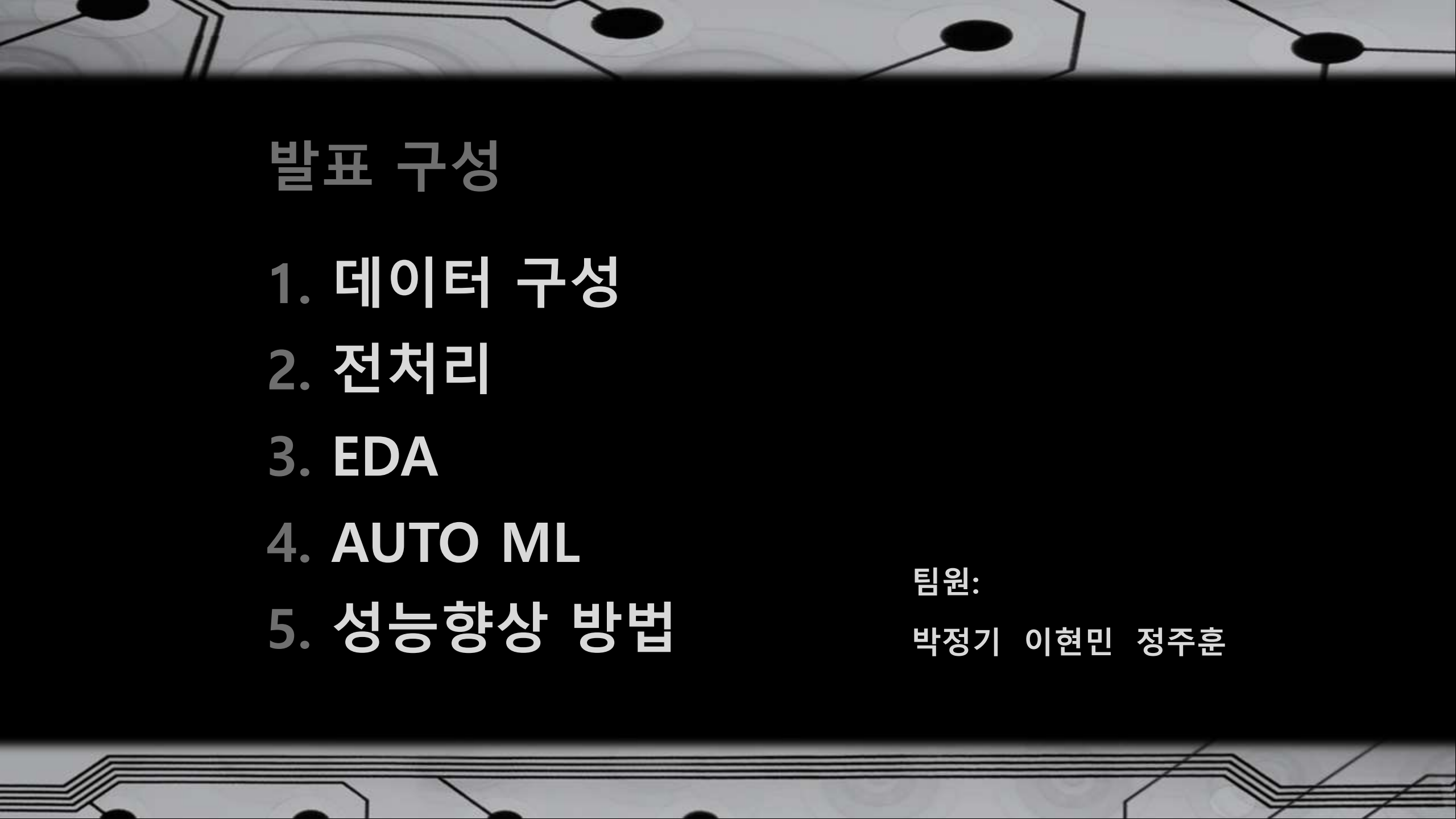


시스템 품질 변화로 인한 사용자 불편 예지 AI 경진대회

정형data / 성능평가지표: AUC

The background of the slide features a dark, textured surface with a light-colored circuit board pattern. This pattern includes various lines, pads, and circular components, resembling a printed circuit board (PCB) layout, which is visible at the top and bottom edges of the slide.

발표 구성

1. 데이터 구성
2. 전처리
3. EDA
4. AUTO ML
5. 성능향상 방법

팀원:

박정기 이현민 정주훈



DACON

커뮤니티

대회

교육

랭킹

더보기

시스템 품질 변화로 인한 사용자 불편 예지 AI 경진대회

LG | 채용 | 시스템 | 사용자 불편 | 정형 | AUC

₩ 상금 : 총 1,000만원

🕒 2021.01.06 ~ 2021.02.03 18:00

+ Google Calendar

대회안내

데이터

코드 공유

토크

리더보드

제출

☰ 개요

📄 규칙

2.목적

데이터를 통해 사용자가 불편을 느끼는 원인 분석

프로젝트 가이드라인

1. ML WorkFlow 구현
2. 적절한 성능평가
 - 과제에서 주어진 평가지표(AUC) 사용
 - train, valid, test set 구분
3. 다양한 ML 모델 사용
4. 모델보다 **Data**에 집중하여 많은 인사이트 도출
5. 처음부터 완벽 모델X
 - 작동하는 pipe line 구현 후 개선 시도

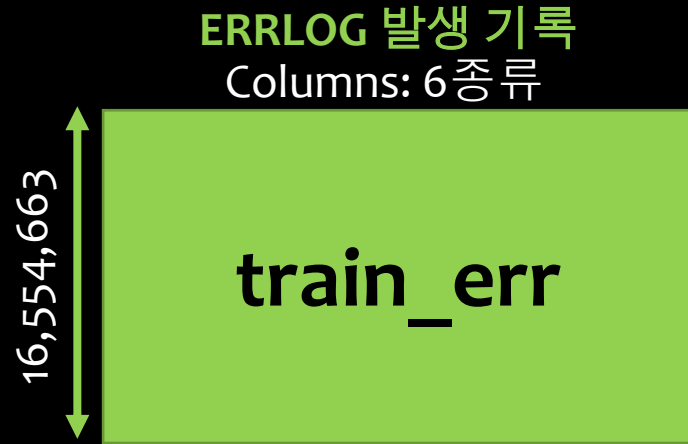
Dacon 평가 기준

구분	항목 및 상세		배점 (만점)
분석	모델 성능	리더보드 Private 점수	10
	분석능력	사용자 불만 접수 원인 분석	15
		err_data간 err 관계 해석	10
		quality_data 수치 해석	10
		err_data와 quality_data간 관계 해석	10
		결과, 비즈니스 분석	10
발표	자료	마크다운, 코드 품질	10
		PPT 자료 완성도	10
	전달	발표 내용, 시간 준수, 질의 응답	15

1. 데이터 구성



1. 데이터 구성



```
err_train.head(3)
```

[6] ✓ 0.5s

	user_id	time	model_nm	fwver	errtype	errcode
0	10000	20201101025616	model_3	05.15.2138	15	1
1	10000	20201101030309	model_3	05.15.2138	12	1
2	10000	20201101030309	model_3	05.15.2138	11	1

user_id : 10000~14999 (15,000명)

time : ERRLOG 발생된 **초단위 시각** (2020-10-31 23:59:59 ~ 2020-12-02 18:51:52)

model_nm : ERRLOG 발생시의 모델 (9종류)

fwver : ERRLOG 발생시의 펌웨어 버전 (37종류)

errtype : 발생한 ERRLOG의 타입 (41종류)

errcode : 발생한 에러코드 (2805종류)

Categorical

1. 데이터 구성



```
quality_train.head(3)
```

[10] ✓ 0.1s

...

	time	user_id	fwver	quality_0	quality_1	quality_12
0	20201129090000	10000	05.15.2138	0.0	0	0
1	20201129090000	10000	05.15.2138	0.0	0	0
2	20201129090000	10000	05.15.2138	0.0	0	0

user_id : 10000~14999 (15,000명 중, 8281명)

time : err quality 기록 시각 (2020-10-31 23:50:00 ~ 2020-11-30 23:40:00)
→ 최소 구분단위: 10분

fwver : err quality 기록시의 펌웨어 버전 (28개)

Categorical

quality_0~12 : err quality 기록시의 quality 수치 (-1~1,910,175 정수)

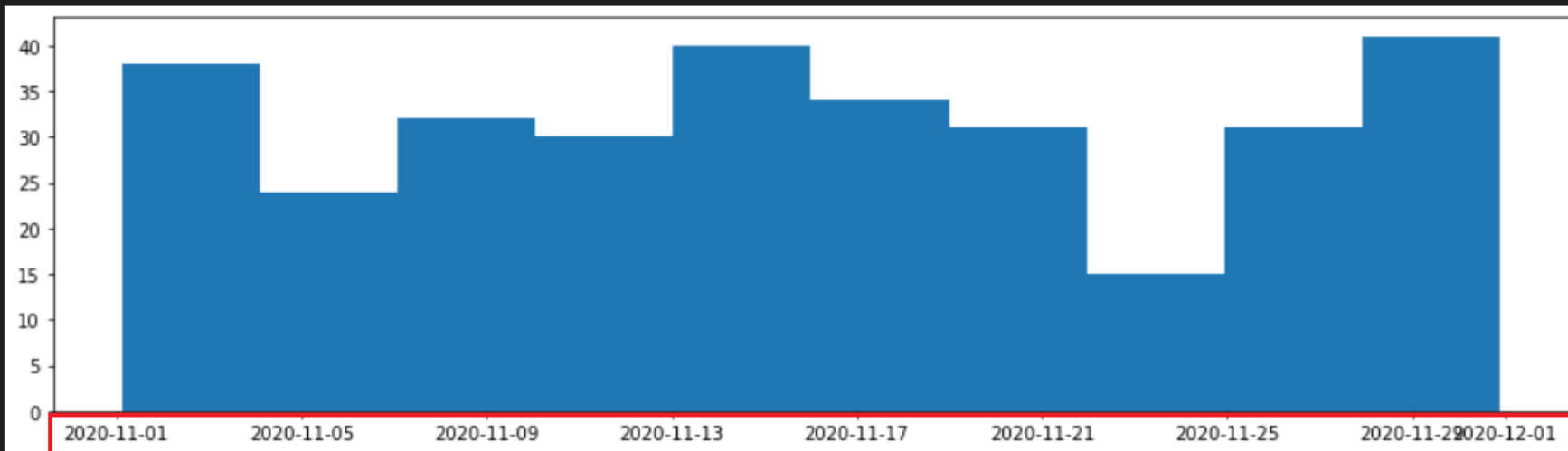
Numeric

1. 데이터 구성

특정유저의
err(1600만)
quality(80만)
time 분포
비교

```
# 10000 유저의 err_time 분포  
plt.figure(figsize=(15,4))  
plt.hist( err.loc[ err["user_id"]==10000, "time" ] )  
plt.show()
```

✓ 0.3s



```
# 10000 유저의 quality_time 분포  
quality.loc[ quality["user_id"]==10000, "time" ]
```

✓ 0.9s

0	2020-11-29	09:00:00
1	2020-11-29	09:00:00
2	2020-11-30	21:00:00
3	2020-11-30	21:00:00

Name: time, dtype: datetime64[ns]

1. 데이터 구성

사용자의 불편신고 기록
Columns: 2종류

5,429
train_problem

```
problem.head(3)
```

[53]	✓	0.9s
...		
	user_id	time
0	19224	20201102200000
1	23664	20201116140000
2	15166	20201114130000

분류 후
확률정보(predict_proba)를
사용하거나,

회귀를 통해서 직접 산출?

