

ML Competition

- 신용카드 대금 연체 정도 예측

Team 머러터저

김서령 김예향 류병하 이은지 황건하

Index

01

EDA
데이터 전처리
분석

02

Feature 생성

03

Model 적용

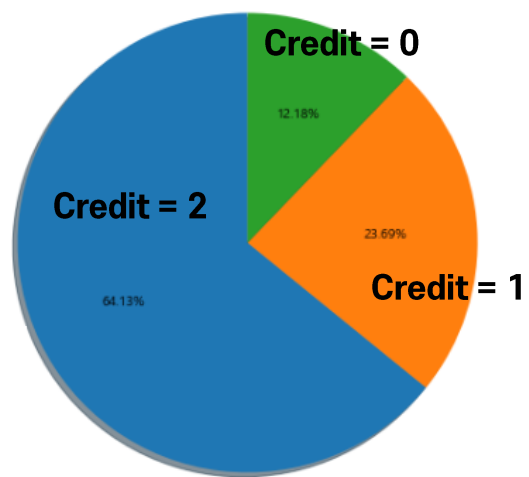
- lgbm
- gb
- catboost

04

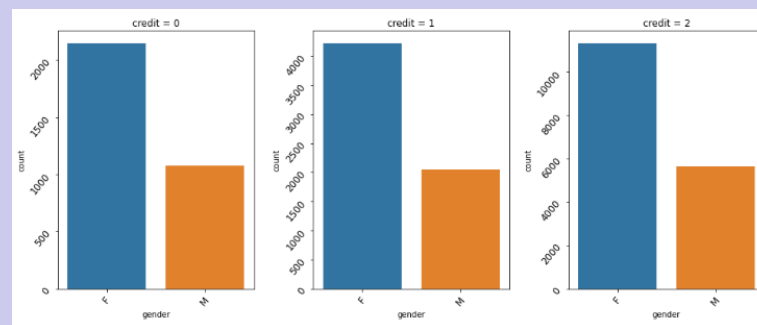
결론
아쉬운 점

EDA

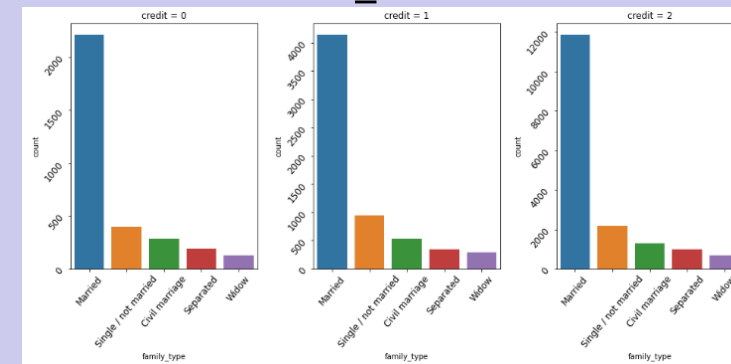
신용등급 비율



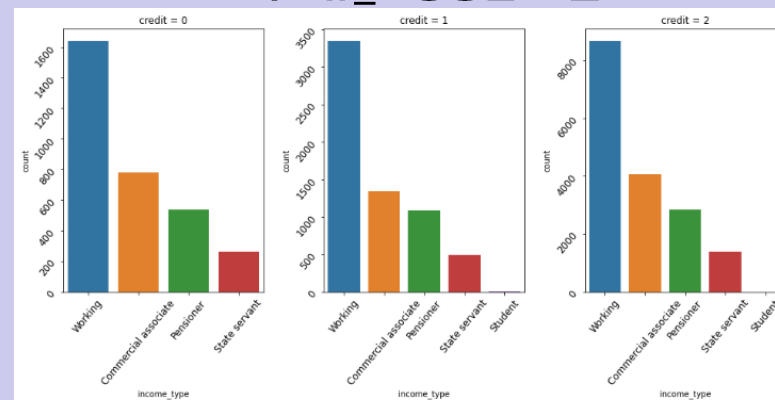
성별_신용등급 비율



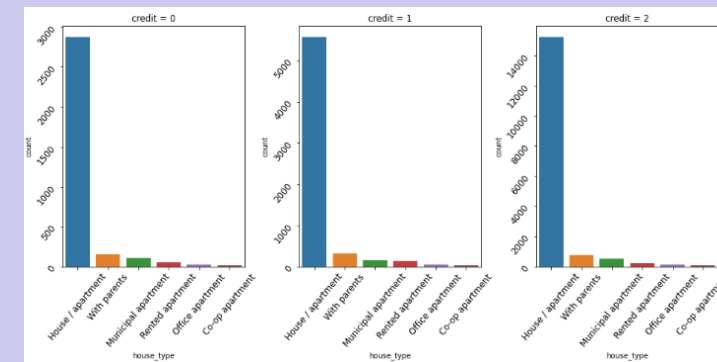
결혼여부_신용등급 비율



소득분류_신용등급 비율



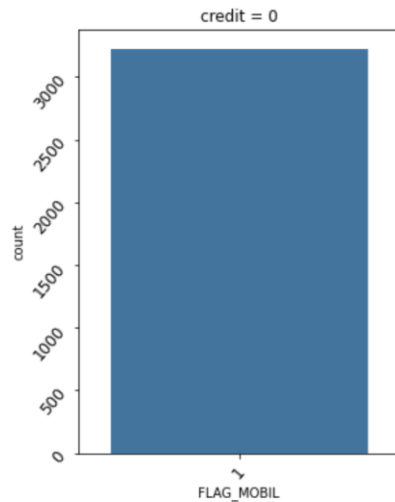
생활방식_신용등급 비율



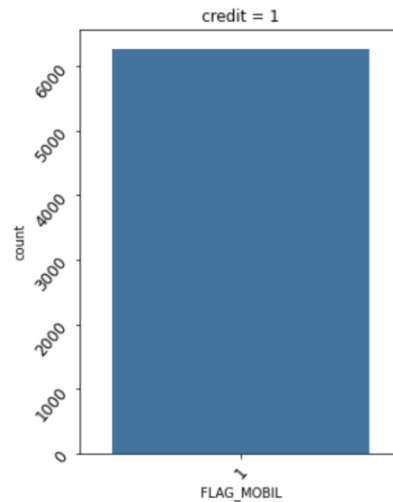
EDA

FLAG_MOBIL

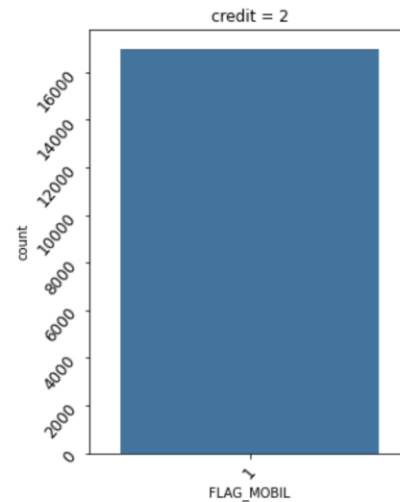
Credit = 0



Credit = 1



