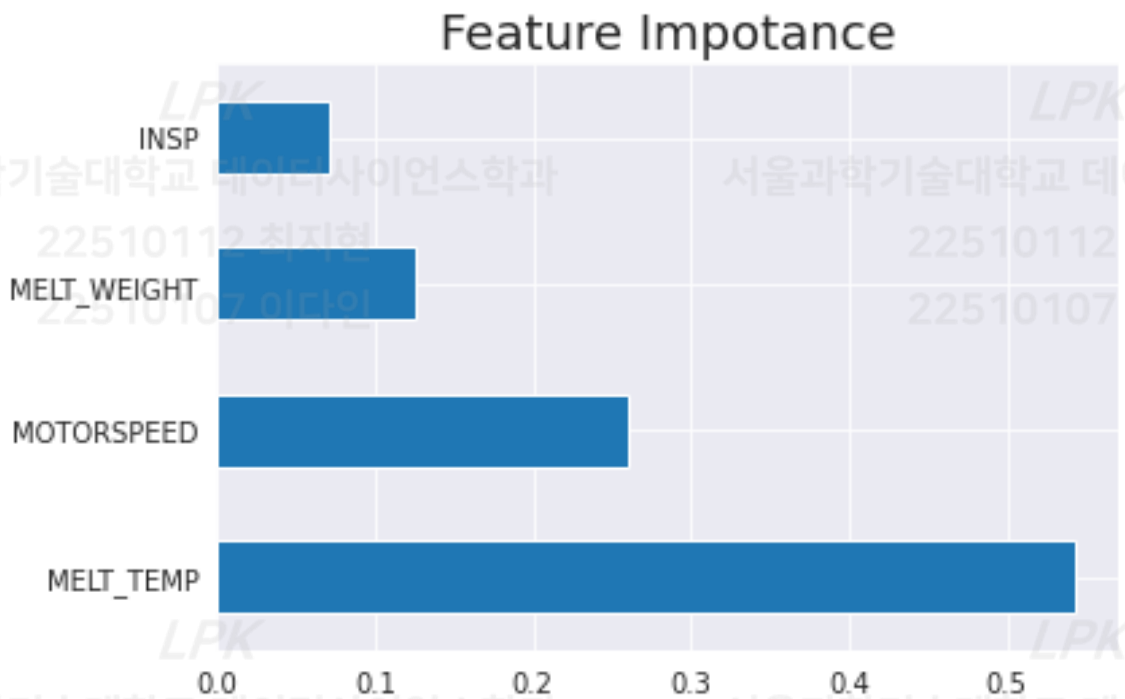
    → 이를 FP/N = (False Positive)/(Negative) 라고 명명함

Machine Learning



Model	Precision	Recall	F1-score	Accuracy	FP/N
Decision Tree	0.8105	0.9560	0.8772	0.7890	0.8337
Ada Boost	0.7978	0.9864	0.8821	0.7922	0.9320
Gradient Boost	0.7955	0.9955	0.8843	0.7947	0.9541
Random Forest	0.7978	0.9907	0.8839	0.7947	0.9361

- Random Forest
  - Raw data로 Random Forest 수행 후 Feature Importance를 확인하면 INSP가 가장 낮은 수치를 보인다.
  - 따라서 목표변수에 중요하지 않은 변수라고 판단했다. → INSP 컬럼 제거



Preprocessing

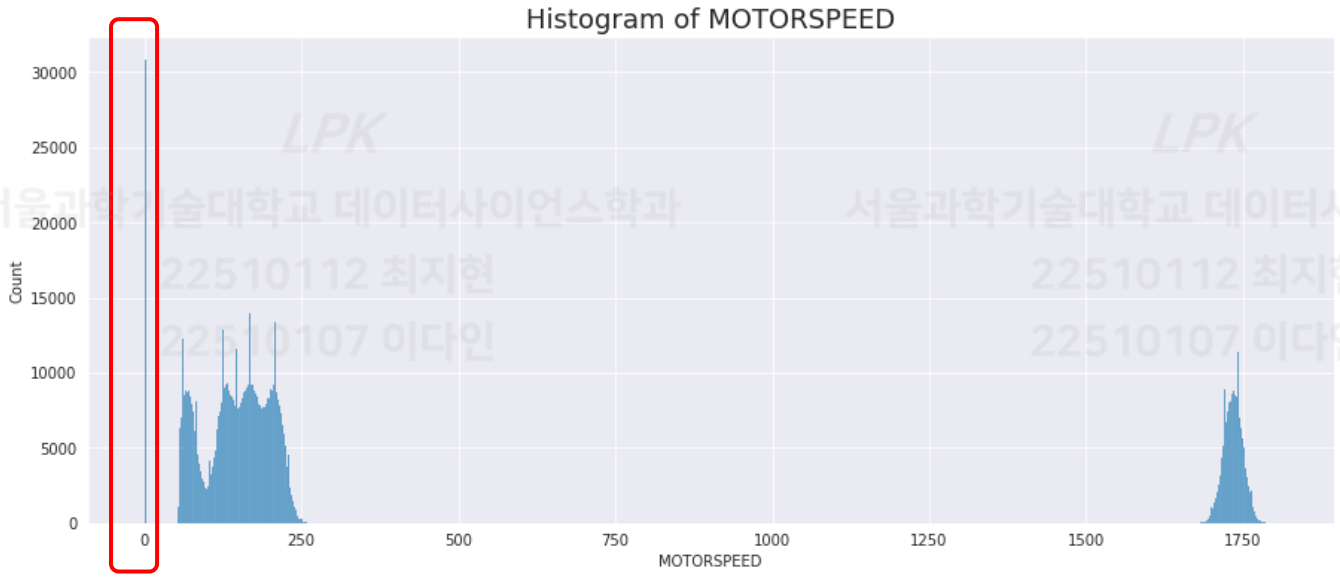
- STD_DT의 데이터 타입을 object에서 timestamp로 변경하고, 시계열 분석을 위해 인덱스로 지정했다.

	STD_DT	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	INSP	TAG
0	2020-03-04 0:00	489	116	631	3.19	OK
1	2020-03-04 0:00	433	78	609	3.19	OK
2	2020-03-04 0:00	464	154	608	3.19	OK
3	2020-03-04 0:00	379	212	606	3.19	OK
4	2020-03-04 0:00	798	1736	604	3.21	OK

STD_DT	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	INSP	TAG
2020-03-04	489	116	631	3.19	OK
2020-03-04	433	78	609	3.19	OK
2020-03-04	464	154	608	3.19	OK
2020-03-04	379	212	606	3.19	OK
2020-03-04	798	1736	604	3.21	OK

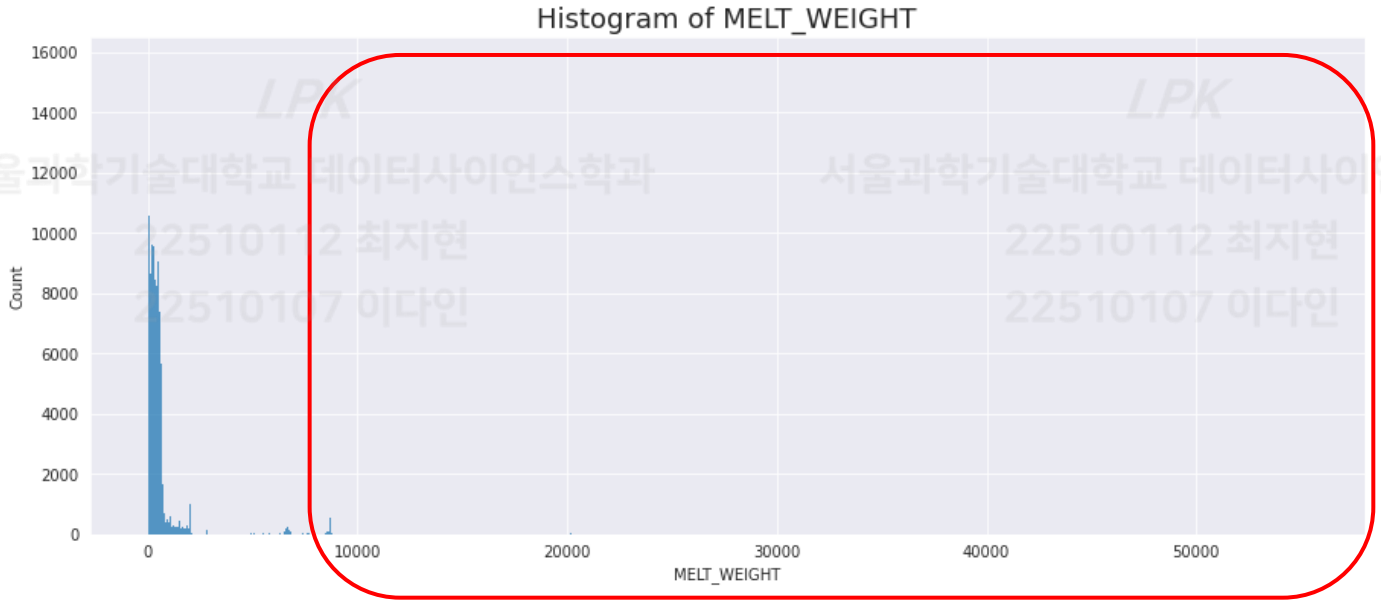
Preprocessing

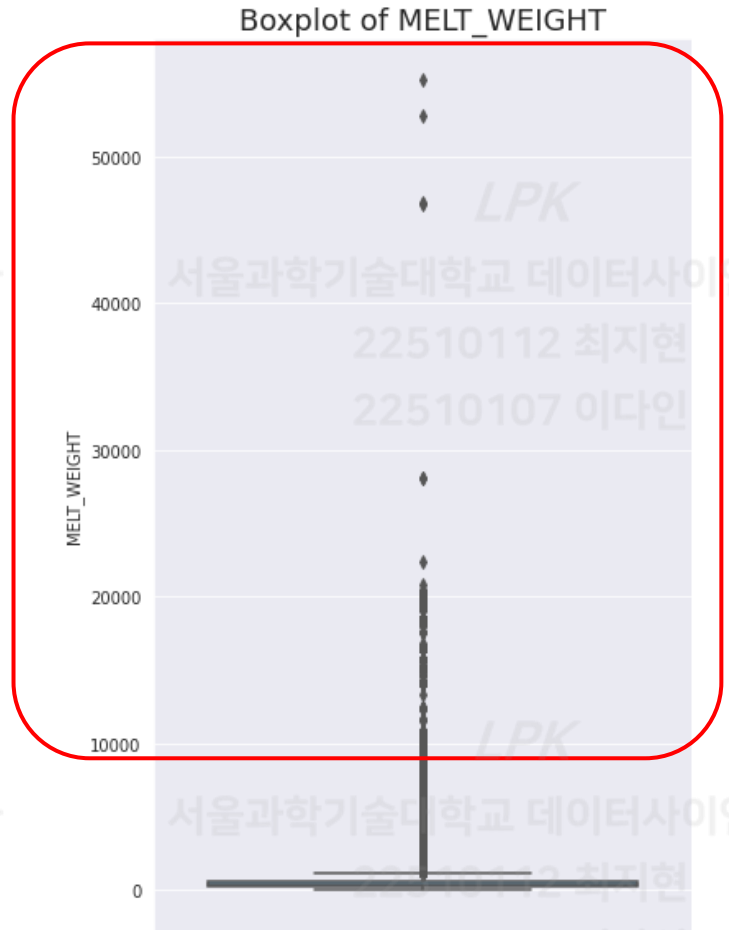
- 이상치 탐지
  - MOTOR SPEED = 0인 데이터가 굉장히 많다. → 이상치로 판단
  - 제거 후 분석을 수행했다.



Preprocessing

- 이상치 탐지
  - MELT_WEIGHT는 10,000 이상의 데이터가 별로 없다. → 이상치로 판단
  - 제거해서 분석을 수행했다.