user_id : 10000~14999 (15,000명 중, **5000명**)

time : ERRLOG 발생 시각

특이사항 : 제출할 최종 파일이 불편신고 '유무'가 아니라
불편신고 '확률' 이라는 점

각 사용자별 불편신고확률

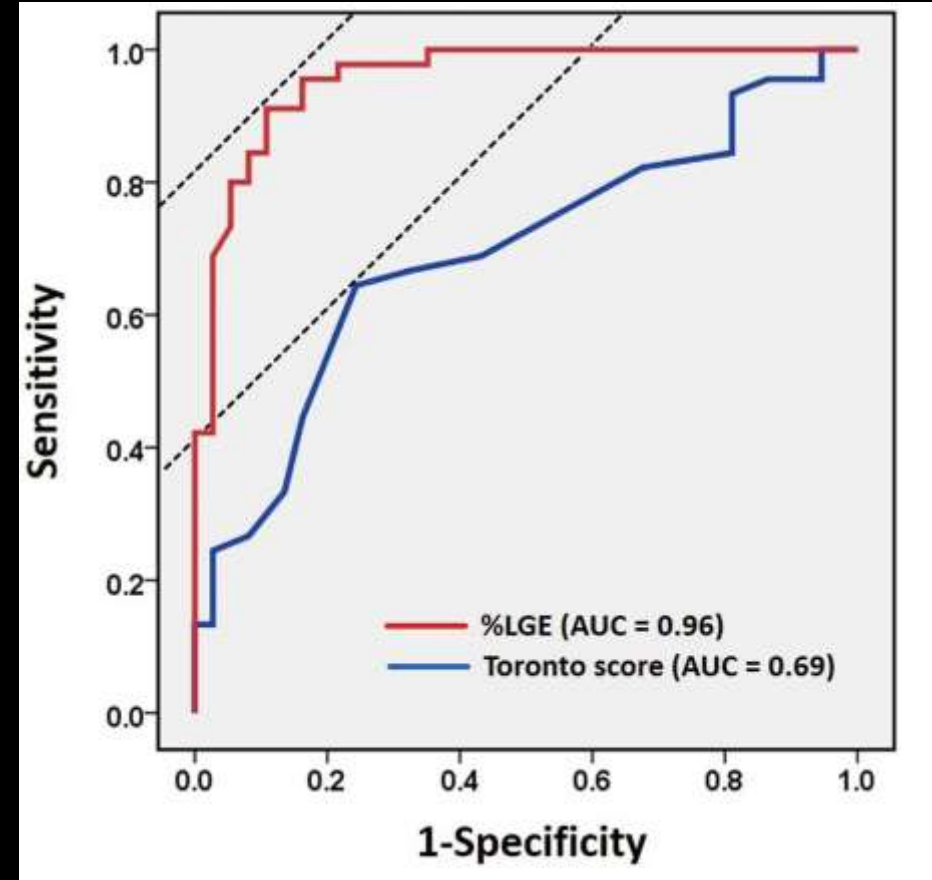
submission

Goal →

1. 데이터 구성

AUC

- 분류모델의 평가지표
- ROC(확률곡선) 아래 면적
- 0~1 실수값
- 1에 가까울 수록 좋은 성능



1. 데이터 구성

각 테이블의 time 컬럼 비교

err: 초

```
err[["user_id", "time"]].head(10)
```

✓ 0.1s

	user_id	time
0	10000	2020-11-01 02:56:16
1	10000	2020-11-01 03:03:09
2	10000	2020-11-01 03:03:09
3	10000	2020-11-01 05:05:14
4	10000	2020-11-01 05:05:15
5	10000	2020-11-01 05:05:16
6	10000	2020-11-01 05:05:22
7	10000	2020-11-01 06:09:03
8	10000	2020-11-01 17:47:54
9	10000	2020-11-01 17:47:56

1600만 건

quality: 10분

```
quality["time"].value_counts()
```

✓ 0.1s

2020-11-10	14:30:00	282
2020-11-10	14:40:00	252
2020-11-08	23:00:00	241
2020-11-20	21:00:00	241
2020-11-10	14:20:00	234
...		
2020-11-10	03:30:00	5
2020-11-14	05:40:00	5
2020-11-08	04:10:00	4
2020-11-13	06:10:00	3
2020-11-03	04:10:00	2

Name: time, Length: 4319, dtype: int64

80만 건

problem: 시간

```
problem.head(10)
```

✓ 0.1s

	user_id	time
0	19224	2020-11-02 20:00:00
1	23664	2020-11-16 14:00:00
2	15166	2020-11-14 13:00:00
3	12590	2020-11-08 21:00:00
4	15932	2020-11-03 21:00:00
5	16852	2020-11-19 15:00:00
6	23427	2020-11-21 11:00:00
7	13507	2020-11-11 16:00:00
8	11274	2020-11-18 12:00:00
9	20610	2020-11-27 23:00:00

5400 건

2. 전처리: 공통 과정

1) 중복 data 제거

```
# drop_duplicates: 완전 중복행 제거  
func01 = lambda df: df.drop_duplicates(subset=None, keep='first')
```

✓ 0.8s

Python

```
apply_(func01,0)
```

✓ 16.3s

Python

before: err_train:	(16554663, 6)	err_test:	(16532648, 6)	quality_train:	(828624, 16)	quality_test:	(747972, 16)
after : err_train:	(15368002, 6)	err_test:	(15527225, 6)	quality_train:	(284202, 16)	quality_test:	(243839, 16)

2. 전처리: 공통 과정

2) dtype 변경: ['time'] int → datetime

20201102220000 → 2020-11-02 22:00:00

```
# ['time'] dtype: int --> datetime64 변경
def func02(df):
    df['time'] = pd.to_datetime( df['time'], format='%Y%m%d%H%M%S' )
    return df
```

✓ 0.9s

Python

```
apply_(func02,0)
```

✓ 1m 17.6s

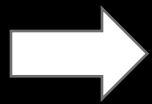
Python

before: err_train:(15368002, 6) err_test:(15527225, 6) quality_train:(284202, 16) quality_test:(243839, 16)
after : err_train:(15368002, 6) err_test:(15527225, 6) quality_train:(284202, 16) quality_test:(243839, 16)

2. 전처리: 공통 과정

3) quality table: 각 quality_0~12 컬럼 값

-1, 0, 1, ..., 1,910,175 의 정수 값



문제점1) int, str 혼용 사용됨 → 5, "5"

문제점2) str type 일부 comma(,) 사용됨 → "1,317"

```
list_ = []
for v in train_qual.quality_8.unique():
    list_.append(v)
print( list_ )
```

Python

```
[0, 1, 8, 5, 11, 2, 6, 20, 4, 3, 7, 12, 16, 14, 38, 10, 13, 15, 9, 29, 27, 18, 43, 17, 22, 42, 19, 125, 21,
25, 23, 32, 68, 73, 26, 35, 28, 31, '0', '7', '1', '2', '4', '6', '3', '5', '9', '12', '8', '1,317', '10',
'17', '27', '15', '24', 47, 37]
```

2. 전처리: 공통 과정

3) quality table: 각 quality_0~12 컬럼 값

-1, 0, 1, ..., 1,910,175 의 정수 값

```
# 콤마 제거 함수
str_to_int = lambda x: int(x.replace(",", "")) if type(x) == str else x

# int로 일괄 변환 함수: quality 테이블의 3번 컬럼부터가 [quality_N]
def func03(df):
    for i in df.columns[3:]:
        df[i] = df[i].apply(str_to_int)
    return df
```

✓ 0.9s

Python

```
apply_(func03,2)
```

✓ 2.6s

Python

before: quality_train:(284202, 16) quality_test:(243839, 16)

after : quality_train:(284202, 16) quality_test:(243839, 16)

2. 전처리: 공통 과정

4) quality table: 일부 컬럼 drop

```
# quality 테이블의 두 컬럼 [quality_3],[quality_4]
# 고유값이 1개씩 밖에 없다. ---> drop

print( len(data[2][1].quality_3.unique()), end=" ")
print( len(data[2][1].quality_4.unique()), end=" ")
print( len(data[3][1].quality_3.unique()), end=" ")
print( len(data[3][1].quality_4.unique()) )
```

✓ 0.7s

Python

1 1 1 1

```
# [quality_3],[quality_4] 컬럼 드랍하는 함수
def func04(df):
    df = df.drop( ["quality_3","quality_4"], axis=1 )
    return df
```

✓ 0.1s

Python

```
apply_(func04,2)
```

✓ 0.1s

Python

before: quality_train:(284202, 16) quality_test:(243839, 16)
after : quality_train:(284202, 14) quality_test:(243839, 14)

2. 전처리: 결측치

각 Dataframe별 결측치 확인

<train_err>

user_id	0
time	0
model_nm	0
fwver	0
errtype	0
errcode	1
dtype:	int64

<train_quality>

time	0
user_id	0
fwver	40080
quality_0	144432
quality_1	0
quality_2	40113
quality_3	0
quality_4	0
quality_5	20
quality_6	0
quality_7	0
quality_8	0
quality_9	0
quality_10	0
quality_11	0
quality_12	0
dtype:	int64

<test_err>

user_id	0
time	0
model_nm	0
fwver	0
errtype	0
errcode	4
dtype:	int64

<test_quality>

time	0
user_id	0
fwver	22764
quality_0	106584
quality_1	11
quality_2	21115
quality_3	0
quality_4	0
quality_5	44
quality_6	0
quality_7	0
quality_8	0
quality_9	0
quality_10	0
quality_11	0
quality_12	0
dtype:	int64

2. 전처리 : 결측치

train_err: errcode

```
1 # errcode 40013일 때와 user_id, time, model_nm, fwver, errtype모두 같다
2 train_err.iloc[3825742:3825747]
```

	user_id	time	model_nm	fwver	errtype	errcode
3825742	13639	2020-11-21 17:40:58	model_2	04.33.1261	31	1
3825743	13639	2020-11-21 17:41:10	model_2	04.33.1261	31	0
3825744	13639	2020-11-21 19:17:18	model_2	04.33.1261	5	NaN
3825745	13639	2020-11-21 19:17:18	model_2	04.33.1261	5	40013
3825746	13639	2020-11-21 22:09:19	model_2	04.33.1261	15	1

```
1 # 따라서 같은 errcode일 것임을 유추하여 40013값을 넣어준다.
2 train_err.errcode = train_err.errcode.fillna('40013')
```

2. 전처리 : 결측치

test_err: errcode

	user_id	time	model_nm	fwver	errtype	errcode	
	937965	30820	2020-11-15 03:59:00	model_2	04.33.1261	40	1
	937966	30820	2020-11-15 03:59:02	model_2	04.33.1261	40	0
	937967	30820	2020-11-15 04:43:17	model_2	04.33.1261	5	NaN
	937968	30820	2020-11-15 04:43:17	model_2	04.33.1261	5	40053
	937969	30820	2020-11-15 09:10:24	model_2	04.33.1261	15	1
	user_id	time	model_nm	fwver	errtype	errcode	
	4038890	33681	2020-11-03 11:01:47	model_2	04.33.1185	14	14
	4038891	33681	2020-11-03 11:02:59	model_2	04.33.1185	7	14
	4038892	33681	2020-11-03 11:02:59	model_2	04.33.1185	5	NaN
	4038893	33681	2020-11-03 11:02:59	model_2	04.33.1185	5	40053
	4038894	33681	2020-11-03 11:03:00	model_2	04.33.1185	6	14
	user_id	time	model_nm	fwver	errtype	errcode	
	9486879	38991	2020-11-27 18:59:52	model_2	04.33.1261	5	B-A8002
	9486880	38991	2020-11-27 21:38:38	model_2	04.33.1261	5	40053
	9486881	38991	2020-11-27 21:38:38	model_2	04.33.1261	5	NaN
	9486882	38991	2020-11-28 00:13:46	model_2	04.33.1261	26	1
	user_id	time	model_nm	fwver	errtype	errcode	
	10425470	39894	2020-11-28 14:46:05	model_1	04.16.3553	20	1
	10425471	39894	2020-11-28 14:46:21	model_1	04.16.3553	26	1
	10425472	39894	2020-11-28 14:47:12	model_1	04.16.3553	5	-1010
	10425473	39894	2020-11-28 14:47:12	model_1	04.16.3553	5	NaN
	10425474	39894	2020-11-28 14:47:57	model_1	04.16.3553	32	80



```
1 # 결측치 채우기
2 test_err.iloc[937967, 5] = '40053'
3 test_err.iloc[4038892, 5] = '40053'
4 test_err.iloc[9486881, 5] = '40053'
5 test_err.iloc[10425473, 5] = '-1010'
```

2. 전처리 : 결측치

train_quality: fwver

```
1 # 1. train_err에서 fwver가 업그레이드 된 경우가 많이 있다.
2 # 2. trian_quality의 time은 분, 초가 생략되어있다.
3 # 3. 따라서 어느 fwver일 때 quality로그가 찍혔는지 알 수 없다.
4 # 4. fwver의 결측치 비율이 적다 (4.8%)
5 # 따라서 fwver의 결측치는 drop
6
7 fwver_null_index = train_quality[train_quality.fwver.isnull()].index
8
9 train_quality = train_quality.drop(fwver_null_index, axis=0)
10 train_quality.info()
```

test_quality: fwver

```
# fwver결측치 제거

fwver_null_index = list(test_quality[test_quality.fwver.isnull()].index)

test_quality = test_quality.drop(fwver_null_index, axis=0)
test_quality.info()
```