Credit = 2



핸드폰 소유 여부
['FLAG_MOBIL']

가치는 값이 모두 1이기 때문에
feature 제거

EDA

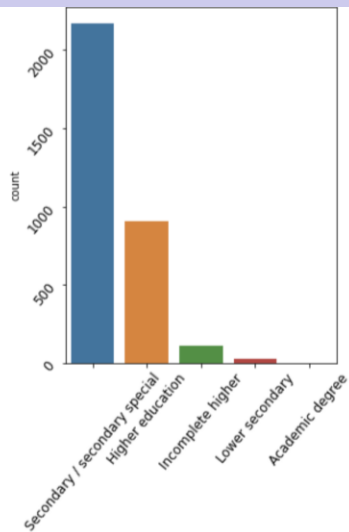
['edu_type']
교육수준

['income_type']
소득분류

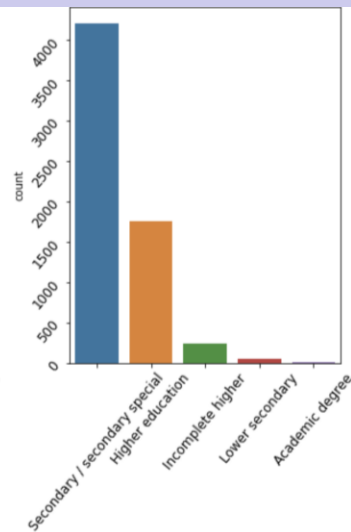
```
train.groupby(['edu_type', 'target'])[['target']].agg('count')
```

Edu_type

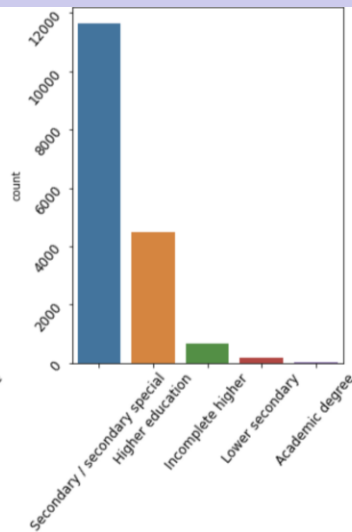
Credit = 0



Credit = 1



Credit = 2



target

edu_type target

Academic degree	0.0	2
	1.0	7
	2.0	14
Higher education	0.0	909
	1.0	1751
	2.0	4502
Incomplete higher	0.0	114
	1.0	246
	2.0	660
Lower secondary	0.0	28
	1.0	59
	2.0	170
Secondary / secondary special	0.0	2169
	1.0	4204
	2.0	11622

