신용카드 사용자 연체 예측

3조

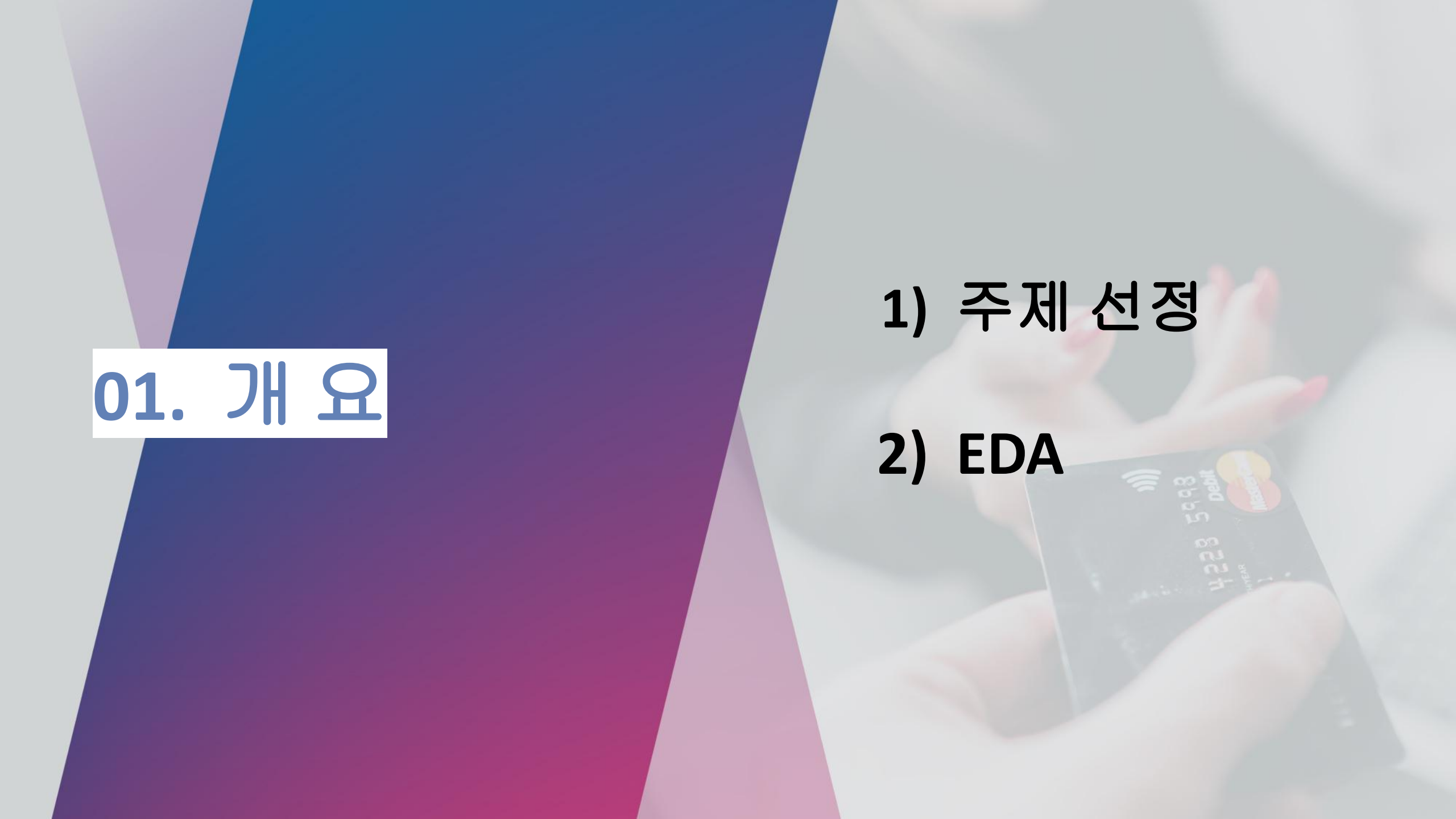
김도균, 박강인, 반위홍, 심승현,
안병윤, 이태훈, 정우영, 최준호

CONTENTS

01. 개 요

1) 주제 선정

2) EDA



주제 선정

아파트 경매가격 예측 경진대회

금융 | 부동산 아파트 경매 빅데이터와 AI로 경매가 예측 분석 | 회귀 | RMSE

심리 성향 예측 AI 경진대회

월간 데이콘 8 | 심리 테스트 분석 | AUC | 분류

구내식당 식수 인원 예측 AI 경진대회

정형 | 한국토지주택공사 | 식수예측 | MAE

데이터 **부정확**, 도메인 **호불호**,
많은 양의 데이터 등으로 기각

신용카드 사용자 연체 예측 AI 경진대회

월간 데이콘 14 | 금융 | 정형 | Logloss

상금 : 100만원

2021.04.05 ~ 2021.05.24 17:59 + Google Calendar

2,427명 마감

정형 데이터 셋으로 비교적 수상자 코드가 **잘 정리**되어 있음.

현 Dacon, Kaggle 등에서 가장 많이 사용되는 알고리즘인 ***Catboost*** 스터디에 용이.

EDA(변수정의)

- gender: 성별
- car: 차량 소유 여부
- reality: 부동산 소유 여부
- income_type: 소득 분류
['Commercial associate', 'Working', 'State servant', 'Pensioner', 'Student']
- edu_type: 교육 수준 ['Higher education' ,'Secondary / secondary special', 'Incomplete higher', 'Lower secondary', 'Academic degree']
- family_type: 결혼 여부['Married', 'Civil marriage', 'Separated', 'Single / not married', 'Widow']
- house_type: 생활 방식
['Municipal apartment', 'House / apartment', 'With parents', Co-op apartment', 'Rented apartment', 'Office apartment']
- FLAG_MOBIL: 핸드폰 소유 여부
- work_phone: 업무용 전화 소유 여부
- phone: 전화 소유 여부
- email: 이메일 소유 여부
- occyp_type: 직업 유형

EDA(변수정의)

numeric feature :

- *child_num: 자녀 수*
- *income_total: 연간 소득*
- *DAYS_BIRTH: 출생일(데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 하루 전에 태어났음을 의미)*
- *DAYS_EMPLOYED: 업무 시작일(데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 하루 전부터 일을 시작함을 의미), 양수 값은 고용되지 않은 상태를 의미함*
- *family_size: 가족 규모*
- *begin_month: 신용카드 발급 월(데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 한 달 전에 신용카드를 발급함을 의미)*

target feature:

- *credit: 사용자의 신용카드 대금 연체를 기준으로 한 신용도 => 낮을 수록 높은 신용의 신용카드 사용자를 의미함*

19개의 feature 변수와 1개의 target 변수(credit)으로 구성되어있음

EDA(Target 값 분포)

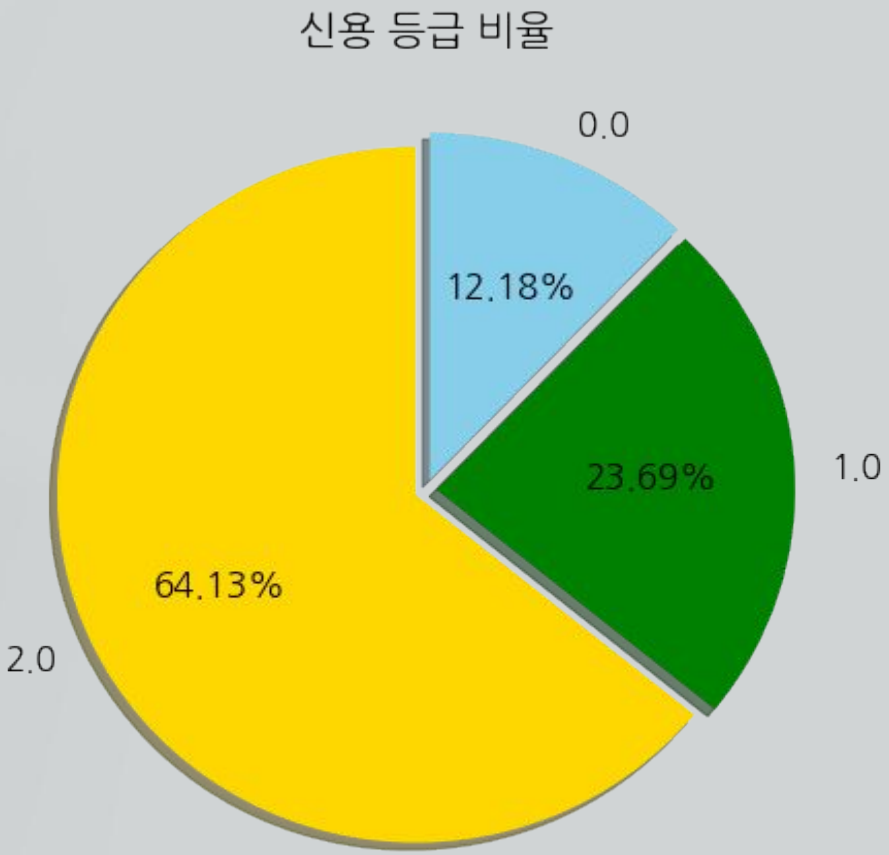
"0" = 높은 신용등급의 사용자

-> 신용도가 낮은 사용자 비율이 높음

% 신용 등급 비율 분포

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

```
2.0    16962
1.0     6267
0.0     3222
Name: credit, dtype: int64
```