2. 전처리 : 결측치

train_quality: quality_N

```
1 train_quality.quality_0.value_counts()
```

✓ 0.4s

0.0	542790
-1.0	130828
1.0	2097
2.0	1252
3.0	518

```
1 train_quality.iloc[:, 3:].describe()
```

✓ 0.2s

	quality_0	quality_1	quality_2	quality_3	quality_4	quality_6	quality_11	quality_12
count	684192.000000	828624.000000	788511.000000	828624.0	828624.0	828624.000000	828624.000000	828624.000000
mean	4.148701	-0.171782	4.751094	0.0	0.0	2.043391	-0.181638	0.045878
std	479.315029	0.692386	586.252469	0.0	0.0	32.695380	0.397767	0.302452
min	-1.000000	-1.000000	-1.000000	0.0	0.0	-1.000000	-1.000000	0.000000
25%	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.000000
max	157667.000000	171.000000	191859.000000	0.0	0.0	600.000000	14.000000	14.000000



```
1 # case1. 최빈값 채우기
2 # case2. 평균값 채우기
3
4 # -1, 0, 1에 데이터가 편향되어있고, max값과 차이가 많이 난다.
5 # 따라서 case1. 최빈값으로 채우기 선택
6
7 for i in train_quality.columns[3:]:
8     train_quality[i] = train_quality[i].fillna(train_quality[i].mode()[0])
```


2. 전처리 : 결측치

test_quality: quality_N

```
1 test_quality.quality_1.value_counts()
✓ 0.5s
```

0	576597
-1	134061
0	24668
-1	7844
1	2436
2	1050

```
1 test_quality.iloc[:, 3:].describe()
✓ 0.2s
```

	quality_0	quality_2	quality_3	quality_4	quality_6	quality_11	quality_12
count	641388.000000	726857.000000	747972.0	747972.0	747972.000000	747972.000000	747972.000000
mean	2.062694	6.286763	0.0	0.0	2.118419	-0.186356	0.040558
std	218.919971	1199.262744	0.0	0.0	33.651757	0.401407	0.337775
min	-1.000000	-1.000000	0.0	0.0	-1.000000	-1.000000	0.000000
25%	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.000000
max	93038.000000	636619.000000	0.0	0.0	600.000000	17.000000	19.000000

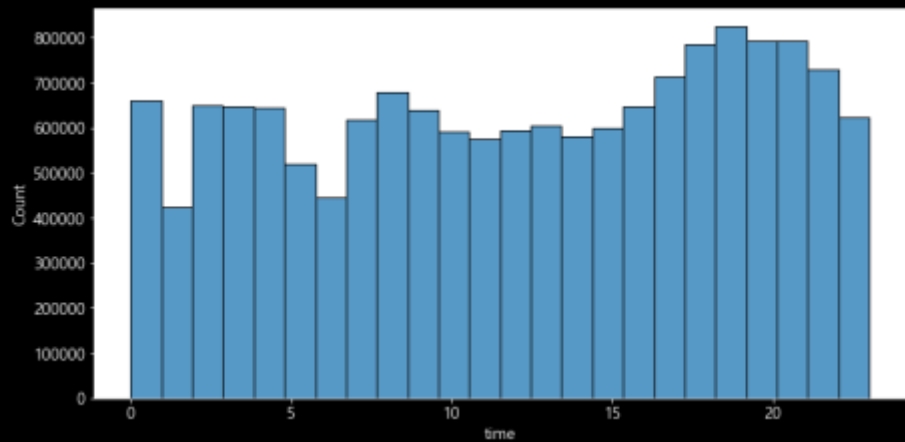


```
1 #최빈값으로 채우기
2
3 for i in test_quality.columns[3:]:
4     test_quality[i] = test_quality[i].fillna(test_quality[i].mode()[0])
```

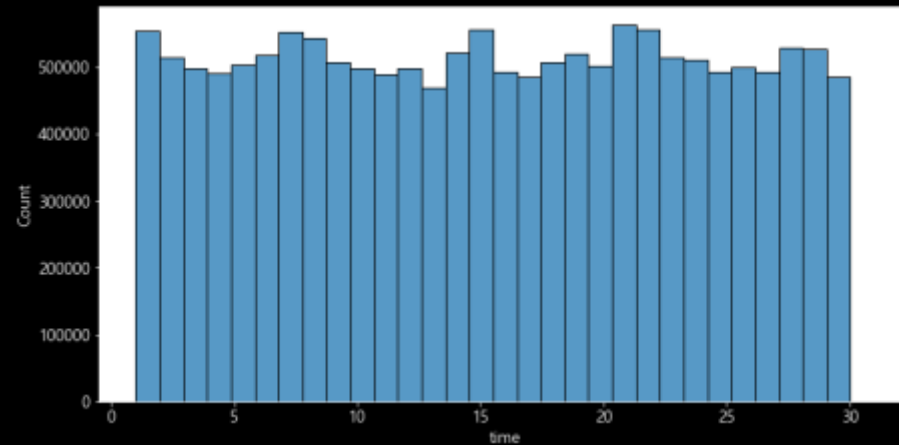
3. EDA: train_err

1. time column

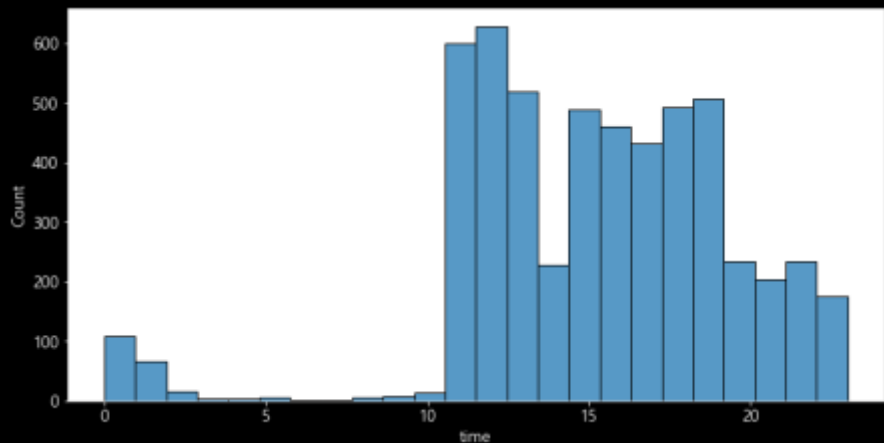
<시간대별 에러 발생 횟수>



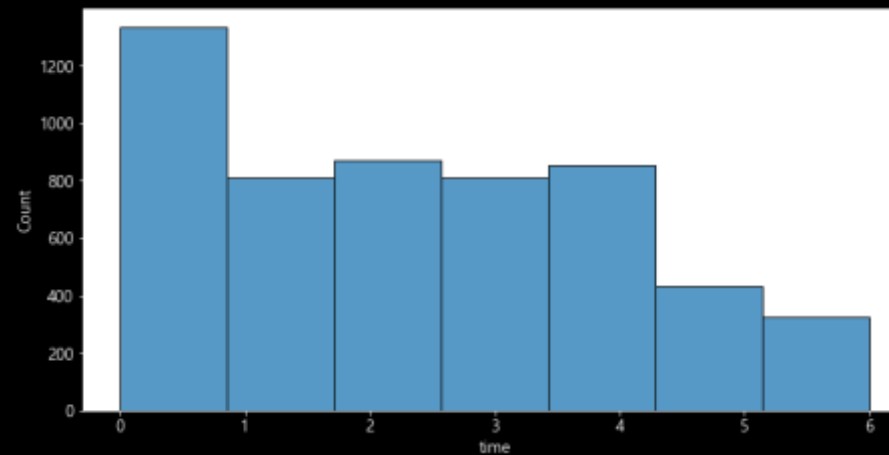
<일별 에러 발생 횟수>



<시간대별 불만 제기 횟수>



<요일별 불만 제기 횟수>



3. EDA: train_err

1. time column

<에러 발생 기간에 따른 유저의 불만 제기율>

```
1 # 에러 발생 기간이 20일 미만인 유저의 불만제기율
2 # 14%
3 df1 = user_day_df[user_day_df['period'] < 20]
4 print(len(train_problem[train_problem.user_id.isin(df1.user_id)].user_id.unique()))
5 print(round(187/1315*100, 2))
```

```
187
14.22
```

```
1 # 평균 에러 발생 기간보다 짧게 오류가 나온 유저의 불만제기율
2 # 22%
3 df2 = user_day_df[user_day_df['period'] < 28]
4 print(len(train_problem[train_problem.user_id.isin(df2.user_id)].user_id.unique()))
5 print(round(712/3217*100, 2))
```

```
712
22.13
```

```
1 # 평균 에러 발생 기간보다 더 길게 오류가 나온 유저의 불만제기율
2 # 36%
3 df3 = user_day_df[user_day_df['period'] > 27]
4 print(len(train_problem[train_problem.user_id.isin(df3.user_id)].user_id.unique()))
5 print(round(4288/11783*100, 2))
```

```
4288
36.39
```


3. EDA: train_err

1. model_nm, fwver column

<model_nm – fwver의 관계>

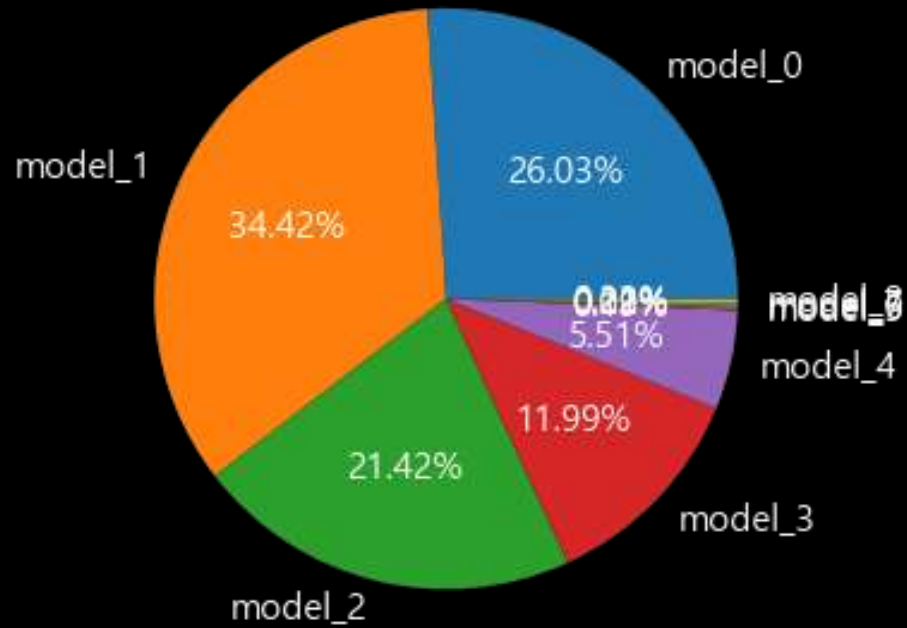
```
1 # model_6를 제외하고 모델별로 4번째까지 같은 펌웨어 번호를 사용한다.
2
3 for nm in range(len(model_fwver.index)):
4     print(f"{model_fwver.model_nm[nm]} -> \n {sorted(model_fwver['fwver'][nm])}")

model_0 ->
['04.22.1442', '04.22.1656', '04.22.1666', '04.22.1684', '04.22.1750', '04.22.1778']
model_1 ->
['04.16.2641', '04.16.3345', '04.16.3439', '04.16.3553', '04.16.3569', '04.16.3571']
model_2 ->
['04.33.1095', '04.33.1125', '04.33.1149', '04.33.1171', '04.33.1185', '04.33.1261']
model_3 ->
['05.15.2090', '05.15.2092', '05.15.2114', '05.15.2120', '05.15.2122', '05.15.2138', '05.15.3104']
model_4 ->
['03.11.1141', '03.11.1149', '03.11.1167']
model_5 ->
['04.82.1684', '04.82.1730', '04.82.1778']
model_6 ->
['10', '8.5.3']
model_7 ->
['05.66.3237', '05.66.3571']
model_8 ->
['04.73.2237', '04.73.2571']
```