신용도 0의 비율

Academic degree 0.08

Higher education 0.126

Incomplete higher 0.111

Lower secondary 0.108

Secondary / secondary special
0.12

전처리 - 이상치

['Family_size']

```
train['family_size'].value_counts()
```

2.0	14106
1.0	5109
3.0	4632
4.0	2260
5.0	291
6.0	44
7.0	9
15.0	3
9.0	2
20.0	1

Name: family_size, dtype: int64

begin_month 열을 제외한 모든 값이 중복

중복

```
1 train[train['family_size'] == 9]
```

	index	gender	car	reality	child_num	income_total	income_type	edu_type	family_type	house_type	DAYS_BIRTH	DAYS_EMPLOYED
25313	25313	F	N	N	7	157500.0	Working	Secondary / secondary special	Married	House / apartment	-13827	-1649
25638	25638	F	N	N	7	157500.0	Working	Secondary / secondary special	Married	House / apartment	-13827	-1649

중복

```
1 train[train['family_size'] == 15]
```

	index	gender	car	reality	child_num	income_total	income_type	edu_type	family_type	house_type	DAYS_BIRTH	DAYS_EMPLOYED
8462	8462	M	Y	Y	14	225000.0	Working	Secondary / secondary special	Separated	House / apartment	-17754	-1689
9021	9021	M	Y	Y	14	225000.0	Working	Secondary / secondary special	Separated	House / apartment	-17754	-1689
25390	25390	M	Y	Y	14	225000.0	Working	Secondary / secondary special	Separated	House / apartment	-17754	-1689

['family_size']의 값이 9, 15, 20 인 데이터가
각각 한명인 동일인물 데이터라 판단하고, 7보다 큰 값 제거

전처리 - 결측치

column:	index	Percent of NaN value: 0.00%
column:	gender	Percent of NaN value: 0.00%
column:	car	Percent of NaN value: 0.00%
column:	reality	Percent of NaN value: 0.00%
column:	child_num	Percent of NaN value: 0.00%
column:	income_total	Percent of NaN value: 0.00%
column:	income_type	Percent of NaN value: 0.00%
column:	edu_type	Percent of NaN value: 0.00%
column:	family_type	Percent of NaN value: 0.00%
column:	house_type	Percent of NaN value: 0.00%
column:	DAYS_BIRTH	Percent of NaN value: 0.00%
column:	DAYS_EMPLOYED	Percent of NaN value: 0.00%
column:	FLAG_MOBIL	Percent of NaN value: 0.00%
column:	work_phone	Percent of NaN value: 0.00%
column:	phone	Percent of NaN value: 0.00%
column:	email	Percent of NaN value: 0.00%
column:	occyp_type	Percent of NaN value: 30.88%
column:	family_size	Percent of NaN value: 0.00%
column:	begin_month	Percent of NaN value: 0.00%
column:	credit	Percent of NaN value: 0.00%

직업유형

['occpy_type']

```
1 train.fillna('NaN', inplace=True)
2 test.fillna('NaN', inplace = True)
```

→ 신용도와 관계가 있을 것이라고 판단

→ NaN 값으로 대체

['occpy_type'] 열에만 약 30%의 결측치가 존재함

전처리

['DAYS_BIRTH'] ['DAYS_EMPLOYED'] ['begin_month']

```
new_features = ['DAYS_BIRTH', 'DAYS_EMPLOYED', 'begin_month']  
for features in new_features:  
    X_train[features] = np.abs(X_train[features])  
    X_test[features] = np.abs(X_test[features])
```

음수 값을 갖는 feature → 양수 값으로 변환



DAYS_BIRTH	DAYS_EMPLOYED	begin_month
13899	4709	6.0
11380	1540	5.0
19087	4434	22.0
15088	2092	37.0
15037	2105	26.0

Feature 생성 방식

1. 도메인 지식 이용

- ['edu_type'] 별 신용도 0의 비율
- ['possible']

3. Groupby → 통계수치 이용

- 소득분류별 연간소득
- 교육수준별 연간소득
- 결혼여부별 연간소득
- 생활방식별 연간소득
- 나이대별 연간소득
- 직업유형별 연간소득

2. 연속형 변수 → 범주화

- ['Age'], ['Age_group']
- ['Years of service'], ['Years_group']
- ['DAYS_BIRTH'] 범주화
- ['begin_month'] 범주화
- ['before_employed']
- ['begore_employed_week']
- ['before_employed_month']

Feature 생성

1. ['edu_type'] 별 신용도 0의 비율

Target과 직접적인 연관이 있는 feature들을 확인

→ ['edu_type']이 target과의 관계에서 가장 유의미한 결과를 보였음.