After Preprocessing

- 전처리 과정을 거친 데이터 셋을 사용해 분석했다.
  - INSP 제거
  - MOTOR SPEED = 0 인 데이터 제거
  - MELT_WEIGHT 값이 10,000 이상인 데이터 제거

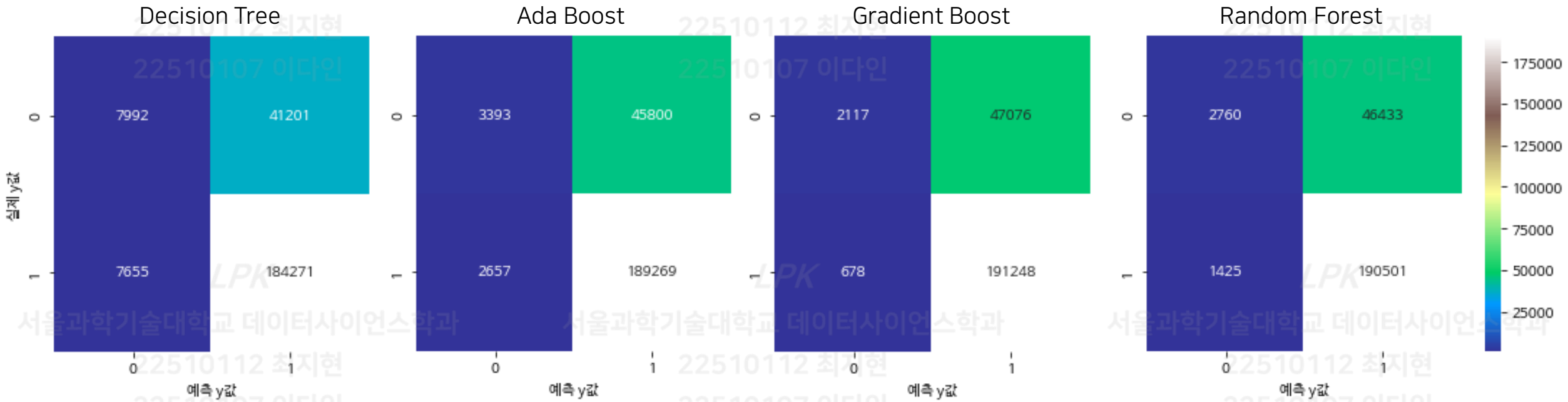
	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	INSP	TAG
0	489	116	631	3.19	1
1	433	78	609	3.19	1
2	464	154	608	3.19	1
3	379	212	606	3.19	1
4	798	1736	604	3.21	1
...	...	...	...	...	...
835195	755	1743	318	3.21	1
835196	385	206	317	3.19	1
835197	465	148	316	3.20	1
835198	467	0	314	3.19	1
835199	453	125	312	3.20	1

835200 rows × 5 columns

	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	TAG
0	489	116	631	1
1	433	78	609	1
2	464	154	608	1
3	379	212	606	1
4	798	1736	604	1
...	...	...	...	...
835194	749	1740	319	1
835195	755	1743	318	1
835196	385	206	317	1
835197	465	148	316	1
835199	453	125	312	1

803729 rows × 4 columns

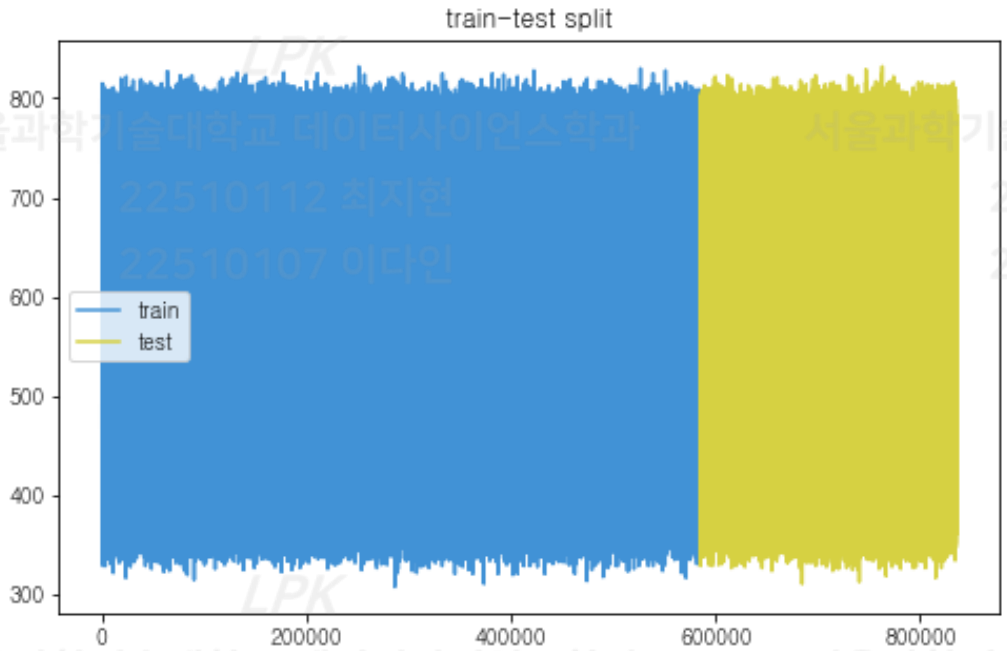
Machine Learning



Model	Precision	Recall	F1-score	Accuracy	FP/N
Decision Tree	0.8173	0.9601	0.8830	0.7974	0.8374
Ada Boost	0.8052	0.9862	0.8865	0.7990	0.9310
Gradient Boost	0.8025	0.9965	0.8890	0.8019	0.9570
Random Forest	0.8040	0.9925	0.8884	0.8015	0.9439

Deep Learning Model

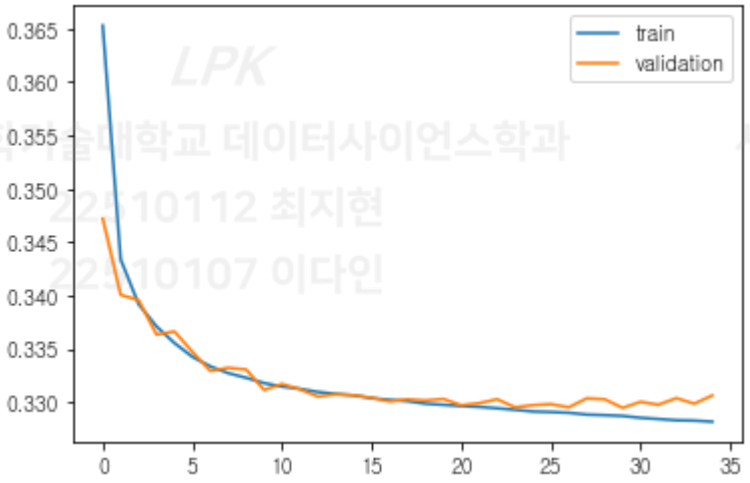
- 이상치 제거
  - MELT_WEIGHT 10000 이상인 행 제외
  - MOTOR_SPEED 0인 행 제외
- Train : Test = 70 : 30

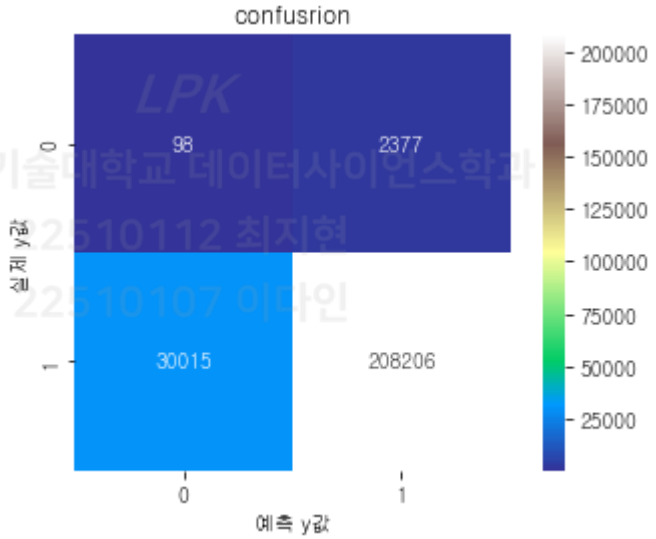


Deep Learning Model

```
Model: "sequential_23"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_24 (LSTM)              (None, 50)                10800
 dense_17 (Dense)            (None, 1)                 51
=================================================================
Total params: 10,851
Trainable params: 10,851
Non-trainable params: 0
_________________________________________________________________
```

- 단일 LSTM
  - 학습이 진행될 때 마다 loss 값이 줄어드는 모습을 확인할 수 있다.
  - confusion matrix를 보면 실제 1(OK)을 1(OK)로 잘 예측하는 모습을 보인다.
    - 실제 0(NG)을 0(NG)로 잘 분류하지 못하고 있다. ⇒ FP/N이 높다





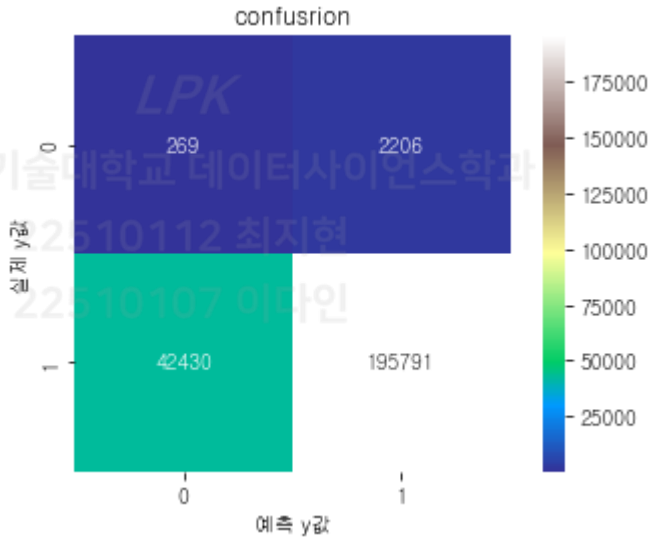
- 이에 따른 성능 지표는 다음과 같다.

성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.9887	0.8740	0.9278	0.8654	0.9404

Deep Learning Model

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_2 (GRU)                 (None, 50)                8250
 dense_2 (Dense)             (None, 1)                 51
=================================================================
Total params: 8,301
Trainable params: 8,301
Non-trainable params: 0
_________________________________________________________________
```

- 단일 GRU
  - 학습이 진행될 때 마다 loss 값이 줄어드는 모습을 확인할 수 있다.
  - confusion matrix를 보면 실제 0(NG)을 0(NG)로 단일 LSTM보다 더 잘 분류한다. ⇒ FP/N 낮아짐
    - 실제 1(OK)을 1(OK)로 분류하는 게 단일 LSTM보다 분류하지 못한다.



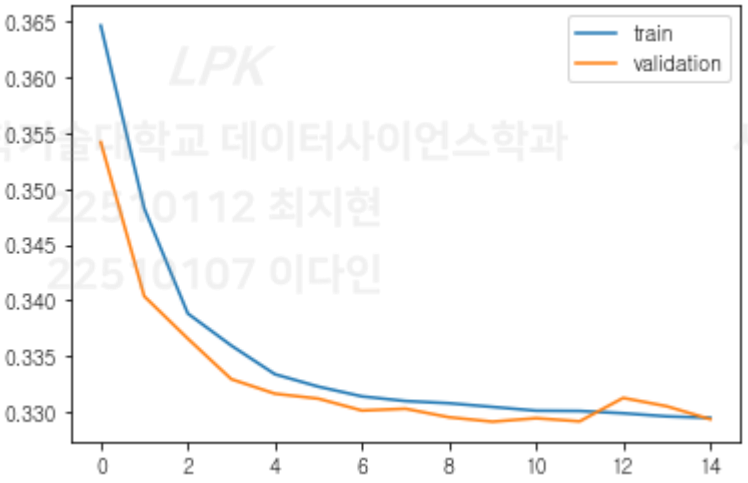
- 이에 따른 성능 지표는 다음과 같다.

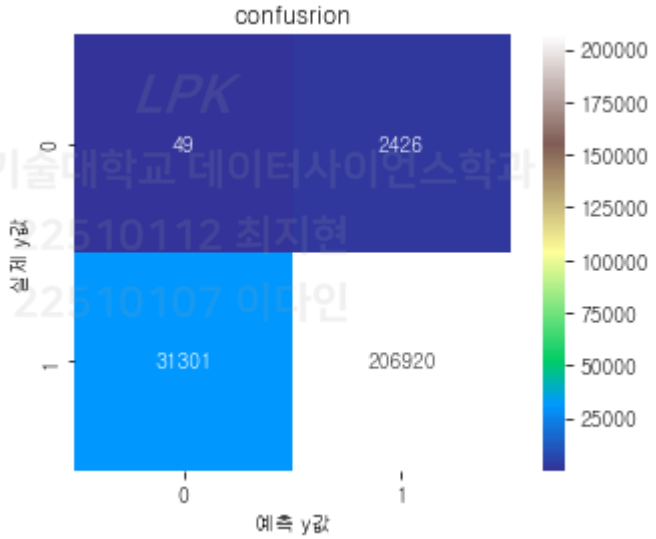
성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.9889	0.8219	0.8977	0.8146	0.8913

Deep Learning Model

- 다중 LSTM
  - 학습이 진행될 때 마다 loss 값이 줄어드는 모습을 확인할 수 있다.
  - confusion matrix를 보면 단일 LSTM과 약간 더 높은 성능을 보여준다.
  - epochs가 단일 LSTM, 단일 GRU보다 적어 모델 학습이 빨리 완료된다.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 10, 50)            10800
dropout (Dropout)            (None, 10, 50)            0
lstm_1 (LSTM)                (None, 10, 50)            20200
dropout_1 (Dropout)          (None, 10, 50)            0
lstm_2 (LSTM)                (None, 50)                20200
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 51,251
Trainable params: 51,251
Non-trainable params: 0
_________________________________________________________________
```



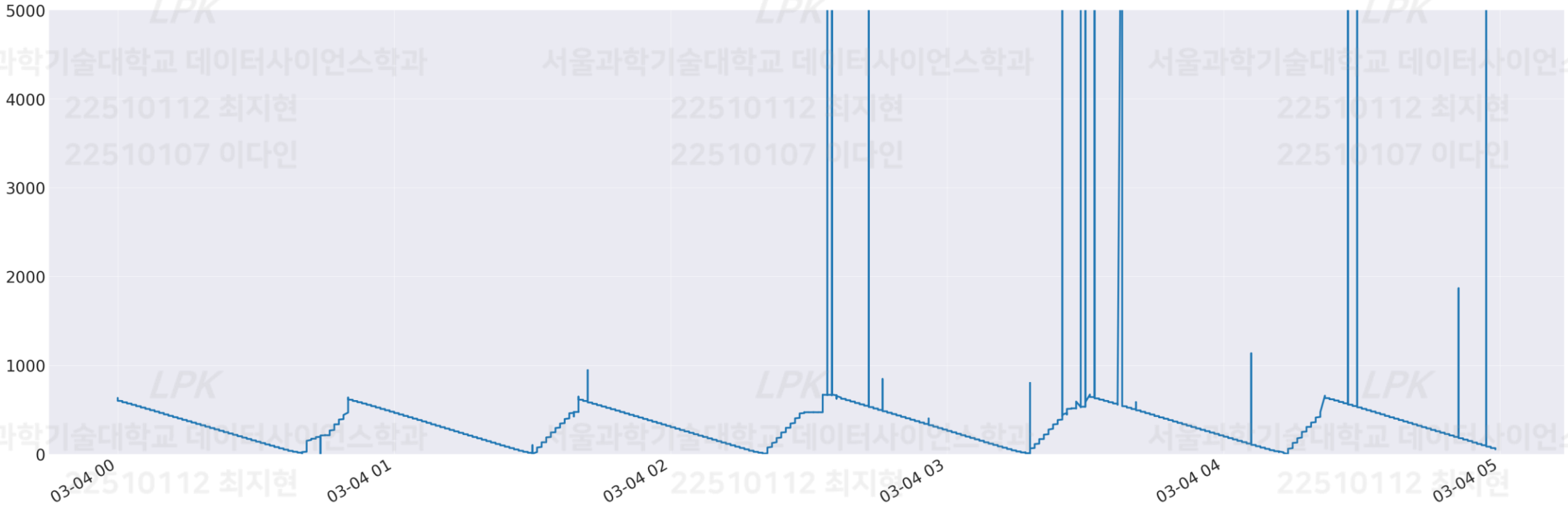


- 이에 따른 성능 지표는 다음과 같다.

