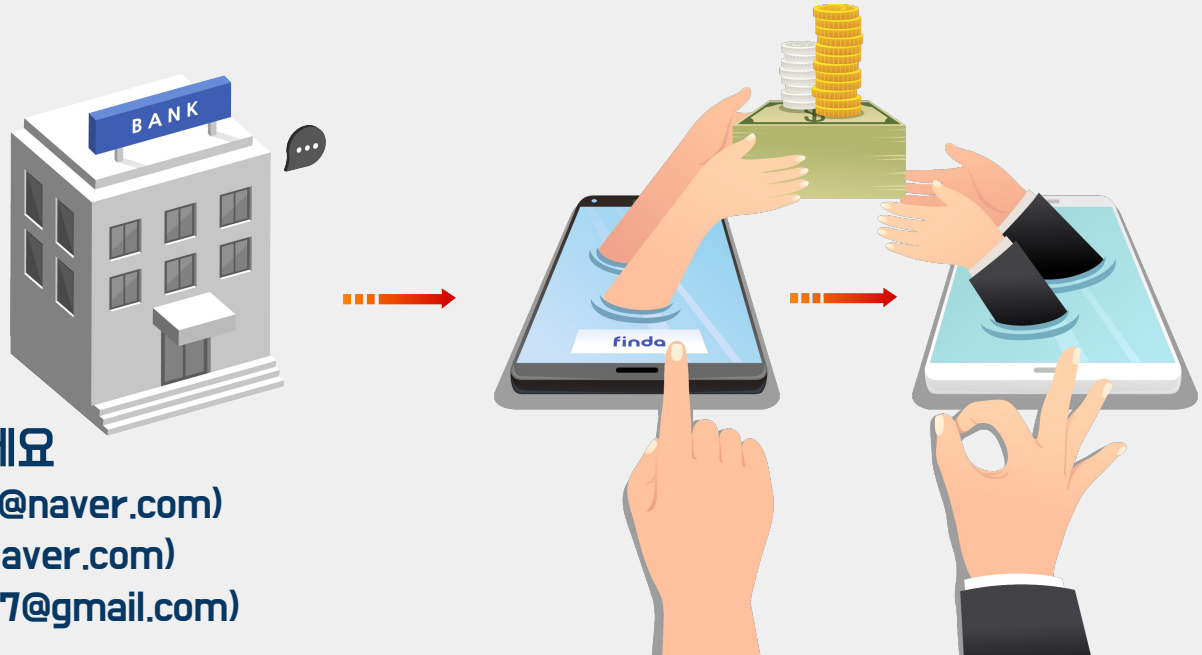
2022빅콘테스트 X finda

# 앱 사용성 데이터를 통한 대출 신청 예측 분석




Team. 우 걱정마세요
김혜현 (hyehyeon57@naver.com)
임성수 (eric_best@naver.com)
황성아 (sunga.kr0717@gmail.com)

# Index

2022빅콘테스트 X finda
Team. 우 걱정마세요

## 01. 분석 주제 이해

- 주제 요약 및 관련 이슈
- 분석 목표
- 아이디어 회의

## 02 데이터 이해 및 탐색

- 데이터 이해
- 데이터 탐색

## 03 데이터 전처리

- user_spec
- loan_result
- log_data
- 최종 데이터 셋

## 04 모델링

- 모델링 준비
- 모델링 진행

## 05 모델 비교 및 선정

- 모델 비교
- 최종 모델 선정

## 06 활용 방안 및 기대 효과

- 활용 방안 및 기대 효과

# 01. 분석 주제 이해

# 01. 분석 주제 이해



"대출이 필요한데. 대출시 고려해야 하는 변동요인도 늘고... 대출 도대체 어떻게 받아야 할까요?"


'대출금리 어쩌나'...8월 은행 가계대출 금리 4.76%

경제 인사이트] 가계대출 총량 규제, 절망하는 서민

우리 집 주택담보대출금리는 고정금리로 바꿔야할까?

DSR 3단계 규제 시행…

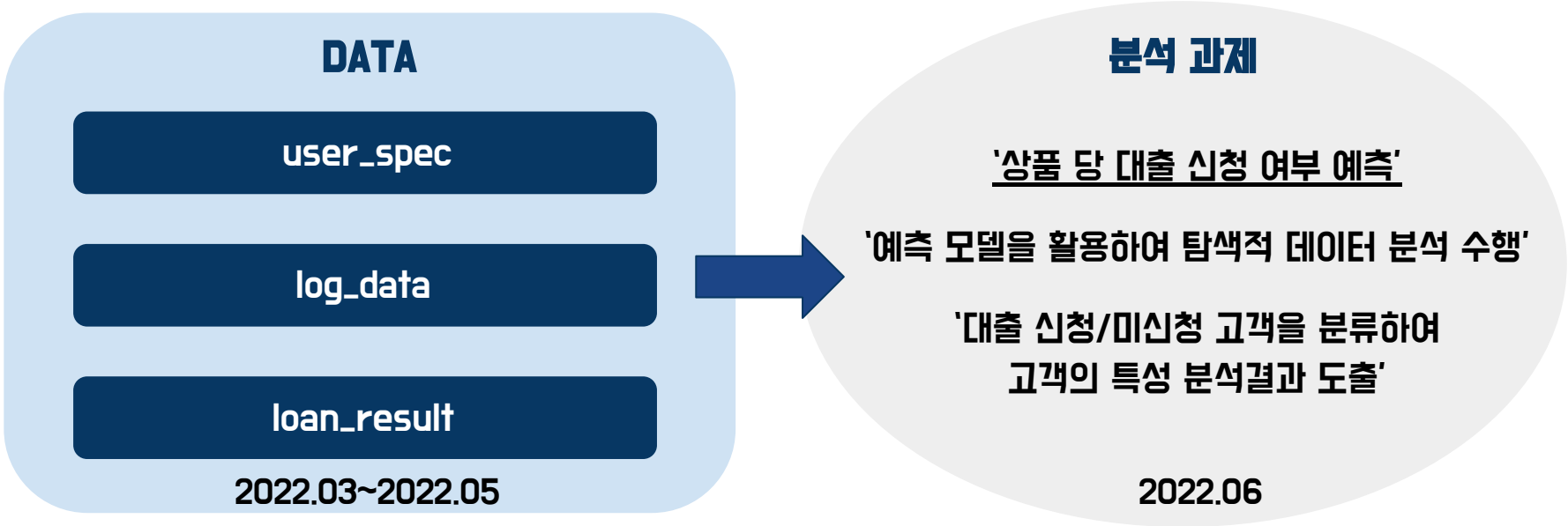
7월부터 'DSR 40%' 규제 대상 확대

핀다, 중저신용 고객 4명 중 1명은 9%대 이하 금리로 대출받아

핀다는 여러 대출상품을 한번의 조회만으로 고객에게 가장 현명한 판단을 내리게 도와주는 플랫폼이다.
고신용자를 포함하여 중-저신용자 모두에게 변동요인을 고려한 알맞은 서비스를 제공해주고 있다.

# 01. 분석 주제 이해






앱 사용성 데이터를 통한 분석을 진행하면서, 고객의 대출신청여부를 예측하고,
'핀다'는 기업내 수익예측과 예산의 올바른 분배를 이루고, 더 나아가 새로운 고객유입이라는 파생효과 기대

