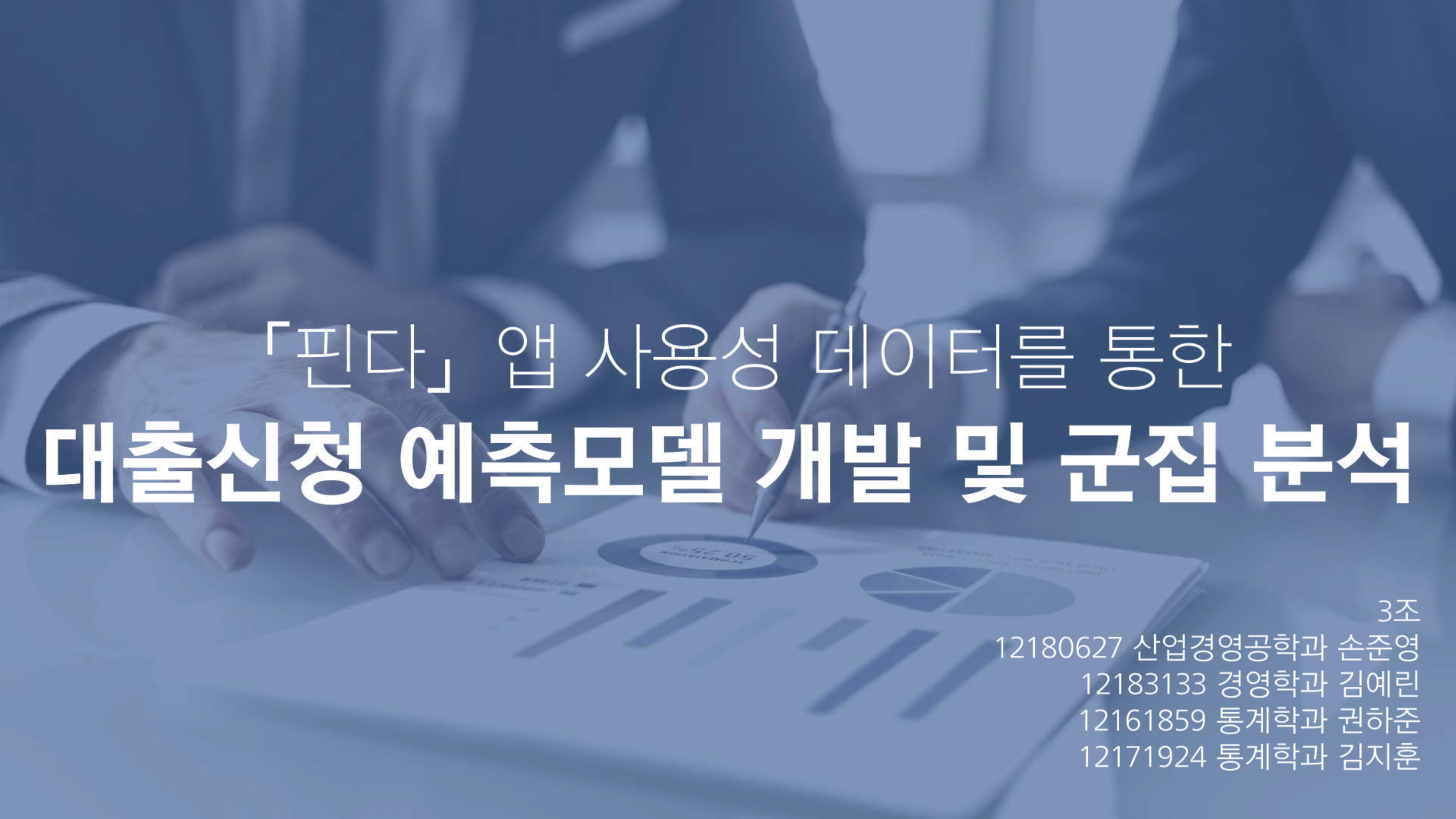


「핀다」 앱 사용성 데이터를 통한

대출신청 예측모델 개발 및 군집 분석

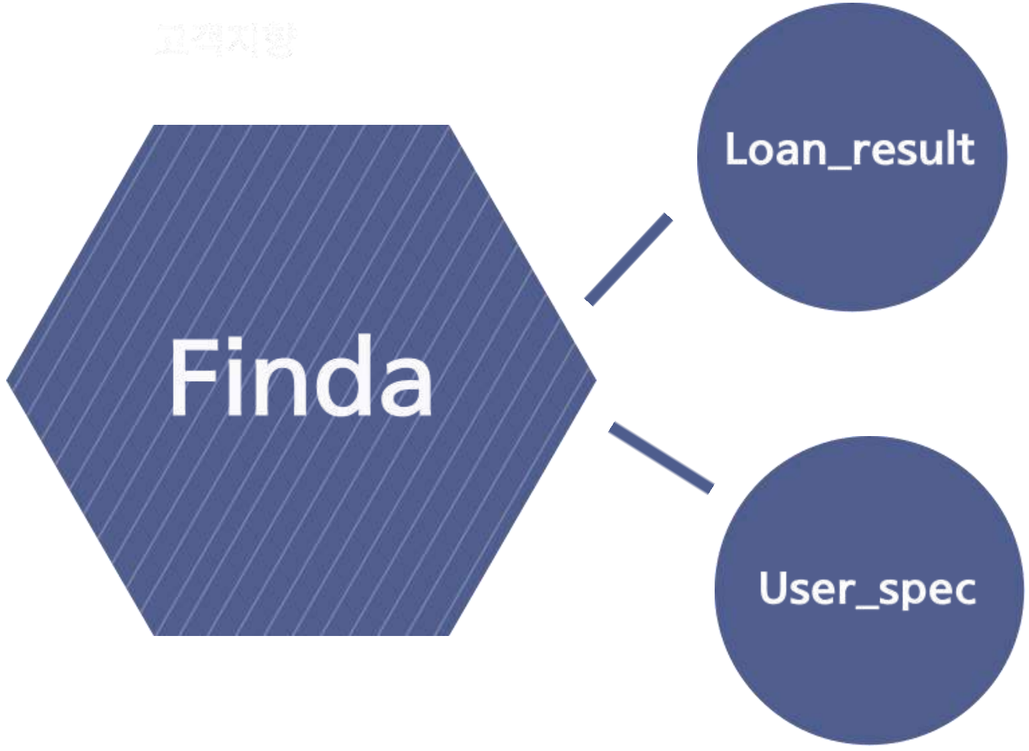
3조
12180627 산업경영공학과 손준영
12183133 경영학과 김예린
12161859 통계학과 권하준
12171924 통계학과 김지훈