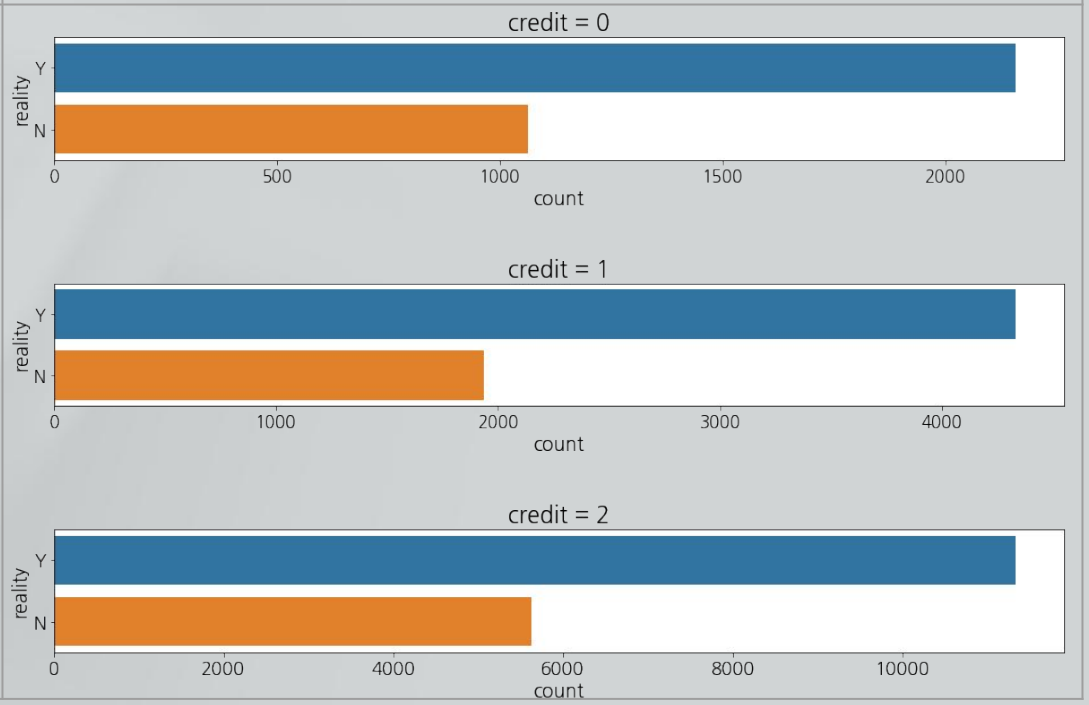
EDA(Target 값에 따른 변수 분포)

성별에 따른 신용등급

"모든 신용등급에서 **남성의** 사용자가 많음"

부동산 유무에 따른 신용등급

" 모든 신용등급에서 부동산을 **소유한** 사용자가 많음"



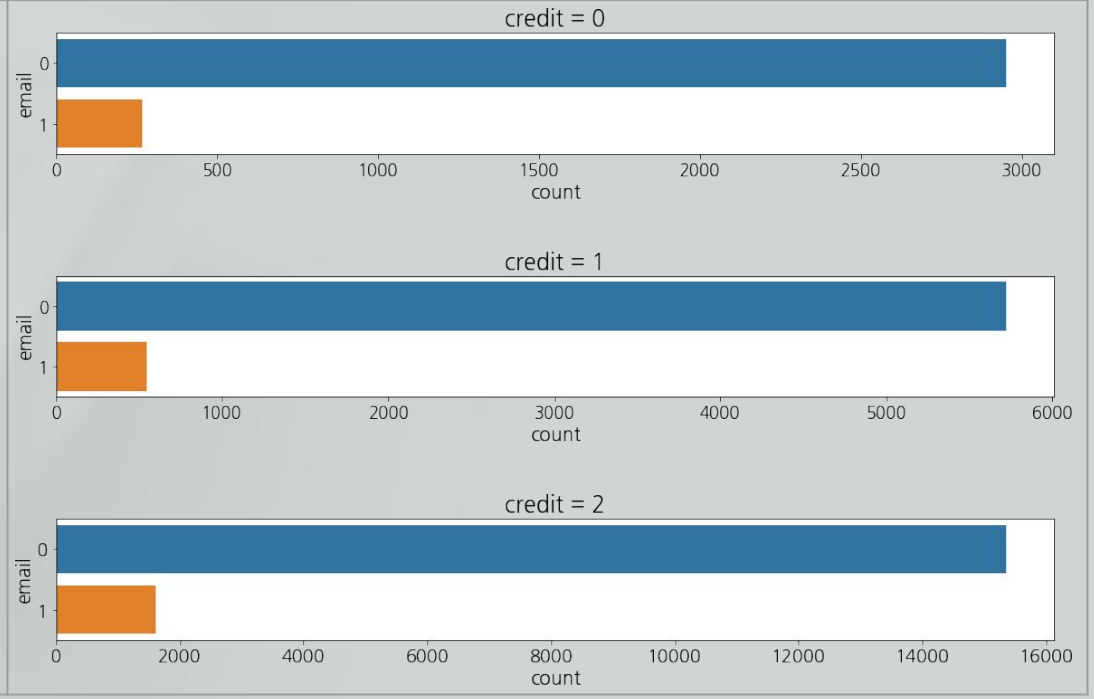
EDA(Target 값에 따른 변수 분포)

자가용 유무에 따른 신용등급

“ 모든 신용등급에서 자가용이 소유하지 않은 사용자가 많음”

이메일 유무에 따른 신용등급

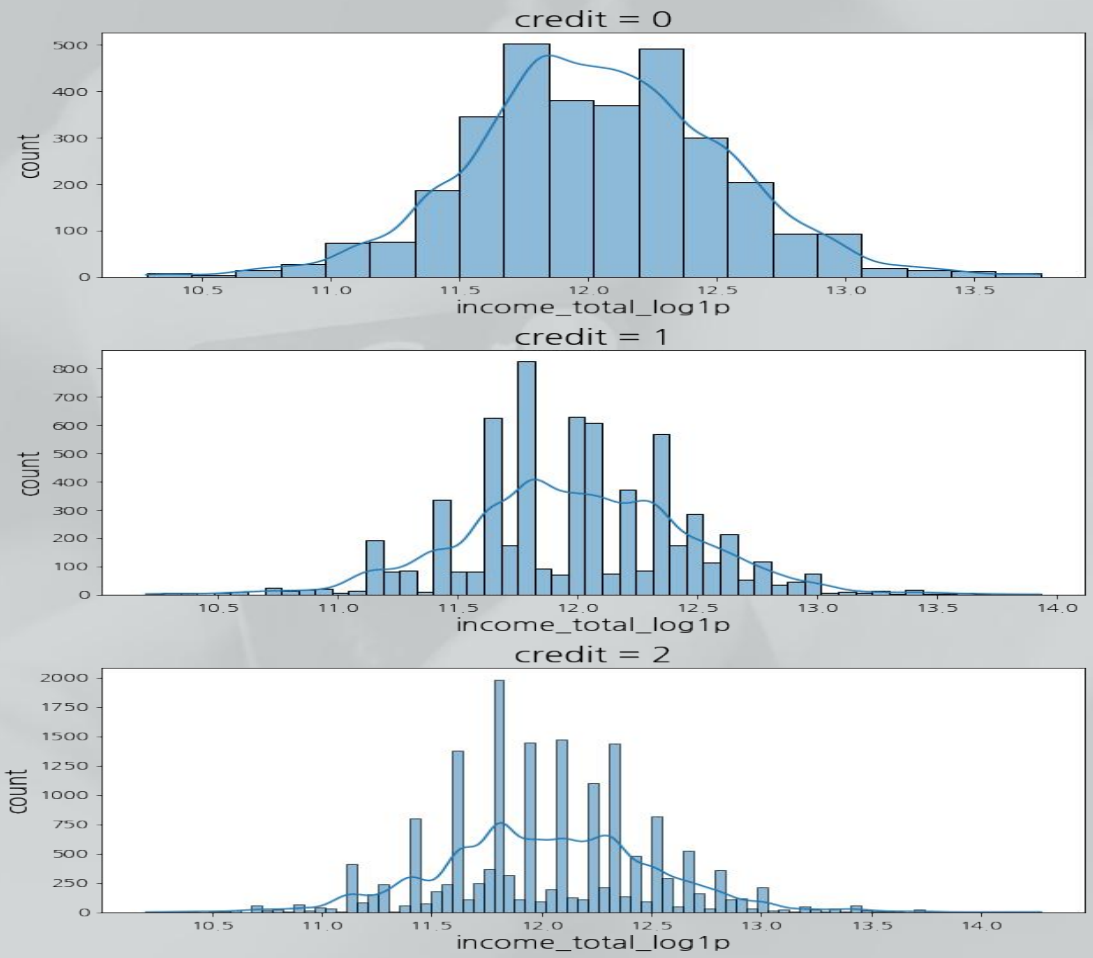
“ 모든 신용등급에서 이메일을 소유한 사용자가 많음”