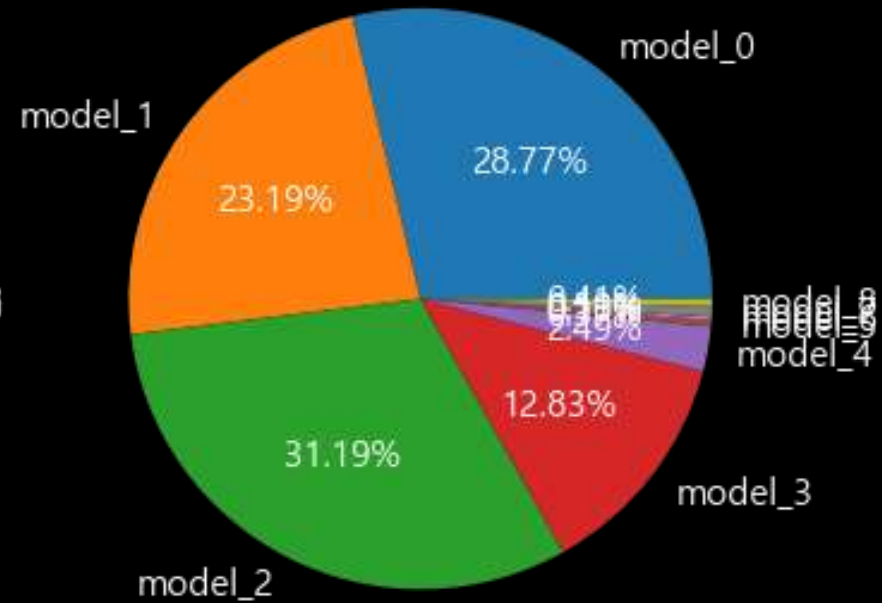
3. EDA: train_err

1. model_nm, fwver column

<모델별 에러 발생율>



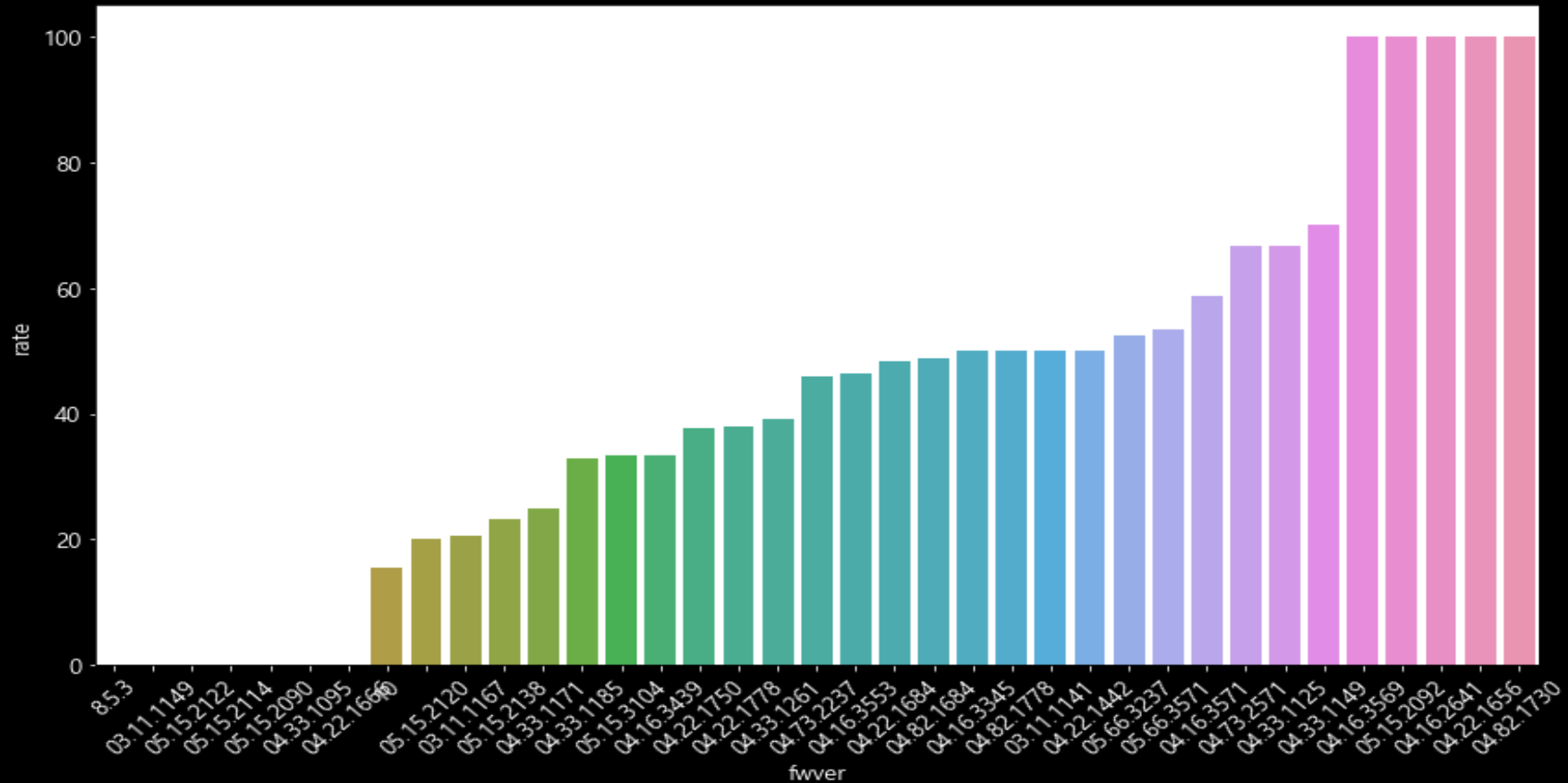
<모델별 불만 제기율>



3. EDA: train_err

1. model_nm, fwver column

<fwver별 불만 제기율>



3. EDA: train_err

1. model_nm, fwver column

<model 변경 후 유저들의 불만 제기율>

```
1 user_sum = len(user_has_2models)
2 user_prob_sum = len(list(train_problem[train_problem['user_id'].isin(user_has_2models)].user_id.unique()))
3
4 print(f"모델이 변경되고 불만을 제기한 사람 {user_prob_sum/user_sum*100}% ")
```

모델이 변경되고 불만을 제기한 사람 91.4651493598862%

<fwver변경(업데이트) 후 불만 제기율>

```
1 user_sum = len(user_has_2fwvers)
2 user_prob_sum = len(list(train_problem[train_problem['user_id'].isin(user_has_2fwvers)].user_id.unique()))
3
4 print(f"fwver가 변경되고 불만을 제기한 사람 {user_prob_sum/user_sum*100}% ")
```

fwver가 변경되고 불만을 제기한 사람 39.696969696969695%

3. EDA: train_err

1. errtype, errcode column

<errtype- errcode의 관계>

```
1 train_err.groupby('errtype')['errcode'].unique().to_frame().reset_index()
```

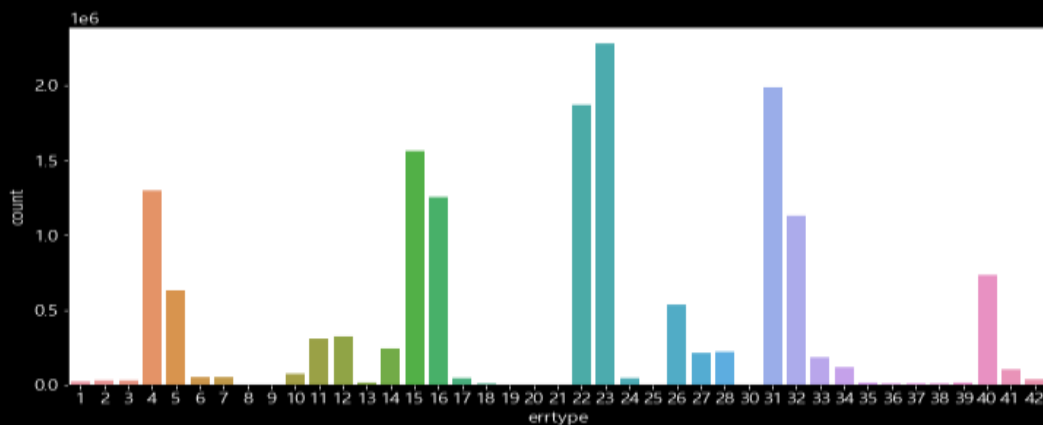
	errtype	errcode
0	1	[0, P-44010, P-41011, P-41007, P-44010, P-41...
1	2	[1, 0]
2	3	[1, 2, 0]
3	4	[0, 1]
4	5	[B-A8002, Q-64002, S-61001, U-81009, V-21008, ...
5	6	[1, 14]
6	7	[1, 14]
7	8	[PHONE_ERR, PUBLIC_ERR, 20]
8	9	[V-21002, V-21005, 1, C-14014, V-21008, C-1203...
9	10	[1]
10	11	[1]
11	12	[1]
12	13	[1]
13	14	[1, 14, 13]
14	15	[1]
15	16	[1]
16	17	[14, 13, 1, 21, 12]
17	18	[1]
18	19	[1]
19	20	[1]
20	21	[1]
21	22	[1]
22	23	[standby, active, connection timeout, terminat...
23	24	[1]

24	25	[2, scanning timeout, 1, UNKNOWN, terminate by...
25	26	[1]
26	27	[1]
27	28	[1]
28	30	[4, 0, 1, 3, 2]
29	31	[1, 0]
30	32	[80, 79, 81, 86, 84, 77, 78, 85, 90, 89, 88, 8...
31	33	[2, 3, 1]
32	34	[4, 1, 2, 3, 6, 5]
33	35	[1]
34	36	[8, 0]
35	37	[0, 1]
36	38	[6796, 5738, 6467, 4893, 5507, 39391, 3113, 36...
37	39	[1, 0]
38	40	[1, 0]
39	41	[NFANDROID2]
40	42	[3, 2]

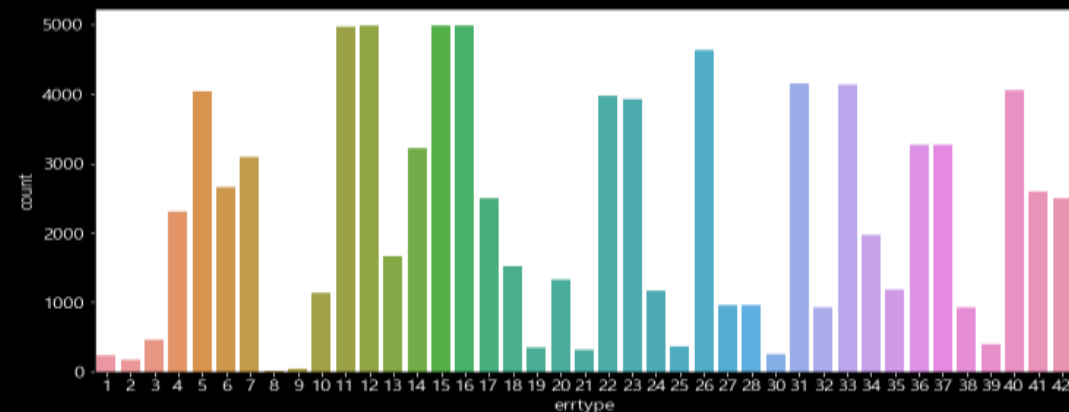
3. EDA: train_err

1. errtype, errcode column

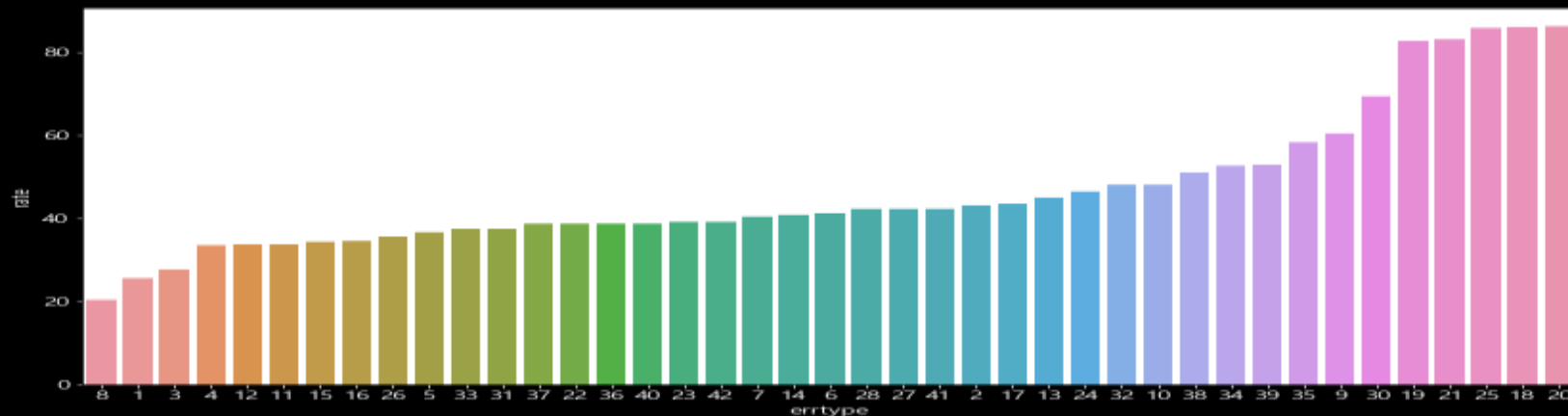
<errtype별 에러 발생 횟수>



<errtype별 불만 제기 횟수>



<errtype별 불만 제기 비율>



3. EDA: train_quality

< -1값을 에러값이라 가정하고 여러 개의 -1값을 갖는 유저들의 불만제기율 분석 >

```
1 # 상위 100명
2 problem_user = len(train_problem[train_problem['user_id'].isin(has_minus_user[:100]['user_id'])].user_id.unique())
3 print(f"상위 100명중 불만을 제기한 사람: {problem_user/100 * 100}%")
4
5 # 상위 500명
6 problem_user = len(train_problem[train_problem['user_id'].isin(has_minus_user[:500]['user_id'])].user_id.unique())
7 print(f"상위 500명중 불만을 제기한 사람: {problem_user/500 * 100}%")
8
9 # 전체 유저 수
10 problem_user = len(train_problem[train_problem['user_id'].isin(has_minus_user['user_id'])].user_id.unique())
11 print(f"전체 유저 중 불만을 제기한 사람: {problem_user/5567 * 100}%")
```