빅콘테스트: 데이터분석리그

POINT 1

퓨처스 부문	앱 사용성 데이터를 통한 대출신청 예측분석 ▪ 가명화된 데이터를 기반으로 고객의 대출상품 신청여부 예측 (2022년 3~5월 데이터제공 / 2022년 6월 예측) ▪ 예측모델을 활용하여 탐색적 데이터 분석 수행 ▪ 대출신청, 미신청 고객을 분류하여 고객의 특성 분석결과 도출

데이터셋 설명



Loan_result

01 핀다 앱을 통해 신청한 금융사별 대출 상품 승인 결과

shape : (13527363, 7)

User_spec

02 가명화된 핀다 앱 사용자의 개인 정보

shape : (1394216, 17)

	application_id	loanapply_insert_time	bank_id	product_id	loan_limit	loan_rate	is_applied
0	2157865	2022-05-09 08:44:59	54	235	20000000.0	16.5	1.0
1	576643	2022-05-09 10:54:53	54	235	11000000.0	16.5	0.0
2	576643	2022-05-09 10:54:53	11	118	3000000.0	20.0	0.0
3	2136706	2022-05-09 10:41:06	42	216	10000000.0	13.5	0.0
4	2136706	2022-05-09 10:41:07	25	169	22000000.0	15.9	0.0

- application_id, bank_id, product_id : 대출 신청서, 은행, 대출 상품 각각의 id
- loanapply_insert_time : 한도조회 일시
- loan_limit : 대출 승인 한도(십만 단위 반올림)
- loan_rate : 대출 승인 금리(둘째 자리 반올림)
- is_applied : 대출 승인 여부

	application_id	user_id	birth_year	gender	insert_time	credit_score	yearly_income	income_type	company_enter_month	employment_type	houseo
0	1249046	118218	1985.0	1.0	2022-06-07 06:28:18	660.0	108000000.0	PRIVATEBUSINESS	20151101.0	기타	
1	954900	553686	1968.0	1.0	2022-06-07 14:29:03	870.0	30000000.0	PRIVATEBUSINESS	20070201.0	정규직	기타
2	137274	59516	1997.0	1.0	2022-06-07 21:40:22	710.0	30000000.0	FREELANCER	20210901.0	기타	기타
3	1570936	167320	1989.0	1.0	2022-06-07 09:40:27	820.0	62000000.0	EARNEDINCOME	20170101.0	정규직	
4	967833	33400	2000.0	1.0	2022-06-07 08:55:07	630.0	36000000.0	EARNEDINCOME	20210901.0	정규직	기타

- user_id : 고객 id
- birth_year : 생년월일
- gender : 성별(0 : 여자 / 1 : 남자)
- insert_time : 생성 일시
- credit_score : 한도조회 당시 유저 신용점수
- yearly_income : 연소득
- income_type : 근로형태
- company_enter_month : 입사연월
- employment_type : 고용형태
- houseown_type : 주거소유형태
- desired_amount : 대출희망금액
- purpose : 대출 목적
- personal_rehabilitation_yn : 개인회생자 여부
- personal_rehabilitation_complete_yn : 개인회생자 납입 여부 완료 여부
- existing_loan_cnt : 기대출 수
- existing_loan_amt : 기대출 금액

결측치 수

application_id	0
loanapply_insert_time	0
bank_id	0
product_id	0
loan_limit	7495
loan_rate	7495
is_applied	3257239

Loan_result 데이터의 결측치 수

application_id	0
user_id	0
birth_year	8593
gender	8593
insert_time	0
credit_score	81769
yearly_income	1
income_type	0
company_enter_month	92314
employment_type	0
houseown_type	0
desired_amount	0
purpose	0
personal_rehabilitation_yn	417763
personal_rehabilitation_complete_yn	843993
existing_loan_cnt	146290
existing_loan_amt	225046

User_spec 데이터의 결측치 수

유추 가능한 경우의 결측치 처리

application_id	user_id	birth_year	gender
132643	49072	NaN	NaN
484786	49072	1985.0	0.0
858081	49072	1985.0	0.0
1967101	49072	1985.0	0.0

user_id를 통해 유추 가능한 고정적인 정보

(생년월일, 성별, 개인회생자 여부 등)는 적절하게 대체

-> 결측치만 있는 user_id 컬럼 데이터는 채워넣을 방법이 없으므로 삭제

신용점수(credit_score) 컬럼의 결측치 처리

application_id	user_id	birth_year	gender	insert_time	credit_score
368073	858775	1996.0	0.0	2022-03-29 10:14:05	NaN
203848	858775	1996.0	0.0	2022-05-20 16:49:25	560.0
1573721	858775	1996.0	0.0	2022-05-22 12:30:54	530.0

신용점수는 시간에 따라 변화할 수 있는 가변적인 데이터

: 처음에는 결측값인 반면, 이후에 신용점수가 업데이트

-> 결측치 이후 신용점수를 처음으로 알 수 있었던 시기와 가장 가까운 날짜의 신용점수로 대체

기대출 수 결측치 처리

```
1  dat1['existing_loan_cnt'].describe()
```

```
count    1.195660e+06
mean     4.743685e+00
std      4.327669e+00
min      1.000000e+00
25%      2.000000e+00
50%      4.000000e+00
75%      6.000000e+00
max      2.780000e+02
Name: existing_loan_cnt, dtype: float64
```


기대출 수의 결측치를 제외한 분포는 최소값(min)이 1로

기대출 횟수가 아예 없는 0회에 해당하는 데이터는 존재 x

-> 기대출 수가 결측치인 값은 기대출이 아예 없는 사람으로 0으로 대체

-> 기대출이 없으면 기대출 금액도 없으므로 기대출 금액 또한 0으로 대체

```
dat1['existing_loan_cnt'].isnull().sum()
```

```
146290
```

```
q="""SELECT * FROM dat1 WHERE user_id IN (SELECT user_id FROM dat1 WHERE existing_loan_cnt IS NULL)"""
exloannone=sqldf(q,locals())
# 기대출수가 결측치인 값을 보유한 user_id의 데이터를 전부 불러옴.
```

```
len(exloannone)
```

```
146290
```