성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.9887	0.8686	0.9246	0.8699	0.9802

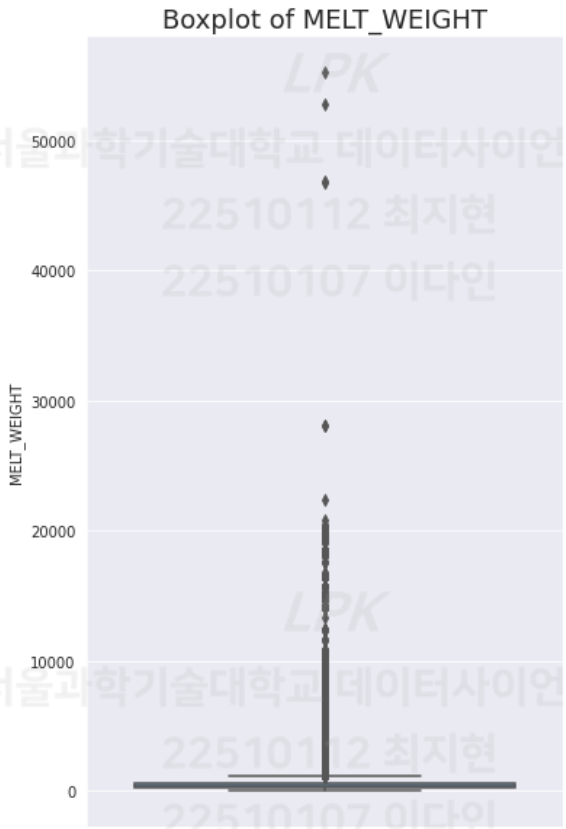
특이사항

- MELT_WEIGHT 특이점 탐지
  - MELT_WEIGHT 주기 확인해보자.
  - 시간을 X축으로 MELT_WEIGHT의 그래프를 그려보면 다음과 같은 주기를 확인할 수 있다.
  - 이전 값에 비해 10% 증가 혹은 감소하는 부분이 존재한다.



특이사항

- MELT_WEIGHT 특이점 탐지
  - MELT_WEIGHT의 중앙값(Median)과 최댓값(Max)의 차이가 굉장히 크다.
  - Boxplot을 보아도 대부분 10,000 이하에 분포하는데, 50,000 이 넘는 데이터가 존재한다.
  - 용해 탱크에도 용량이 존재할 텐데 왜 이러한 특이점이 보이는지 알아보고자 했다.

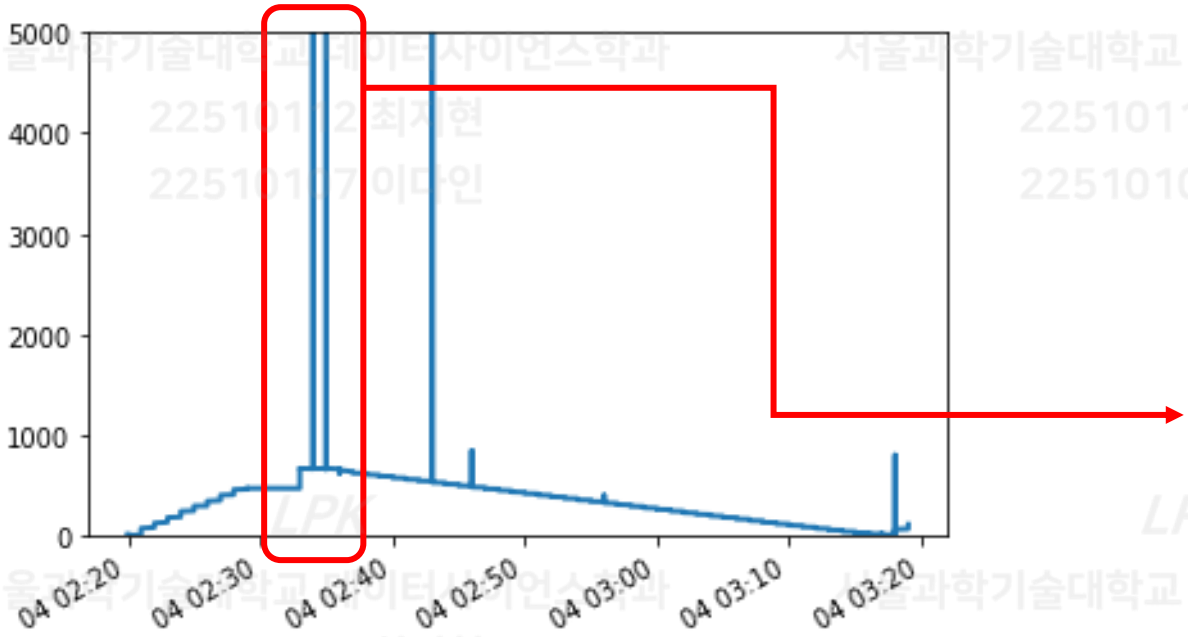


MELT_WEIGHT의 요약 통계량

	MELT_WEIGHT
Mean	582.962
Std	1217.604
Median	383
Min	0
Max	55252

특이사항

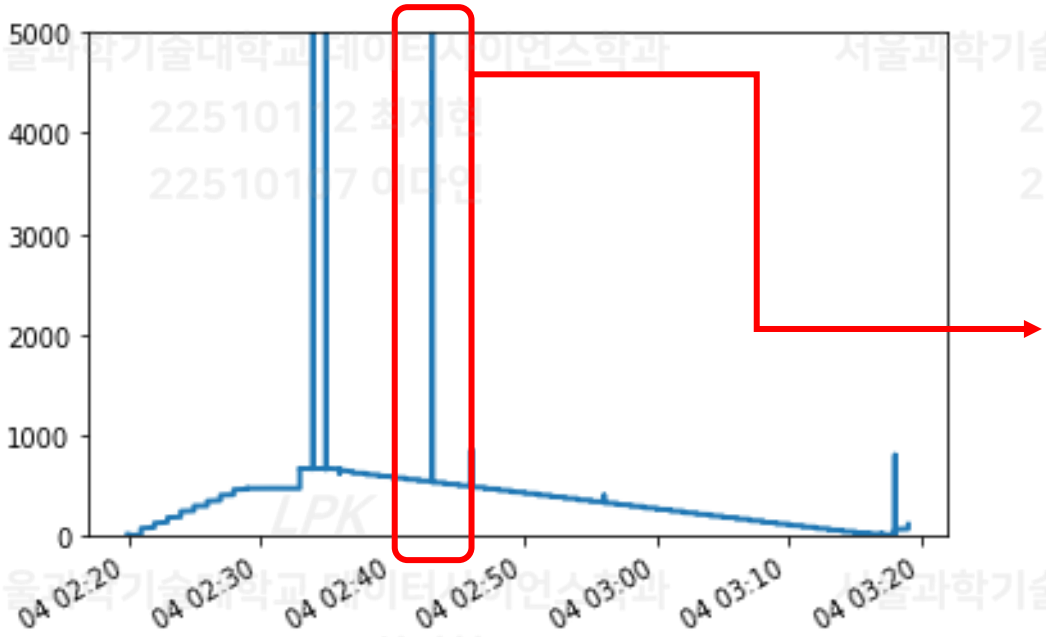
- MELT_WEIGHT 특이점 탐지
  - 2020/03/04 02:20 ~ 2020/03/04 03:20 구간만 잘라서 확인
  - 앞의 값으로 대체
  - 맨 뒤의 숫자를 지우고 앞과 뒤의 숫자 관계를 보면 이어지는 숫자 임을 확인했다.



STD_DT	NUM	MELT_TEM	MOTORSP	MELT_WEI	INSP	TAG
2020/03/04 2:35	1550	442	132	666	3.2	OK
2020/03/04 2:35	1551	416	61	665	3.19	OK
2020/03/04 2:35	1552	475	136	664	3.2	OK
2020/03/04 2:35	1553	399	230	6633	3.19	OK
2020/03/04 2:35	1554	735	1717	661	3.21	OK
2020/03/04 2:35	1555	743	1739	660	3.21	OK
2020/03/04 2:35	1556	405	191	662	3.18	OK
2020/03/04 2:35	1557	460	174	6622	3.19	OK
2020/03/04 2:35	1558	443	78	662	3.19	OK
2020/03/04 2:35	1559	466	125	668	3.19	OK
2020/03/04 2:36	1560	476	117	660	3.2	OK

특이사항

- MELT_WEIGHT 특이점 탐지
  - 2020/03/04 02:20 ~ 2020/03/04 03:20 구간만 잘라서 확인
  - 앞의 값으로 대체
  - 디지털 숫자 0을 8로 인식한 것으로 보인다.
    - 앞의 숫자와 뒤의 숫자를 보고 8을 지우면 이상치가 정상 데이터로 나온다.

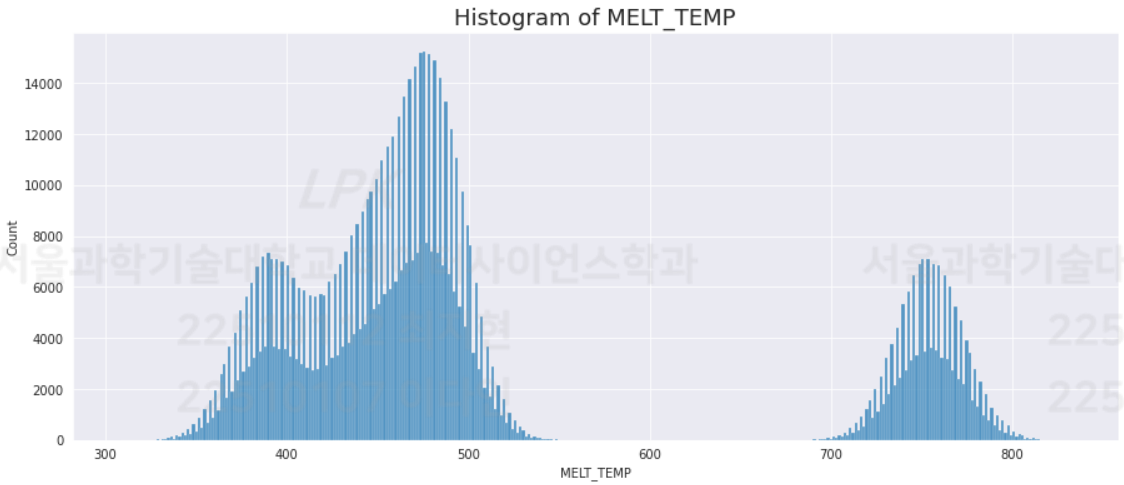


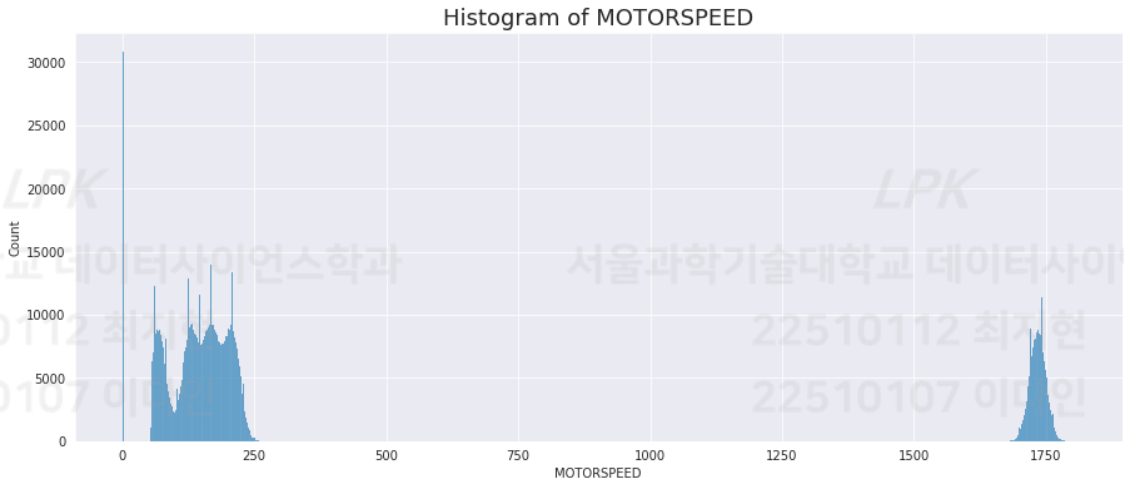
STD_DT	NUM	MELT_TEM	MOTORSP	MELT_WEI	INSP	TAG
2020/03/04 2:43	1632	490	164	539	3.19	OK
2020/03/04 2:43	1633	398	212	537	3.19	OK
2020/03/04 2:43	1634	746	1690	536	3.22	OK
2020/03/04 2:43	1635	757	1748	8535	3.21	OK
2020/03/04 2:43	1636	347	191	532	3.19	OK
2020/03/04 2:43	1637	462	158	531	3.19	OK
2020/03/04 2:43	1638	468	59	530	3.19	OK

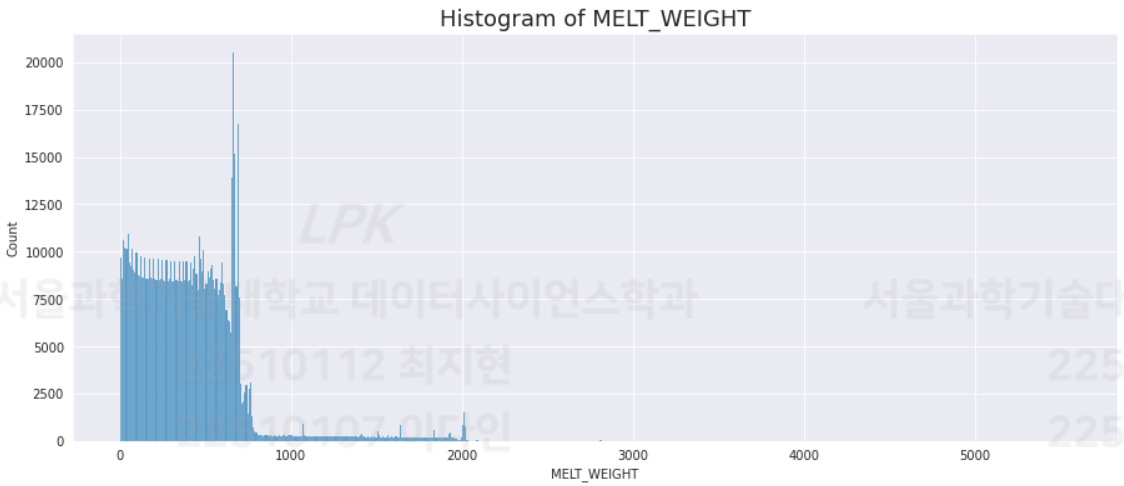
MELT_WEIGHT 수정

EDA

- Histogram
  - MELT_TEMP와 MOTOR SPEED는 수정한 부분이 없으므로 처음과 동일하다.
  - MELT_WEIGHT 수정 후, 처음 분포 보다 일관성 있게 분포하는 것을 확인할 수 있다.