상위 100명중 불만을 제기한 사람: 71.0%

상위 500명중 불만을 제기한 사람: 61.4%

전체 유저 중 불만을 제기한 사람: 41.67415124842824%

3. EDA: train_quality

<double checking>

```
1 # 1값을 갖는 유저들
2 has_1_user = train_quality[train_quality.quality_1 == 1].groupby('user_id')['quality_1'].count().to_frame().reset_index()
3 has_1_user = has_1_user.sort_values(ascending=False, by='quality_1')
4
5
6 # 상위 100명
7 problem_user = len(train_problem[train_problem['user_id'].isin(has_1_user[:100]['user_id'])].user_id.unique())
8 print(f"상위 100명중 불만을 제기한 사람: {problem_user/100 * 100}%")
9
10 # 상위 500명
11 problem_user = len(train_problem[train_problem['user_id'].isin(has_1_user[:500]['user_id'])].user_id.unique())
12 print(f"상위 500명중 불만을 제기한 사람: {problem_user/500 * 100}%")
13
14 # 전체 유저 수
15 problem_user = len(train_problem[train_problem['user_id'].isin(has_1_user['user_id'])].user_id.unique())
16 print(f"전체 유저 중 불만을 제기한 사람: {problem_user/len(has_1_user.index) * 100}%")
```

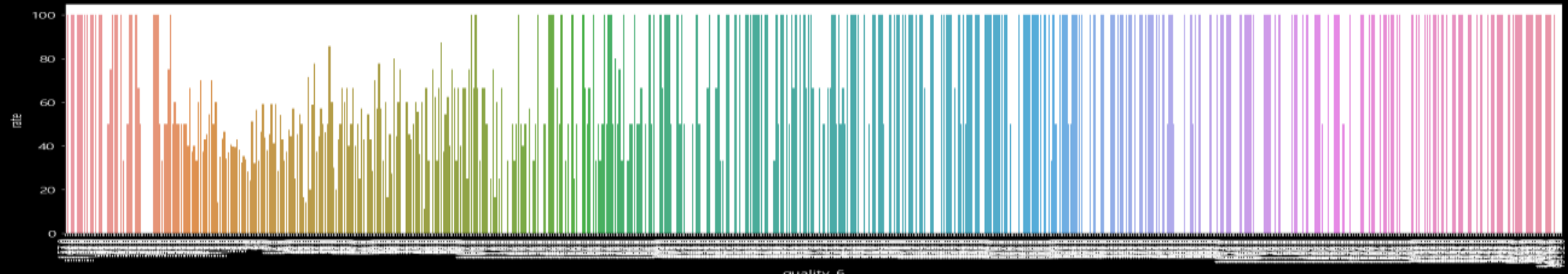
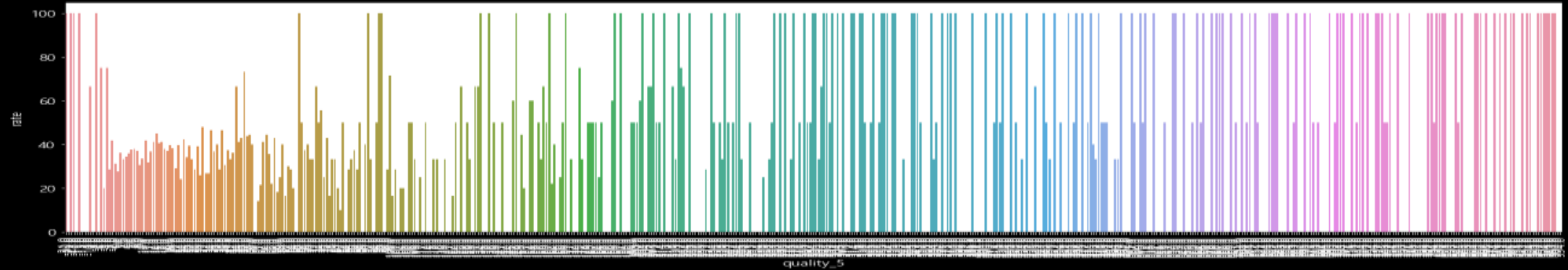
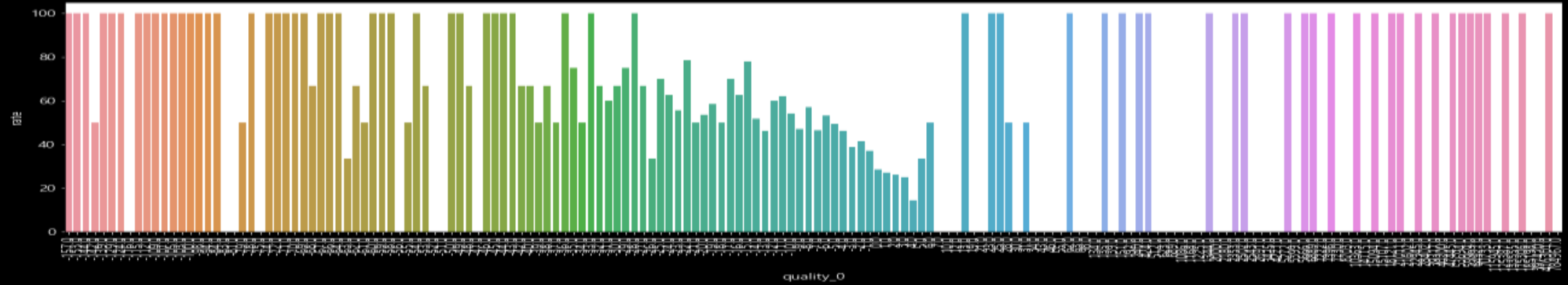
상위 100명중 불만을 제기한 사람: 61.0%
상위 500명중 불만을 제기한 사람: 56.2%
전체 유저 중 불만을 제기한 사람: 53.36225596529284%

```
1 # 0값을 갖는 유저들
2 has_0_user = train_quality[train_quality.quality_0 == 0].groupby('user_id')['quality_0'].count().to_frame().reset_index()
3 has_0_user = has_0_user.sort_values(ascending=False, by='quality_0')
4
5
6 # 상위 100명
7 problem_user = len(train_problem[train_problem['user_id'].isin(has_0_user[:100]['user_id'])].user_id.unique())
8 print(f"상위 100명중 불만을 제기한 사람: {problem_user/100 * 100}%")
9
10 # 상위 500명
11 problem_user = len(train_problem[train_problem['user_id'].isin(has_0_user[:500]['user_id'])].user_id.unique())
12 print(f"상위 500명중 불만을 제기한 사람: {problem_user/500 * 100}%")
13
14
15 # 전체 유저 수
16 problem_user = len(train_problem[train_problem['user_id'].isin(has_0_user['user_id'])].user_id.unique())
17 print(f"전체 유저 중 불만을 제기한 사람: {problem_user/len(has_0_user.index) * 100}%")
```

상위 100명중 불만을 제기한 사람: 67.0%
상위 500명중 불만을 제기한 사람: 56.99999999999999%
전체 유저 중 불만을 제기한 사람: 38.39698370226222%

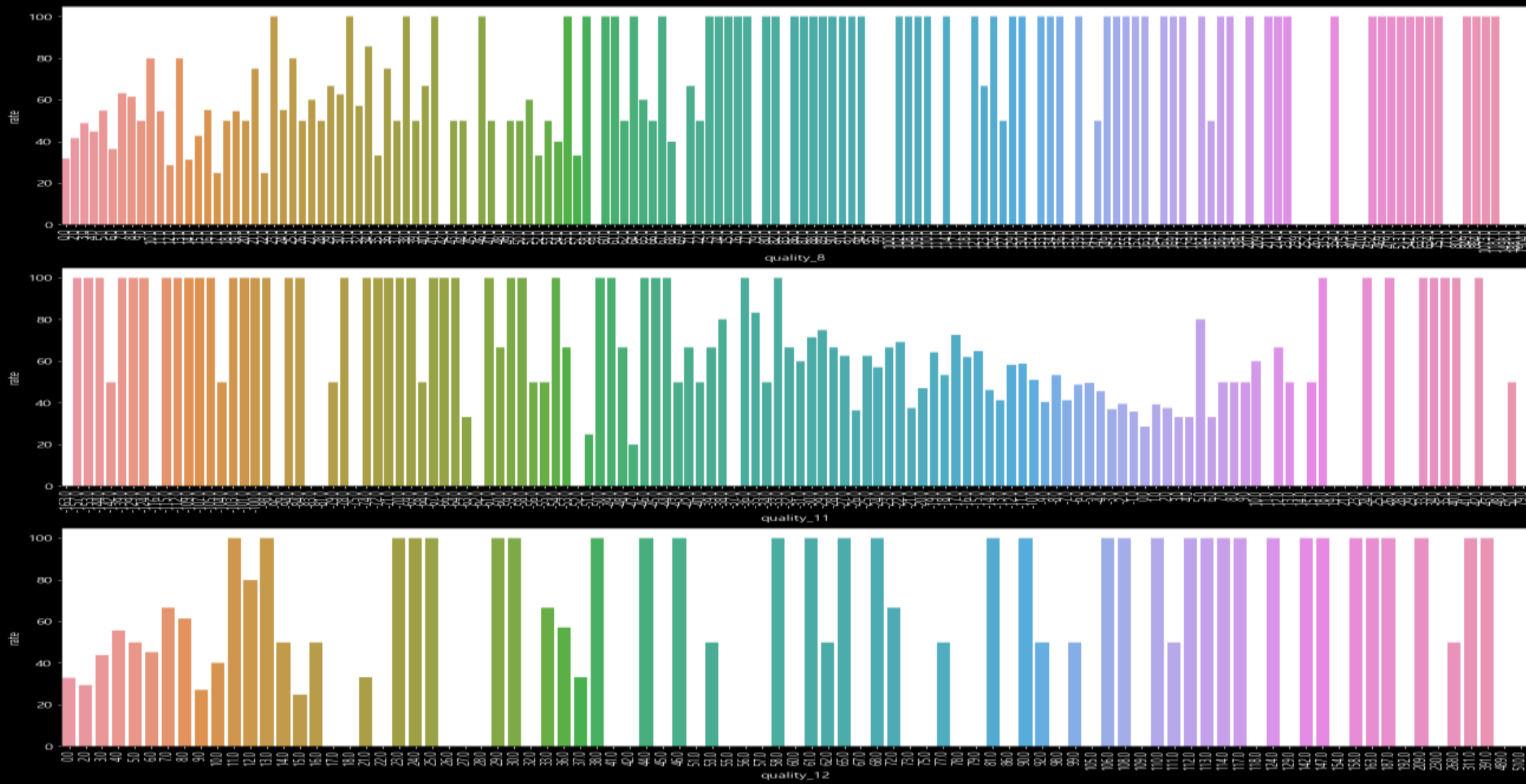
3. EDA: train_quality

<모든 quality값의 합>



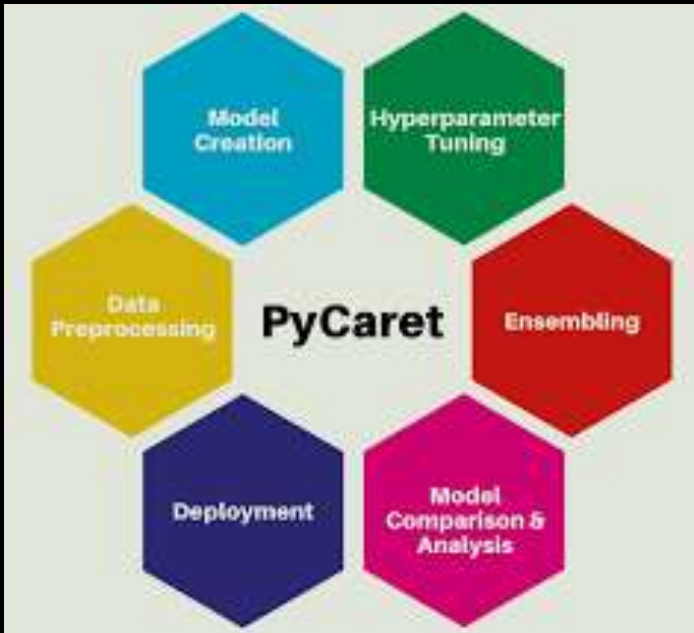
3. EDA: train_quality

<모든 quality값의 합>



4. AUTO ML

Pycaret



- 오픈소스 머신러닝 라이브러리
- machine learning workflows 자동화
- 적은 코드로 강력한 end-to-end ML 솔루션
- 단 몇줄만으로 대부분의 ML 모델로 학습 및 평가 가능
- Kaggle, Dacon에서 많은 사람들이 사용중