기대출 수가 결측치인 데이터 수 = 기대출 수에 결측치 값이 있는 사용자의 수

-> 처음부터 기대출이 없었던 것으로 판단

기대출 금액 결측치 처리

existing_loan_cnt	existing_loan_amt
1.0	None
1.0	None
1.0	None
1.0	None
1.0	None

기대출 수가 0인 고객의 기대출 금액을 0으로 대체

기대출 수가 1 이상인 경우 시간의 흐름에 따라 변할 수 있어 예측이 어려움

-> 데이터 삭제

기타 결측치 처리

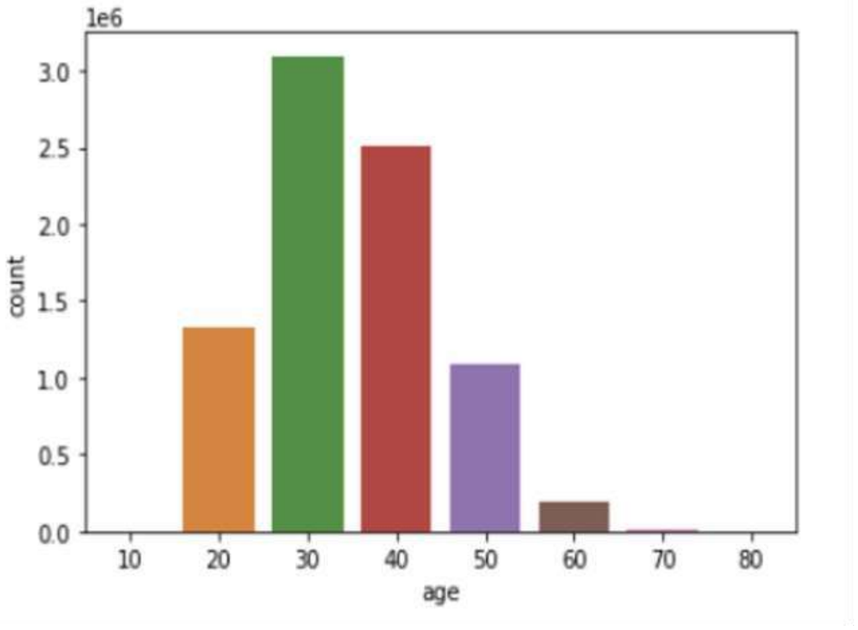
application_id	0
user_id	0
birth_year	0
gender	0
insert_time	0
credit_score	0
yearly_income	0
income_type	0
company_enter_month	0
employment_type	0
houseown_type	0
desired_amount	0
purpose	0
personal_rehabilitation_yn	0
personal_rehabilitation_complete_yn	0
existing_loan_cnt	0
existing_loan_amt	0
loanapply_insert_time	0
bank_id	0
product_id	0
loan_limit	0
loan_rate	0
is_applied	0

개인 회생자 신청 여부, 개인 회생자 납입 여부의 경우
같은 user_id를 통하여 유추할 수 있는 데이터는 그에 맞게 처리
나머지 알 수 없는 결측치는 개인회생자에 해당되지 않는 고객으로 간주 -> 0으로 대체

입사년월의 경우,
user_id를 통하여 입사 날짜를 구할 수는 있지만, 퇴사나 재입사 등의 요인 발생 가능
-> 입사년월의 결측치 데이터 삭제

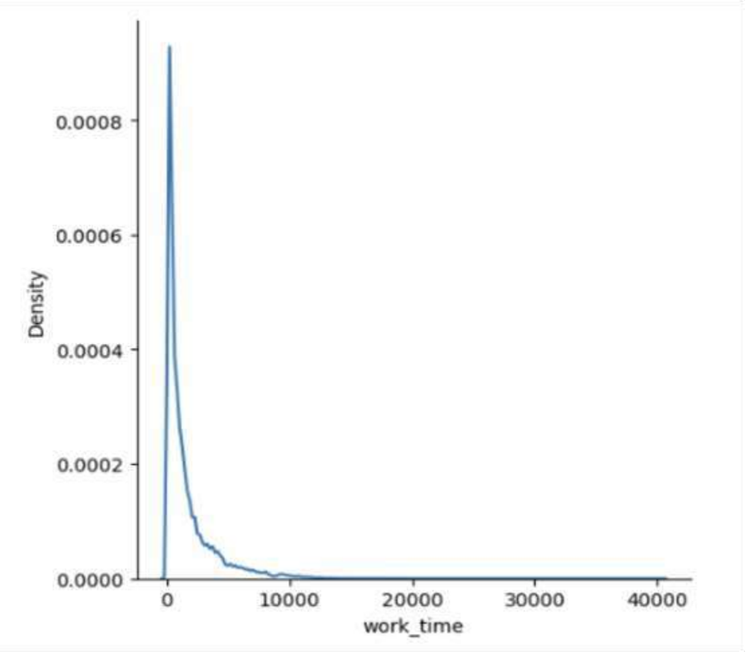
대출한도나 대출금리의 경우,
데이터 출처에 따르면 은행에서 제공해주지 않은 데이터이므로 무시해도 좋다고 명시
-> 대출 한도 및 대출 금리 결측치 데이터 삭제

연령대(Age)



대출 신청일 – 생년월일 -> ***연령 데이터*** 생성

근무일수(Work_time)



입사년월: 대출 신청일 – 입사년월 -> ***근무일수*** 데이터 생성

입사년도의 오기입으로 인한 분포 상 이상치 존재

(음수 혹은 20000일 이상의 지나치게 큰 데이터)

-> 해당 데이터 삭제

재직 등급(Work_rank)

```
dat['work_time'] = dat['work_time']/365
# 근무일수의 단위를 연으로 바꿈.

bins = [0,1/12,0.5,1,100]

labels = [0,1,2,3]

dat['work_rank']=pd.cut(dat['work_time'],bins,right=False,labels=labels)
```

은행사를 통해 재직 기간 (1개월, 6개월, 1년이상)에
따라 대출 상품의 수와 조건이 달라진다는 것 확인
-> 이 기준으로 근무일수를 분리하여 ***재직 등급 데이터*** 생성

기대출(Ex_loan)