“edu_type 별 target 비율”

```
train.groupby(['edu_type', 'target'])[['target']].agg('count')
```

		target	
	edu_type	target	
Academic degree	0.0	2	
	1.0	7	
	2.0	14	
Higher education	0.0	909	
	1.0	1751	
	2.0	4502	
Incomplete higher	0.0	114	
	1.0	246	
	2.0	660	
Lower secondary	0.0	28	
	1.0	59	
	2.0	170	
Secondary / secondary special	0.0	2169	
	1.0	4204	
	2.0	11622	

```
a = []  
  
for i in X_train['edu_type']:  
    if i == 'Academic degree':  
        a.append(0.08)  
    elif i == 'Higher education':  
        a.append(0.126)  
    elif i == 'Incomplete higher':  
        a.append(0.111)  
    elif i == 'Lower Secondary':  
        a.append(0.108)  
    else:  
        a.append(0.12)
```

```
X_train['edu_type_num'] = a
```

```
a = []  
  
for i in X_test['edu_type']:  
    if i == 'Academic degree':  
        a.append(0.08)  
    elif i == 'Higher education':  
        a.append(0.126)  
    elif i == 'Incomplete higher':  
        a.append(0.111)  
    elif i == 'Lower Secondary':  
        a.append(0.108)  
    else:  
        a.append(0.12)
```

```
X_test['edu_type_num'] = a
```

Feature 생성

2. ['possible']

개인의 **상환능력**이 신용등급과 연관이 있을 것이라고 예상

➡ 데이터에서 feature들을 조합해 상환능력을 대체할 수 있는 feature를 생성함.

```
X_train['possible'] = X_train['income_total'] / (X_train['DAYS_BIRTH'] + X_train['DAYS_EMPLOYED'])  
X_test['possible'] = X_test['income_total'] / (X_test['DAYS_BIRTH'] + X_test['DAYS_EMPLOYED'])
```

$$X_train['possible'] = \frac{X_train['income_total']}{X_train['DAYS_BIRTH'] + X_train['DAYS_EMPLOYED']}$$

Feature 생성

3. ['ID']

begin_month 열을 제외하고 중복값 확인 → 총 8759개의 중복값

```
1 X_train.drop(columns=['begin_month', 'index']).drop_duplicates().shape  
(8759, 16)
```


begin_month 열을 제외한 모든 열을 합해 ['id'] feature 생성

```
['child_num']+['income_total']+['DAYS_BIRTH']+['DAYS_EMPLOYED']+['family_size']+['gender']+['car']+['reality']  
+['income_type']+['edu_type']+['family_size']+['house_type']+['work_phone']+['phone']+['email']+['occyp_type']
```

Feature 생성

3. ['ID']

```
X.loc[:, 'id_factorize'] = pd.factorize(X['id'])[0].reshape(-1,1)
```



수치형으로 변환

	index	begin_month	id	id_factorize
	0	0	6.0 0202500.01389947092.0FNNCommercial associateHi...	0
	1	1	5.0 1247500.01138015403.0FNYCommercial associateSe...	1
	2	2	22.0 0450000.01908744342.0MYWWorkingHigher educatio...	2
	3	3	37.0 0202500.01508820922.0FNYCommercial associateSe...	3
	4	4	26.0 0157500.01503721052.0FYState servantHigher ed...	4

9995	36452	19.0	0202500.01859354342.0FYWWorkingIncomplete high...	3480
9996	36453	34.0	0202500.01088613152.0MYWWorkingSecondary / sec...	1109
9997	36454	55.0	0292500.021016140182.0FNYWorkingSecondary / se...	4991
9998	36455	33.0	0180000.01654110852.0FYCommercial associateSe...	1898
9999	36456	11.0	0270000.091541872.0FNYWorkingHigher education2...	6468

Feature 생성

4. ['Age'] (나이)

‘DAYS_BIRTH’으로 ‘Age’ feature 생성

DAYS_BIRTH	age
13899	38
11380	31
19087	52
15088	41
15037	41
13413	37
17570	48
14896	41
15131	41
15785	43

['DAYS_BIRTH']를 365로 나눈 뒤
round를 통해 0번째 자릿수까지 출력

['Age_group'] (나이대)