# 01. 분석 주제 이해



## 대출 여부에 영향을 미치는 요인은?

### 01. 유저의 기본정보를 담은 스펙데이터

➡ 고객의 직군(근로형태), 대출신청시 신용점수, 대출목적, 대출희망금액, 기대출수, 개인회생자 여부 등

### 02. 유저의 앱 사용로그 데이터

➡ 일 코드, 행동일시, 행동명(회원가입,로그인,핀다 앱 실행 … KCB 신용정보조회) 등

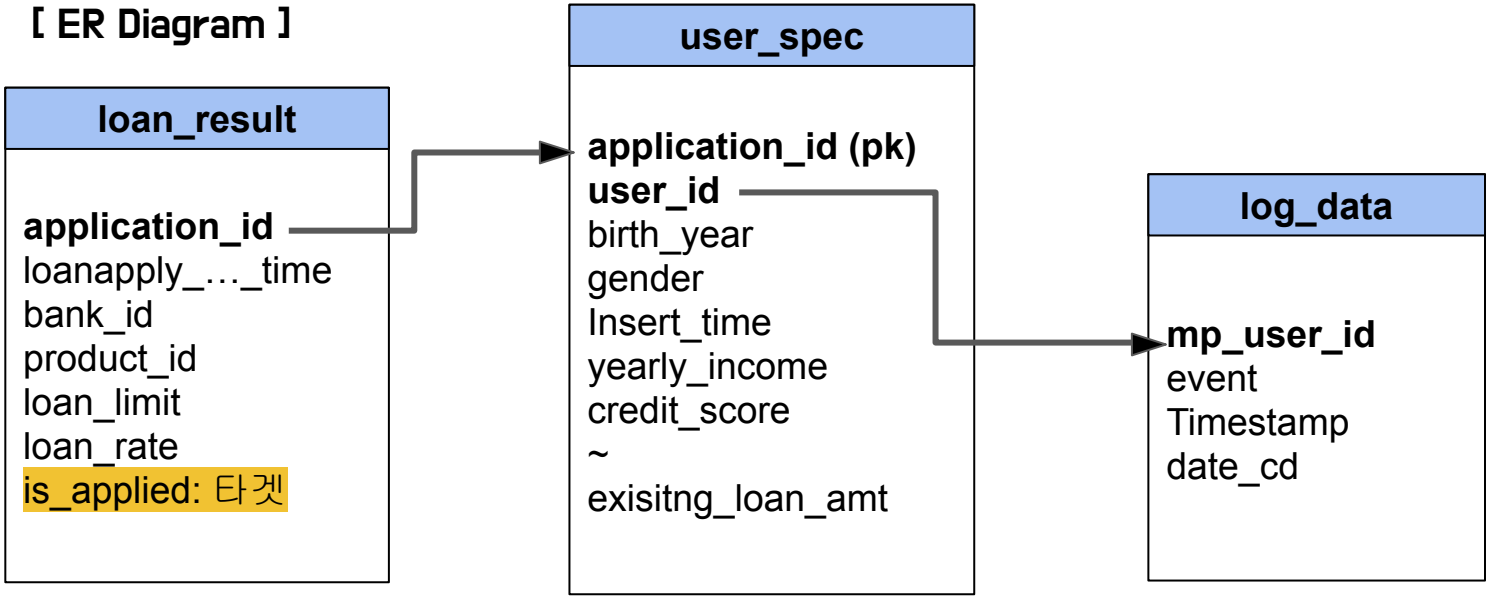
### 03. 대출별 금융사별 승인결과 데이터

➡ 한도조회 일시, 금융사 번호, 상품번호, 승인한도, 승인금리 등

# 02. 데이터 이해 및 탐색

## Q. 주어진 데이터 셋의 구조는?

[ ER Diagram ]

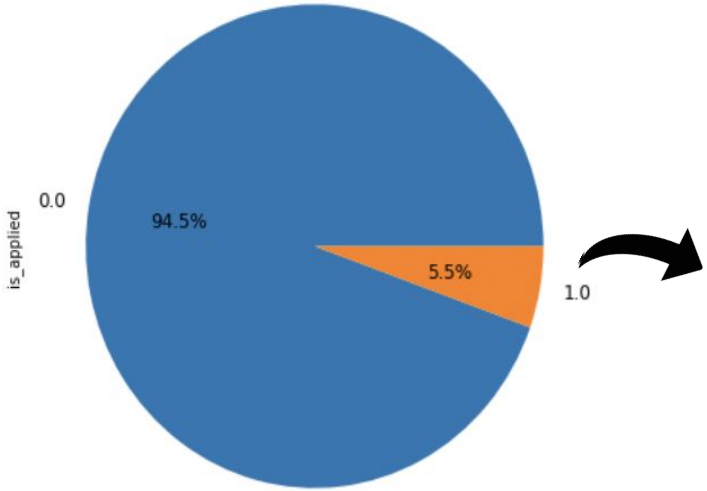


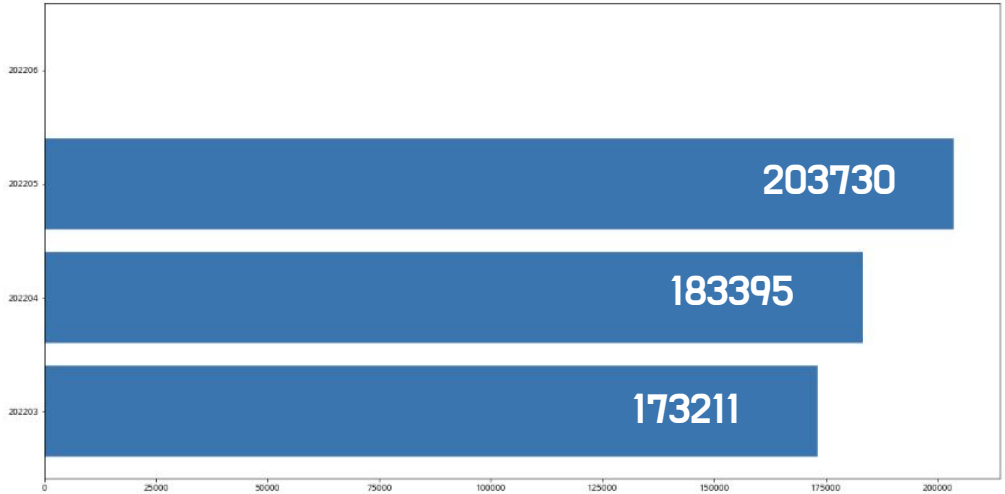

**[분석 Insight] 최종적으로 loan_result의 application_id 기준으로 합쳐 예측 모델을 구축하자.**

# 02. 데이터 이해 및 탐색



## Q. finda의 대출 현황은?




사용자가 조회한 상품 중 5.5% 정도가 최종적으로 대출 신청했으며.
최종 대출 신청 수가 점점 증가하고 있음을 확인하였다.
[분석 Insight] 데이터 불균형 문제가 존재한다고 판단하였다.

# 02. 데이터 이해 및 탐색



## Q. 앱 사용자들의 행동 패턴은 ?

[ finda 앱 탐색 ]

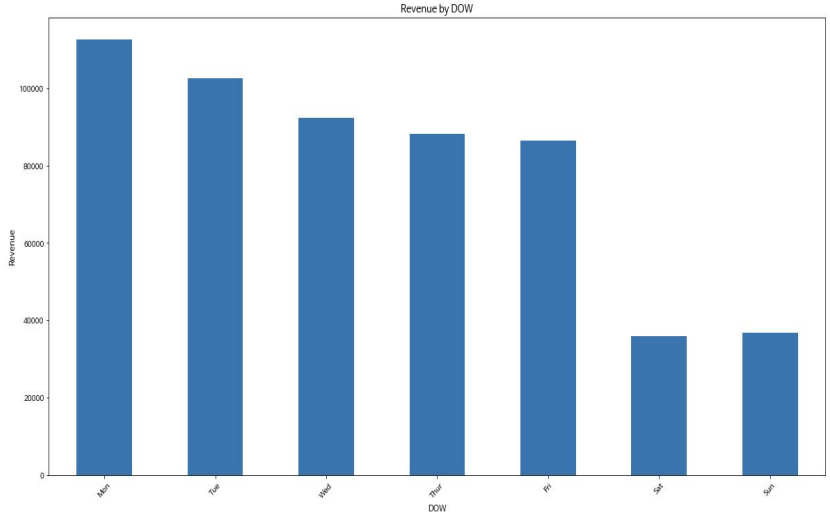

로그 데이터 행동 변수들은 개별적인 순서를 가지고 있어.
행동의 순서는 분석에 활용하기 어렵다고 판단하였다.
[분석 Insight] 시간으로 세션을 정해 앱 사용자들의 행동을 확인해보자.