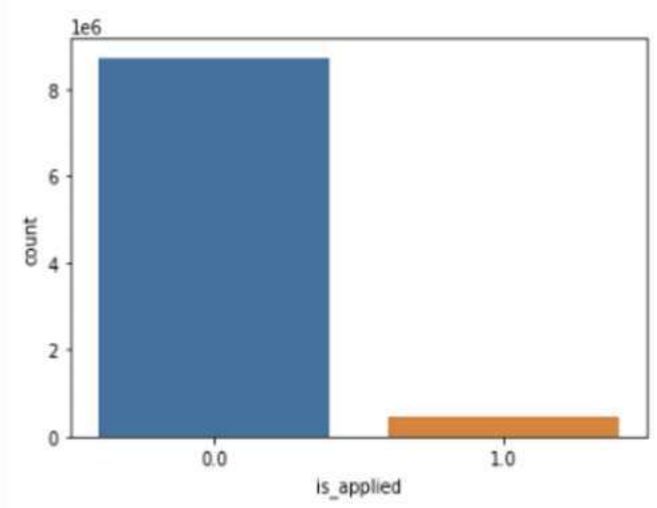
```
dat['exloan']=0

dat.loc[dat['existing_loan_cnt']>0,'exloan']=1

dat.loc[dat['existing_loan_amt']>2*dat['yearly_income'],'exloan']=2
```

기대출 과다자 : 연간 수입보다 기대출 금액이 2배 이상인 사람
기대출 수와 기대출 금액 데이터를 이용하여
기대출이 없는 사람 / 기대출이 있지만 과다자가 아닌 사람 / 기대출 과다자로 분리

| 4. 향후 프로젝트 계획




```
dat['personal_rehabilitation_complete_yn'] = dat['personal_rehabilitation_complete_yn'].astype('category')
dat['gender'] = dat['gender'].astype('category')
dat['is_applied'] = dat['is_applied'].astype('category')
dat['exloan'] = dat['exloan'].astype('category')
dat['age'] = dat['age'].astype('category')
dat['work_rank']=dat['work_rank'].astype('category')
dat['yearly_income']=dat['yearly_income'].astype('float32')
dat['desired_amount']=dat['desired_amount'].astype('float32')
dat['loan_limit']=dat['loan_limit'].astype('float32')
dat['loan_rate']=dat['loan_rate'].astype('float16')
dat['credit_score']=dat['credit_score'].astype('int16')
dat['work_time'] = dat['work_time'].astype('float32')
# 메모리 크기를 줄이기 위하여 타입 변경.
```

오버샘플링(SMOTE) 기법을 활용한 데이터 불균형 문제 해소

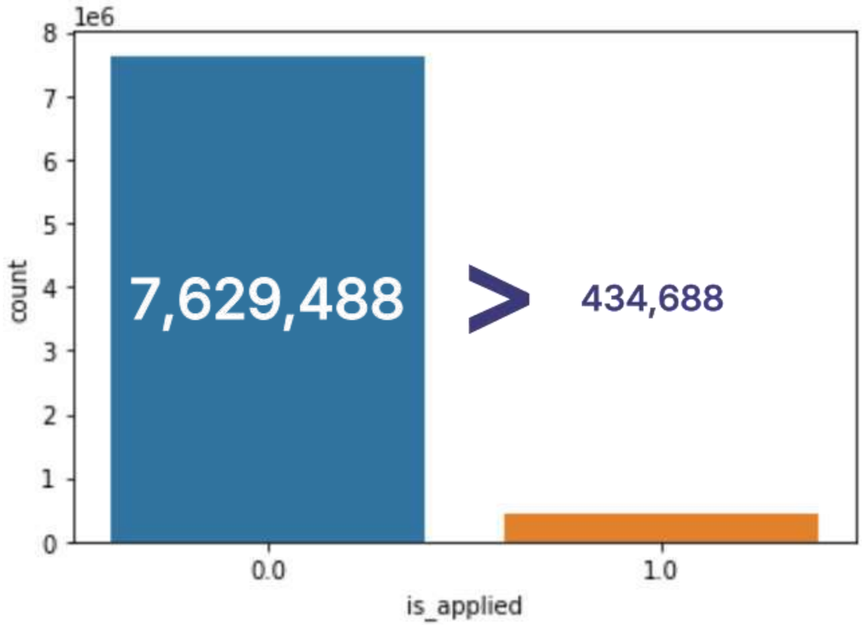
범주형 데이터에 대한 라벨 인코딩 수행

-> n개의 범주형 데이터를 0~n-1수치의 수치 데이터로 변경

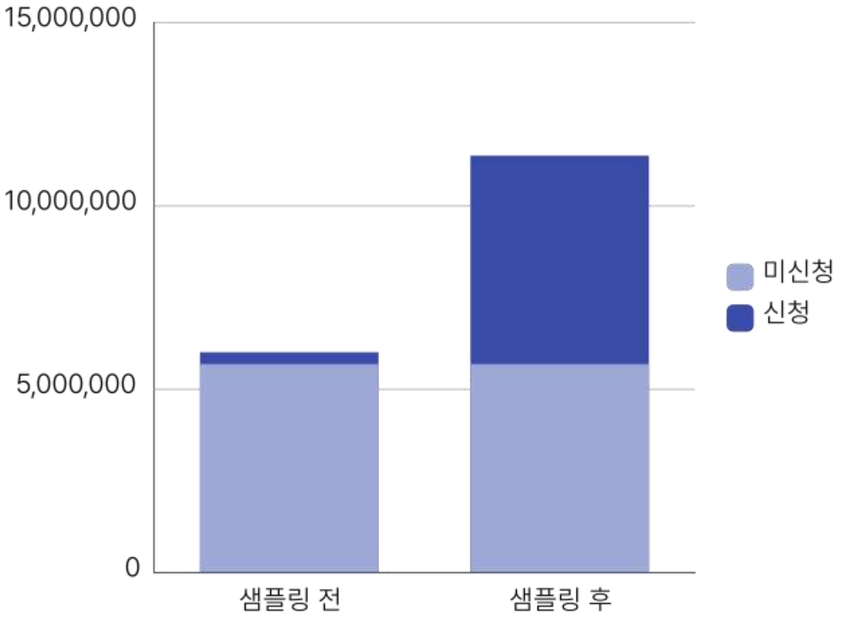
데이터 메모리 크기를 줄이기 위하여 타입 변경

-> 메모리 크기가 크면 분석 시 시간이 매우 많이 소요되므로 크기를 줄이는 것이 중요

Oversampling



17:1의 불균형 데이터셋



1:1의 균형 데이터셋

Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

불균형이 심한 데이터에서는
비중이 높은 클래스에 대한 예측만 하더라도
높은 정확도가 나올 수 있음→ 부적합한 평가지표

F1 - score

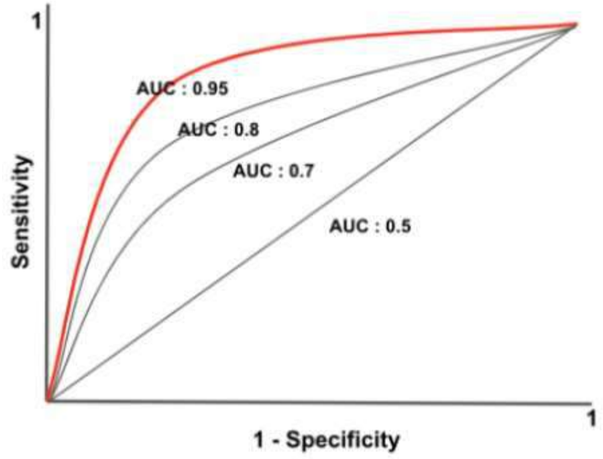
$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