나이를 기준으로 그룹화

```
X_train['age_group'].value_counts()
```

```
30대      7505  
40대      6886  
50대      5695  
20대      3284  
60대      3081  
Name: age_group, dtype: int64
```

```
X_test['age_group'].value_counts()
```

```
30대      2805  
40대      2608  
50대      2209  
20대      1191  
60대      1187  
Name: age_group, dtype: int64
```

Feature 생성

5. ['Years of service'] (근무경력)

['DAYS_EMPLOYED'] (업무 시작일) 로
['year_of_service'] feature 생성

DAYS_EMPLOYED	year_of_service
4709	13
1540	4
4434	12
2092	6
2105	6
4996	14
1978	5
5420	15
1466	4
1308	4

'DAYS_EMPLOYED'을 365로 나눈 뒤
round를 통해 0번째 자릿수까지 출력

['Years_group']

근무경력을 기준으로 그룹화

```
X_train['years_group'].value_counts()
```

```
1~3년      6649
4~6년      5264
근무경력없음  5118
7~9년      3775
10~15년     3477
16~20년     1043
21년~25년     582
26년~30년     335
31년이상      208
```

```
Name: years_group, dtype: int64
```

```
X_test['years_group'].value_counts()
```

```
1~3년      2430
4~6년      2055
근무경력없음  1922
7~9년      1478
10~15년     1282
16~20년      412
21년~25년     235
26년~30년     106
31년이상       80
```

```
Name: years_group, dtype: int64
```

Feature 생성

6. ['DAYS_BIRTH'] 범주화

수치형 feature을 범주화

```
a = []

for i in X_train['DAYS_BIRTH']:
    if i <= X_train['DAYS_BIRTH'].quantile(q=0.25):
        a.append(1)
    elif i <= X_train['DAYS_BIRTH'].quantile(q=0.5):
        a.append(2)
    elif i <= X_train['DAYS_BIRTH'].quantile(q=0.75):
        a.append(3)
    else:
        a.append(4)

X_train['DAYS_BIRTH_class'] = a

a = []

for i in X_test['DAYS_BIRTH']:
    if i <= X_test['DAYS_BIRTH'].quantile(q=0.25):
        a.append(1)
    elif i <= X_test['DAYS_BIRTH'].quantile(q=0.5):
        a.append(2)
    elif i <= X_test['DAYS_BIRTH'].quantile(q=0.75):
        a.append(3)
    else:
        a.append(4)

X_test['DAYS_BIRTH_class'] = a
```

1사분위수(0.25) -> 1
2사분위수(0.5) -> 2
3사분위수(0.75)-> 3
4사분위수(1)-> 4

['begin_month'] 범주화

```
a = []

for i in X_train['begin_month']:
    if i <= X_train['begin_month'].quantile(q=0.25):
        a.append(1)
    elif i <= X_train['begin_month'].quantile(q=0.5):
        a.append(2)
    elif i <= X_train['begin_month'].quantile(q=0.75):
        a.append(3)
    else:
        a.append(4)

X_train['begin_month_class'] = a

a = []

for i in X_test['begin_month']:
    if i <= X_test['begin_month'].quantile(q=0.25):
        a.append(1)
    elif i <= X_test['begin_month'].quantile(q=0.5):
        a.append(2)
    elif i <= X_test['begin_month'].quantile(q=0.75):
        a.append(3)
    else:
        a.append(4)

X_test['begin_month_class'] = a
```


Feature 생성

7. ['before_employed']

취업하기 전까지의 일수

```
X_train['before_EMPLOYED']  
= X_train['DAYS_BIRTH'] - X_train['DAYS_EMPLOYED']
```

['before_employed_week']

취업하기 전까지의 주수

```
np.floor((X_train['before_EMPLOYED'])/7)  
- ((np.floor((X_train['before_EMPLOYED'])/7)/4).astype(int)*4)
```

['before_employed_month']

취업하기 전까지의 개월수

```
np.floor((X_train['before_EMPLOYED'])/30)  
- ((np.floor((X_train['before_EMPLOYED'])/30)/12).astype(int)*12)
```

['DAYS_BIRTH']와 ['DAYS_EMPLOYED'] feature에도
동일한 방식으로 month와 week feature 생성

Feature 생성

8. 통계수치 feature

Category 와 numerical feature 의 조합으로

mean, median, variance, standard deviation, 변동계수(표준편차/평균)을 사용

```
object1 = X_train.groupby('income_type')['income_total'].agg(['income_type_총income_total', np.sum),
                                                             ('income_type_평균income_total', np.mean),
                                                             ('income_type_최대income_total', np.max),
                                                             ('income_type_최소income_total', np.min),
                                                             ('income_type_income_total표준편차', np.std),
                                                             ('income_type_income_total변동계수',
                                                              lambda x : np.std(x)/np.mean(x))].reset_index().fillna(0)
X_train= pd.merge(X_train, object1, on = 'income_type', how='left')

object2 = X_test.groupby('income_type')['income_total'].agg(['income_type_총income_total', np.sum),
                                                             ('income_type_평균income_total', np.mean),
                                                             ('income_type_최대income_total', np.max),
                                                             ('income_type_최소income_total', np.min),
                                                             ('income_type_income_total표준편차', np.std),
                                                             ('income_type_income_total변동계수',
                                                              lambda x : np.std(x)/np.mean(x))].reset_index().fillna(0)
X_test= pd.merge(X_test, object2, on = 'income_type', how='left')
```