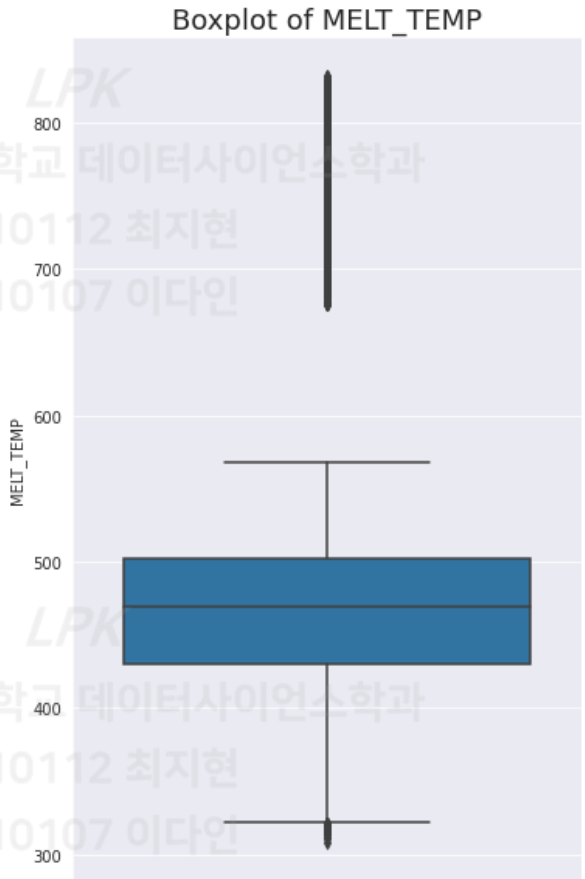


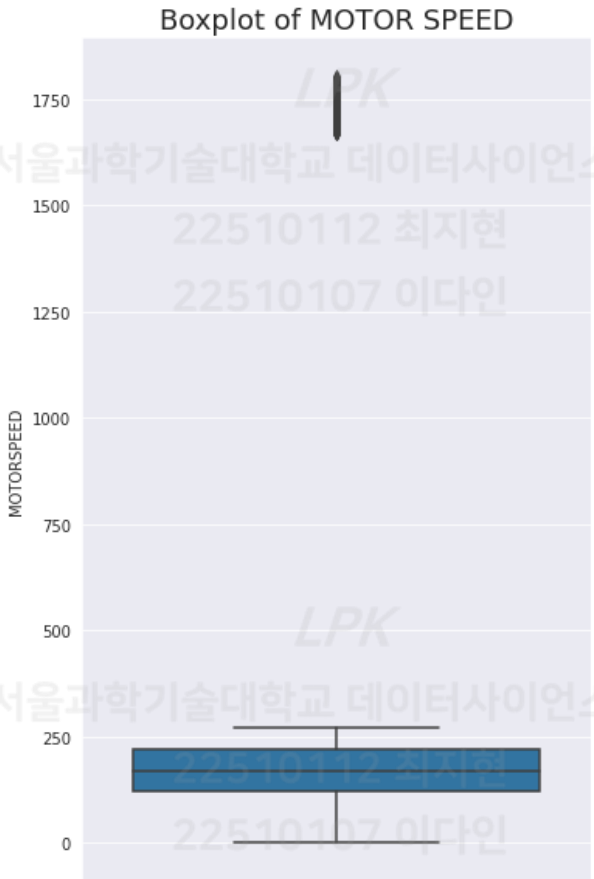


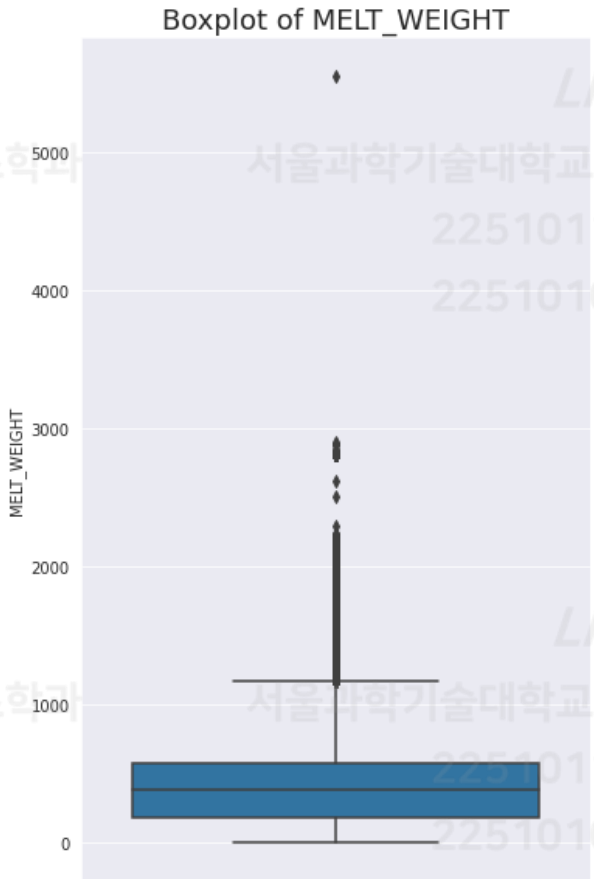


EDA

- Boxplot
  - MELT_WEIGHT 수정 후, boxplot도 마찬가지로 처음보다 데이터가 일관되게 분포하는 것을 확인할 수 있다.

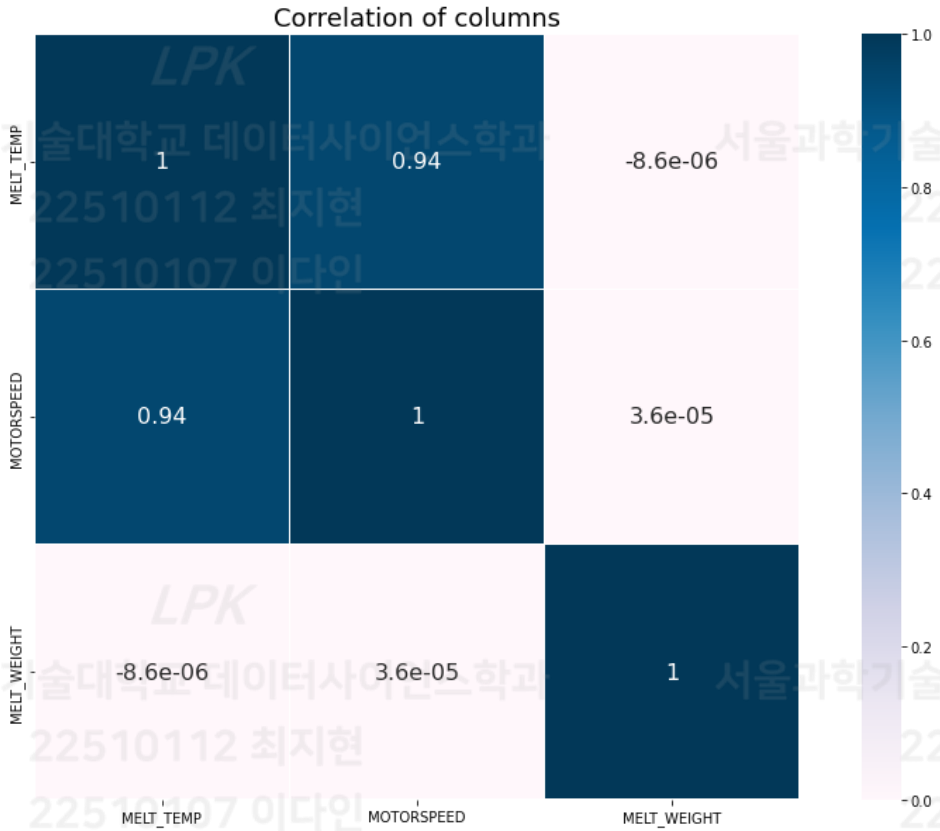






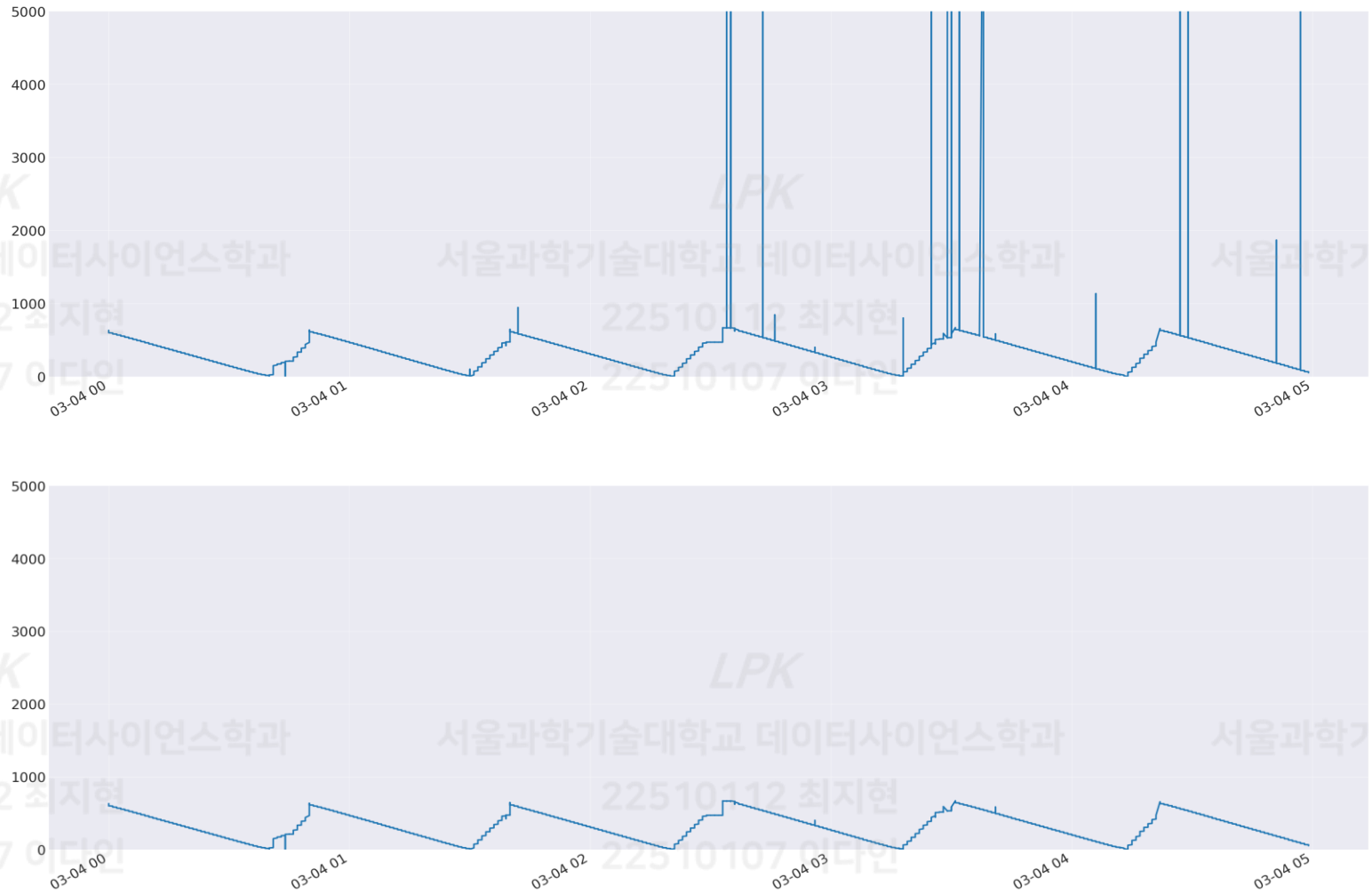
EDA

- Heatmap
  - MELT_WEIGHT 수정 후
  - MELT_WEIGHT의 상관관계를 보면
  - 아주 약간의 수가 올랐지만 여전히 다른 변수와 관계가 없다고 볼 수 있다.