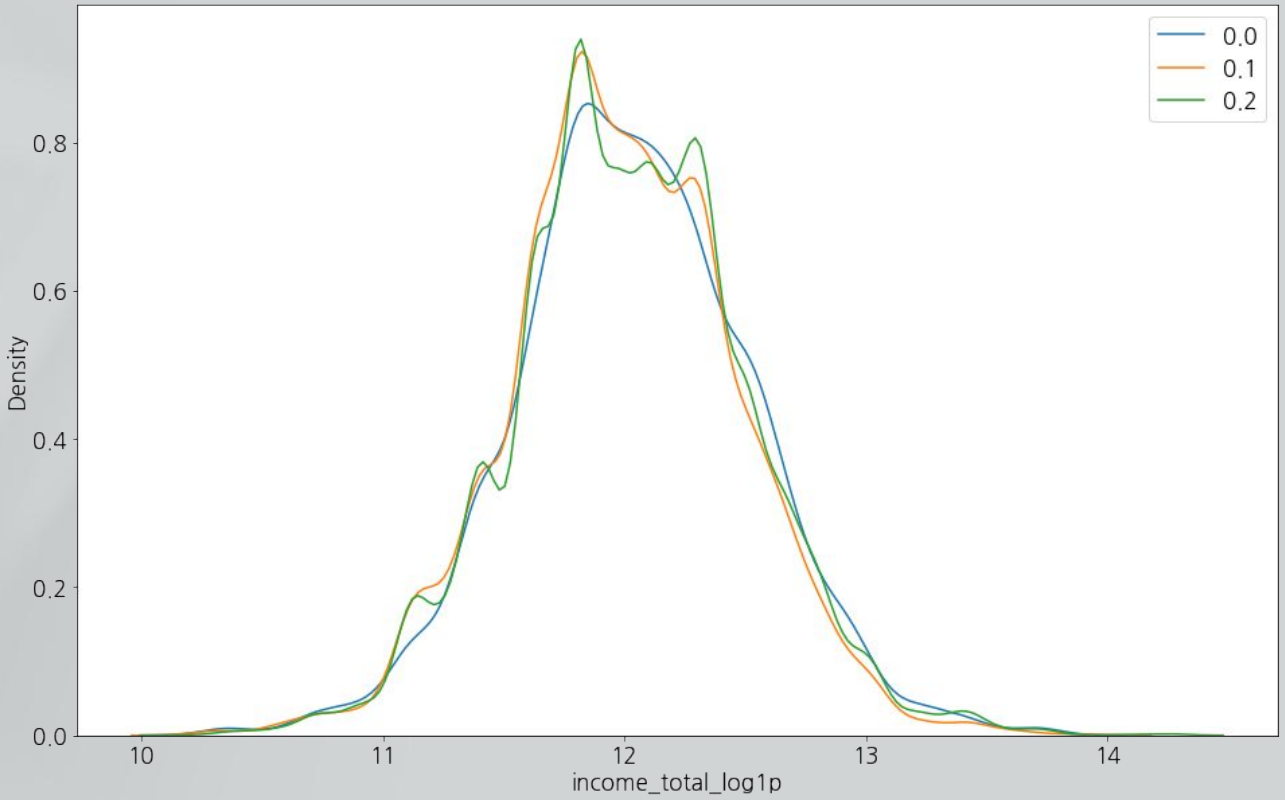


EDA(Target 값에 따른 변수 분포)

-신용등급에 따른 연간 소득

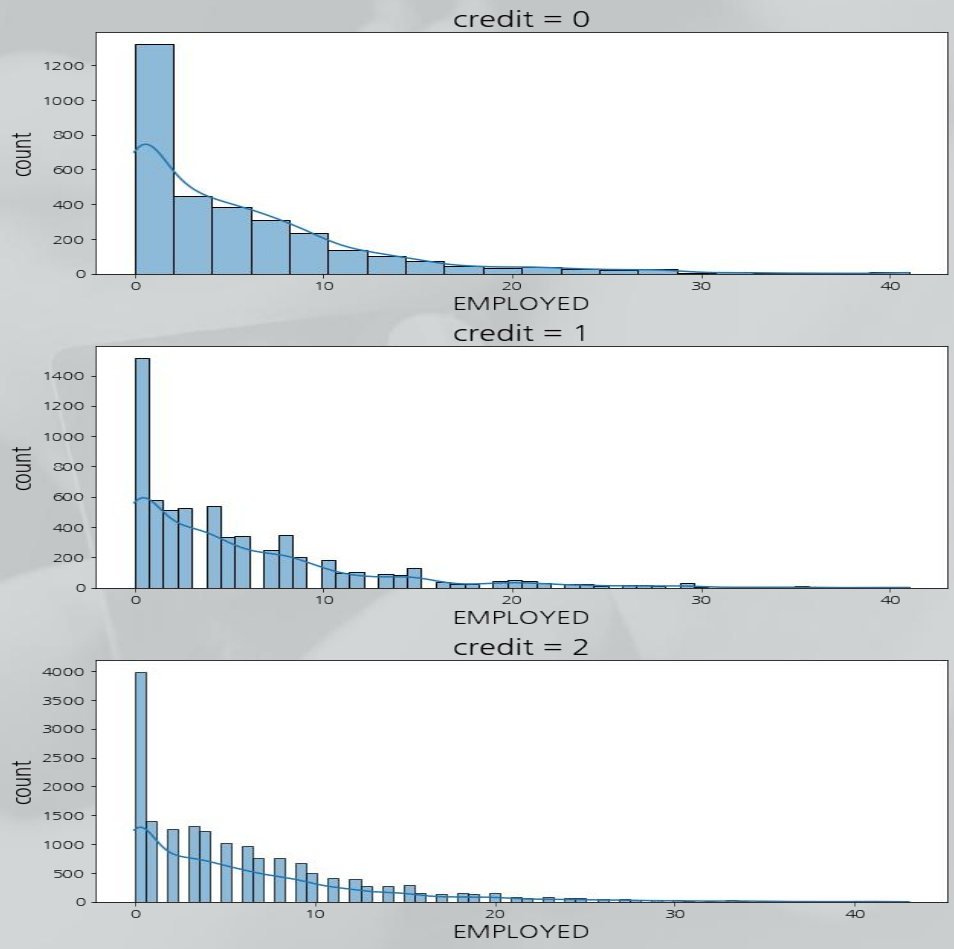
->신용등급에 따라 연간 소득 차이는 크게 없다

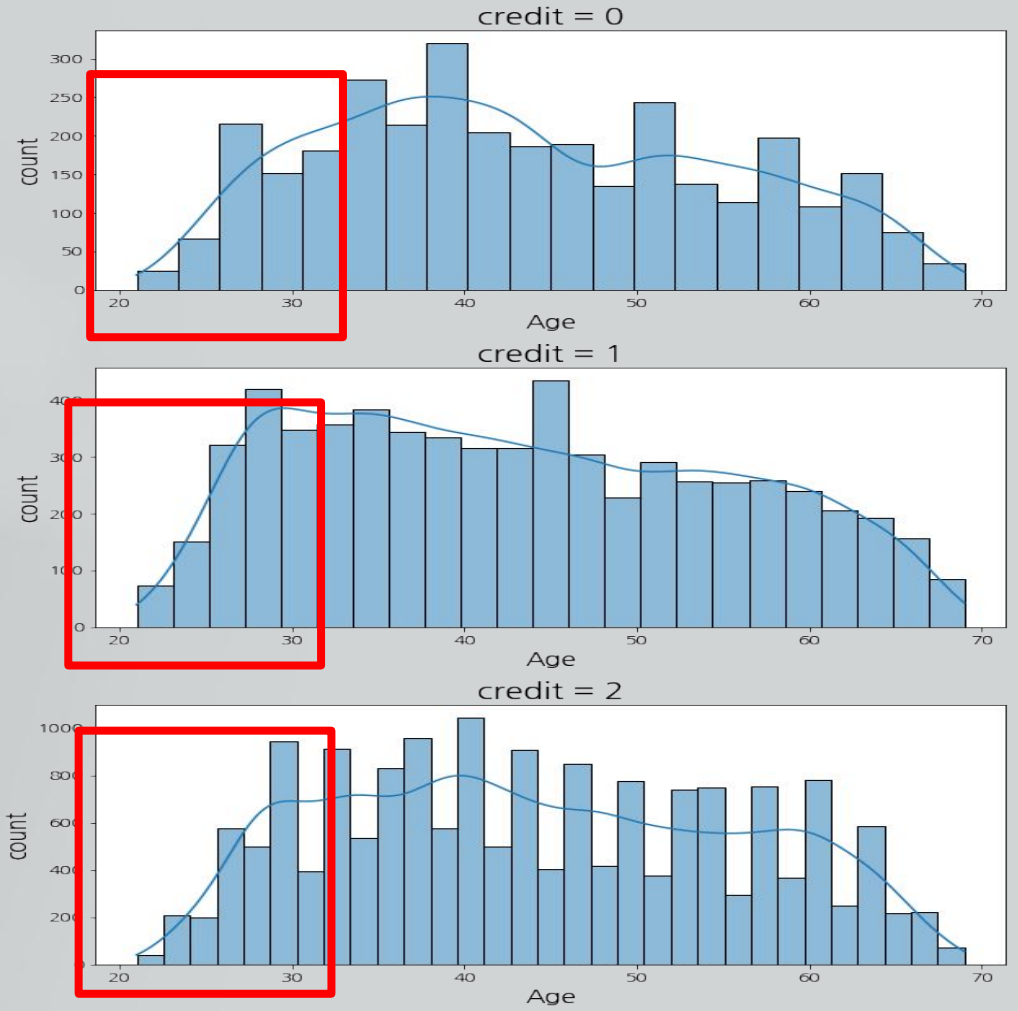




EDA(Target 값에 따른 변수 분포)

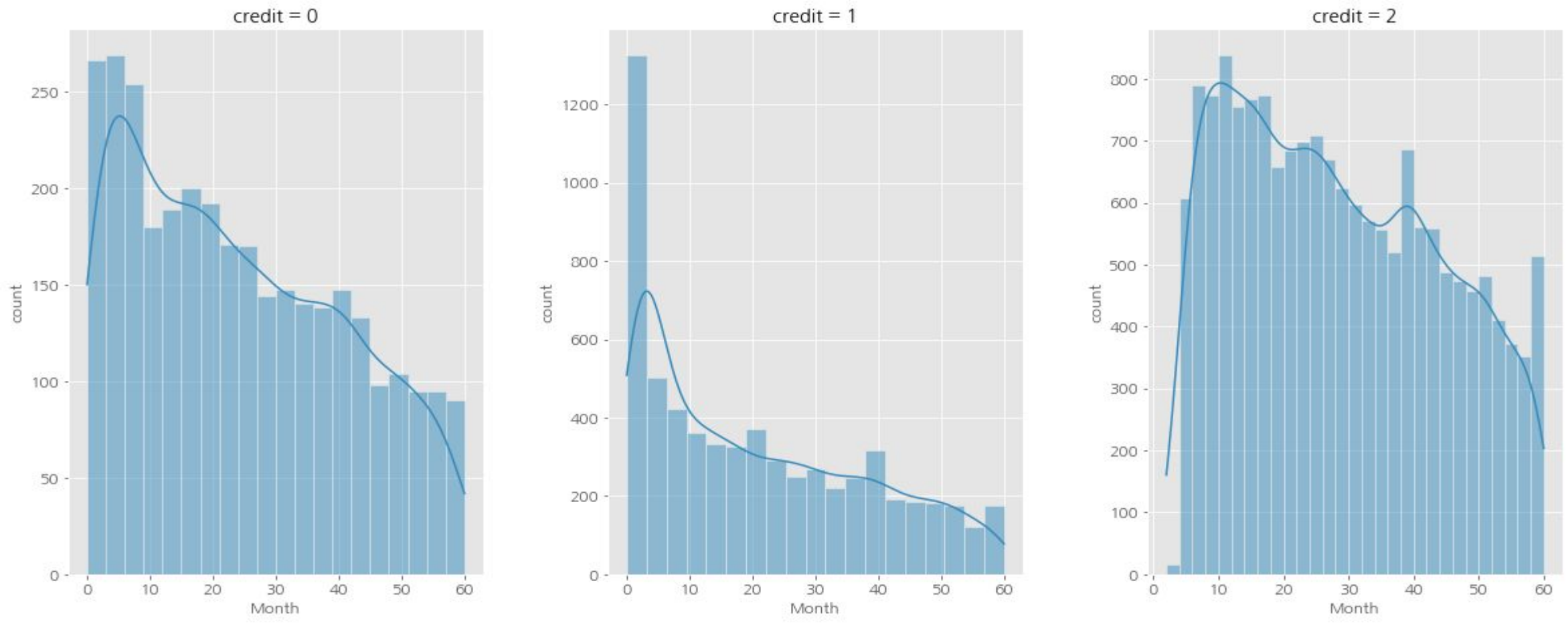
연도별 업무 기간/나이에 따른 신용등급 분포





EDA(Target 값에 따른 변수 분포)