4. AUTO ML

Pycaret 예제

```
from pycaret.classification import *
import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
from tqdm import tqdm
import gc
import random
import lightgbm as lgb
import re
from sklearn.metrics import *
from sklearn.model_selection import KFold
import warnings
warnings.filterwarnings(action='ignore')

#clf = setup(data = train, target = "problem")
clf = setup(train, target = 'problem', train_size = 0.85)

# best 3 모델 구하기
best_3 = compare_models(sort = 'AUC', n_select = 3)

# best 3 모델 blending하여 하나의 모델 만들기
blended = blend_models(estimator_list = best_3, fold = 5, method = 'soft')

# hyper parameter tuning
tuned_final_model = tune_model(blended, n_iter = 50)

# tuned_final_model 모델 성능평가
pred_holdout = predict_model(tuned_final_model)
```

- 단 5줄로 데이터 전처리,
모델 학습, best model, blending,
하이퍼 파라미터 튜닝,
성능 평가 가능
- setup() 함수에 많은 옵션들이 존재
(PCA, feature_selection,
normalize, log_experiment 등)
- 더 많은 내용은 Pycaret docs 참고>
<https://pycaret.gitbook.io/docs/>

4. AUTO ML

Pycaret 장단점

장점

편리성

1. 모델 학습 및 예측 과정을 간단한 코드로 대체
2. 다양한 모델들을 이용하여 **best** 모델을 찾고, **blending** 또한 용이
3. scaling, PCA, 교차검증 등 **부가적인 처리과정들이**
hyper parameter 튜닝을 통해 **옵션으로 제공**

단점

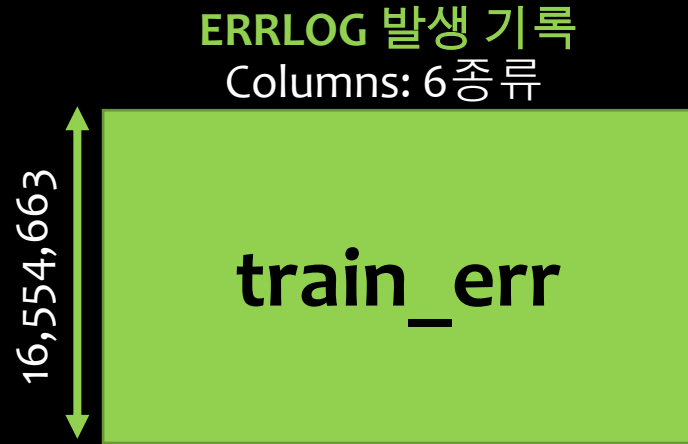
정확한 이해
필요

1. 여러 기능 이용을 위해서는 정확한 사용법(docs) 파악 필수
2. 한계와 제약 사항(많음)을 파악할 필요
3. Pycaret의 다양한 옵션들을 사용하지 못한다면
누구나 도달 가능한 같은 결론(경쟁력 ↓)

5. 성능향상 방법

- 1) Feature 생성 및 선별 ← (가장 큰 요인)
- 2) (분류/회귀 방법 변경)
- 3) 다양한 ML모델 적용 및 보팅
- 4) PCA, scaling, 부차적 과정

5. 성능향상 방법: 1) feature 생성 및 선별



```
err_train.head(3)
```

[6] ✓ 0.5s

	user_id	time	model_nm	fwver	errtype	errcode
0	10000	20201101025616	model_3	05.15.2138	15	1
1	10000	20201101030309	model_3	05.15.2138	12	1
2	10000	20201101030309	model_3	05.15.2138	11	1

model_nm : ERRLOG 발생시의 모델 (9종류)

fwver : ERRLOG 발생시의 펌웨어 버전 (37종류)

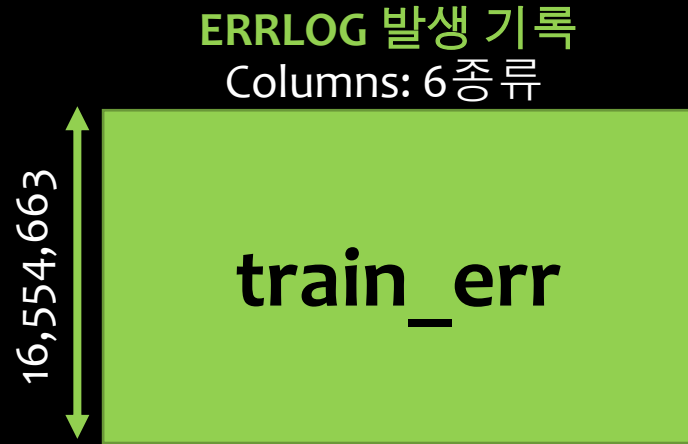
errtype : 발생한 ERRLOG의 타입 (41종류)

errcode : 발생한 에러코드 (2805종류)

One-Hot-Encoding

가공하여 활용

5. 성능향상 방법: 1) feature 생성 및 선별



```
err_train.head(3)
```

[6] ✓ 0.5s

	user_id	time	model_nm	fwver	errtype	errcode
0	10000	20201101025616	model_3	05.15.2138	15	1
1	10000	20201101030309	model_3	05.15.2138	12	1
2	10000	20201101030309	model_3	05.15.2138	11	1

model_nm : ERRLOG 발생시의 모델 (9종류)
fwver : ERRLOG 발생시의 펌웨어 버전 (37종류)
errtype : 발생한 ERRLOG의 타입 (41종류)

One-Hot-Encoding

1인당 평균 1,000건의 ERRLOG
ERRLOG 발생 대상이
같은 모델, 같은 펌웨어버전,
같은 에러 타입만 있는 것이 아니다



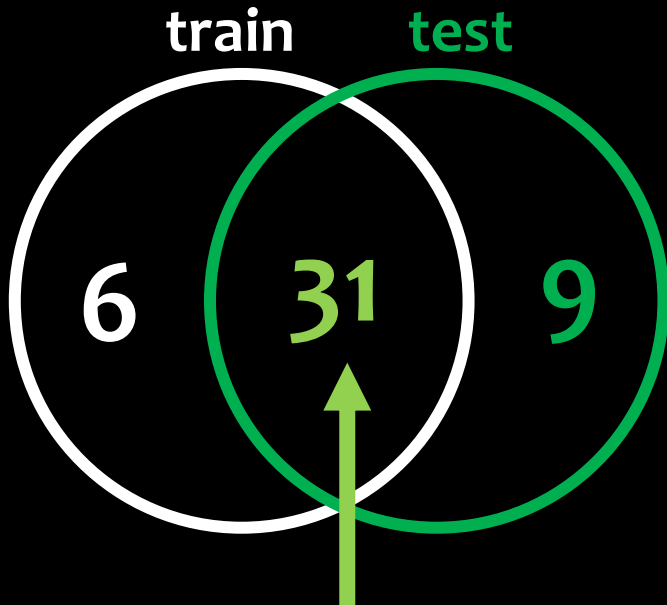
`groupby(user_id).max()` : 유무
`groupby(user_id).sum()` : 횟수

5. 성능향상 방법: 1) feature 생성 및 선별

One Hot Encodig Issue

fwver : ERRLOG 발생시의 펌웨어 버전
unique_values: train: 37종류, test: 40종류

One-Hot-Encoding



31 개의 공통소속 unique_values만 이용 → over fitting 방지

```
pd.get_dummies( data=train, columns=["fwver"] )  
pd.get_dummies( data=test , columns=["fwver"] )
```

Python

```
all_df = pd.concat( [train, test], axis=0 )  
pd.get_dummies( data=all_df, columns=["fwver"] )  
train = all_df.loc( all_df["user_id"]<30000 )  
test = all_df.loc( all_df["user_id"]>=30000 )
```

Python

각각 size 달라짐

→ 37, 40개

동일기준 적용

→ 46, 46개

5. 성능향상 방법: 1) feature 생성 및 선별

ERRLOG 발생 기록
Columns: 6종류

16,554,663
train_err

```
err_train.head(3)
```

	user_id	time	model_nm	fwver	errtype	errcode
0	10000	20201101025616	model_3	05.15.2138	15	1
1	10000	20201101030309	model_3	05.15.2138	12	1
2	10000	20201101030309	model_3	05.15.2138	11	1

errcode: 발생한 에러코드

(2805종류)

errcode별 신고율

```
print( errcode_rate.shape )  
display( errcode_rate.head() )
```

(2805, 4)

	errcode	count_all	count_1	complain
0	1	8097696	3576404	0.441657
1	0	2594264	928249	0.357808

Step1) 2,805개의 각 **errcode**별 불편신고율 계산
Step2) 각 유저별 발생한 **errcode**들에 따라
해당 **errcode**들의 각각의 불편신고율을
sum 또는 **max**하여 feature로 활용

5. 성능향상 방법: 1) feature 생성 및 선별



```
quality_train.head(3)
```

[10] ✓ 0.1s

...

	time	user_id	fwver	quality_0	quality_1	quality_12
0	20201129090000	10000	05.15.2138	0.0	0	0
1	20201129090000	10000	05.15.2138	0.0	0	0
2	20201129090000	10000	05.15.2138	0.0	0	0

user_id : 15,000명 중, 8281명(절반의 인원이 결측)

time : 일부 기간의 기록만 존재
(예) 10000 유저 11월 한달 중 29, 30일 뿐

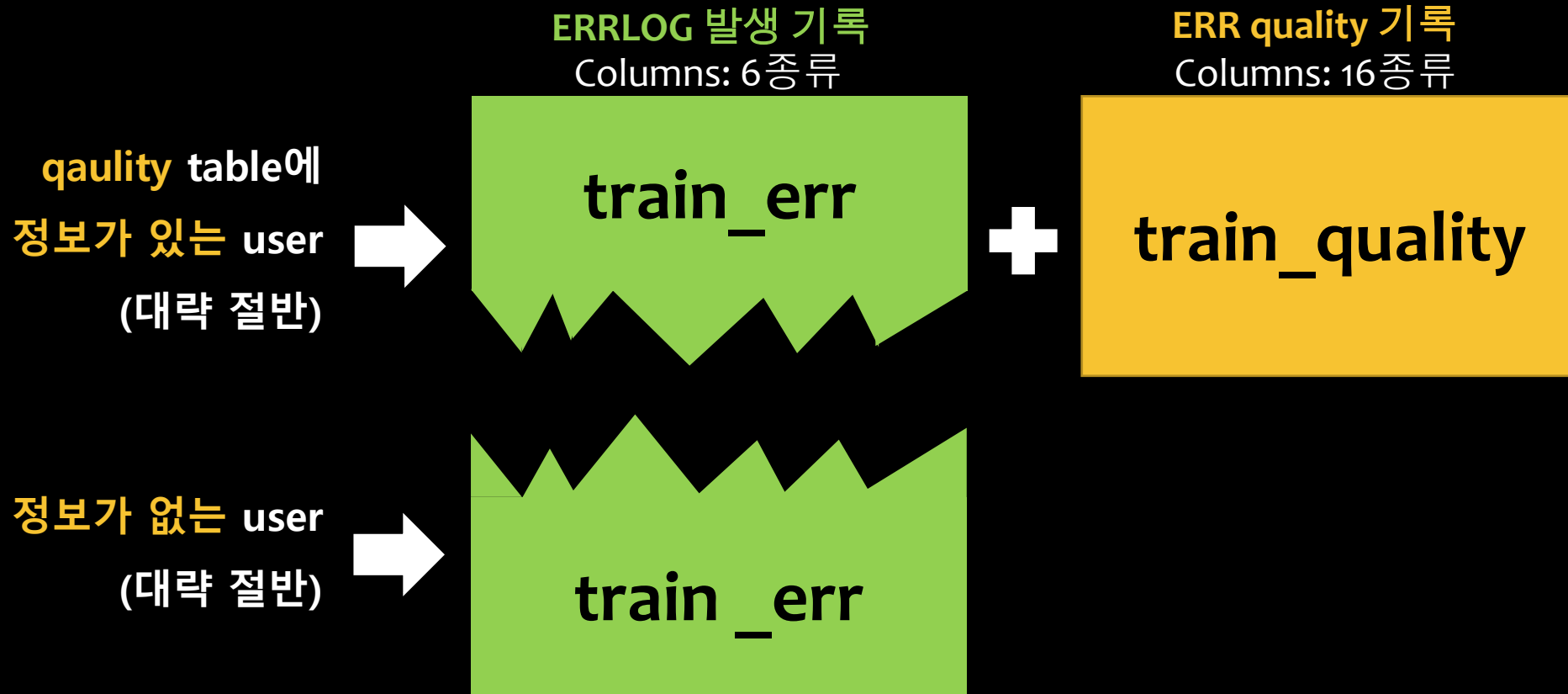
quality_0~12 : quality_0 컬럼의 경우 결측치 10% 이상



굳이 사용을
해야 하는가?