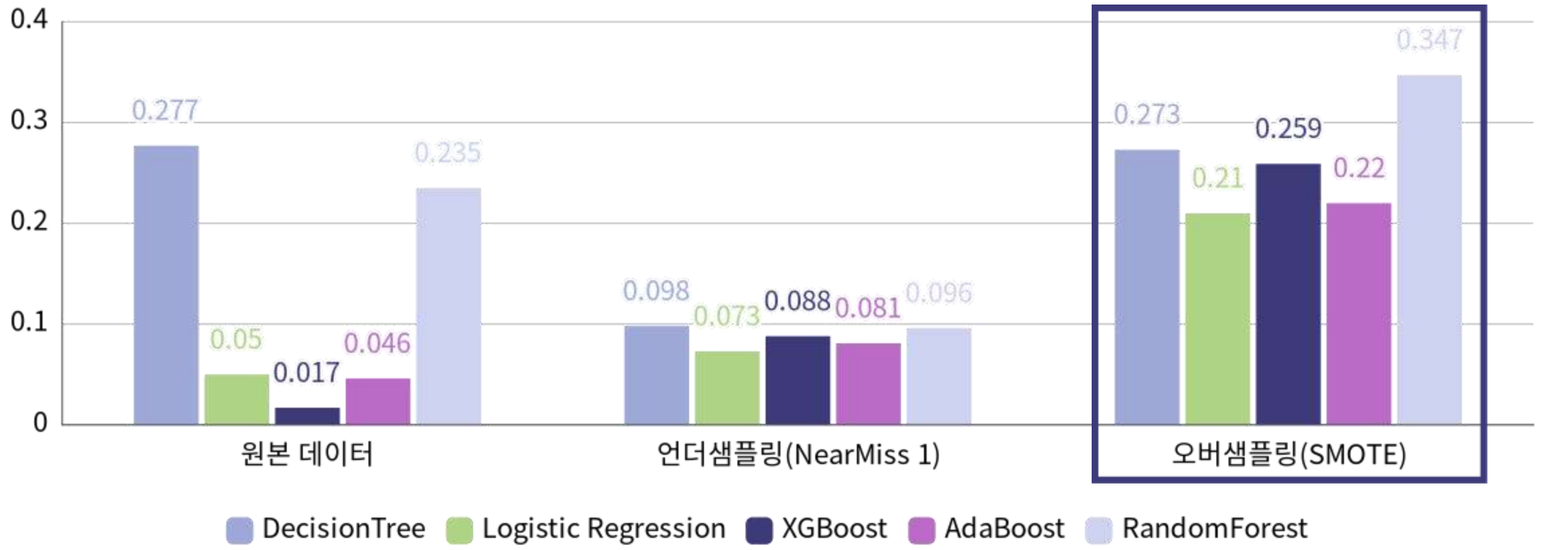
정밀도와 재현율의 수치가
적절하게 조합되어 사용
→ 1에 가까울수록 성능이 좋음