# 02. 데이터 이해 및 탐색

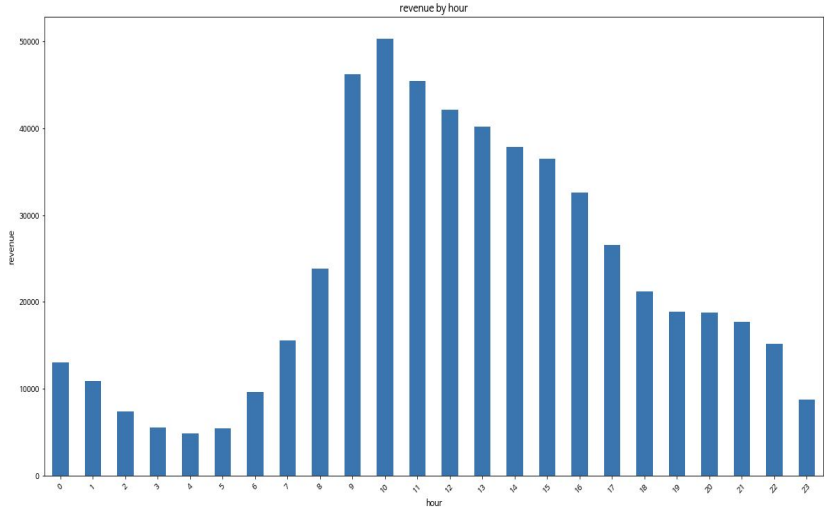


## Q. finda의 요일/시간별 대출 흐름은?

[ 요일별 대출량 ]




[ 시간별 대출량 ]

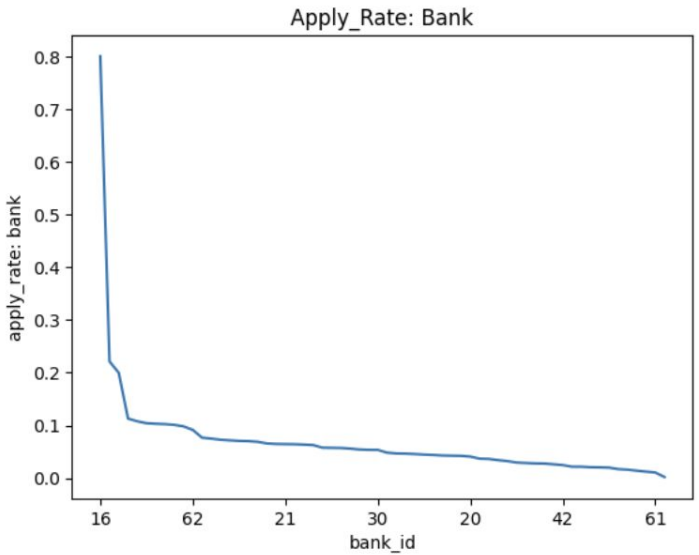


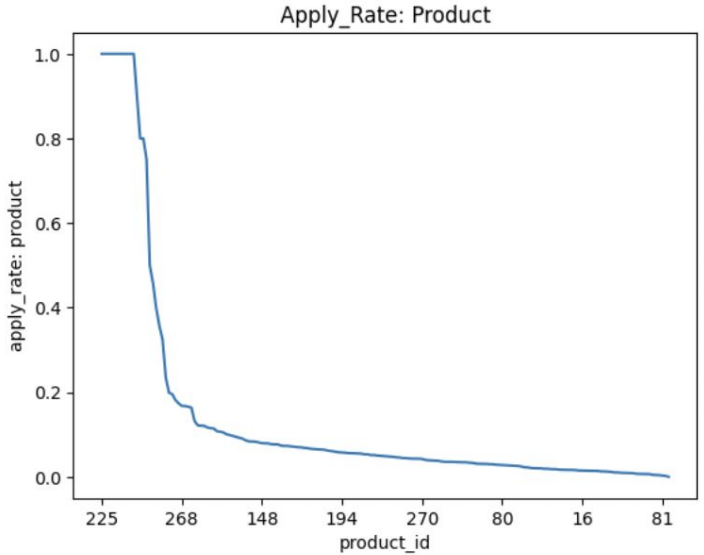

요일/시간별 대출량 그래프를 보면, 평일의 대출량이 더 많았다.
시간별 대출량 그래프를 통해서 주말과 평일의 시간별 대출량의 추이는 비슷한 것을 확인하였다.
[분석 Insight] 요일과 시간이 대출에 영향을 줄 것이라 판단하였다.

# 02. 데이터 이해 및 탐색



## Q. 은행/상품별 대출 신청률은 ?



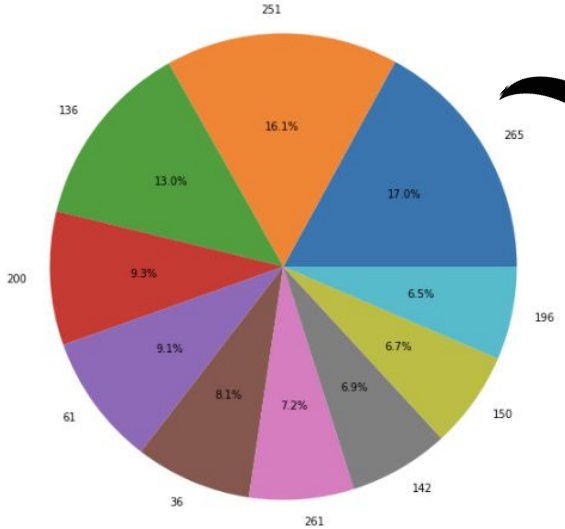

은행과 대출 상품이 최종 대출 신청 여부에 많은 영향을 미칠 것이라 예상해 보았다.
대출률이 높은 은행/상품과 낮은 은행/상품의 차이가 눈에 띄게 존재한다.
[분석 Insight] 은행/대출 상품들의 대출 신청률을 분석에 활용하자.

# 02. 데이터 이해 및 탐색



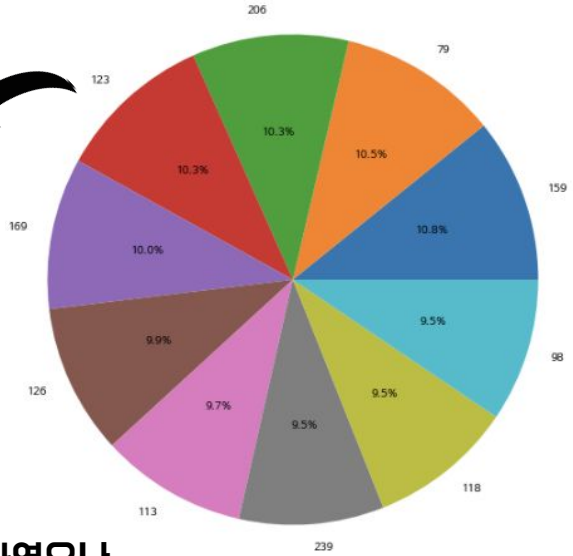
## Q. 대출량이 많은 Top10 상품은 ?

[ 상품별 대출량 Top10 ]




| Top5 상품 | | Top5 저금리 |
|---|---|---|
| 265 | | 119 |
| 251 | ≠ | 234 |
| 136 | | 250 |
| 200 | | 183 |
| 61 | | 188 |

[ 상품별 저금리 Top10 ]




금리가 낮은 상품의 대출량이 많을 것으로 예상하였으나,
금리가 낮은 Top5 상품이 대출량이 많은 Top5 상품에 없음을 확인하였다.
[분석 Insight] 은행사와 상품을 군집화하여 대출량이 많은 상품의 특징을 확인해보자.