EDA

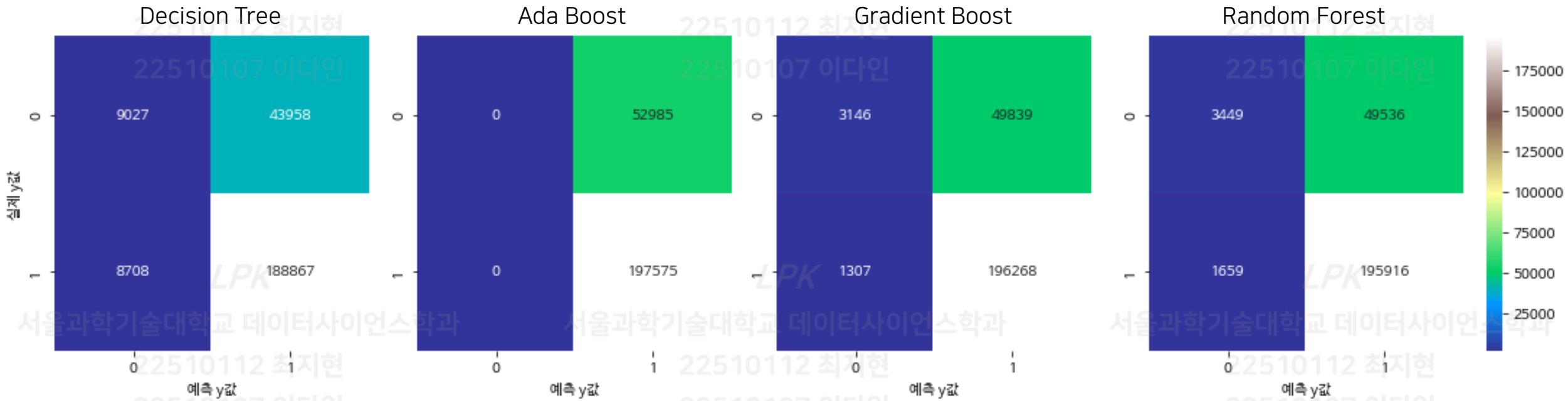
- WEIGHT 주기 확인
  - 이전 값에 비해 10% 뛰던 값이 안정화 된다.



EDA

- 불량과 양품 비교
  - 어떤 시간(혹은 날짜)에 불량(=0)이 더 많을지 확인하기 위해 시간을 X축으로 TAG에 대한 그래프를 그렸다.
  - 그래프를 보면 3월 중순 ~ 4월 중순 까지만 불량(=0, 파란색)이 존재하는 것을 알 수 있다.
  - 실제로 원데이터 csv파일을 확인하면 03/17 ~ 04/14 까지만 불량(=0)이 존재하는 것을 확인할 수 있다.

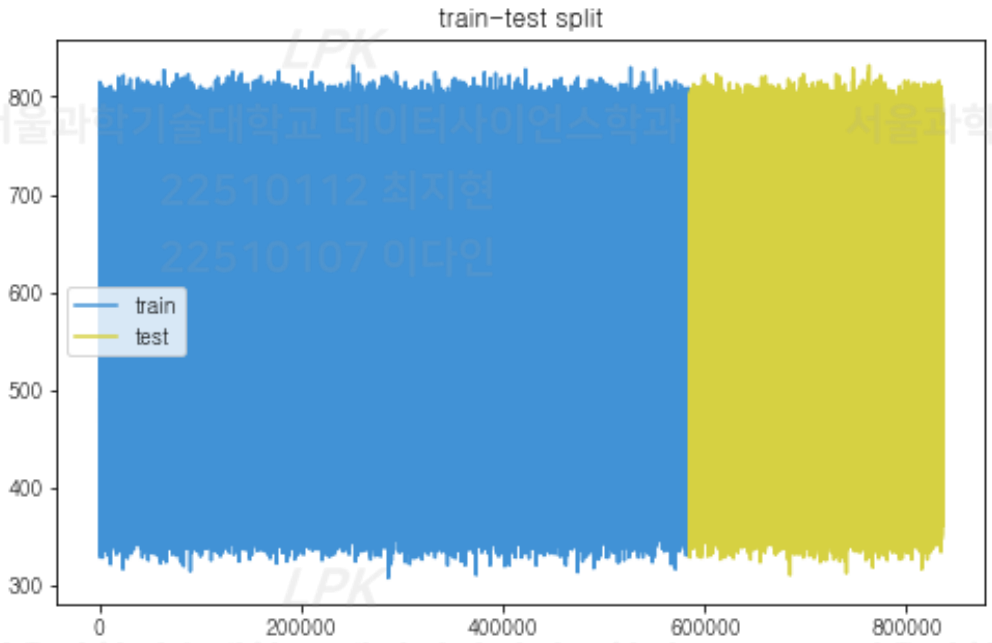
Machine Learning



Model	Precision	Recall	F1-score	Accuracy	FP/N
Decision Tree	0.7908	0.9559	0.8776	0.7898	0.8296
Ada Boost	0.7885	1	0.8818	0.7885	1
Gradient Boost	0.7975	0.9934	0.8847	0.7959	0.9406
Random Forest	0.7982	0.9916	0.8844	0.7957	0.9349

Deep Learning Model

- 이상치 제거
  - MELT_WEIGHT 이상(weird)
  - 이상치 탐지하여 제거
- Train : Test = 70 : 30

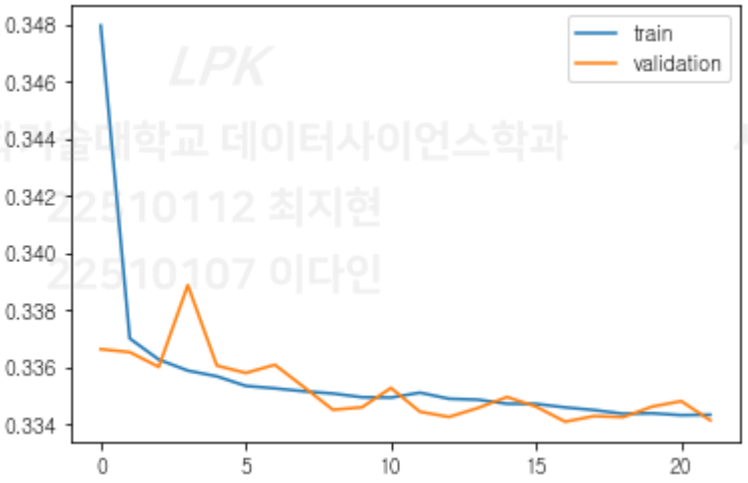


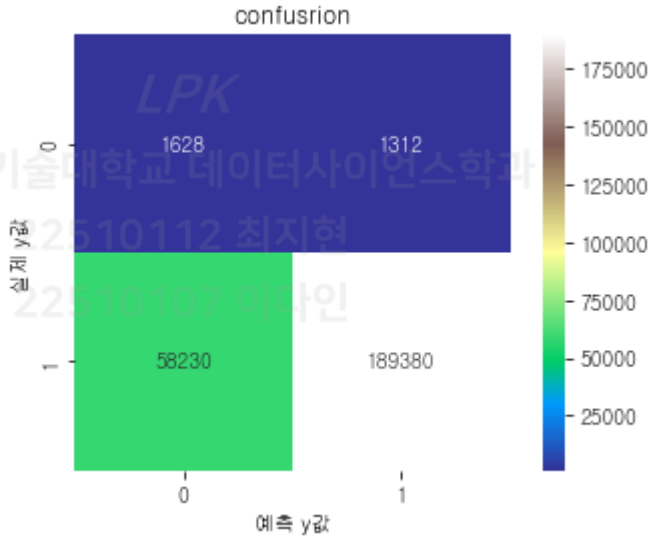
Deep Learning Model

단일LSTM : MOTOR SPEED = 0, MELT_WEIGHT는 10,000 이상의 데이터

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50)                10800
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 10,851
Trainable params: 10,851
Non-trainable params: 0
_________________________________________________________________
```

- 단일 LSTM
  - 학습이 진행될 때 마다 loss 값이 줄어드는 모습을 확인할 수 있다.
  - confusion matrix를 보면 실제 0(NG)을 0(NG)로 잘 예측하는 모습을 보인다.
    - 중량 이상치 제거 후 실제 0(NG)을 1(OK)로 예측하는 비율이 단일LSTM보다 적어진다. ⇒ FP/N 하락





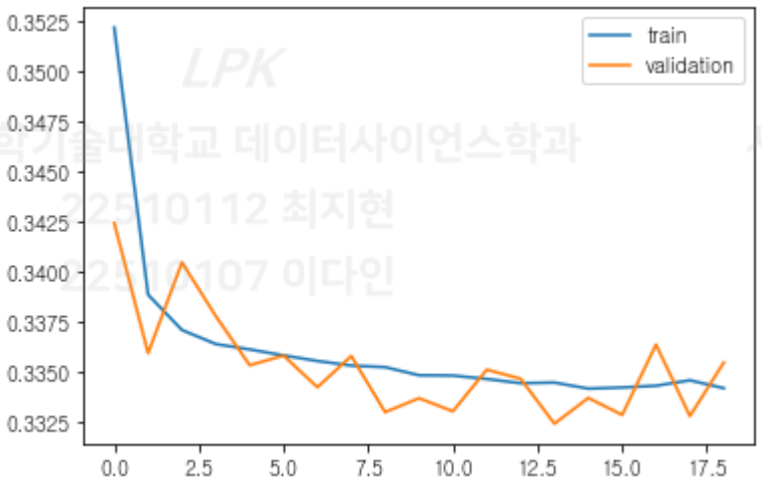
- 이에 따른 성능 지표는 다음과 같다.

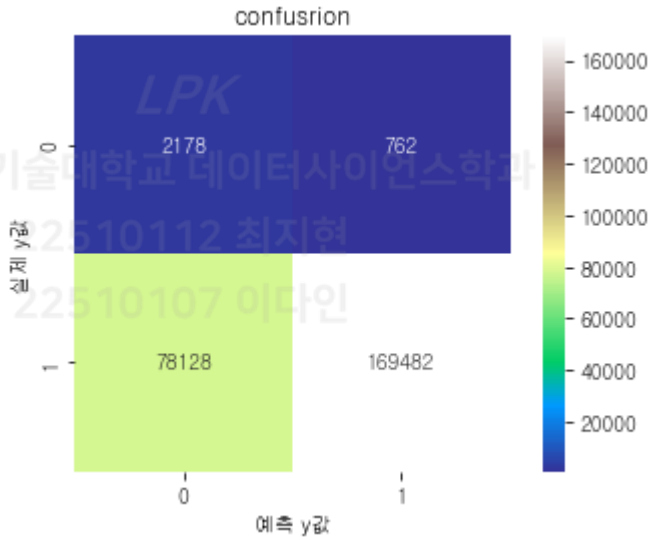
성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.9931	0.7648	0.8642	0.76235	0.4463

Deep Learning Model

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_2 (GRU)                  (None, 50)                8250
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
```

- 단일 GRU
  - 학습이 진행될 때 마다 loss 값이 약간 튀지만 전반적으로 줄어드는 모습을 확인할 수 있다.
  - confusion matrix를 보면 실제 0(NG)을 0(NG)로 잘 예측하는 모습을 보인다. ⇒ FP/N 작아짐.
    - 실제 1(OK)을 0(NG)로 예측하는 비율이 상당히 높다.





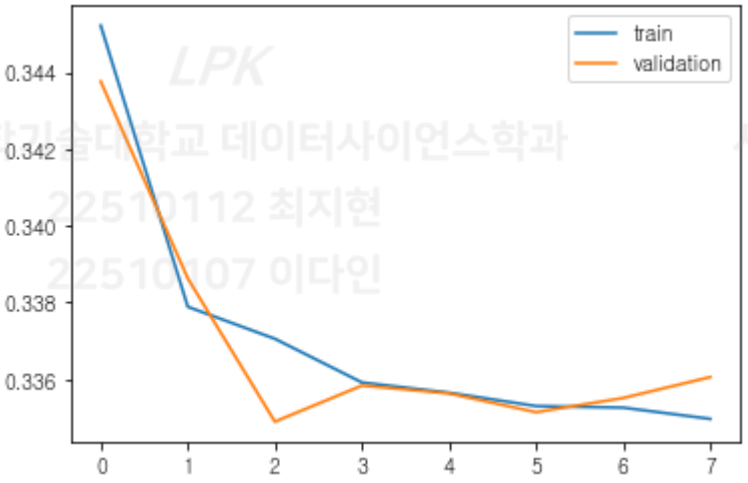
- 이에 따른 성능 지표는 다음과 같다.