발급월수에 따른 신용도 분포 -> 신용등급별 차이가 있음.



1등 코드는..?

```
corr_ = train.corr()['credit'].reset_index()
corr_.sort_values(by='credit', ascending=False)
```

	index	credit
9	credit	1.000000
8	begin_month	0.143323
2	DAYS_BIRTH	0.024332
3	DAYS_EMPLOYED	0.019905
6	email	0.012133
7	family_size	0.009937
0	child_num	0.005825
1	income_total	0.005274
5	phone	0.003388
4	work_phone	-0.002498

"19개의 feature 변수 중 target 값과 상관관계가 높은 feature가 1개(?)..."

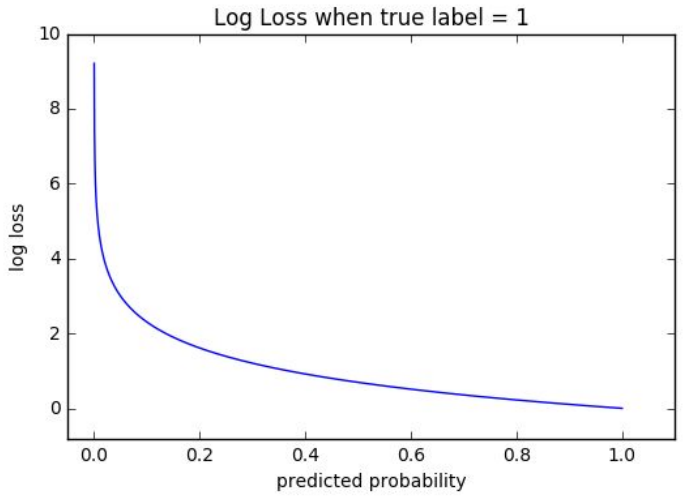
02. 1등 코드 분석

프로젝트 진행과정 - 1등 코드 분석

1	소회의실		0.6581	77	8달 전

신용카드 예측 데이터 대회는 ***logloss*** *의 평가 점수가* ***0.6581*** *이 가장 낮았음.*

여기서, 평가지표로 활용되는 'logloss'란 무엇일까?



Log loss 란..?

일반적으로 정답을 맞춘다는 것은 '정답을 맞출 확률'을 의미 하지만,

"Logloss는 얼마나 정확하게 정답을 맞출 것인지에 대한 척도를 의미"

정답을 **100%** 확률로 맞춤 -log(1.0) = **0**

정답을 **60%** 확률로 맞춤 -log(0.6) = **0.510**

정답을 **52%** 확률로 맞춤 -log(0.52) = **0.653**

확률이 높을수록 log loss는 낮아짐, 즉 log loss가 0에 가까울수록 정답을 높은 확률로 맞춘다는 것을 의미.

어떻게 '소회의실'팀은 1등을 했을까

ID 컬럼 생성

새로운 특성
(Feature)
추가

Catboost
알고리즘 사용

1) ID컬럼 생성과정

ID 컬럼 생성 코드

*-> **begin_month를 제외**한 나머지 변수를 ID변수로 생성*

```
#ID 생성: 각 컬럼의 값들을 더해서 고유한 사람을 파악(*한 사람이 여러 개 카드를 만들 가능성을 고려해 begin_month는 제외함)
df['ID'] = ₩
df['child_num'].astype(str) + '_' + df['income_total'].astype(str) + '_' +₩
df['DAYS_BIRTH'].astype(str) + '_' + df['DAYS_EMPLOYED'].astype(str) + '_' +₩
df['work_phone'].astype(str) + '_' + df['phone'].astype(str) + '_' +₩
df['email'].astype(str) + '_' + df['family_size'].astype(str) + '_' +₩
df['gender'].astype(str) + '_' + df['car'].astype(str) + '_' +₩
df['reality'].astype(str) + '_' + df['income_type'].astype(str) + '_' +₩
df['edu_type'].astype(str) + '_' + df['family_type'].astype(str) + '_' +₩
df['house_type'].astype(str) + '_' + df['occyp_type'].astype(str)
```

1) ID - 중복 데이터 확인

Begin_month 제외 한 나머지 컬럼의 특성

-> begin_month를 제외한 데이터에서 중복데이터가 다수 발생(80% 이상이 중복데이터)

```
#중복데이터 확인
#인덱스제거
for df in [train,test]:
    df.drop('index',axis=1,inplace=True)
#중복데이터를 확인하는 함수 정의
def dup(df,col):
    collist = list(df.columns)
    for i in col:
        collist.remove(i)
    return(df.duplicated(subset = collist, keep = False).sum())
```

```
len(train.index)
```

```
26457
```

```
for i in collist:
    print('%-13s' % i, f' 외 중복 데이터 : {dup(train,[i])}건')
```

```
gender        외 중복 데이터 : 3159건
car           외 중복 데이터 : 3155건
reality       외 중복 데이터 : 3155건
child_num     외 중복 데이터 : 3155건
income_total  외 중복 데이터 : 3157건
income_type   외 중복 데이터 : 3159건
edu_type      외 중복 데이터 : 3155건
family_type   외 중복 데이터 : 3157건
house_type    외 중복 데이터 : 3155건
DAYS_BIRTH    외 중복 데이터 : 3437건
DAYS_EMPLOYED 외 중복 데이터 : 3155건
FLAG_MOBIL    외 중복 데이터 : 3155건
work_phone    외 중복 데이터 : 3155건
phone         외 중복 데이터 : 3155건
email         외 중복 데이터 : 3157건
occyp_type    외 중복 데이터 : 3155건
family_size   외 중복 데이터 : 3159건
begin_month   외 중복 데이터 : 20375건
credit        외 중복 데이터 : 4497건
```

1) ID - ID컬럼의 특성

ID	count	people
0_297000.0_15519_-3234_0_0_0_1.0_F_N_Y_Commercial associate_Secondary / secondary special_Single / not married_Rented apartment_Laborers	35	1.0
2_225000.0_16768_-3088_1_0_0_4.0_M_N_N_Working_Higher education_Civil marriage_House / apartment_Laborers	24	2.0
1_157500.0_12676_-1350_0_0_0_2.0_F_N_Y_State servant_Secondary / secondary special_Widow_House / apartment_Waiters/barmen staff	21	1.0
0_112500.0_9952_-1613_0_0_0_1.0_M_Y_Y_Working_Secondary / secondary special_Single / not married_House / apartment_freelancer	20	5.0
0_202500.0_21126_365243_0_0_0_1.0_F_N_Y_Pensioner_Secondary / secondary special_Widow_House / apartment_inoccupation	19	1.0
0_225000.0_12322_-3717_0_0_0_2.0_F_Y_Y_Working_Higher education_Married_House / apartment_Core staff	18	2.0
0_225000.0_22976_365243_0_0_0_1.0_F_N_Y_Pensioner_Secondary / secondary special_Single / not married_House / apartment_inoccupation	17	1.0
1_135000.0_10112_-3170_1_1_0_3.0_F_N_Y_Working_Secondary / secondary special_Married_House / apartment_Laborers	16	7.0
1_562500.0_13790_-5639_1_1_0_3.0_M_N_Y_Working_Incomplete higher_Married_House / apartment_freelancer	15	10.0
0_130500.0_13520_-5488_0_0_0_2.0_F_N_Y_Working_Secondary / secondary special_Married_House / apartment_Core staff	14	15.0

```
len(train.index)
```

```
26457
```

```
temp['people'].sum()
```

```
8756.0
```

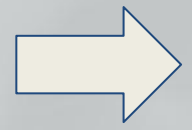
“ 동일 ID의 개수를 활용하여
여러번 카드를 발급한 사람이 많았음을 확인 ”