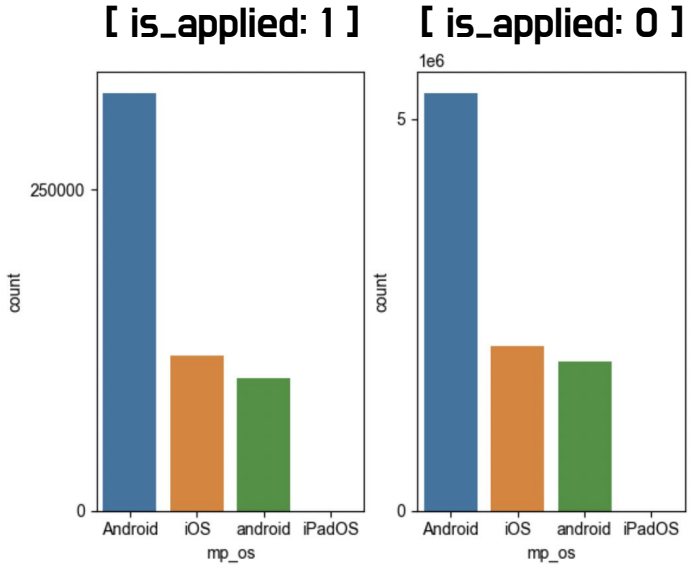
# 02. 데이터 이해 및 탐색



## Q. 앱에 접속한 OS에 따른 차이는?

| | user_id | timestamp | event | mp_os | mp_app_version |
|---|---|---|---|---|---|
| **11709372** | 1 | 2022-05-03 14:52:28 | 11-11-5-7-11 | android | 464 |
| **10428909** | 7 | 2022-05-22 16:39:49 | 11 | android | 465 |
| **9627339** | 9 | 2022-05-21 23:37:58 | 11-11-11 | android | 465 |
| **9505105** | 11 | 2022-03-24 10:53:59 | 4-11-5-11-6-1-2-8-3-5-5-11-4-11-5-2-8-3-11-5-5... | iOS | 3.6.1 |
| **1447554** | 12 | 2022-03-14 01:13:11 | 4-7-1-2-8-3-3-3-4-7-1-2-8-3-3-3-4-7-1-2-8-3-3-... | Android | 3.8.0 |
| **...** | ... | ... | ... | ... | ... |
| **2012632** | 879693 | 2022-05-13 11:29:49 | 1-2-8-3-4-1-3-4-5-11-1-3-4-1-2-8-1-2-8-1-3-3-3... | iOS | 3.10.2 |
| **14380169** | 879694 | 2022-03-31 20:07:23 | 1-2-8-3-3-3 | iOS | 3.6.1 |
| **6068504** | 879695 | 2022-05-27 12:48:32 | 1-2-8-3 | iOS | NaN |
| **7298665** | 879696 | 2022-03-14 05:35:34 | 7-5-11-11-5-11-4-7-2-8-3 | Android | 3.8.2 |
| **8248077** | 879698 | 2022-05-24 22:33:24 | 4-1-2 | iOS | NaN |

"행동(event)에 임의로 숫자를 부여해 같은 user_id끼리 timestamp 순서에 따라 합친 데이터 셋(relog) 활용"




Android, android, iOS, iPadOS 에 따라 대출 신청 여부에 차이가 있는지 살펴 보았지만,
별 다른 차이가 없음을 확인하였다.

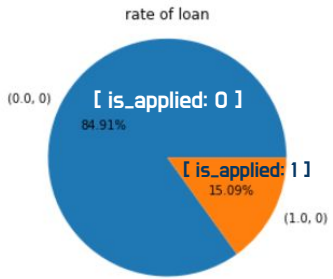
[분석 Insight] log_data의 mp_os, mp_app_version 변수는 제외하고 분석을 진행하자.

# 02. 데이터 이해 및 탐색

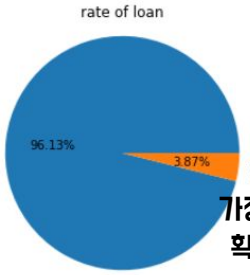
## Q. 앱 행동 패턴에 따른 대출 신청여부 차이는?

### [ log_data에서 유저 행동 군집화 결과]

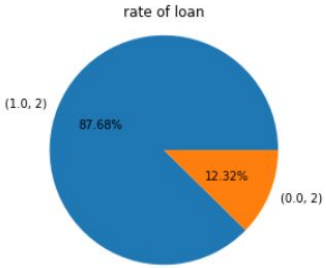
* 행동(event)의 총합, 접속일 수, 접속시간, 유저의 첫 행동(one-hot), 유저의 각 행동의 횟수를 이용하여 (총 20개의 컬럼) 군집화



군집 1

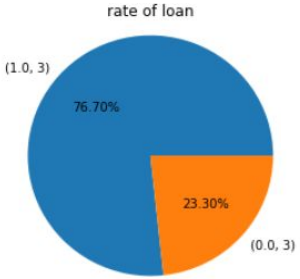


군집 2

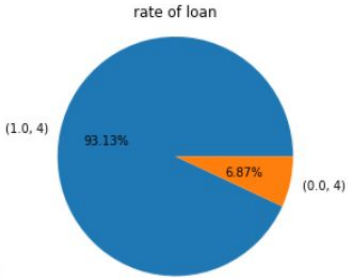


군집 3

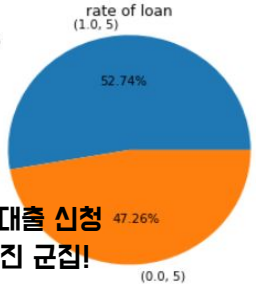
가장 낮은 대출 신청 확률을 가진 군집!



군집 4



군집 5



군집 6

가장 높은 대출 신청 확률을 가진 군집!

유저의 앱 행동 데이터들로 군집화 한 결과를 보아, 군집 별로 대출 신청 여부에 차이를 나타내는 것을 확인하였다.

[분석 Insight] log_data에서 유저들의 행동 패턴을 분석에 이용하자.

# 03. 데이터 전처리

# 02. 데이터 전처리



### [ user_spec 변수 당 결측값 개수 ]

| | |
|---|---|
| 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 |

* loan_result에 가지고 있지 않은 application_id는 제외하고 진행

### [log_data의 주요 행동 횟수]


**신용 접수가 가장 중요한 변수라고 판단하여, 나머지 변수들의 결측값을 먼저 채운 후 신용접수를 예측하여 결측값을 채우는 방향으로 진행**

# 02. 데이터 전처리



## 파생변수 1: 나이대 (user_spec: reage)

[ birth_year ]

[ age ]

```
respec['age'] = 2023 - respec['birth_year']
```

[ reage ]

```
respec.loc[respec['age'] < 20, 'reage'] = 10
respec.loc[(respec['age'] >= 20) & (respec['age'] <30), 'reage'] = 20
respec.loc[(respec['age'] >= 30) & (respec['age'] <40), 'reage'] = 30
respec.loc[(respec['age'] >= 40) & (respec['age'] <50), 'reage'] = 40
respec.loc[(respec['age'] >= 50) & (respec['age'] <60), 'reage'] = 50
respec.loc[(respec['age'] >= 60) & (respec['age'] <70), 'reage'] = 60
respec.loc[respec['age'] >= 70, 'reage'] = 70
```

* 70이상부터는 이상치라 판단하여
70이상은 한 범주로 묶어 주었음

## 파생변수 2: 근속 연수 (user_spec: career)

[ company_enter_month ]

[ career ]

```
respec['career'] = 2023 - respec['company_enter_month']
```

"날짜 형태로 되어 있는 데이터들을 직관적으로
알아보기 쉬운 수치 형태로 변환하기 위해 진행"

# 02. 데이터 전처리



## [ reage ]

yearly_income을 기준으로
범위를 나눠 결측값 채움

| | reage | yearly_income |
|---|---|---|
| 0 | 20.0 | 2.987041e+07 |
| 1 | 30.0 | 4.132548e+07 |
| 2 | 40.0 | 5.027102e+07 |
| 3 | 50.0 | 5.942931e+07 |
| 4 | 60.0 | 4.667257e+07 |
| 5 | 70.0 | 3.630668e+07 |

*나이에 따라 연소득이
가장 차이를 크게 보인다고 판단

## [ gender ]

Income_type 별 성별의 비율을 구하여 해당 비율에 맞게 결측값 채움

ex)

```
sample1 = respec[(respec['gender'].notnull()) & (respec['income_type'] == 1)]['gender'].value_counts()

per_0 = sample1[0] / (sample1[1] + sample1[0])
per_0 = (per_0*100).round(0)
```

```
# income_type == 1
rnd_num = np.random.uniform(0, 1, 1)

if rnd_num > per_0:
  respec.loc[(respec['gender'].isnull()) & (respec['income_type'] == 1), 'gender'] = 1
else:
  respec.loc[(respec['gender'].isnull()) & (respec['income_type'] == 1), 'gender'] = 0
```

*직종에 따라 성별 비율의 차이가 있다고 판단

## [ yearly_income ]

결측값 한 개는
무직이라고 판단하여 0으로 채움

## [ career ]

나이대을 기준으로
각 나이대의 평균으로 결측값 채움

# 02. 데이터 전처리



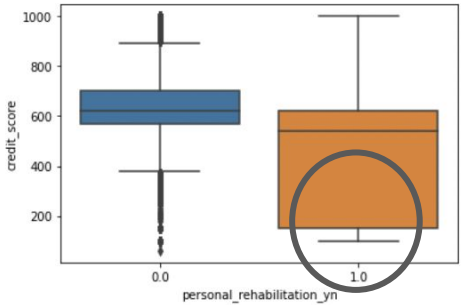
### [ existing_loan_cnt ]