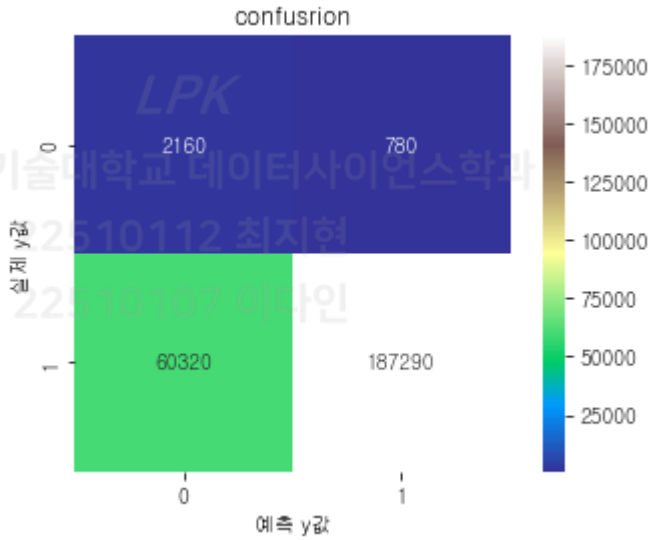
성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.9955	0.6845	0.8112	0.6851	0.2592

Deep Learning Model

- 다중 LSTM
  - 학습이 진행될 때 마다 loss 값이 줄어드는 모습을 확인할 수 있다.
  - confusion matrix를 보면 단일 LSTM보다 나은 성능을 보여준다. ⇒ FP/N 하락
    - epochs가 가장 적다.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 10, 50)            10800
dropout (Dropout)            (None, 10, 50)            0
lstm_1 (LSTM)                (None, 10, 50)            20200
dropout_1 (Dropout)          (None, 10, 50)            0
lstm_2 (LSTM)                (None, 50)                20200
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 51,251
Trainable params: 51,251
Non-trainable params: 0
_________________________________________________________________
```





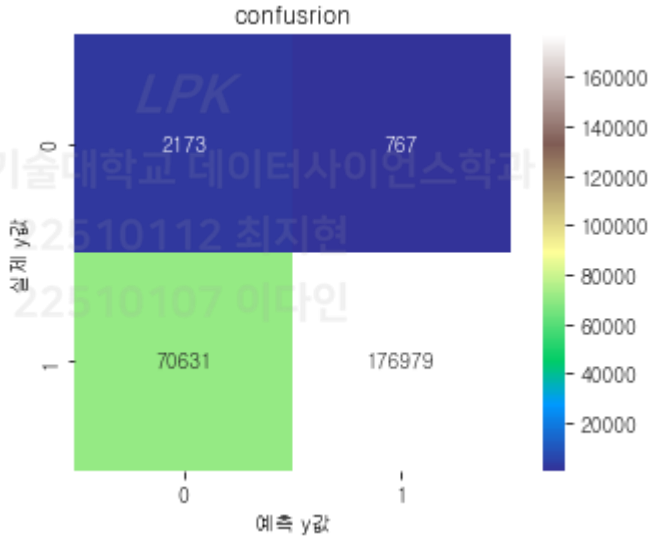
- 이에 따른 성능 지표는 다음과 같다.

성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.9959	0.7564	0.8598	0.7561	0.2653

Deep Learning Model

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 tcn_1 (TCN)                 (None, 20)                13700
 dense_1 (Dense)             (None, 1)                 21
=================================================================
Total params: 13,721
Trainable params: 13,721
Non-trainable params: 0
_________________________________________________________________
```

- TCN
  - 학습이 진행될 때 마다 loss 값이 튀는 모습을 확인할 수 있다.
  - confusion matrix를 보면 실제 0(NG)을 0(NG)로 잘 예측하는 모습을 보인다.
    - 성능이 좋은 편이지만 학습이 느리다.



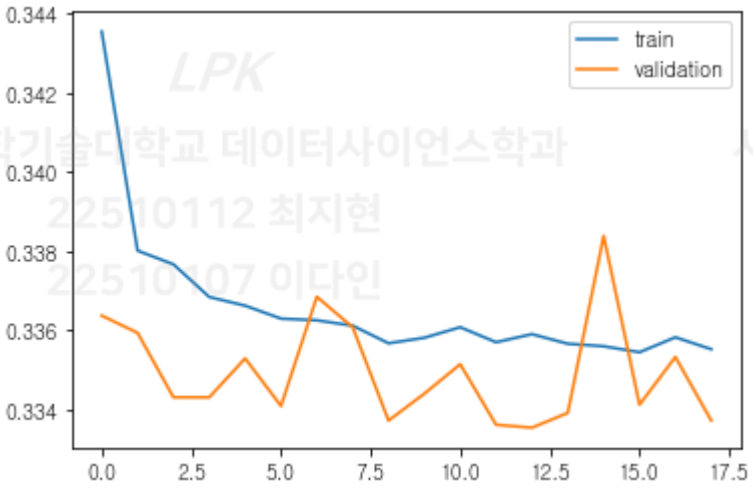
- 이에 따른 성능 지표는 다음과 같다.

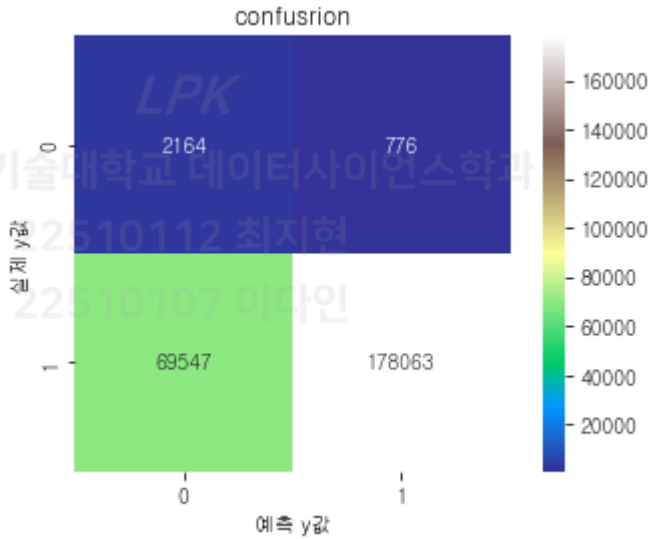
성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.9915	0.7900	0.8793	0.7858	0.2608

Deep Learning Model

- TCN & LSTM
  - 학습이 진행될 때 마다 loss 값이 많이 줄어들지 않고 튀는 모습을 확인할 수 있다.
  - confusion matrix를 보면 실제 0(NG)을 0(NG)로 잘 예측하는 모습을 보인다.
    - 다중 LSTM보다 학습 속도가 느리다.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 10, 50)            10800
dropout_4 (Dropout)          (None, 10, 50)            0
lstm_6 (LSTM)                (None, 10, 50)            20200
dropout_5 (Dropout)          (None, 10, 50)            0
tcn_1 (TCN)                  (None, 50)                90600
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 121,651
Trainable params: 121,651
Non-trainable params: 0
_________________________________________________________________
```





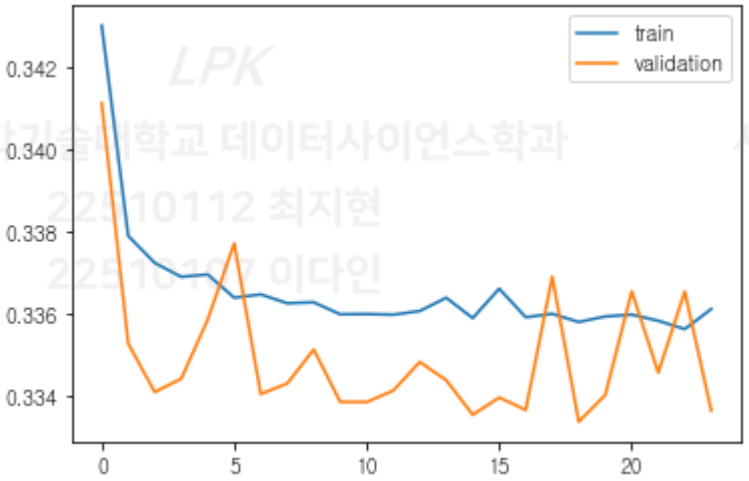
- 이에 따른 성능 지표는 다음과 같다.

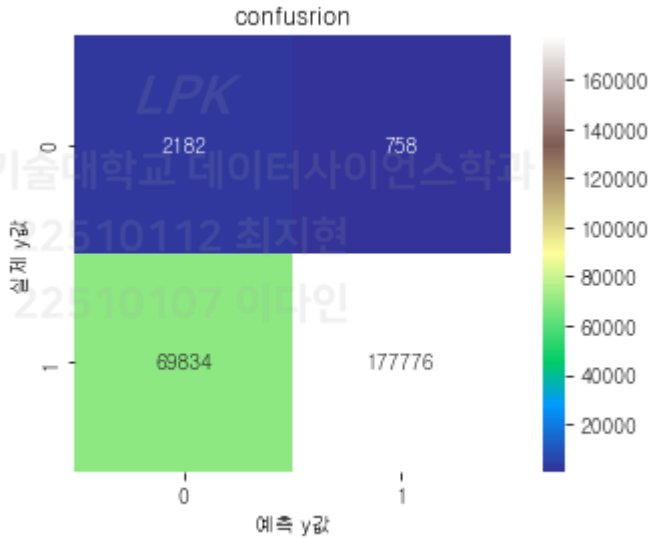
성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.9957	0.7191	0.8351	0.7193	0.2639

Deep Learning Model

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_4 (GRU)                  (None, 10, 50)            8250
dropout_10 (Dropout)         (None, 10, 50)            0
tcn_6 (TCN)                  (None, 20)                17460
dense_5 (Dense)              (None, 1)                 21
=================================================================
```

- TCN & GRU 1
  - 학습이 진행될 때 마다 loss 값이 상당히 크고 튀는 모습을 확인할 수 있다.
  - confusion matrix를 보면 실제 0(NG)을 0(NG)로 잘 예측하는 모습을 보인다. ⇒ FP/N 최소
    - 학습하는 데 시간이 오래 걸린다.





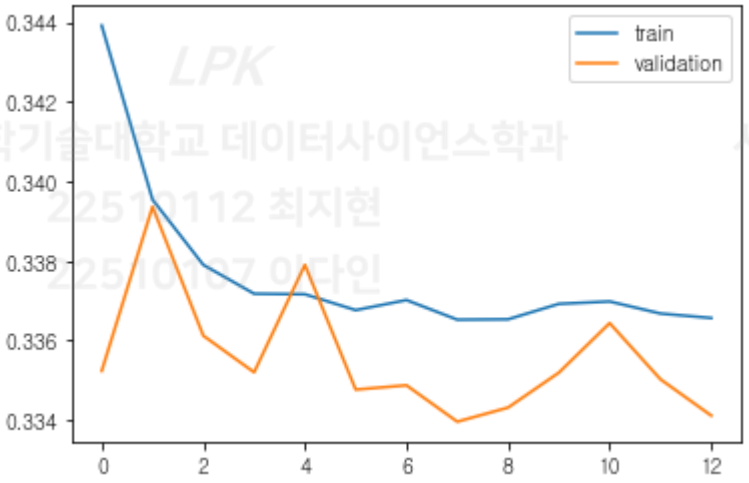
- 이에 따른 성능 지표는 다음과 같다.

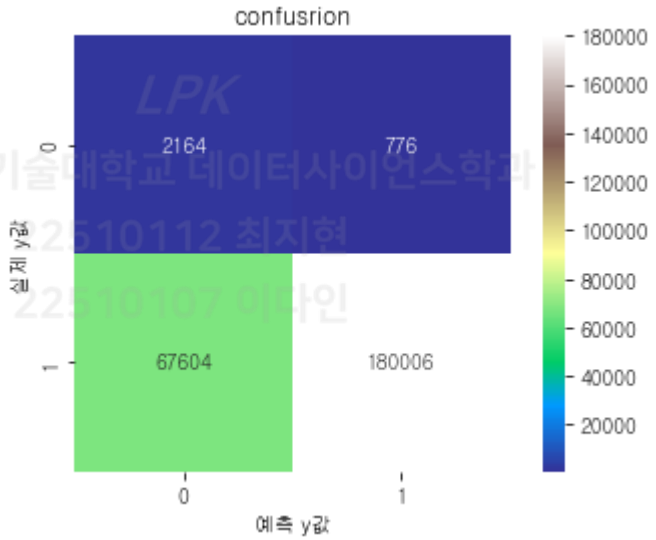
성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.9958	0.7180	0.8344	0.7183	0.2578