2) 새로운 특성 추가

Feature importance

```
# before_EMPLOYED: 고용되기 전까지의 일수
df['before_EMPLOYED'] = df['DAYS_BIRTH'] - df['DAYS_EMPLOYED']
df['income_total_befofeEMP_ratio'] = df['income_total'] / df['before_EMPLOYED']
df['before_EMPLOYED_m'] = np.floor(df['before_EMPLOYED'] / 30) - ((np.floor(df['before_EMPLOYED'] / 30) / 12).astype(int) * 12)
df['before_EMPLOYED_w'] = np.floor(df['before_EMPLOYED'] / 7) - ((np.floor(df['before_EMPLOYED'] / 7) / 4).astype(int) * 4)

#DAYS_BIRTH 파생변수- Age(나이), 태어난 월, 태어난 주(출생연도의 n주차)
df['Age'] = df['DAYS_BIRTH'] // 365
df['DAYS_BIRTH_m'] = np.floor(df['DAYS_BIRTH'] / 30) - ((np.floor(df['DAYS_BIRTH'] / 30) / 12).astype(int) * 12)
df['DAYS_BIRTH_w'] = np.floor(df['DAYS_BIRTH'] / 7) - ((np.floor(df['DAYS_BIRTH'] / 7) / 4).astype(int) * 4)


#DAYS_EMPLOYED_m 파생변수- EMPLOYED(근속연수), DAYS_EMPLOYED_m(고용된 달) ,DAYS_EMPLOYED_w(고용된 주(고용연도의 n주차))
df['EMPLOYED'] = df['DAYS_EMPLOYED'] // 365
df['DAYS_EMPLOYED_m'] = np.floor(df['DAYS_EMPLOYED'] / 30) - ((np.floor(df['DAYS_EMPLOYED'] / 30) / 12).astype(int) * 12)
df['DAYS_EMPLOYED_w'] = np.floor(df['DAYS_EMPLOYED'] / 7) - ((np.floor(df['DAYS_EMPLOYED'] / 7) / 4).astype(int) * 4)

#ability: 소득/(살아온 일수+ 근무일수)
df['ability'] = df['income_total'] / (df['DAYS_BIRTH'] + df['DAYS_EMPLOYED'])

#income_mean: 소득/ 가족 수
df['income_mean'] = df['income_total'] / df['family_size']
```

```
train.shape
(26451, 18)
```

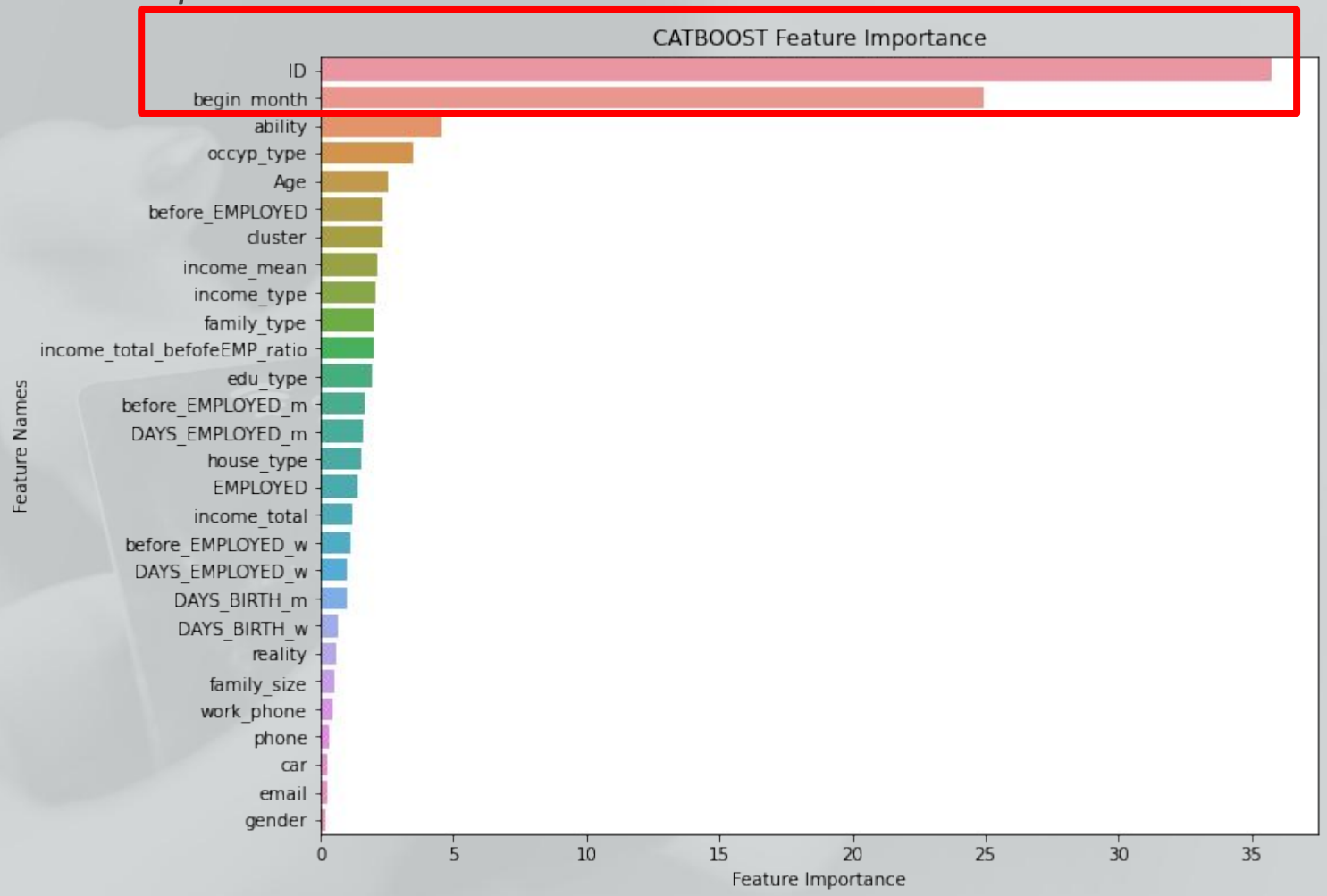


```
train.shape
(24822, 28)
```

"기존 변수를 활용한 feature 10개 생성"

2) 새로운 특성 추가

Feature importance



모델의 <u>영향</u>을 미치는 요인으로
<u>상위</u>에 랭크된 변수

기존변수 : **begin_month**
<u>추가</u> 변수
:ID, ability, Age

3) CatBoost?

Categorical Boosting 의 약자로, 이름 그대로 범주형 변수 처리에 특화된 부스팅 기반 알고리즘 모델

3) CatBoost

기존 부스팅 기법

1. 실제 값들의 평균과 각 실제 값들의 잔차를 이용해 학습하는 모델 생성
2. 예측된 잔차에 Learing_rate 를 곱해 평균과 더하여 예측값 갱신
3. 위의 과정 반복

한계점

1. 이전 데이터를 반복적으로 쓰는 과정에서 발생되는 **"과적합 문제"**
2. 범주형 변수를 one-hot-encoding 할때, **"변수의 수가 급증하는 문제"**