기존 데이터에 기대출수가 0인 값이 존재하고 있지 않는 것을 보아 **결측값이 기대출수가 0인 유저라 판단**하여 0으로 채움

### [ existing_loan_amt ]

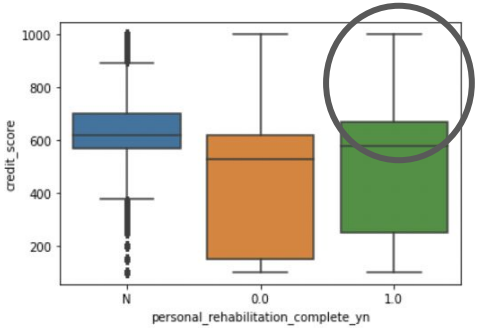
1. **기대출수가 0이면 총 금액도 0**
2. 나머지는 이상치를 제외한 **제1~3사분면에 속한 데이터들의 평균**으로 채움

### [ personal_rehabilitaion_yn ]




1. **신용점수 400이하는 1**
2. 나머지는 **0과 1의 비율**만큼 랜덤으로 채움

### [ personal_rehabilitation_complete_yn ]




1. 개인 회생 여부가 0인데 완료 여부도 0인 **데이터는 오류**라 판단하여 **N으로 변경**
2. 개인 회생 여부가 0인 유저는 N으로, 1인 유저는 **신용점수 600 이상이면 1.** 나머지는 0으로 채움

# 02. 데이터 전처리



## "Credit score 결측값 예측"

### XGBoost

- 머신러닝 기법들 중 뛰어난 성능
- 과적합 규제 기능
- 병렬 CPU 환경에서의 학습
- 내장된 교차검증

### LightGBM

- 머신러닝 기법들 중 뛰어난 성능
- **Leaf - wise**
- **GOSS**
  **(Gradient based One Side ampling**
- **EFB(Exclusive Feature Buning)**

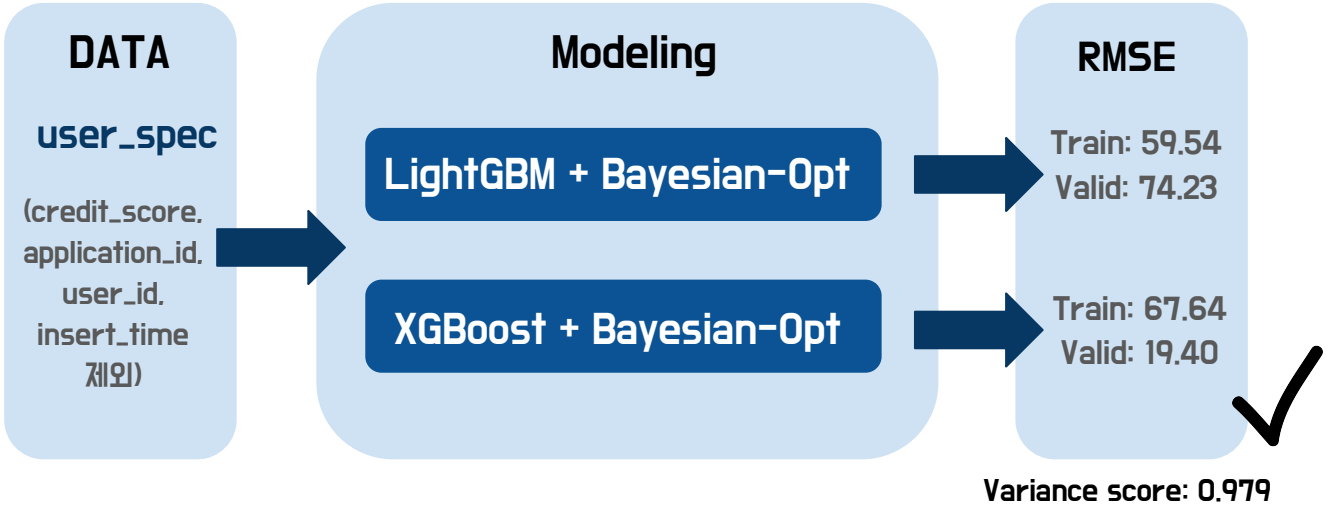
### Bayesian-Optimization

- 불필요한 하이퍼 파라미터 반복 탐색을 줄여 보다 빠르게 최적 하이퍼 파라미터를 찾을 수 있는 최적화 방법

# 02. 데이터 전처리



## "Credit score 결측값 예측"




| | KCB |
|---|---|
| 1등급 | 942~1000 |
| 2등급 | 891~941 |
| 3등급 | 832~930 |
| 4등급 | 768~831 |
| 5등급 | 698~767 |
| 6등급 | 630~697 |
| 7등급 | 530~629 |
| 8등급 | 454~529 |
| 9등급 | 335~453 |
| 10등급 | 0~334 |

XGBoost 알고리즘을 Bayesian-optimization을 통해 최적화 한 예측 모델을 사용하여
credit_score의 결측값을 처리한 후, KCB신용점수 기준에 따라 등급으로 대체

# 02. 데이터 전처리

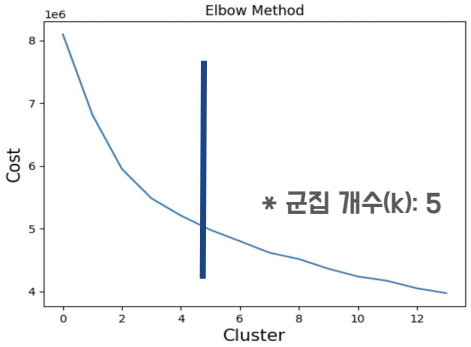


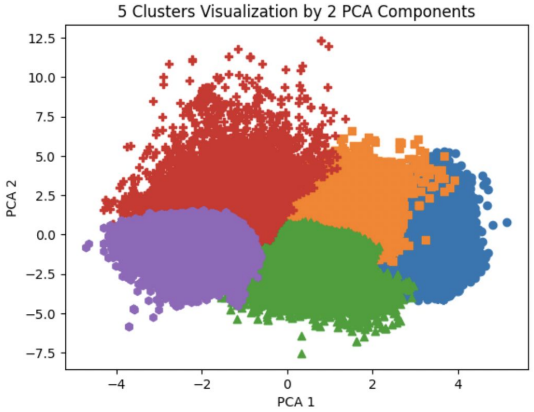
## 파생변수 3: 고객 분류 군집화 (user_spec: spec_clust)

**[ K-Prototypes ]**

**연속형 속성과 범주형 속성이 혼합된 데이터를 군집분석 할 수 있는 알고리즘.**

* K는 그룹화 할 그룹 수




1. K개의 부분집합으로 객체 분할
2. 각 군집의 중심값 계산
3. 군집의 중심값과 가장 가까운 객체를 군집에 재 할당
4. 2번으로 돌아가서 객체를 반복적으로 재 할당

* 군집의 중심값은 군집의 연속형 속성의 평균값과 범주형 속성의 최빈값으로 정한다.

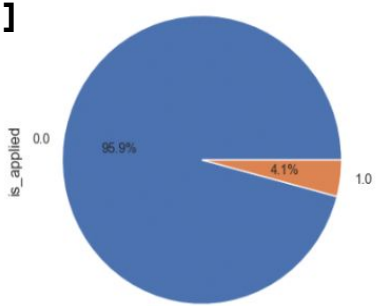
**다양한 데이터 타입의 고객 변수들로 비슷한 특징을 가진 고객들끼리 묶어 분석에 이용하고자 K-Prototypes 군집분석 진행**

# 02. 데이터 전처리

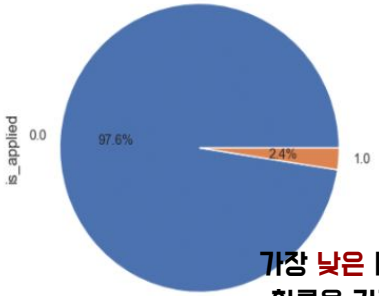


## 파생변수 3: 고객 분류 군집화 (user_spec: spec_clust)

[ 군집 결과 ]

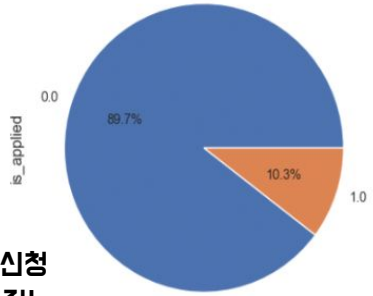



군집 1

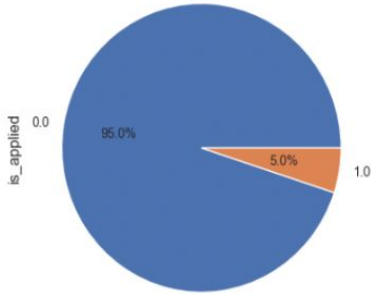



군집 2

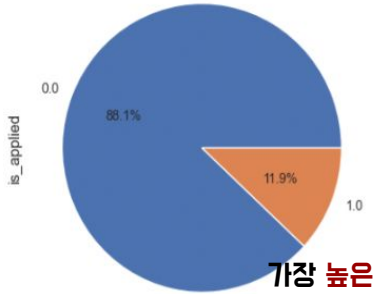
가장 낮은 대출 신청 확률을 가진 군집!