Deep Learning Model

- TCN & GRU 2
  - 학습이 진행될 때 마다 loss 값이 크고, 튀는 모습을 확인할 수 있다.
    - TCN & GRU 1보다 loss값이 많이 안정 돼있는 모습을 볼 수 있다.
  - confusion matrix를 보면 실제 0(NG)을 0(NG)로 잘 예측하는 모습을 보인다.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_2 (GRU)                  (None, 10, 50)            8250
dropout_8 (Dropout)          (None, 10, 50)            0
gru_3 (GRU)                  (None, 10, 30)            7380
dropout_9 (Dropout)          (None, 10, 30)            0
tcn_5 (TCN)                  (None, 25)                23950
dense_4 (Dense)              (None, 1)                 26
=================================================================
Total params: 39,606
Trainable params: 39,606
Non-trainable params: 0
_________________________________________________________________
```





- 이에 따른 성능 지표는 다음과 같다.

성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.9957	0.7280	0.8404	0.7271	0.2639

Deep Learning 사후확률

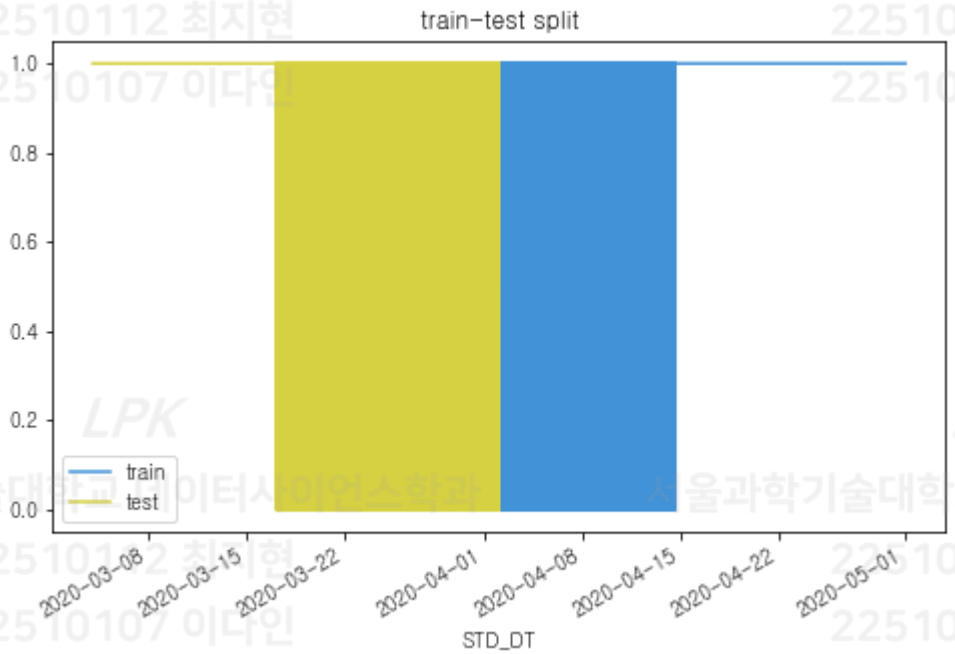
- 방법론
  1. 모델을 만들고 이 모델에 테스트 셋을 넣으면 데이터의 예측 결과 즉, 사후확률이 나오게 된다.
     - 여기까지는 일반적인 딥러닝 모델로 블랙박스이다.

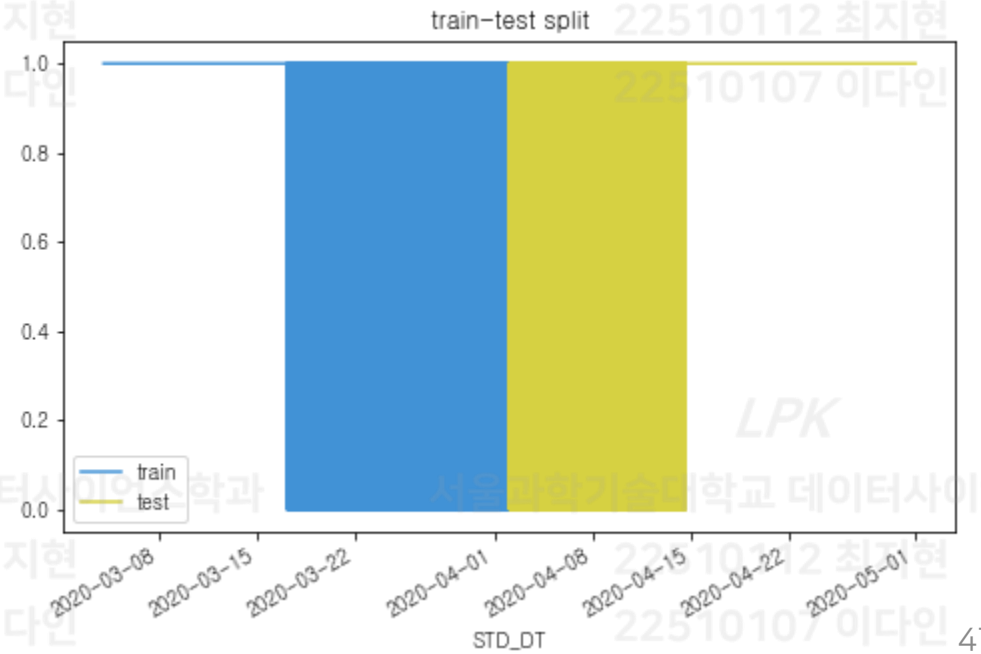
  2. 이걸 전체 데이터 셋에 대해 실행해 전체 데이터 셋에 대한 사후확률을 받는다.
     - Train과 Test를 다르게 해 보면서 사후확률을 구한다.

  3. 받은 사후확률로 새로운 데이터 셋을 만들고 이 새로운 데이터 셋으로 decision tree를 그려서 원래의 데이터 셋과 rule, 성능 지표를 비교해 본다.
     - rule를 받아오면서 화이트박스로 만들어 진다.

Deep Learning 사후확률

- 먼저 데이터 셋은 이상치제거데이터을 사용한 데이터 셋을 이용했다.
- 모델 선정
  - 실제 0(NG)을 0(NG)로 잘 예측하는 모델 중 가장 빠른 다중 LSTM 모델을 사용했다.
- Train – Test은 아래 그림과 같이 나눠 딥러닝을 두 번 시행했다.
  - 종속변수의 값 중 0(NG)의 값이 가운데에 집중 되어있다.
  - 0(NG)와 1(OK)가 비교적 균등하게 들어갈 수 있도록 절반씩 나누어 진행했다.





Deep Learning 사후확률

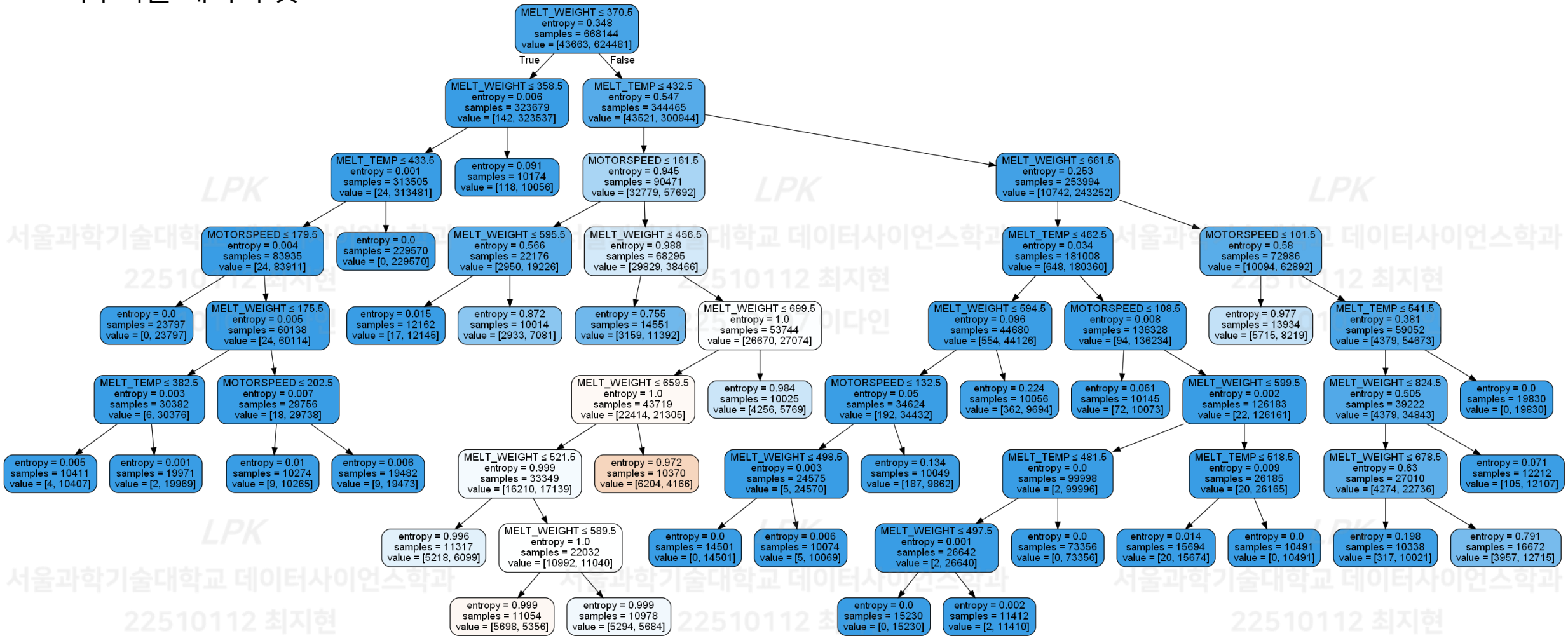
- 사후확률을 받아 0.5 이상이면 1, 0.5 미만이면 0으로 치환한다.
- 이렇게 만들어진 최종 데이터 셋은 다음과 같다.

	STD_DT	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	TAG	predict
0	2020-03-04 00:01:00	507	128	596	1	1
1	2020-03-04 00:01:00	408	66	595	1	1
2	2020-03-04 00:01:00	474	138	594	1	1
3	2020-03-04 00:01:00	358	201	592	1	1
4	2020-03-04 00:01:00	740	1740	590	1	1
...	...	...	...	...	...	...
417585	2020-04-30 23:59:00	755	1743	318	1	1
417586	2020-04-30 23:59:00	385	206	317	1	1
417587	2020-04-30 23:59:00	465	148	316	1	1
417588	2020-04-30 23:59:00	467	0	314	1	1
417589	2020-04-30 23:59:00	453	125	312	1	1

- predict column이 사후확률을 받아 만든 새로운 column이다.

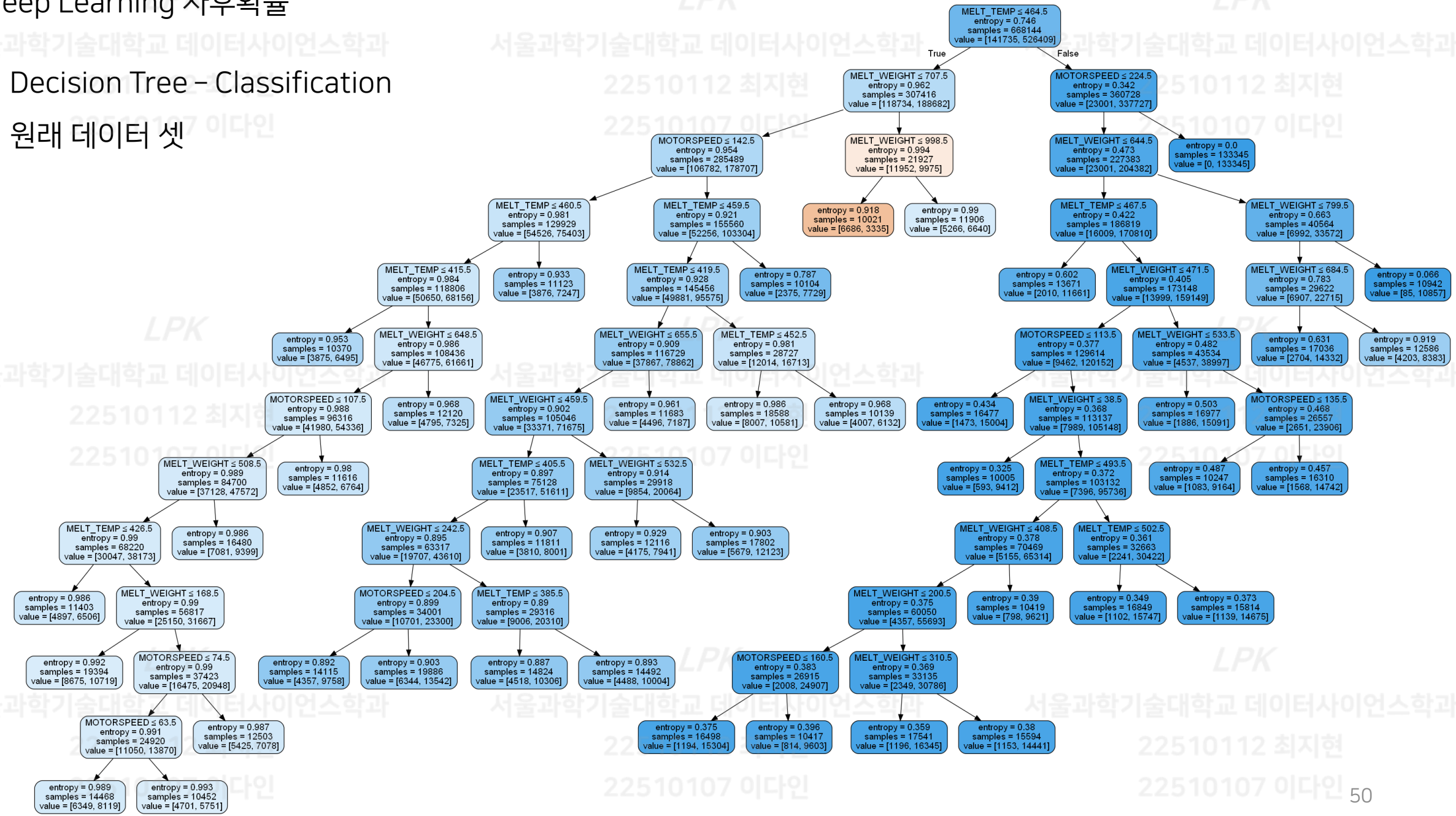
Deep Learning 사후확률

- Decision Tree – Classification
- 사후확률 데이터 셋



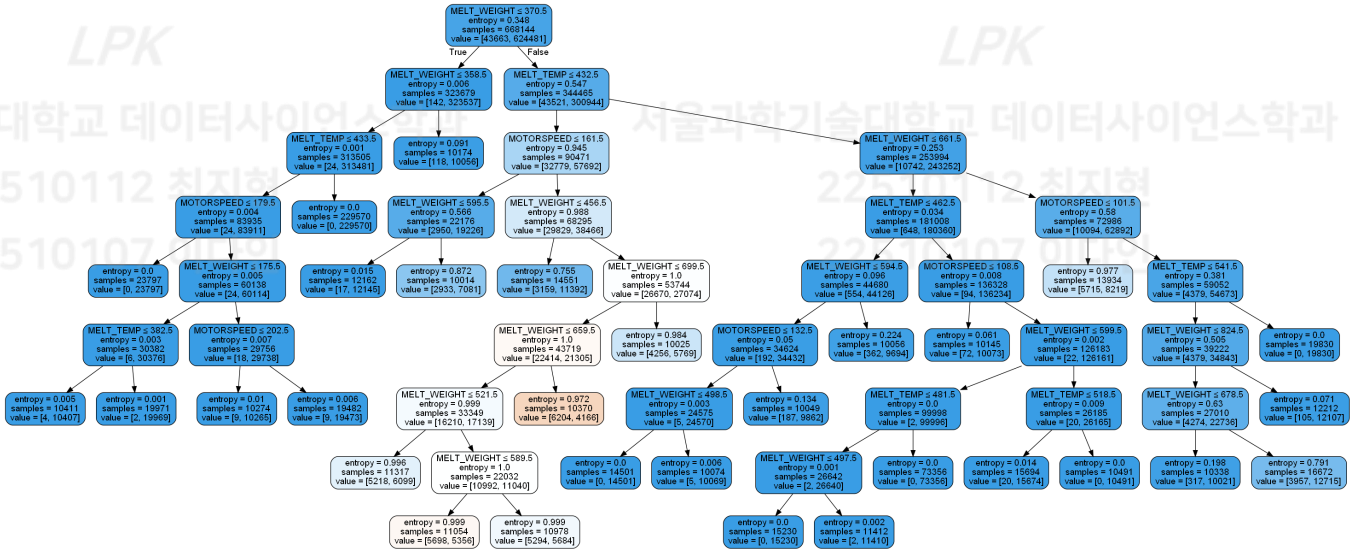
Deep Learning 사후확률

- Decision Tree – Classification
- 원래 데이터 셋

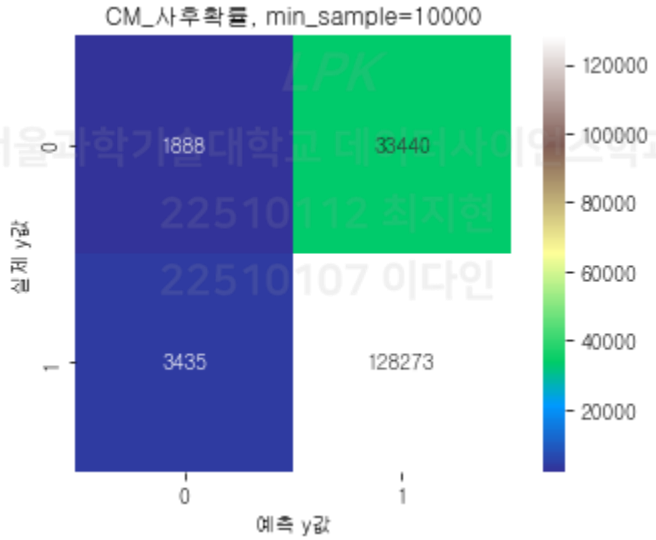


Deep Learning 사후확률

- Decision Tree – Classification
- 사후확률 데이터 셋



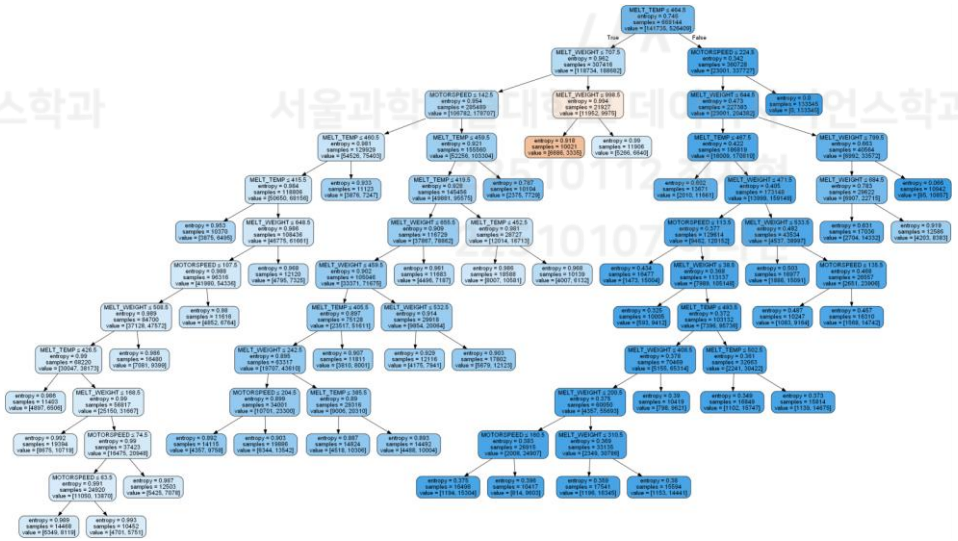
- confusion matrix와 성능 지표는 아래와 같다.



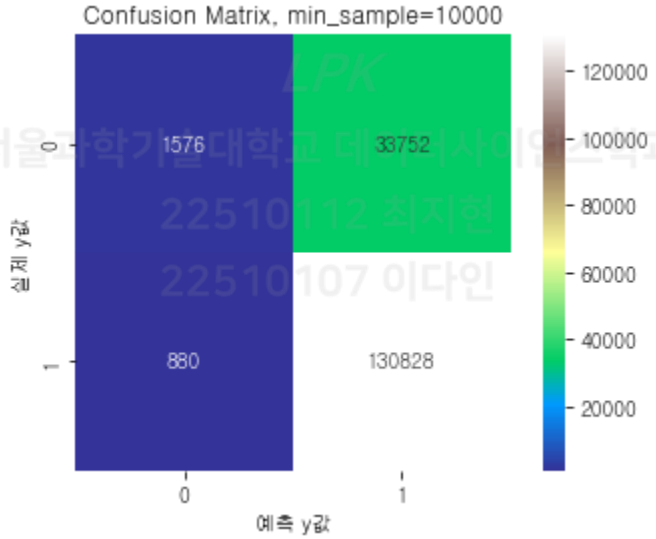