5. 성능향상 방법: 1) feature 생성 및 선별

대안: 처음부터 user기준, quality table에 정보 존재 여부에 따라 data 분할



5. 성능향상 방법: 1) feature 생성 및 선별



```
quality_train.head(3)
```

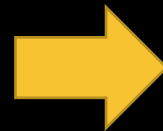
[10] ✓ 0.1s

...

	time	user_id	fwver	quality_0	quality_1	quality_12
0	20201129090000	10000	05.15.2138	0.0	0	0
1	20201129090000	10000	05.15.2138	0.0	0	0
2	20201129090000	10000	05.15.2138	0.0	0	0

quality_0~12 : -1, 0, 1, ..., 1,910,175 (정수)
대부분의 값이 -1, 0 에 분포
quality_0 컬럼의 경우 결측치 10% 이상

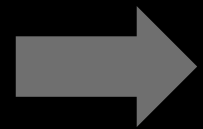
1인당 평균 34건의 quality 기록
1인당 각 quality_N 컬럼들의 값을
새로운 피쳐로 가공해야 한다



수치값 빈도수에 따라 0, 1, 양수 범주로 구분
groupby(user_id).max(): 유무
groupby(user_id).sum(): 횟수

5. 성능향상 방법: 1) feature 생성 및 선별

feature 생성만큼 중요한 feature 선별 (다중공선성 완화)



1. feature별 유의미성 직접 판단

2. 상관계수corr() 이용

3. PCA

```
err_train.head(3)
```

[6] ✓ 0.5s

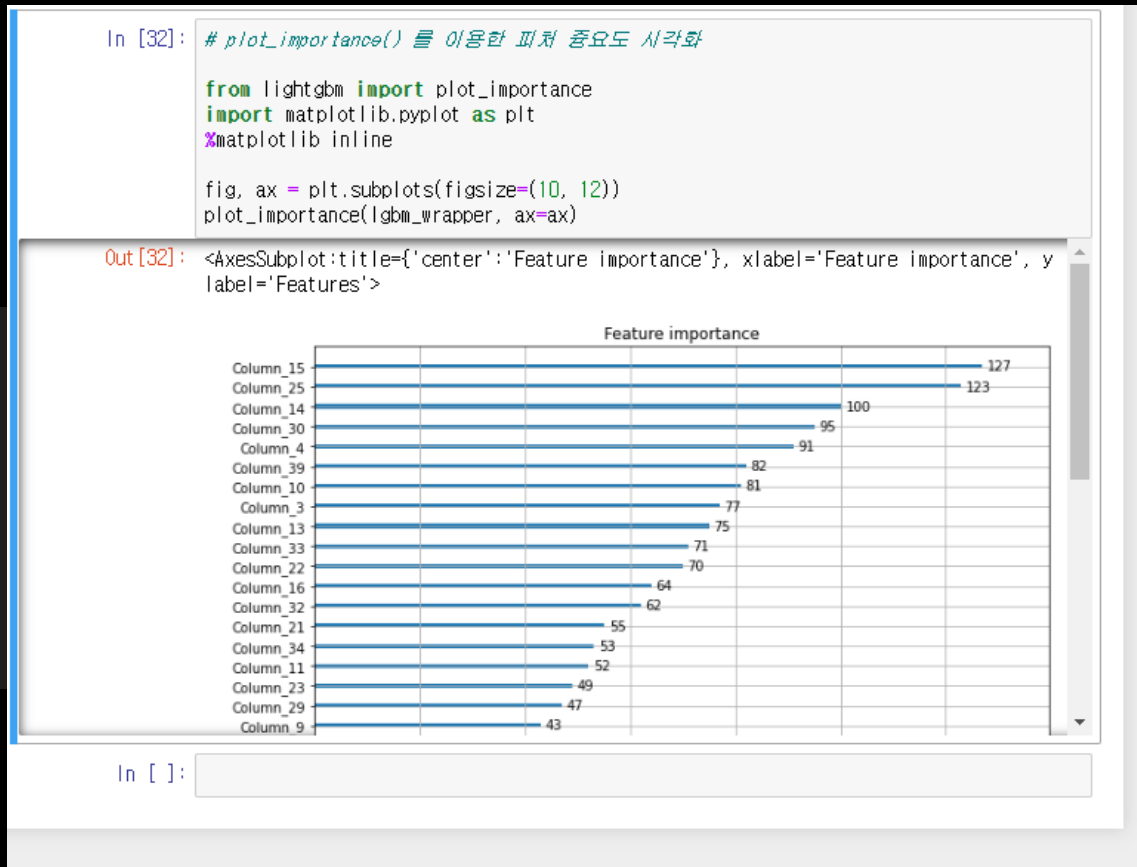
	user_id	time	model_nm	fwver	errtype	errcode
0	10000	20201101025616	model_3	05.15.2138	15	1
1	10000	20201101030309	model_3	05.15.2138	12	1
2	10000	20201101030309	model_3	05.15.2138	11	1

어떤 columns 사용?

5. 성능향상 방법: 1) feature 생성 및 선별

feature 생성만큼 중요한 feature 선별 (다중공선성 완화)

➡ 1. feature별 유의미성 직접 판단



plot_importance()

→ 컬럼별 중요도 파악

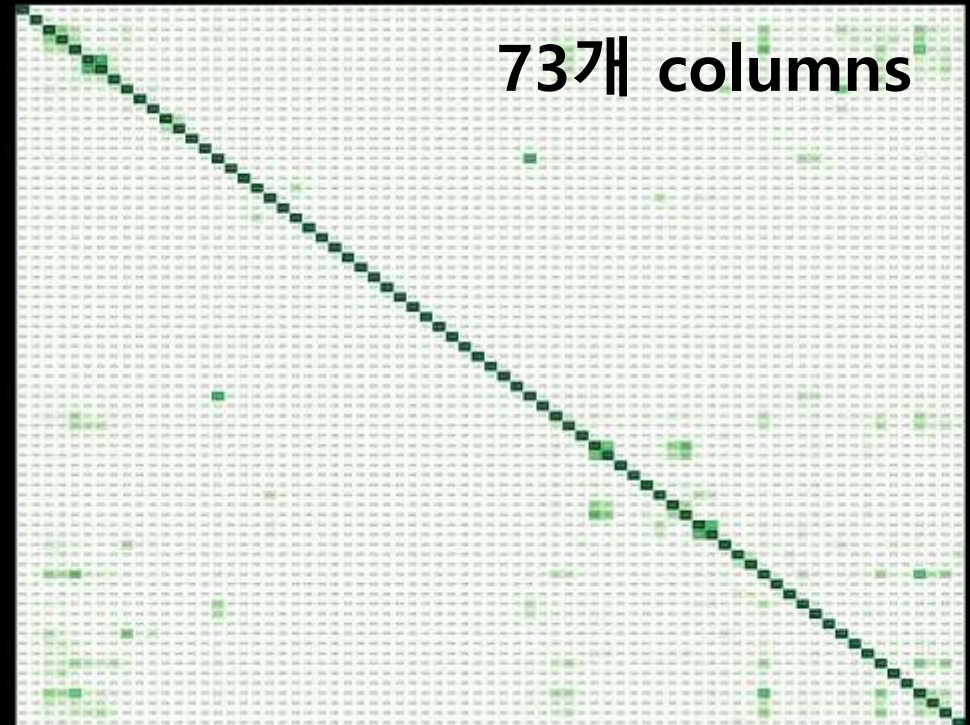
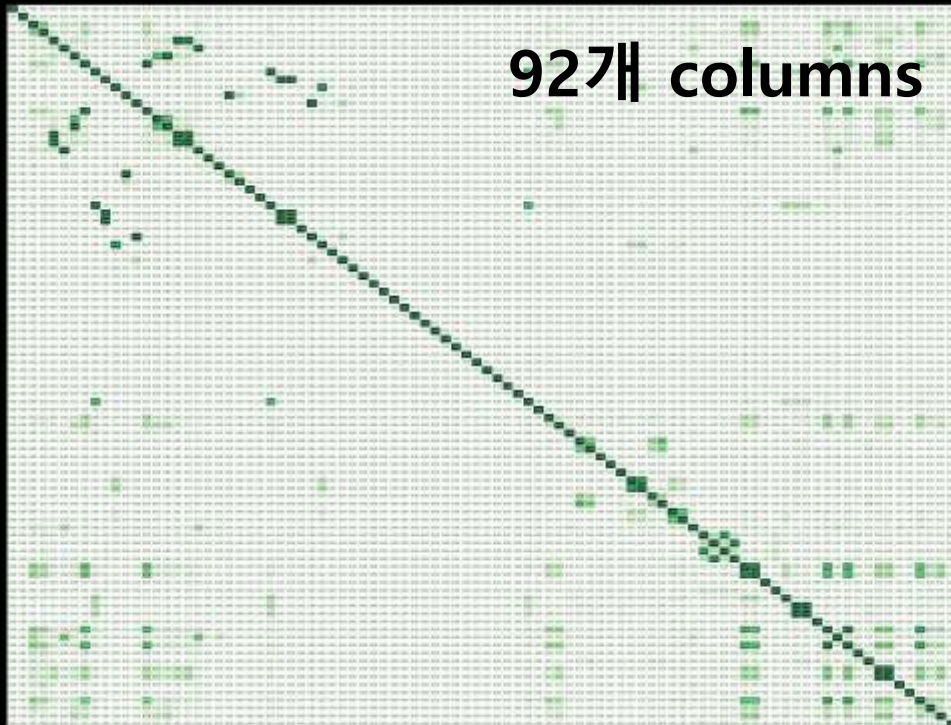
5. 성능향상 방법: 1) feature 생성 및 선별

feature 생성만큼 중요한 feature 선별 (다중공선성 완화)

➡ 1. feature별 유의미성 직접 판단

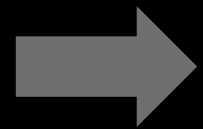
2. 상관계수 `corr()` 이용

상관계수 0.9 넘는 컬럼들 제거



5. 성능향상 방법: 1) feature 생성 및 선별

feature 생성만큼 중요한 feature 선별 (다중공선성 완화)



1. feature별 유의미성 직접 판단

2. 상관계수corr() 이용

3. PCA

```
● # test data에 PCA 적용하는 잘못된 방법
train = pd.DataFrame( PCA().fit_transform(train) )
test  = pd.DataFrame( PCA().fit_transform(test) )

# test data에 PCA 적용하는 올바른 방법
train = pd.DataFrame( PCA().fit(train).transform(train) )
test  = pd.DataFrame( PCA().fit(train).transform(test) )
```

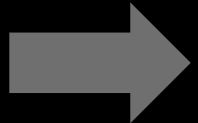
train 으로 fit하고,
train, test에 동일하게 적용한다

5. 성능향상 방법: 2) 분류/회귀 방법 변경

사건의 발단

분류모델의 성능평가지표: AUC → 분류문제.