군집 3




군집 4




군집 5

가장 높은 대출 신청 확률을 가진 군집!

# 02. 데이터 전처리



## 파생변수 5: 은행 별 대출 신청률 (loan_result: bank_apply_rate)

| bank_id | bank_apply_rate |
|---|---|
| 28 | 0.001962 |
| 61 | 0.010598 |
| 46 | 0.012281 |
| 12 | 0.014004 |
| 55 | 0.016036 |
| ... | ... |
| 49 | 0.107746 |
| 60 | 0.112861 |
| 4 | 0.199582 |
| 29 | 0.221203 |
| 16 | 0.800000 |

은행 id 당 대출 신청(is_applied)를 얼마나 했는지 알기 위한 변수

ex) 은행16의 is_applied가 0: 1회, 1: 4회
-> (0 + 1 + 1 + 1 + 1) / 5 = 0.8

## 파생변수 6: 대출 상품 별 대출 신청률 (loan_result: product_apply_rate)

| product_id | product_apply_rate |
|---|---|
| 211 | 0.000000 |
| 217 | 0.001962 |
| 81 | 0.002831 |
| 40 | 0.003859 |
| 139 | 0.004386 |
| ... | ... |
| 232 | 1.000000 |
| 116 | 1.000000 |
| 76 | 1.000000 |
| 21 | 1.000000 |
| 165 | 1.000000 |

상품 id 당 대출 신청(is_applied)를 얼마나 했는지 알기 위한 변수

ex) 상품21의 is_applied가 0: 0회, 1: 2회
-> (1 + 1) / 2 = 1

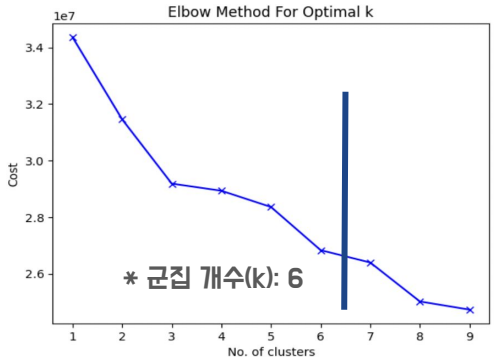
# 02. 데이터 전처리

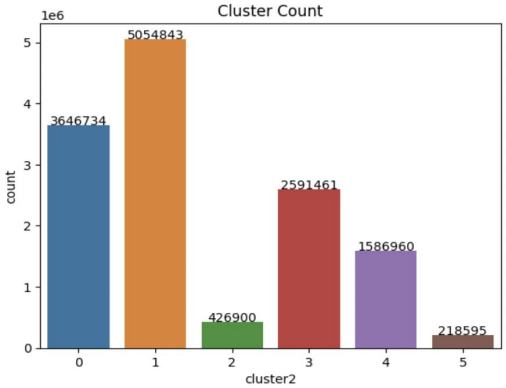


## 파생변수 4: 고객 유형과 은행사, 상품의 군집화 (loan_result: cluster2)

[ K-Modes ]

범주형 데이터를 군집분석 할 수 있는 알고리즘
K는 그룹화 할 그룹 수




1. K개의 부분집합으로 객체 분할
2. 각 군집의 중심값 계산
3. 군집의 중심값과 가장 가까운 객체를 군집에 재 할당
4. 2번으로 돌아가서 객체를 반복적으로 재 할당

* 군집의 중심값은 최빈값으로 정한다.
* 군집의 중심값과 객체간의 거리는 동일하면 0, 동일하지 않으면 1을 주는 방식 사용한다.

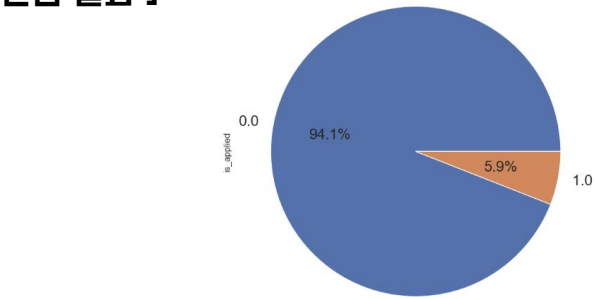
범주형 데이터 타입의 변수들로 비슷한 종류의 상품을 묶어 분석에 이용하고자
K-Modes 군집분석 진행

# 02. 데이터 전처리

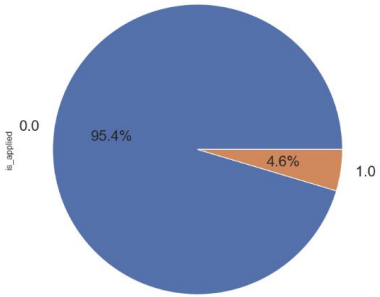


## 파생변수 4: 고객 유형과 은행사, 상품의 군집화 (loan_result: cluster2)

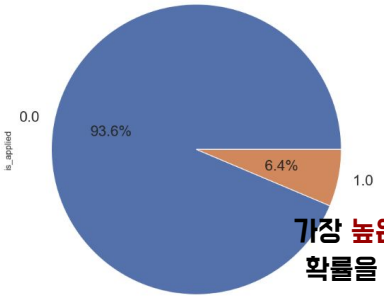
[ 군집 결과 ]




군집 1

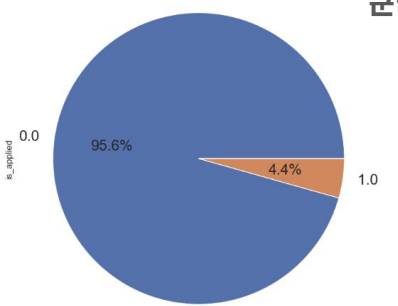



군집 2

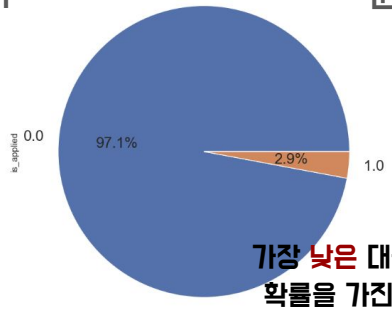



군집 3

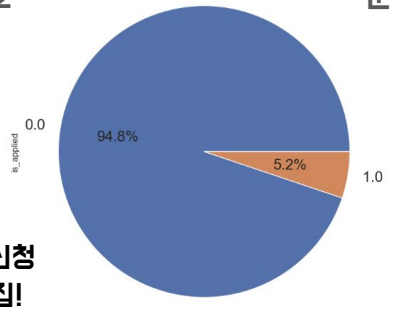
가장 높은 대출 신청 확률을 가진 군집!




군집 4




군집 5




군집 6

가장 낮은 대출 신청 확률을 가진 군집!