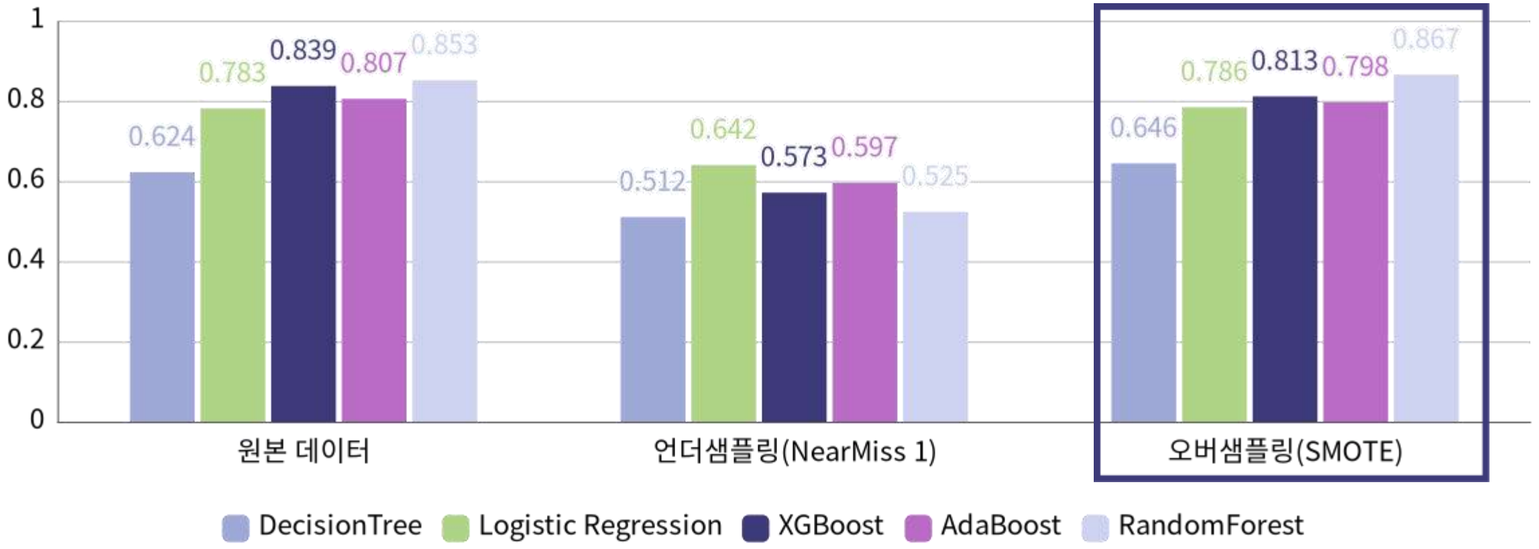
ROC-AUC score



이진 분류의 예측 성능 측정에서
중요하게 사용되는 지표
→ 1에 가까울수록 성능이 좋음

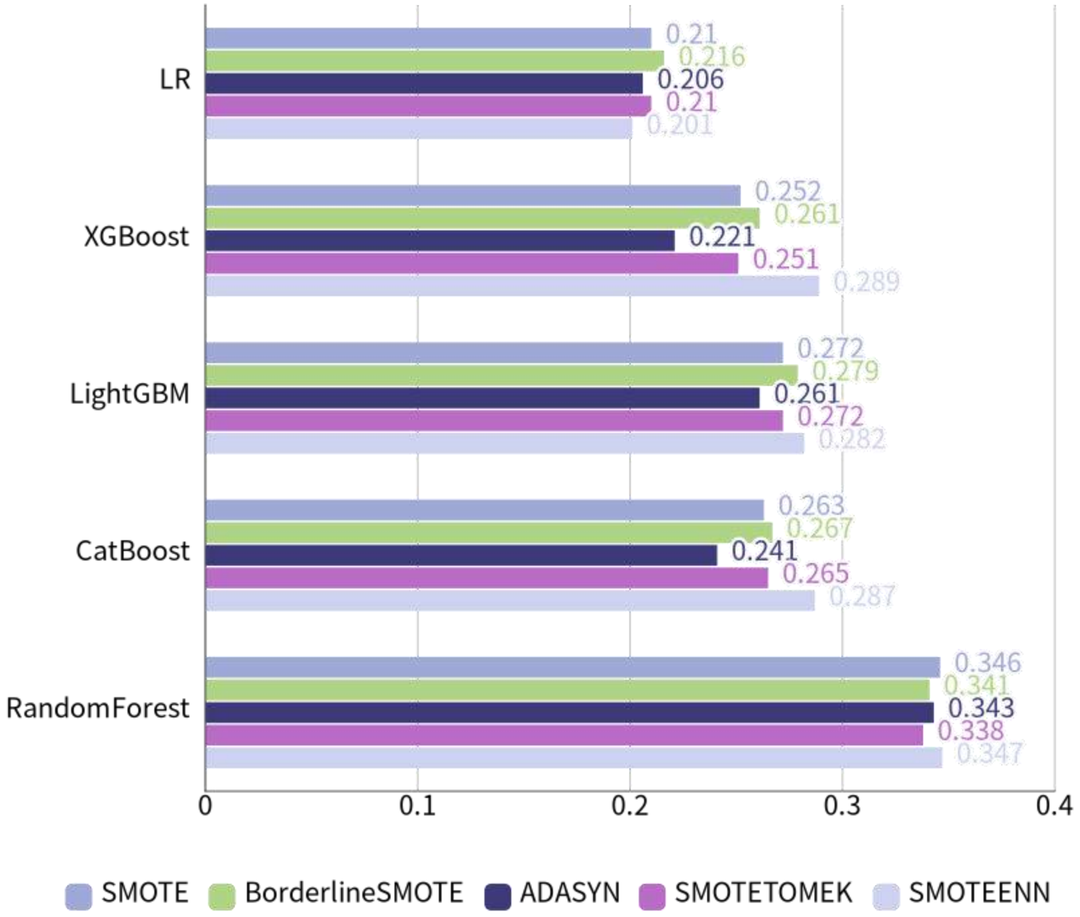
✔ F1-score 기준
샘플링 성능





✔ROC - AUC 기준
샘플링 성능

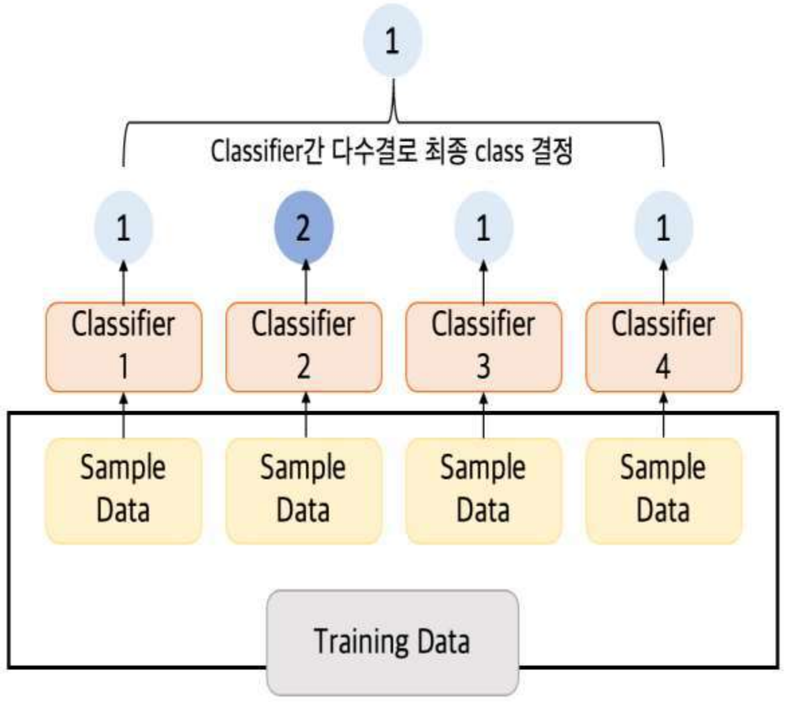
✔ F1-score 기준 샘플링 성능



✔ ROC-AUC score 기준 샘플링 성능

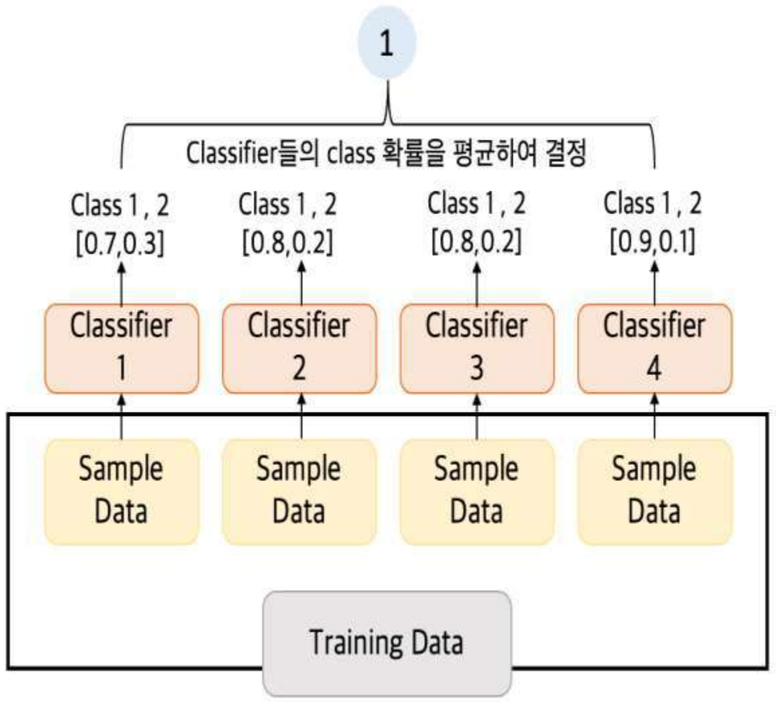
	Hyper Parameter
Logistic Regression	'C' : 1, 'penalty' : l2
XGBoost	'learning_rate' : 1, 'max_depth' : 10, 'min_child_weight' : 5