< 다양한 통계수치 feature 생성 >

- 소득분류별 연간소득
- 교육수준별 연간소득
- 결혼여부별 연간소득
- 생활방식별 연간소득
- 나이대별 연간소득
- 직업유형별 연간소득

Modeling

첫 번째 실험 전략

양상불을 목적으로 각 모델들의 상관관계를 낮추기 위해서
각 팀원들이 각자 다른 모델을 선택해 모델학습을 진행함.

- 1. LightGBM
수치형 feature -> PowerTransformer
범주형 feature -> Ordinal Encoding
- 2. XGBoost
수치형 feature -> PowerTransformer
범주형 feature -> Target Encoding

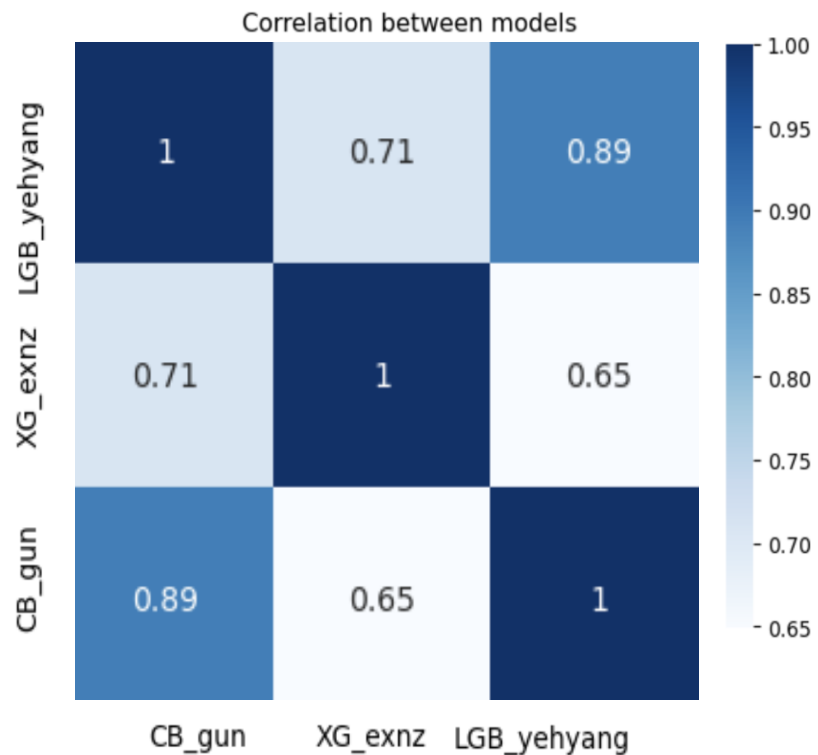
제출 완료한 각각의 성능을 비교 첫 번째 실험 결과

	1. LightGBM	2. XGBoost	3. CatBoost
public	0.7061195353	0.79448610	0.6723580678
private	0.6931036688	0.780129040	0.6641048254

CatBoost 성능이 가장 좋음

Submission ensemble

상관관계 확인



가중치 설정

CatBoost 0.65
LightGBM 0.2
XGB 0.15

가중평균 적용

성능 public 0.6806744368
private 0.6712043469

한계점

가중평균을 이용해 앙상블을 여러 번 진행하였으나
모두 Catboost단일 모델 보다 성능이 낮았음.