# 02. 데이터 전처리



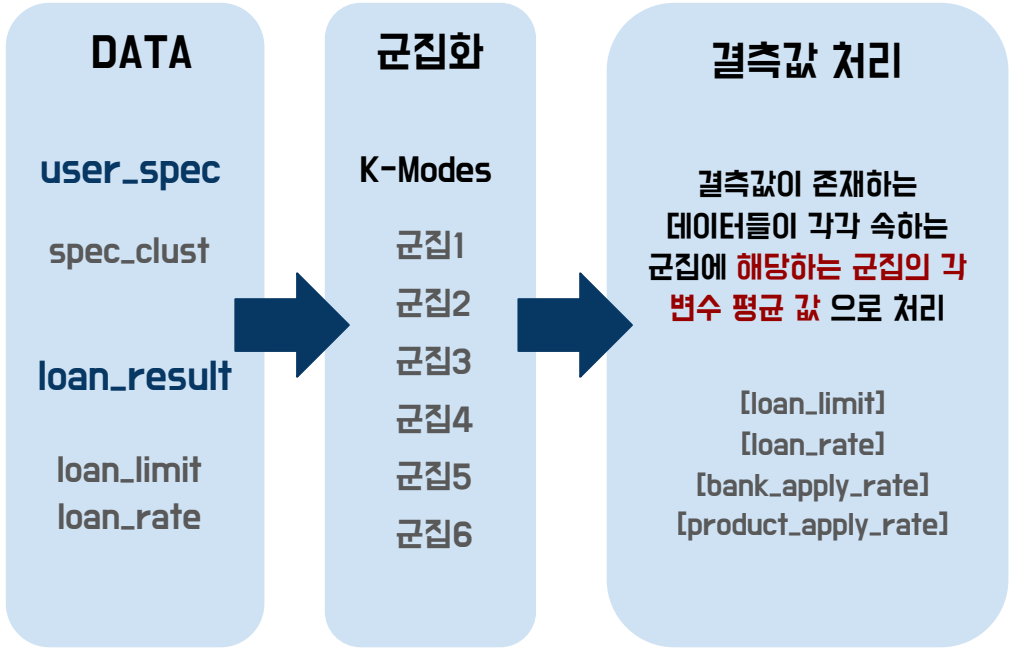
**[ loan_result 변수 당 결측값 개수 ]**

| | |
|---|---|
| application_id | 0 |
| loanapply_insert_time | 0 |
| bank_id | 0 |
| product_id | 0 |
| loan_limit | 5738 |
| loan_rate | 5738 |

* test 데이터 제외

**[ loan_result 파생변수 결측값 개수 ]**

| | |
|---|---|
| bank_apply_rate | 10254 |
| product_apply_rate | 12567 |

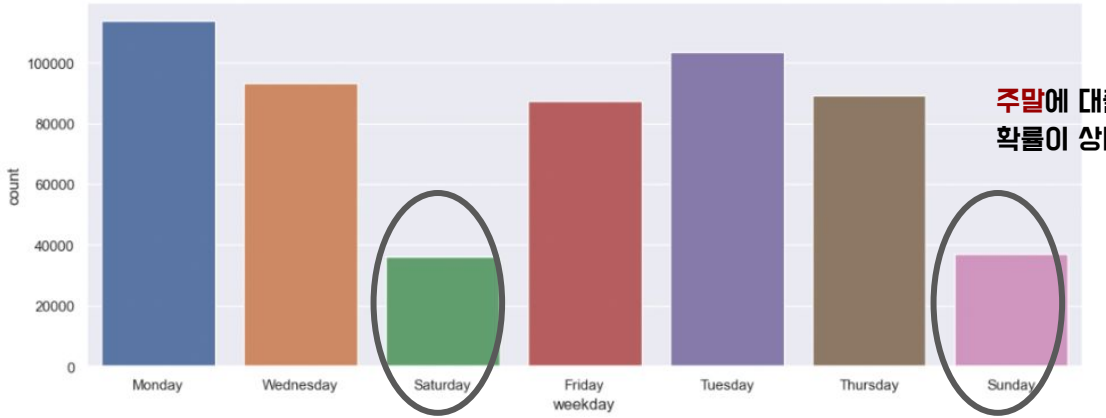


# 02. 데이터 전처리



## 파생변수 7: 대출 신청 요일 (loan_result: weekday)

[ is_applied: 1 ]




주말에 대출 신청을 할 확률이 상대적으로 낮음 !

loanapply_time

```
from datetime import datetime

loan_result['loanapply_insert_time'] = pd.to_datetime(loan_result['loanapply_insert_time'])
```

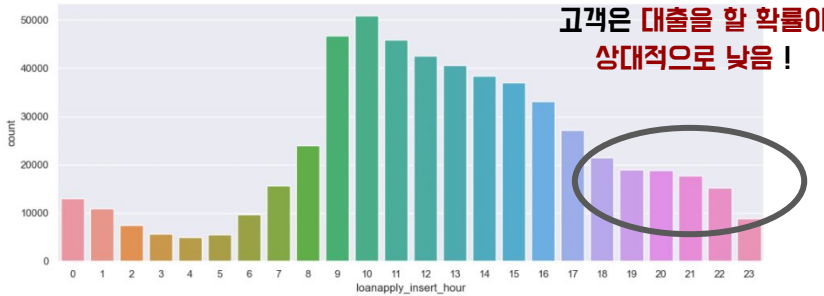
```
loan_result['weekday'] = loan_result['loanapply_insert_time'].dt.day_name()
```

# 02. 데이터 전처리



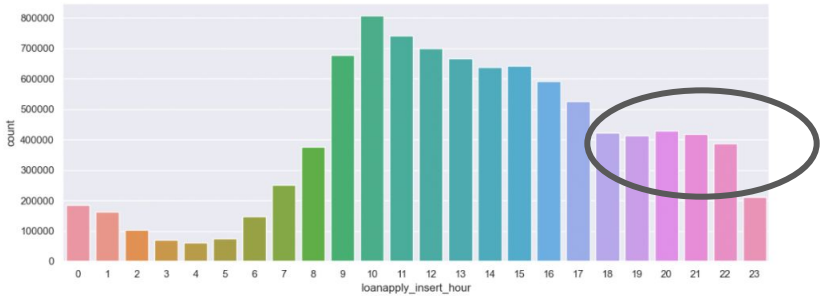
## 파생변수 8: 대출 신청 시간대 (loan_result: loanapply_insert_hour)

[ is_applied: 1 ]




밤 시간에 대출 신청을 한 고객은 대출을 할 확률이 상대적으로 낮음 !

[ is_applied: 0 ]




loanapply_time

```
from datetime import datetime

loan_result['loanapply_insert_time'] = pd.to_datetime(loan_result['loanapply_insert_time'])
```

```
loan_result['loanapply_insert_hour'] = loan_result['loanapply_insert_time'].dt.hour
```

# 02. 데이터 전처리



## 파생변수 9: 앱에서 행동한 횟수 (log_data: action_cnt)

### [ 각 유저가 하루 당 행동한 횟수 ]

```
grouped = log_data.groupby(['user_id', 'date_cd'])['user_id'].count()
grouped
```

| user_id | date_cd | |
|---|---|---|
| 1 | 2022-05-03 | 2 |
| | 2022-06-16 | 3 |
| 7 | 2022-05-22 | 1 |
| 9 | 2022-05-21 | 3 |
| 11 | 2022-03-24 | 12 |
| | | .. |
| 879693 | 2022-06-29 | 1 |
| 879694 | 2022-03-31 | 6 |
| 879695 | 2022-05-27 | 4 |
| 879696 | 2022-03-14 | 11 |
| 879698 | 2022-05-24 | 3 |

**user id** 당 접속 날짜 별 행동(event)의 횟수를 구하여 count.

### [ 유저가 했던 행동을 합해서 하나의 변수로 추출]

```
sample=pd.DataFrame(df_grouped_new['user_id_cnt'].groupby(df_grouped_new['user_id']).sum())
sample
```

| user_id | user_id_cnt |
|---|---|
| 1 | 5 |
| 7 | 1 |
| 9 | 3 |
| 11 | 43 |
| 12 | 120 |
| ... | ... |
| 879693 | 61 |
| 879694 | 6 |
| 879695 | 4 |
| 879696 | 11 |
| 879698 | 3 |

**user id** 당 총 행동의 횟수를 합(sum)한 후 하나의 컬럼으로 추출

user_id 기준으로 아이디 당 행동의 횟수를 하나의 컬럼으로 생성함

-> id의 중복을 없애고 각 유저의 컬럼들을 직관적으로 보기 위함.