성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.7932	0.9739	0.8743	0.7792	0.9466

Deep Learning 사후확률

- Decision Tree – Classification
- 원래 데이터 셋



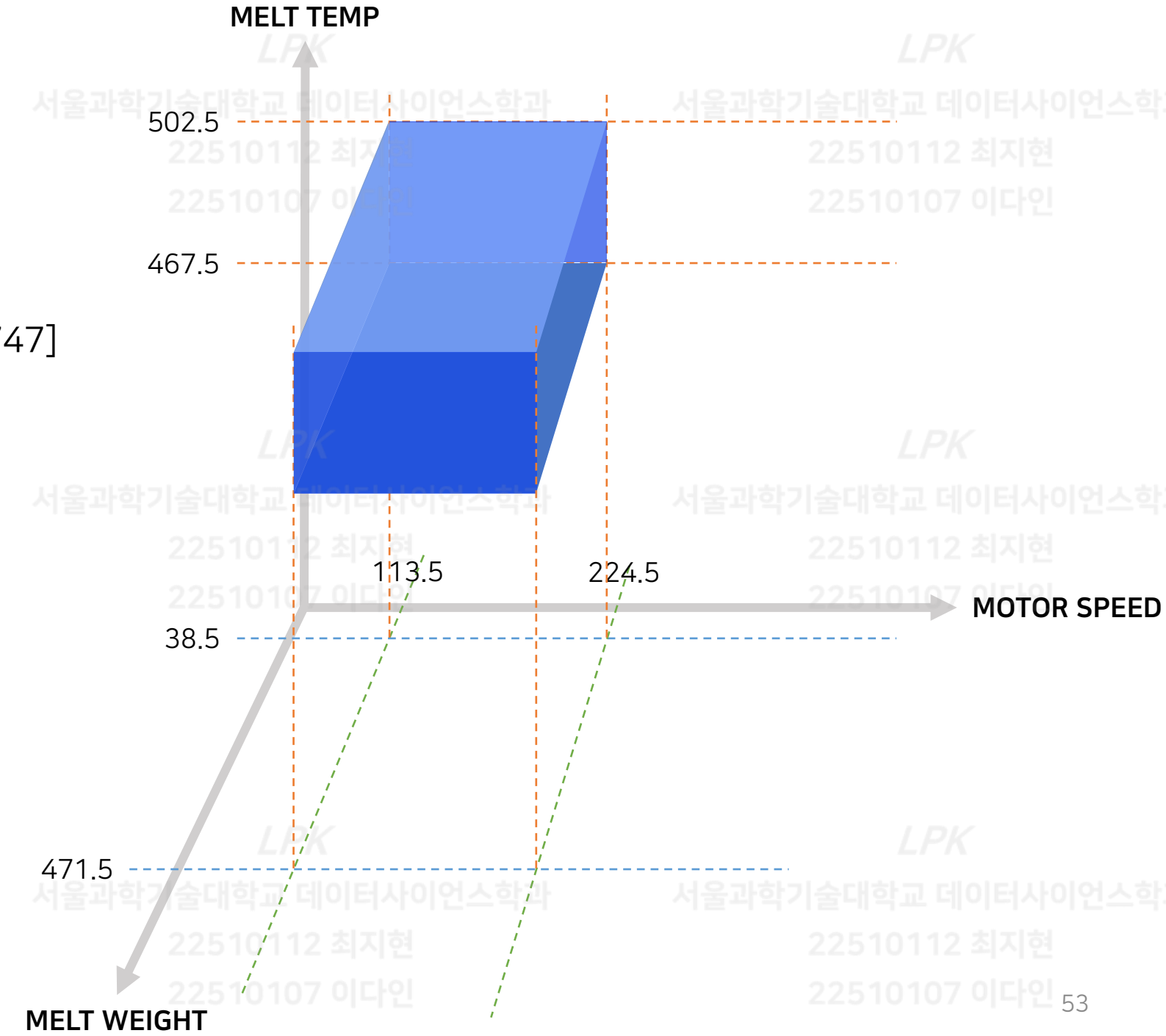
- confusion matrix와 성능 지표는 아래와 같다.



성능 지표	precision	recall	f1-score	accuracy	FP/N
	0.7949	0.9933	0.8831	0.7927	0.9554

Deep Learning 사후확률

- 113.5 > MOTORSPEED ≤ 224.5,
- 467.5> MELT_TEMP ≤ 502.5,
- 38.5 > MELT_WEIGHT ≤ 471.5
- Entropy=0.349, Value=[1102, 15747]



Deep Learning 사후확률

- entropy가 가장 낮고 가장 잘 분류된 rule, 상위 5개

	사후확률 Decision tree / 총 rule 개수: 30	원 데이터 Decision tree / 총 rule 개수 : 41
1	MELT_WEIGHT ≤ 358.5, MELT_TEMP > 433.5 Entropy = 0.0, Value = [0, 229,570]	MELT_TEMP > 464.5, MOTORSPEED > 224.5 Entropy=0.0, Value=[0, 133345]
2	370.5 > MELT_WEIGHT ≤ 599.5, MOTORSPEED > 108.5, MELT_TEMP > 481.5 Entropy = 0.0, Value = [0, 73,356]	MELT_TEMP > 464.5, MOTORSPEED ≤ 224.5, MELT_WEIGHT > 799.5 Entropy=0.066, Value=[85, 10857]
3	MELT_WEIGHT ≤ 358.5, MELT_TEMP ≤ 433.5, MOTORSPEED ≤ 179.5 Entropy = 0.0, Value = [0, 23,797]	113.5 > MOTORSPEED ≤ 224.5, MELT_TEMP > 467.5, MELT_WEIGHT ≤ 38.5 Entropy=0.325, Value=[593, 9412]
4	370.5 > MELT_WEIGHT ≤ 498.5, 432.5 > MELT_TEMP ≤ 462.5, MOTORSPEED ≤ 132.5 Entropy = 0.0, Value = [0, 14,501]	113.5 > MOTORSPEED ≤ 224.5, 467.5 > MELT_TEMP ≤ 502.5, 38.5 > MELT_WEIGHT ≤ 471.5 Entropy=0.349, Value=[1102, 15747]
5	370.5 > MELT_WEIGHT ≤ 497.5, 462.5 > MELT_TEMP ≤ 481.5, MOTORSPEED > 108.5 Entropy = 0.0, Value = [0, 15,230]	113.5 > MOTORSPEED ≤ 224.5, 467.5 > MELT_TEMP ≤ 493.5, 200.5 > MELT_WEIGHT ≤ 310.5 Entropy=0.359, Value=[1196, 16345]

Deep Learning 사후확률

- 나무 안의 색이 같은 색 계열이면 같은 집단으로 분류를 한 것이다.
- 색이 진해질 수록 정확하게 분류를 했다는 것이다.
- 트리를 보면 사후확률로 그린 그래프가 같은 색의 집단으로 분명하게 분류되는 것을 알 수 있다.
- 실제 0을 1로 예측하는 비율이 사후확률 Decision Tree가 더 작다.
    - FP/N 이 더 작음.

- 모든 룰의 entropy 값 0.5 이하임을 알 수 있다.
    - value 안의 값을 보면 사후확률 decision tree의 값이 더 잘 분류된 모습을 볼 수 있다.

최종 결론

- 데이터 결론
  - MELT_WEIGHT에 이상치가 많다.
  - 데이터 베이스에 MELT_WEIGHT의 입력 방식이 화면 인식을 통한 딥러닝 분류 기법인 것 같다.
    - 디지털 숫자를 인식하는데 0과 8이 비슷해 8000대의 이상치가 존재하는 것 같다.
    - (예시)
    - 정확한 MELT_WEIGHT 값이 입력될 수 있는 방법을 찾아야 한다.
  - 설명변수와 목표변수의 관계가 크지 않다.
    - 이는 0과 1의 분류가 잘 되지 않는 문제로 이어진다.
    - 더 큰 상관관계를 갖는 변수를 추가해야 한다.

최종 결론

- 머신 러닝의 성능이 낮은 이유는 데이터 불균형이 심하고, 설명변수와 목표변수의 관계가 없기 때문이다.
- 또한 머신 러닝으로 시계열 분석은 불가능하다.
- 따라서 딥러닝으로 시계열 분석을 진행했다.
- 확실한 성능 향상이 있었지만 시간이 오래 걸리고 만족할 만한 결과를 도출하지 못했다.
    - 머신러닝과 비교해서 성능이 높지만 딥러닝도 0과 1을 제대로 분류하지 못 한다.

- 성능이 좋아도 90% 이상의 정확도를 가지지 않는다.
- 따라서 현장작업자가 양품을 잘 만들 수 있는 규칙이 정립될 수 있으면 좋겠다고 생각했다.
    - 딥러닝을 화이트 박스 모델로 만들어줄 방법을 고안했다.
        - 사후확률 Decision Tree 이다.

- 이 사후확률 규칙을 지킨다면 불량이 발생할 확률이 감소할 것이다.
    - **현장작업자가 직접 용해 상태를 보고 판단하는 것이 아닌 설명변수의 설정값 만으로도 양품을 만들어 낼 수 있을 것이다.**

감사합니다.