new_modeling

두 번째 실험 전략

각자 만든 feature를 모두 합쳐서 Catboost 단일 모델로 모델학습을 진행

```
{['gender', 'car', 'reality', 'child_num', 'income_total', 'income_type',  
  'edu_type', 'family_type', 'house_type', 'DAYS_BIRTH', 'DAYS_EMPLOYED',  
  'work_phone', 'phone', 'email', 'occyp_type', 'family_size',  
  'begin_month', 'id', 'age', 'age_group', 'year_of_service',  
  'years_group', 'DAYS_BIRTH_month', 'DAYS_BIRTH_week',  
  'DAYS_EMPLOYED_month', 'DAYS_EMPLOYED_week', 'before_EMPLOYED',  
  'before_EMPLOYED_month', 'before_EMPLOYED_week', 'DAYS_BIRTH_class',  
  'begin_month_class', 'income_total*10000', 'income_total_dev',  
  'income_total_log', '소득/가족', '소득/자녀', 'income_type_총income_total',  
  'income_type_평균income_total', 'income_type_최대income_total',  
  'income_type_최소income_total', 'income_type_income_total표준편차',  
  'income_type_income_total변동계수', 'edu_type_총income_total',  
  'edu_type_평균income_total', 'edu_type_최대income_total',  
  'edu_type_최소income_total', 'edu_type_income_total표준편차',  
  'edu_type_income_total변동계수', 'family_type_총income_total',  
  'family_type_평균income_total', 'family_type_최대income_total',  
  'family_type_최소income_total', 'family_type_income_total표준편차',  
  'family_type_income_total변동계수', 'house_type_총income_total',  
  'house_type_평균income_total', 'house_type_최대income_total',  
  'house_type_최소income_total', 'house_type_income_total표준편차',  
  'house_type_income_total변동계수', 'Age_type_총income_total',  
  'Age_type_평균income_total', 'Age_type_최대income_total',  
  'Age_type_최소income_total', 'Age_type_income_total표준편차',  
  'Age_type_income_total변동계수', 'occyp_type_총income_total',  
  'occyp_type_평균income_total', 'occyp_type_최대income_total',  
  'occyp_type_최소income_total', 'occyp_type_income_total표준편차',  
  'occyp_type_income_total변동계수', 'EMPLOYED_RATIO',  
  'income_per_days_birth', 'income_per_days_birth_X_DAYS_BIRTH',  
  'begin_month_X_DAYS_BIRTH', 'BIRTH*id', 'EMP*id', 'BIRTH*EMP',  
  'possible', 'possible_class', 'car_reality', 'id_총DAYS_BIRTH',  
  'id_평균DAYS_BIRTH', 'id_최대DAYS_BIRTH', 'id_최소DAYS_BIRTH',  
  'id_DAYS_BIRTH표준편차', 'id_DAYS_BIRTH변동계수', 'id_총begin_months5',  
  'id_평균begin_months5', 'id_최대begin_months5', 'id_최소begin_months5',  
  'id_begin_month표준편차s5', 'id_begin_month변동계수s5', 'pos+beg+dBirth',  
  'income/before_EMPLOYED', 'edu_type_num', 'edu_type_num/DAYS_BIRTH',  
  'edu_type_num*income_per_days_birth'],  
dtype='object')}
```

첫번째 실험 결과

총 98개의 feature 생성

성능 public 0.6723586599
private 0.6632314885

최종 결론

중복되는 Feature를 제거해 기본변수까지 합쳐 **총 98개의 Feature 생성**함.

Feature importance를 직접 확인해서 **중요도가 낮은 피쳐 10개를 삭제**했지만, **Feature 성능이 떨어짐**

```
{['gender', 'car', 'reality', 'child_num', 'income_total', 'income_type',  
'edu_type', 'family_type', 'house_type', 'DAYS_BIRTH', 'DAYS_EMPLOYED',  
'work_phone', 'phone', 'email', 'occyp_type', 'family_size',  
'begin_month', 'id', 'age', 'age_group', 'year_of_service',  
'years_group', 'DAYS_BIRTH_month', 'DAYS_BIRTH_week',  
'DAYS_EMPLOYED_month', 'DAYS_EMPLOYED_week', 'before_EMPLOYED',  
'before_EMPLOYED_month', 'before_EMPLOYED_week', 'DAYS_BIRTH_class',  
'begin_month_class', 'income_total*10000', 'income_total_dev',  
'income_total_log', '소득/가족', '소득/자녀', 'income_type_총income_total',  
'income_type_평균income_total', 'income_type_최대income_total',  
'income_type_최소income_total', 'income_type_income_total표준편차',  
'income_type_income_total변동계수', 'edu_type_총income_total',  
'edu_type_평균income_total', 'edu_type_최대income_total',  
'edu_type_최소income_total', 'edu_type_income_total표준편차',  
'edu_type_income_total변동계수', 'family_type_총income_total',  
'family_type_평균income_total', 'family_type_최대income_total',  
'family_type_최소income_total', 'family_type_income_total표준편차',  
'family_type_income_total변동계수', 'house_type_총income_total',  
'house_type_평균income_total', 'house_type_최대income_total',  
'house_type_최소income_total', 'house_type_income_total표준편차',  
'house_type_income_total변동계수', 'Age_type_총income_total',  
'Age_type_평균income_total', 'Age_type_최대income_total',  
'Age_type_최소income_total', 'Age_type_income_total표준편차',  
'Age_type_income_total변동계수', 'occyp_type_총income_total',  
'occyp_type_평균income_total', 'occyp_type_최대income_total',  
'occyp_type_최소income_total', 'occyp_type_income_total표준편차',  
'occyp_type_income_total변동계수', 'EMPLOYED_RATIO',  
'income_per_days_birth', 'income_per_days_birth_X_DAYS_BIRTH',  
'begin_month_X_DAYS_BIRTH', 'BIRTH*id', 'EMP*id', 'BIRTH*EMP',  
'possible', 'possible_class', 'car_reality', 'id_총DAYS_BIRTH',  
'id_평균DAYS_BIRTH', 'id_최대DAYS_BIRTH', 'id_최소DAYS_BIRTH',  
'id_DAYS_BIRTH표준편차', 'id_DAYS_BIRTH변동계수', 'id_총begin_months5',  
'id_평균begin_months5', 'id_최대begin_months5', 'id_최소begin_months5',  
'id_begin_month표준편차s5', 'id_begin_month변동계수s5', 'pos+beg+dBirth',  
'income/before_EMPLOYED', 'edu_type_num', 'edu_type_num/DAYS_BIRTH',  
'edu_type_num*income_per_days_birth'],  
dtype='object')}
```