3) CatBoost

CatBoost 모델 특징

1. Ordered Boosting
: 일부 샘플에 대한 잔차를 계산하여 모델을 만든 후,
그 뒤의 데이터 잔차는 해당 모델의 예측한 값을 활용

2. Random Permutation
: 데이터셋을 무작위 순열들로 나누어, 랜덤성을
부여한 이후 Ordered Boosting 처리

→ **“ 과적합 문제 해결 “**

3) CatBoost

CatBoost 모델 특징

3. Ordered Target Encoding
: 과거 데이터들로부터만 평균을 내어 범주형 변수값 인코딩

4. Categorical Feature Combinations
: 동일한 대상을 대표하는 특징은 하나의 특징으로 분류

→ “ 범주형 변수 처리 문제 해결 “

3) Catboost - 성능 테스트(Pycaret 활용)

```
1 !pip install pycaret
2
3 from pycaret.classification import *
4 from sklearn.metrics import log_loss
5
6 clf = setup(data = train, target = 'credit', train_size = 0.85)
7 add_metric('logloss', 'LogLoss', log_loss, greater_is_better = False, target="pred_proba")
8 compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
catboost	CatBoost Classifier	0.7023	0.7145	0.4422	0.6860	0.6359	0.2648	0.3322	0.7577
lightgbm	Light Gradient Boosting Machine	0.6992	0.7114	0.4254	0.6912	0.6221	0.2388	0.3234	0.7622
gbc	Gradient Boosting Classifier	0.6934	0.6553	0.4110	0.6575	0.6085	0.2134	0.3052	0.7940
lda	Linear Discriminant Analysis	0.6429	0.6053	0.3377	0.5343	0.5103	0.0130	0.0496	0.8626
nb	Naive Bayes	0.6434	0.6060	0.3355	0.5514	0.5056	0.0071	0.0469	0.8633
lr	Logistic Regression	0.6418	0.5523	0.3333	0.4119	0.5018	0.0000	0.0000	0.8788
dummy	Dummy Classifier	0.6418	0.5000	0.3333	0.4119	0.5018	0.0000	0.0000	0.8812
rf	Random Forest Classifier	0.7009	0.7522	0.5445	0.6806	0.6861	0.3776	0.3827	1.0374
ada	Ada Boost Classifier	0.6910	0.6253	0.4047	0.6197	0.6020	0.1999	0.2990	1.0809
et	Extra Trees Classifier	0.6733	0.7099	0.5247	0.6544	0.6605	0.3264	0.3299	2.8550
knn	K Neighbors Classifier	0.6279	0.6633	0.4709	0.6040	0.6121	0.2276	0.2308	4.0079
dt	Decision Tree Classifier	0.6183	0.6482	0.5069	0.6273	0.6223	0.2772	0.2776	12.7273
qda	Quadratic Discriminant Analysis	0.4513	0.5010	0.3370	0.4866	0.4558	0.0031	0.0031	18.9526

Pycaret

: 기존의 머신러닝 라이브러리들을 결합한 High-Level API 로, 데이터분석부터 모델별 성능 비교까지 간편한 기능으로 제공

Classifier 모델별 성능 비교 결과

: Accuracy 점수와, 프로젝트 평가지표인 LogLoss 점수에서 **catboost**
모델이 가장 높은 성능 기록

03. 성능 향상 방안

소회의실팀의 추가 전처리 과정

다중공선성

"모델의 성능이 좋아도 성능의 **원인을** 명확히 **파악하기** **어렵기** 때문에 특정 변수끼리 상관관계가 높은 파생변수 제거"

스케일링

"**수치형** 데이터의 상대적인 크기 차이를 **표준화하기** 위해 적용"

클러스터링

"feature들의 특징을 군집화 시켜 **target** 분류에 **영향**을 주는 **feature**생성"

새로운 Feature

"기존 feature를 다양한 방식으로 **조합하여** target분류에 영향을 주는 **새로운 feature** 생성"

전처리 과정에 대한 의문점

다중공선성

“분류모델이기 때문에 성능의 원인을 설명할 필요 없음”

스케일링

“트리 기반 모델을 이용하여 훈련을 진행하기 때문에 데이터의 크기를 맞출 필요 없음”

클러스터링

“Target분류에 영향력 없음”

새로운 Feature

“Target분류에 영향력 없음”

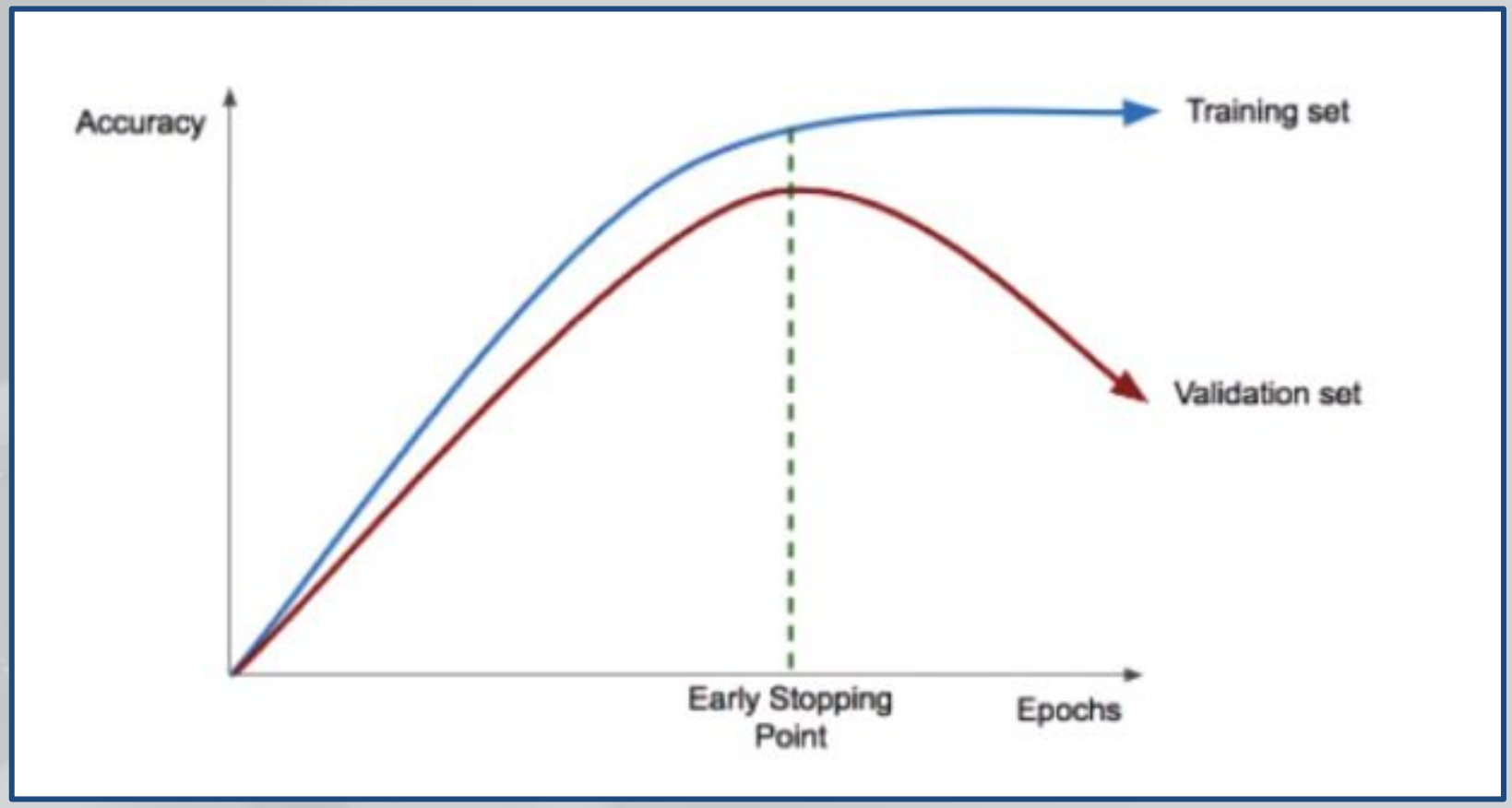
logloss score 향상 ↑

15	새싹밟지마 새싹	0.66688	3	10일 전

대회 출전팀 제침 ↑

22	소회의실	0.66714	77	8달 전

하이퍼파라미터(HyperParameter) 튜닝



- *Eearly_stopping: 최적의 정확도를 측정하기 위한 조기종료 지점.*

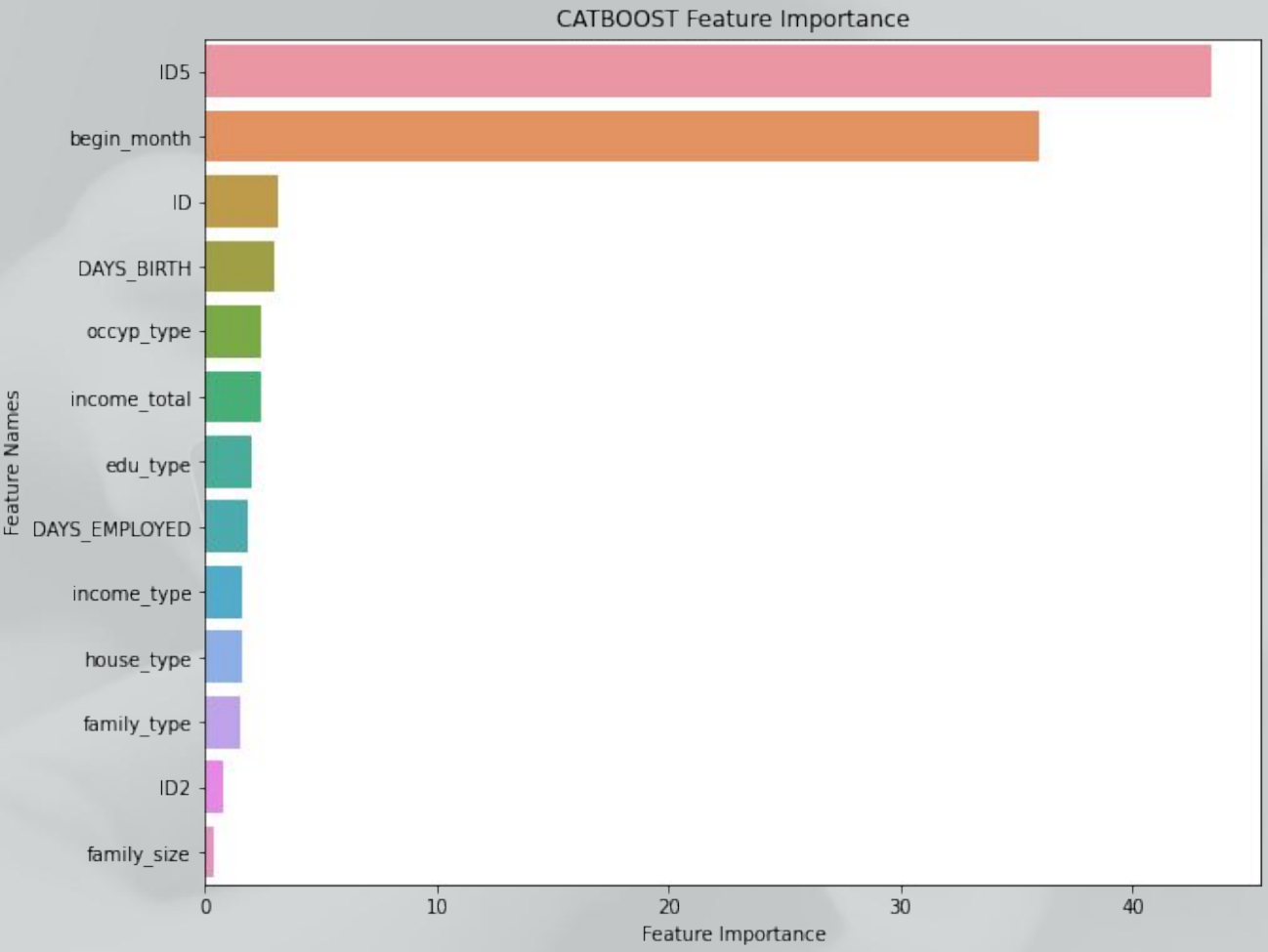
  기본값: 100 round

Score 향상

1	다냐니라	무력	0.66408	36	3시간 전
2	새싹치지마	새싹	0.66448	12	33분 전
3	chopin	ch	0.66457	5	4시간 전

3등 분석 절차

Feature_engineering




```python
# ID2 생성 : 어느정도 범위가 있는 colums을 가지고 ID2 생성.
df['ID2'] = ₩
df['income_total'].astype(str) + '_' + df['DAYS_BIRTH'].astype(str) + '_' +₩
df['DAYS_EMPLOYED'].astype(str) + '_' + df['family_size'].astype(str) + '_' +₩
df['begin_month'].astype(str)

# 파생된 변수들만 가지고 ID5을 생성.
df['ID5'] = ₩
df['before_EMPLOYED'].astype(str) + '_' + df['income_total_befofeEMP_ratio'].astype(str) + '_' +₩
df['before_EMPLOYED_m'].astype(str) + '_' + df['before_EMPLOYED_w'].astype(str) + '_' +₩
df['Age'].astype(str)+ '_' + df['DAYS_BIRTH_m'].astype(str) + '_' +₩
df['DAYS_BIRTH_w'].astype(str) + '_' + df['EMPLOYED'].astype(str) + '_' +₩
df['DAYS_EMPLOYED_m'].astype(str)+ '_' + df['DAYS_EMPLOYED_w'].astype(str) + '_' +₩
df['ability'].astype(str)+ '_' + df['income_mean'].astype(str)
```

public score: 0.6658288686
private score: 0.6574407104

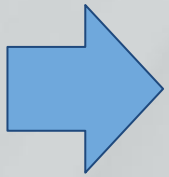
2등 분석 절차

데이터 전처리

결측치 처리