최종적으로 제출할 결과값: 불편신고 유무(0 or 1)가 아닌
불편신고 확률(0~1, 실수값)



회귀모델로 풀면 더 정확할 수도 있지 않을까?

결론

1. Label이 없음: 불편신고 유무만 있음
2. 성능평가지표 AUC를 활용할 수 없음.
→ 문제에서 요구한 최적의 모델 확인 불가

∴ 분류모델보다 성능이 훨씬 떨어짐

5. 성능향상 방법: 3) 다양한 ML모델 적용 및 보팅

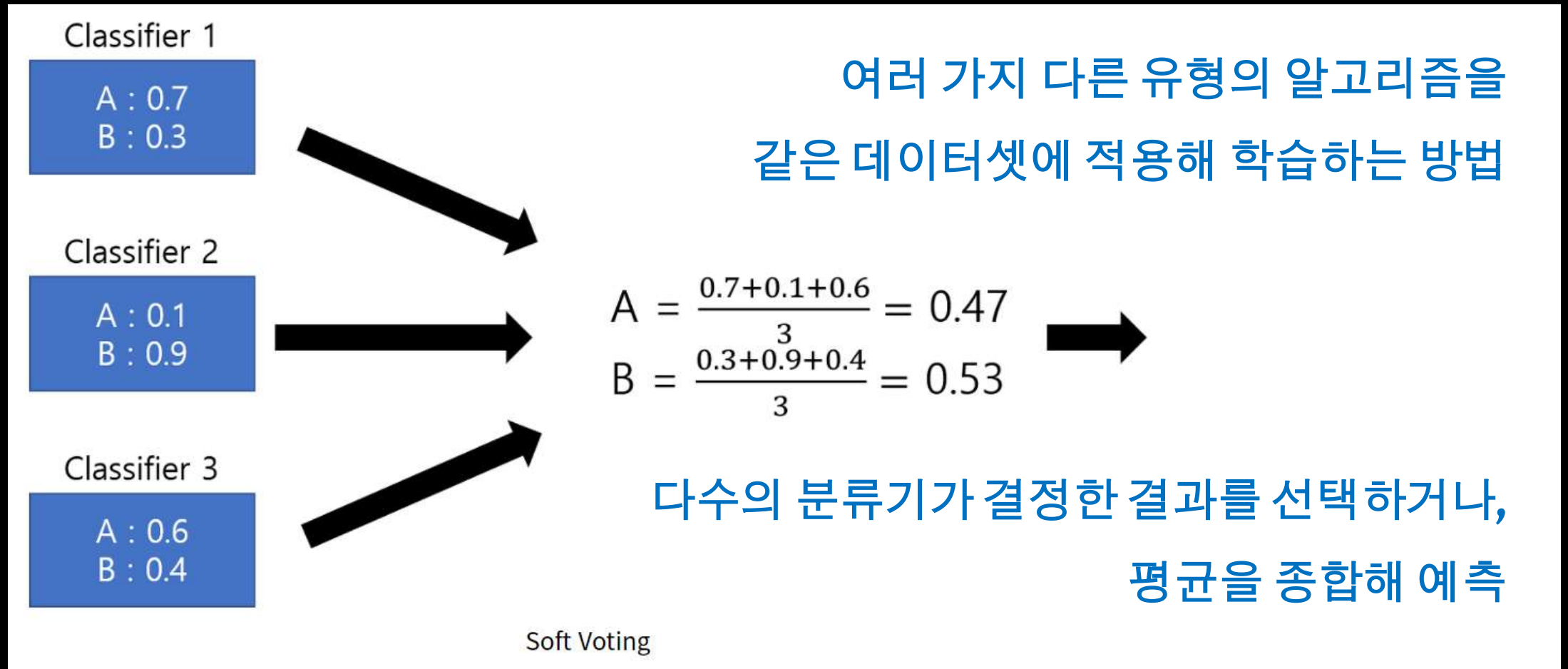
Pycaret을 활용한 AUC기준 최적의 모델 찾기

```
1 best_3 = compare_models(sort = 'AUC', n_select = 3)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.7909	0.8075	0.4779	0.8155	0.6025	0.4735	0.5047	
lightgbm	Light Gradient Boosting Machine	0.7891	0.8046	0.5109	0.7774	0.6164	0.4794	0.4996	
rf	Random Forest Classifier	0.7880	0.8035	0.4885	0.7927	0.6044	0.4705	0.4963	
et	Extra Trees Classifier	0.7816	0.7964	0.4759	0.7800	0.5909	0.4537	0.4795	
ada	Ada Boost Classifier	0.7835	0.7940	0.4842	0.7795	0.5972	0.4601	0.4845	
lr	Logistic Regression	0.7766	0.7696	0.4253	0.8114	0.5579	0.4273	0.4673	
lda	Linear Discriminant Analysis	0.7625	0.7506	0.3564	0.8313	0.4986	0.3741	0.4319	
nb	Naive Bayes	0.6985	0.7342	0.5060	0.6250	0.5271	0.3223	0.3395	
knn	K Neighbors Classifier	0.7455	0.7207	0.4276	0.6874	0.5270	0.3656	0.3850	
dt	Decision Tree Classifier	0.6956	0.6588	0.5505	0.5404	0.5452	0.3166	0.3167	

5. 성능향상 방법: 3) 다양한 ML모델 적용 및 보팅

Pycaret을 활용한 soft voting → 단일모델보다 성능 우수



5. 성능향상 방법: 4) PCA, scaling, 부차적 과정

Scaling 여부가 성능에 큰 차이 주지 않음




before

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Voting Classifier	0.7943	0.8051	0.4938	0.7852	0.6063	0.4767	0.5002

after

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Voting Classifier	0.7967	0.8068	0.4979	0.7910	0.6111	0.4830	0.5067

5. 성능향상 방법: 4) PCA, scaling, 부차적 과정

대회안내	데이터	코드 공유	토크	리더보드	제출
PUBLIC	PRIVATE	AWARDS	RANKING CHART	순위기준	
<div>● WINNER ● 1% ● 4% ● 10%</div>					전체 랭킹 >
#	팀	팀 멤버	최종점수	제출수	등록일
1	핏백runners		0.84231	41	일 년 전
2	Kang		0.84864	85	일 년 전
3	계란후라이		0.84022	63	일 년 전

681777

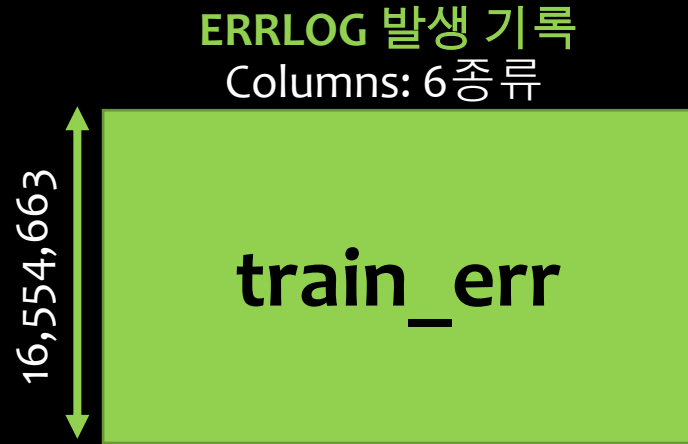
submission_ver6.csv
pycaret_ver6_optuna edit

최종 달성한 모델 성능

2022-06-17
15:42:50

0.8206606995
0.8208881245

5. 성능향상 방법: 4) PCA, scaling, 부차적 과정



```
err_train.head(3)
```

[6] ✓ 0.5s

	user_id	time	model_nm	fwver	errtype	errcode
0	10000	20201101025616	model_3	05.15.2138	15	1
1	10000	20201101030309	model_3	05.15.2138	12	1
2	10000	20201101030309	model_3	05.15.2138	11	1

model_nm : ERRLOG 발생시의 모델 (9종류)

feature



각 유저별 ERRLOG 발생한 model이
2종류 이상인 사람: 1, 나머지: 0

optuna



hyper
parameter

ML 구동

5. 성능향상 방법: 4) PCA, scaling, 부차적 과정

<model 변경 후 유저들의 불만 제기율>

```
1 user_sum = len(user_has_2models)
2 user_prob_sum = len(list(train_problem[train_problem['user_id'].isin(user_has_2models)].user_id.unique()))
3
4 print(f"모델이 변경되고 불만을 제기한 사람 {user_prob_sum/user_sum*100}% ")
```

모델이 변경되고 불만을 제기한 사람 91.4651493598862%

단일 feature: 에러 로그 발생한 모델이 2종류 이상인 사람 (0,1 값)

➡ target 과의 압도적인 상관관계!

이번 프로젝트 과제의 특별했던 점

1. 3개의 train_data
2. Work_Flow 중요성
3. Big_data
4. 시계열 dtype

사용자의 불편신고 기록

train_problem

ERR quality 기록

train_quality

ERRLOG 발생 기록

train_err

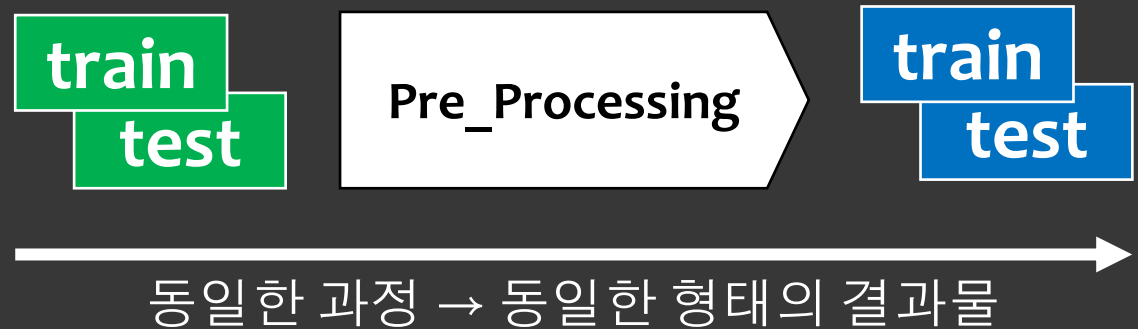
Train data set 3개, Primary Key: 없음 → merge 불가
→ 직접 user별 feature 생성
(한 개의 training set 구성)

이번 프로젝트 과제의 특별했던 점

1. 3개의 train_data
2. Work_Flow 중요성
3. Big_data
4. 시계열 dtype

주어진 Data Table이 많고
품질이 나쁠수록(결측) 복잡한 처리 과정

➡ 체계적인 Work_Flow 설계가 중요!



이번 프로젝트 과제의 특별했던 점

1. 3개의 train_data

2. Work_Flow 중요성

3. Big_data

4. 시계열 dtype

Data Size 커질수록

코딩, ML구동시 성능 뿐 아니라 효율!

※ dtype 변경 예시

202006140000(int) → 2020-06-14 00:00:00(datetime)

24분 err['time'].astype('str').astype('datetime64')

1분

pd.to_datetime(err['time'], format='%Y%m%d%H%M%S')

30초

pd.to_datetime 방식으로 function 제작, 실행

이번 프로젝트 과제의 특별했던 점

1. 3개의 train_data
2. Work_Flow 중요성
3. Big_data
4. 시계열 dtype

```
<class 'pandas.core.frame.DataFrame'>  
Index: 4 entries, 1 to 4  
Data columns (total 4 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   col_A        4 non-null      int64  
1   col_D1        4 non-null      datetime64[ns]  
2   how_long      4 non-null      timedelta64[ns]  
3   day_          4 non-null      period[D]  
dtypes: datetime64[ns](1), int64(1), period[D](1), timedelta64[ns](1)  
memory usage: 332.0+ bytes
```

	col_A	col_D1	how_long	day_
student				
1	10	2022-06-13 15:00:00	163 days 15:00:00	2022-06-13
2	10	2022-06-13 15:10:00	163 days 15:10:00	2022-06-13
3	20	2022-06-13 15:20:00	163 days 15:20:00	2022-06-13
4	20	2022-06-13 15:30:00	163 days 15:30:00	2022-06-13

이번 프로젝트 진행으로 느낀점

1. DataFrame 기초 코딩 능력 및 시각화 테크닉 필수!
2. AutoML 사용시 작동법과 결과물에 대한 이해는 기본
 - 더 나아가 hyper parameter 설정방법 이해 필요
3. ML 모델 선정, hyper parameter 보다 중요한 것은
 - Data의 품질(결측) 및 Data 파악능력 & feature 생성&선택
4. team으로 진행시 다양한 insight 공유 가능