LightGBM	'learning_rate' : 0.2, 'max_depth' : -1, 'min_child_samples' : 15, 'num_leavs' : 80
CatBoost	'depth' : 6, 'iterations' : 1000, 'l2_leaf_reg' : 1e-19, 'leaf_estimation_iterations' : 10, 'leaf_estimation_iterations' : 10

약 1,200만개의 데이터에 대한 튜닝을 위해 **GPU 가속**이 가능한 **부스팅 모델** 위주로 GridsearchCV 진행



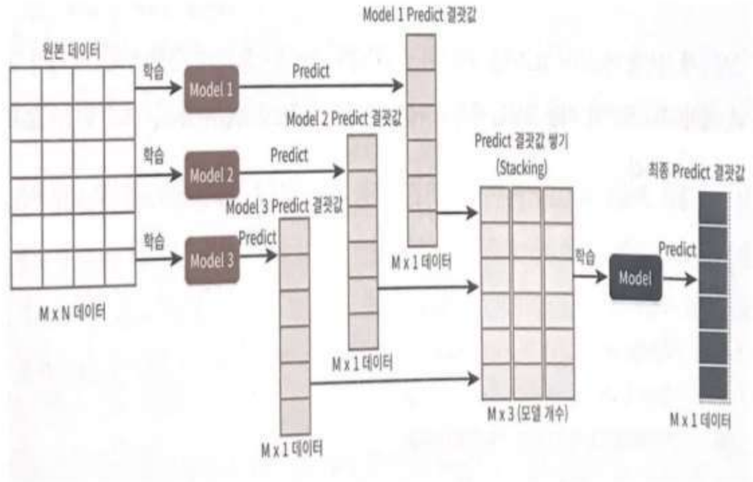
하드 보팅(Hard Voting)

분류기들의 레이블 값 결정 확률을 모두 더하고
이를 평균해서 이들 중 확률이 가장 높은 레이블 값을
최종 보팅 결괏값으로 선정



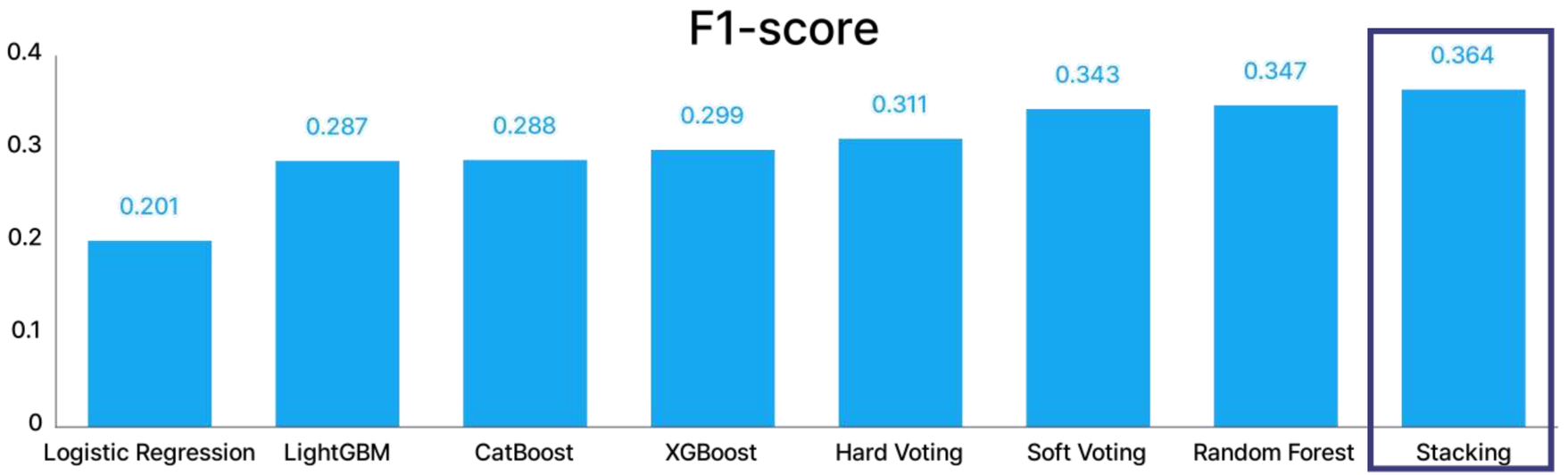
소프트 보팅(Soft Voting)