```
for df in [train,test]:
    print(df['occyp_type'].value_counts().head(5))
```

```
NaN             8171
Laborers        4512
Core staff      2646
Sales staff     2539
Managers        2167
Name: occyp_type, dtype: int64
NaN             3152
Laborers        1699
Sales staff      946
Core staff       945
Managers         845
Name: occyp_type, dtype: int64
```



```
for df in [train,test]:
    print(df['occyp_type'].value_counts().head(5))
```

```
Laborers        4512
inoccupation    4438
freelancer      3733
Core staff      2646
Sales staff     2539
Name: occyp_type, dtype: int64
Laborers        1699
inoccupation    1697
freelancer      1455
Sales staff      946
Core staff       945
Name: occyp_type, dtype: int64
```

- 직업(occyp_type)의 결측치를 freelancer와 inoccupation으로 대체

- freelancer = 노동시간이 있는경우

- inoccupation = 노동시간이 없는경우

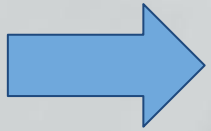
데이터 전처리

family_type

- single이고 결혼하지 않았는데 아이가 있는경우 widow로 대체

```
train.groupby('family_type')['child_num'].sum()
```

```
family_type
Civil marriage          902
Married                9183
Separated               565
Single / not married    607
Widow                    84
Name: child_num, dtype: int64
```

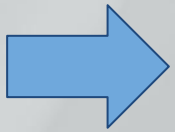


```
train.groupby('family_type')['child_num'].sum()
```

```
family_type
Civil marriage          902
Married                9183
Separated               565
Single / not married      0
Widow                   691
Name: child_num, dtype: int64
```

의미 없는 변수 제거

- index 제거
- FLAG_MOBIL 삭제:모든 값이 1로 동일



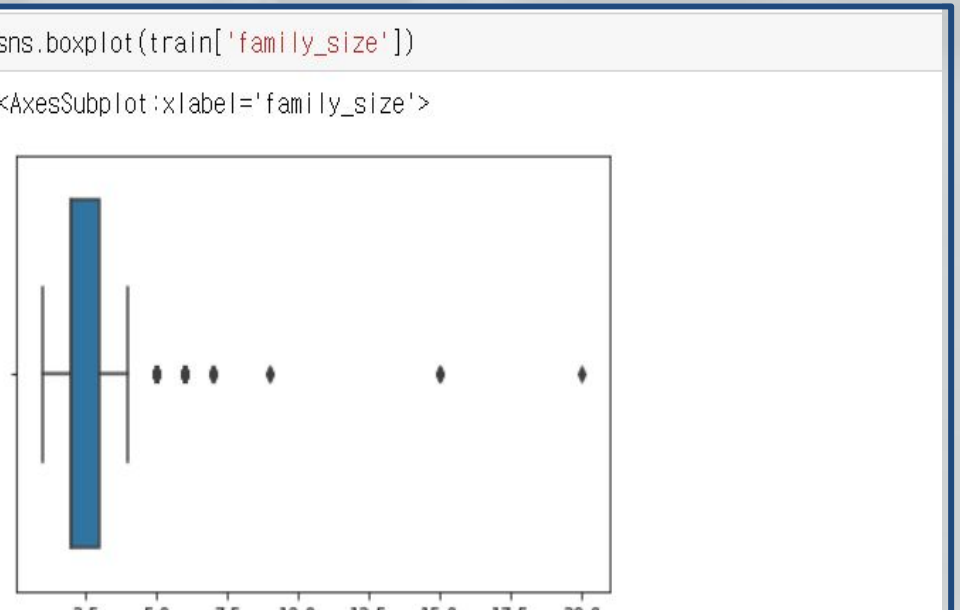
```
train.drop(['index', 'FLAG_MOBIL'], axis=1, inplace=True)
test.drop(['index', 'FLAG_MOBIL'], axis=1, inplace=True)
```

데이터 전처리

이상치 처리

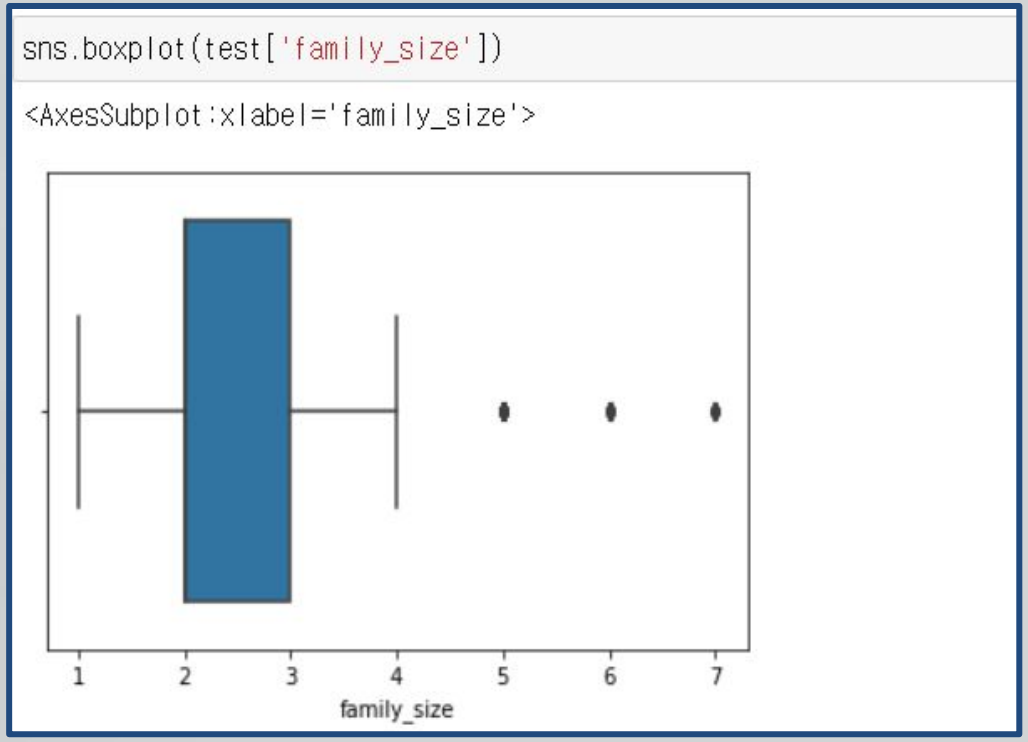
- family_size 이상치 처리 필요

```
sns.boxplot(train['family_size'])
<AxesSubplot:xlabel='family_size'>
```



- family_size > 7 인 데이터 제거(IQR 제거시 logloss 값이 너무 커지기 때문에 7보다 큰 수만 제거)

```
sns.boxplot(test['family_size'])
<AxesSubplot:xlabel='family_size'>
```



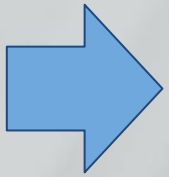
데이터 전처리

DAYS_EMPLOYED

- 양수인 데이터는 현재 무직자로 판단, 0 처리

```
for df in [train,test]:
    print(df['DAYS_EMPLOYED'].value_counts().sort_values(ascending=False).head())
```

```
 365243    4438
-401         57
-1539        47
-200         45
-2087        44
Name: DAYS_EMPLOYED, dtype: int64
 365243    1697
-1678        22
-1661        21
-401         21
-2057        20
Name: DAYS_EMPLOYED, dtype: int64
```