Feature를 모두 합친 CatBoost 모델이 제일 성능이 좋았음.
최종 모델로 Feature를 모두 합친 CatBoost 모델을 선택

NOTE

GBDT에서는 단순한 노이즈가 되는 특징이 있더라도 성능이 쉽게 떨어지지는 않습니다. 대량 생산되는 특징 중에는 노이즈가 되는 특징이 있을 수 있고 완전하지 않을 수 있습니다. 하지만, 이들 특징 중에 유효한 특징이 있어서 모델 성능을 충분히 떨어뜨릴 수 있다면, 특징에서 유효한 특징을 찾아냈을 때의 긍정적인 측면이 더 크다는 생각이 이러한 방법의 배경이 아닐까 생각합니다.

출처: 데이터가 뛰어노는 AI 놀이터, 캐글 p360

아쉬운 점

< Feature 생성 파트 >

['frequency']

각각의 샘플이 가지는 ID의 개수로 feature 생성

```
tr_freq= pd.DataFrame(X_train['id'].value_counts()).reset_index()
tr_freq.columns = ['id', 'frequency']
X_train = pd.merge(X_train, tr_freq, on='id', how='outer')

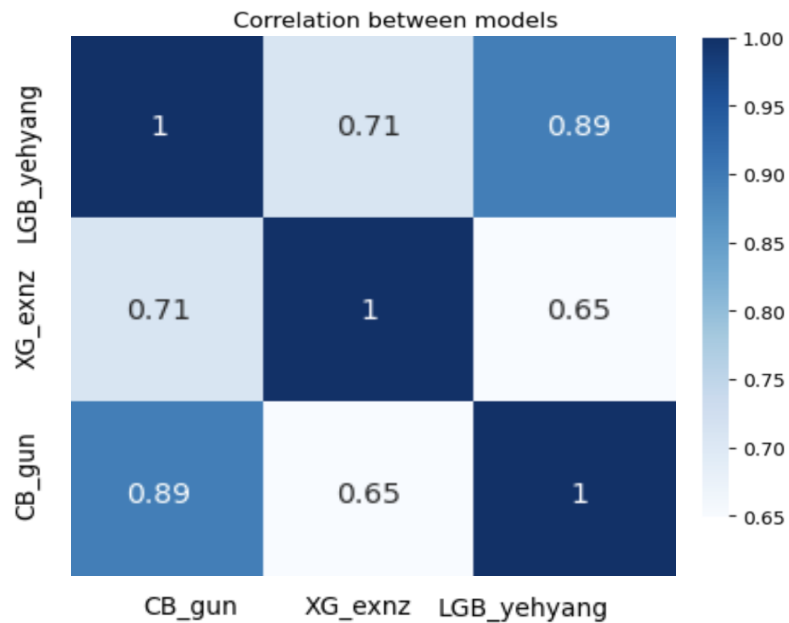
te_freq= pd.DataFrame(X_test['id'].value_counts()).reset_index()
te_freq.columns = ['id', 'frequency']
X_test = pd.merge(X_test, te_freq, on='id', how='outer')
```

Feature importance 확인 시 중요도 ↑
But, 성능 ↓

아쉬운 점

< Ensemble 파트 >

앙상블에서의 성능 향상을 도모하기 위해서는 **데이터 간의 독립성**이 최대한 보장되어 있어야 함.



생성한 feature 셋들이 비슷해서
앙상블 파트에서 성능이 잘 나오지 않음.

감사합니다