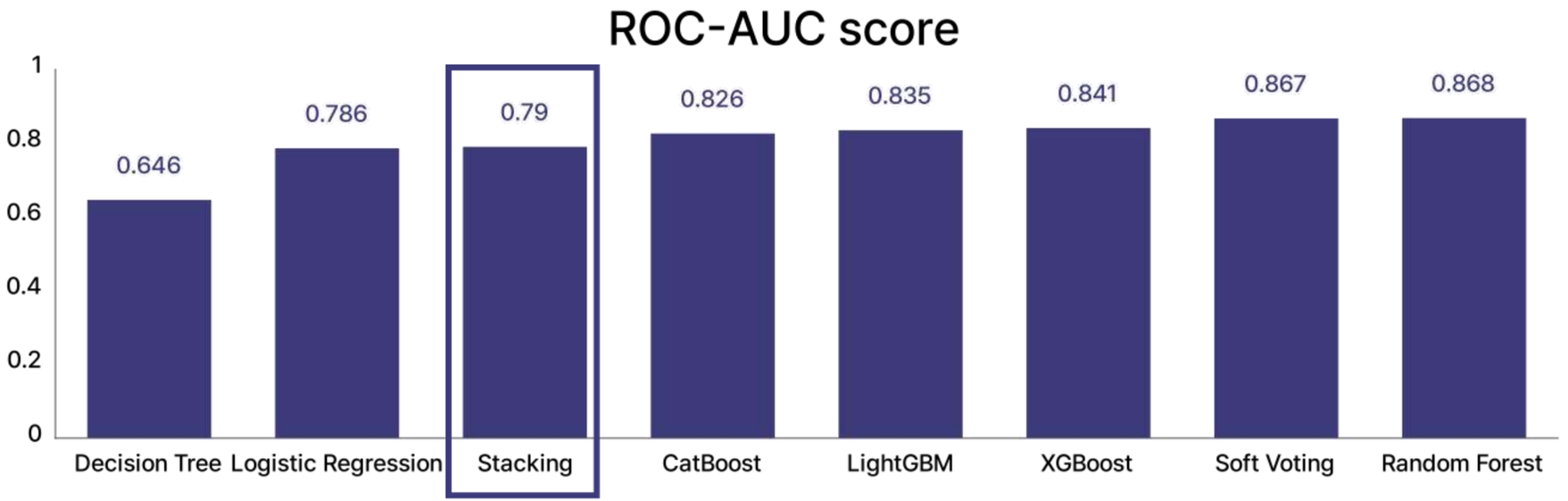
예측 결괏값들중 다수의 분류기가 결정한 예측값을
최종 보팅 결괏값으로 선정



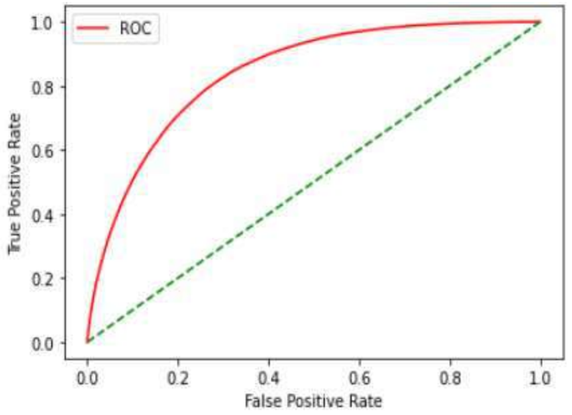
스태킹(Stacking)

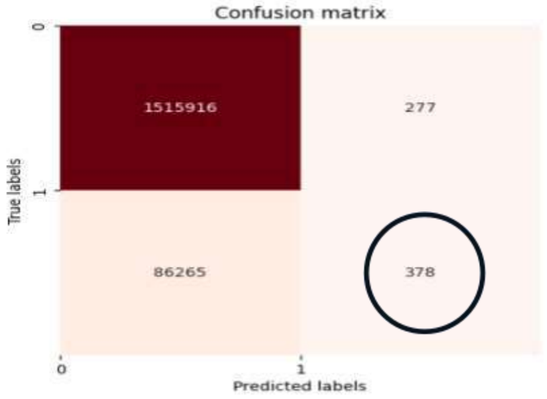
여러 가지 다른 모델의 예측 결과값을
다시 학습 데이터로 만들어 다른 모델(메타 모델)로
재학습시켜 결과를 나타내는 방법


F1-score
0.4
0.3
0.2
0.1
0
0.201
0.287
0.288
0.299
0.311
0.343
0.347
0.364
Logistic Regression
LightGBM
CatBoost
XGBoost
Hard Voting
Soft Voting
Random Forest
Stacking


ROC-AUC score
1
0.8
0.6
0.4
0.2
0
0.646
0.786
0.79
0.826
0.835
0.841
0.867
0.868
Decision Tree
Logistic Regression
Stacking
CatBoost
LightGBM
XGBoost
Soft Voting
Random Forest

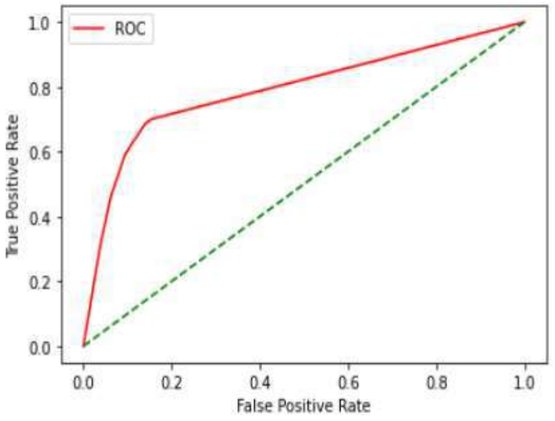
원본 데이터
LightGBM

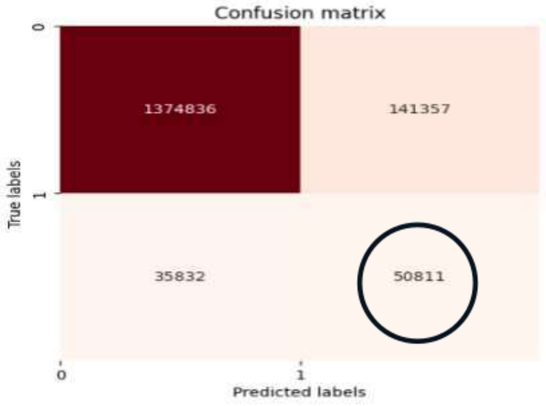




F1 Score : 0.009
ROC-AUC score : 0.841
ACC : 0.946

최종 예측 모델
Stacking(Meta Model : LightGBM)





F1 Score : 0.364
ROC-AUC score : 0.790
ACC : 0.889

정확도와 ROC-AUC score는 소폭 하락했지만 F1 score가 대폭 상승하였고,
실제 86,600개의 1 class에 대하여 378개 맞추던 것이 50,811개까지 맞추면서 성능이 대폭 향상

피처 선택 과정

기존 피처들에서 유의미한 인사이트를 찾지 못함
앱을 직접 사용해보면서 feature selection 진행
→ 분석 대상에서 제외된 피처가 꽤 많음

군집 분석

예측 분류 모델 개발에 더해 계획 했던
고객 특성별 군집 분석을 수행하지 못함

Random Forest 모델의 활용

분석 환경이 부족해 추가적인 활용 불가
추가적인 복합 모델 활용 및
파라미터 튜닝을 하지 못함