```
for df in [train,test]:
    print(df['DAYS_EMPLOYED'].value_counts().sort_values(ascending=False).head())
```

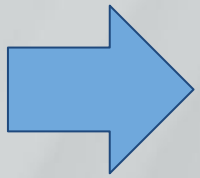
```
 0         4438
-401         57
-1539        47
-200         45
-2087        44
Name: DAYS_EMPLOYED, dtype: int64
 0         1697
-1678        22
-1661        21
-401         21
-2057        20
Name: DAYS_EMPLOYED, dtype: int64
```

데이터 전처리

DAYS_BIRTH, begin_month, DAYS_EMPLOYED

- 음수값 -> 양수 변환

	DAYS_BIRTH	begin_month	DAYS_EMPLOYED
0	-13899	-6.0	-4709
1	-11380	-5.0	-1540
2	-19087	-22.0	-4434
3	-15088	-37.0	-2092
4	-15037	-26.0	-2105
...	...	...	...
26452	-12079	-2.0	-1984
26453	-15291	-47.0	-2475
26454	-10082	-25.0	-2015
26455	-10145	-59.0	-107
26456	-19569	-9.0	-1013

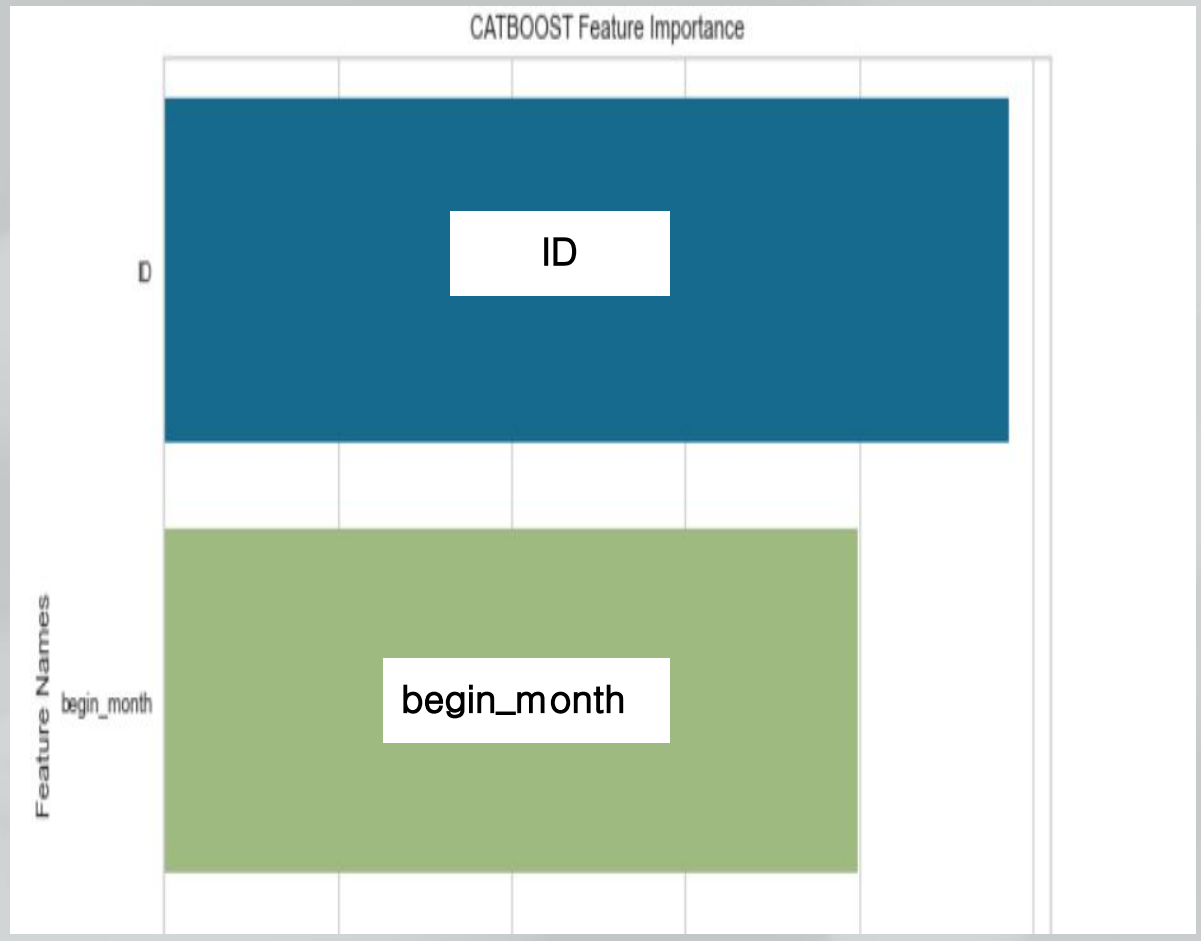


	DAYS_BIRTH	begin_month	DAYS_EMPLOYED
0	13899	6.0	4709
1	11380	5.0	1540
2	19087	22.0	4434
3	15088	37.0	2092
4	15037	26.0	2105
...	...	...	...
26452	12079	2.0	1984
26453	15291	47.0	2475
26454	10082	25.0	2015
26455	10145	59.0	107
26456	19569	9.0	1013

26457 rows × 3 columns

Feature_engineering

Feautre Importance




```
# 속성별로 ID 생성
# ID 생성: raw data의 고유 칼럼을 합침
df['ID'] = ㅐ
df['child_num'].astype(str) + '_' + df['income_total'].astype(str) + '_' +ㅐ
df['DAYS_BIRTH'].astype(str) + '_' + df['DAYS_EMPLOYED'].astype(str) + '_' +ㅐ
df['work_phone'].astype(str) + '_' + df['phone'].astype(str) + '_' +ㅐ
df['email'].astype(str) + '_' + df['family_size'].astype(str) + '_' +ㅐ
df['gender'].astype(str) + '_' + df['car'].astype(str) + '_' +ㅐ
df['reality'].astype(str) + '_' + df['income_type'].astype(str) + '_' +ㅐ
df['edu_type'].astype(str) + '_' + df['family_type'].astype(str) + '_' +ㅐ
df['house_type'].astype(str) + '_' + df['occyp_type'].astype(str)
```

- 중복 데이터의 비율(begin_month 제외)이 높으므로 식별할 수 있는 feature(ID) 생성

- CatBoost 모델링 결과 Feautre Importance 10% 미만 feature 제거 후 학습

public score: 0.665183242
private score: 0.657935256

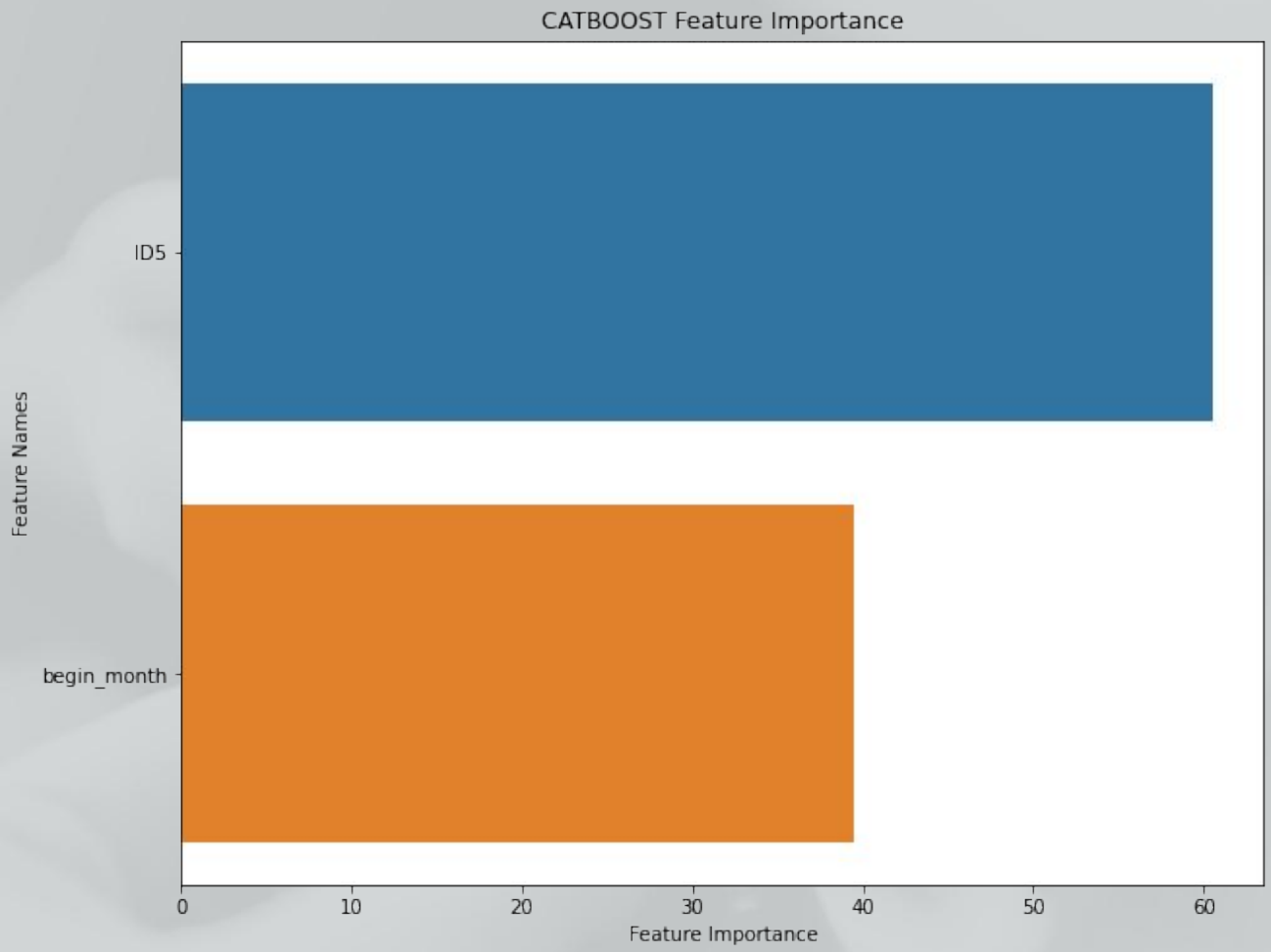
1등 분석 절차

Feature_engineering

```
df['ID5'] = #
df['before_EMPLOYED'].astype(str) + '_' + df['income_total_befofeEMP_ratio'].astype(str) + '_' +#
df['Age'].astype(str)+ '_' + df['EMPLOYED'].astype(str) + '_' +#
df['ability'].astype(str)+ '_' + df['income_mean'].astype(str) #age는 빼는게 좋아보임.

df['ID5'] = #
df['before_EMPLOYED'].astype(str) + '_'+ df['ability'].astype(str)  # log loss: 0.661

df['ID5'] = #
df['before_EMPLOYED'].astype(str) + '_'+ df['income_mean'].astype(str) # log loss: 0.662

df['ID5'] = #
df['income_total_befofeEMP_ratio'].astype(str) + '_'+ df['EMPLOYED'].astype(str) # log loss: 0.661

df['ID5'] = #
df['income_total_befofeEMP_ratio'].astype(str) + '_'+ df['ability'].astype(str) # log loss: 0.662

df['ID5'] = #
df['before_EMPLOYED'].astype(str) + '_'+ df['ability'].astype(str) # log loss : 0.661

df['ID5'] = #
df['before_EMPLOYED'].astype(str) + '_'+ df['ability'].astype(str)  + '_' +#
df['income_mean'].astype(str)  # log loss: 0.661

df['ID5'] = #
df['before_EMPLOYED'].astype(str) + '_'+ df['ability'].astype(str)  + '_' +#
df['income_total_befofeEMP_ratio'].astype(str)  # log loss: 0.661
```

최적 성능 조합 선택

Feature_engineering




```
# 기존 ID5에서 최소한의 columns으로 새로운 ID5 생성
df['ID5'] = ##
df['before_EMPLOYED'].astype(str) + '_'+ df['ability'].astype(str)
```

public score: 0.6640818521
private score:
0.6572957219

04. 결 론

결론

결론 및 해결

1. 80% 이상이 중복데이터 처리

→ 중복 데이터의 특성을 가진 feature생성

2. 모델이 소수의 변수에 의존하는 경우

→ 타겟과 관련성이 적은 특성을 줄여서 모델성능을 높임

3.범주형 데이터를 이용한 데이터 분류 할 때

→ catboost를 이용하여 encoding 없이 학습

THANK YOU