# 02. 데이터 전처리



## 파생변수 10: 앱 접속한 날짜 수 (log_data: use_day_cnt)

| user_id | user_id_day_cnt |
|---|---|
| 1 | 2 |
| 7 | 1 |
| 9 | 1 |
| 11 | 4 |
| 12 | 14 |
| ... | ... |
| 879693 | 8 |
| 879694 | 1 |
| 879695 | 1 |
| 879696 | 1 |
| 879698 | 1 |

유저가 행동(event)을 한 기점.
timestamp를 기반으로 생성된 date_cd

-> 각 유저가 앱을 **총 몇 일** 접속했는지 기준으로 유저당 하나의 컬럼으로 추출

## 파생변수 11: 앱 접속한 총 시간 (log_data: timeout_sum(s))

| user_id | timeout |
|---|---|
| 1 | 0 days 00:00:08 |
| 7 | 0 days 00:00:00 |
| 9 | 0 days 00:00:00 |
| 11 | 0 days 00:00:30 |
| 12 | 0 days 00:03:20 |
| ... | ... |
| 879693 | 0 days 00:01:26 |
| 879694 | 0 days 00:00:00 |
| 879695 | 0 days 00:00:00 |
| 879696 | 0 days 00:00:30 |
| 879698 | 0 days 00:00:08 |

1. 유저의 처음 행동에서 다음 행동까지의 세션의 기준을 **10분**으로 고정
2. timestamp의 시간차이를 구하여 접속한 총시간을 추가

-> 각 유저가 앱을 **총 몇 시간** 접속했는지 기준으로 유저당 하나의 컬럼으로 추출

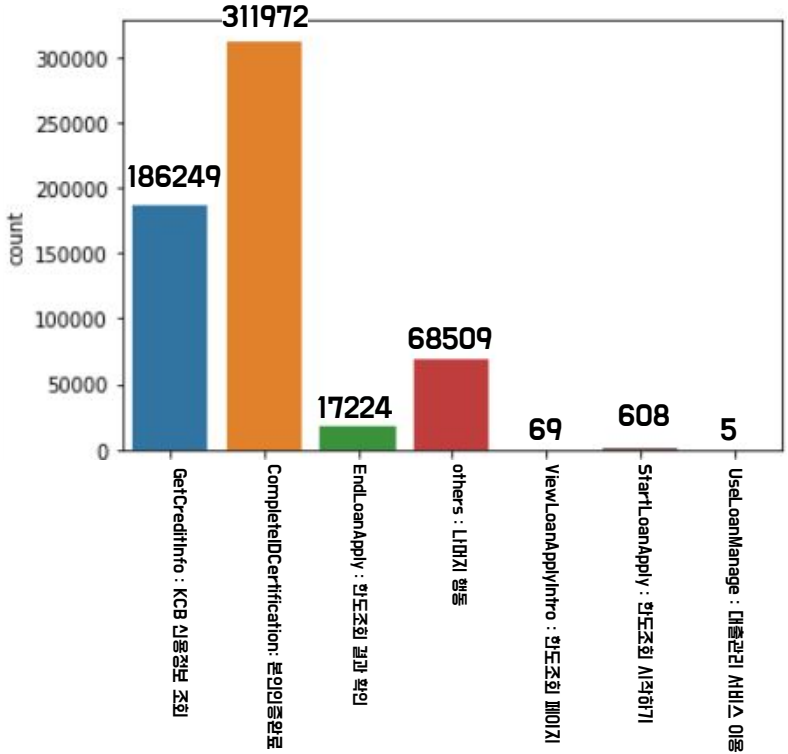
| timeout_sum | timeout_sum(s) |
|---|---|
| 0 days 00:00:08 | 8.0 |
| 0 days 00:00:00 | 0.0 |
| 0 days 00:00:00 | 0.0 |

* 초(s) 단위로 변경

# 02. 데이터 전처리



## 파생변수 12: 앱에서 처음 한 행동 (log_data: first_event)




| user_id | first_event | timestamp |
|---|---|---|
| 1 | GetCreditInfo | 2022-05-03 14:52:28 |
| 7 | GetCreditInfo | 2022-05-22 16:39:49 |
| 9 | GetCreditInfo | 2022-05-21 23:37:58 |
| 11 | CompleteIDCertification | 2022-03-24 10:56:26 |
| 12 | CompleteIDCertification | 2022-03-14 01:13:46 |
| ... | ... | ... |
| 879693 | CompleteIDCertification | 2022-05-13 11:30:06 |
| 879694 | CompleteIDCertification | 2022-03-31 20:07:42 |
| 879695 | CompleteIDCertification | 2022-05-27 12:48:51 |
| 879696 | CompleteIDCertification | 2022-03-14 05:38:16 |
| 879698 | others | 2022-05-24 22:33:24 |

1. 유저가 한 행동중에서 가장 먼저 한 event를 timestamp기준으로 변수('first_event')생성

2. 주요행동 : 한도조회, 출관리서비스, 신용정보조회 등의 서비스와 관련하여 앱을 이용한 유저들에게 주요행동 first_event컬럼 부여

3. 주요행동 이외의 event들은 앱 접속시 행동의 의미가 적다고 생각하여 others로 통일

# 02. 데이터 전처리



## "loan_result 의 application_id 기준으로 JOIN"

| application_id | bank_id | product_id | loan_limit | loan_rate | is_applied | weekday | loanapply_insert_hour | user_id | gender | ... | desired_amount |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1748340 | 7 | 191 | 42000000.0 | 13.6 | NaN | Tuesday | 13 | 430982.0 | 1.0 | ... | 25000000.0 |
| 1748340 | 25 | 169 | 24000000.0 | 17.9 | NaN | Tuesday | 13 | 430982.0 | 1.0 | ... | 25000000.0 |
| 1748340 | 2 | 7 | 24000000.0 | 18.5 | NaN | Tuesday | 13 | 430982.0 | 1.0 | ... | 25000000.0 |
| 1748340 | 4 | 268 | 29000000.0 | 10.8 | NaN | Tuesday | 13 | 430982.0 | 1.0 | ... | 25000000.0 |
| 1748340 | 11 | 118 | 5000000.0 | 16.4 | NaN | Tuesday | 13 | 430982.0 | 1.0 | ... | 25000000.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1969227 | 2 | 7 | 30000000.0 | 13.6 | 0.0 | Monday | 14 | 109899.0 | 1.0 | ... | 20000000.0 |
| 1969227 | 33 | 110 | 9000000.0 | 14.4 | 0.0 | Monday | 14 | 109899.0 | 1.0 | ... | 20000000.0 |
| 1969227 | 50 | 142 | 3000000.0 | 11.2 | 0.0 | Monday | 14 | 109899.0 | 1.0 | ... | 20000000.0 |
| 1969227 | 22 | 100 | 4000000.0 | 15.3 | 0.0 | Monday | 14 | 109899.0 | 1.0 | ... | 20000000.0 |
| 1969227 | 19 | 231 | 9000000.0 | 15.5 | 0.0 | Monday | 14 | 109899.0 | 1.0 | ... | 20000000.0 |

: Target인 is_applied 컬럼이 존재하는
laon_result셋은 application_id를 기준으로 데이터가 존재.

1. user_id와 application_id가 모두 존재하는 user_spec 데이터 셋을 loan_result와 'application_id'를 기준으로 Join

2. 1의 데이터를 application_id를 기준으로 log_data와 Join

# 02. 데이터 전처리



## "JOIN으로 인해 생긴 log_data 파생변수의 결측값 처리"

* JOIN 했을 때 생기는 약 50만개의 결측값 데이터는 주어진 데이터 셋의 한계라고 볼 수 있음

[ 변수 당 결측값 개수 ]

| | |
|---|---|
| timestamp | 488001 |
| event | 488001 |
| action_cnt | 488001 |
| use_day_cnt | 488001 |
| first_event | 488001 |
| timeout_sum(s) | 488001 |

" 앞에서 spec_clust와 bank_id, product_id로 군집화하여 만든 파생변수 'cluster2'를 기반으로 하여 결측값 처리 "

**[Cluster2]**

군집1
군집2
군집3
군집4
군집5
군집6

**결측값 처리**

결측값이 존재하는 데이터들이 각각 속하는 군집에 해당하는 군집의 각 변수 평균 값(수치형)과 최빈값(범주형) 으로 처리

# 04. 모델링

# 04. 모델링



## Q. 불균형 데이터의 문제점은?

모델은 정확도를 높이기 위해 대부분의 데이터를 다수 클래스로 예측하게 됨.
-> 100개의 데이터 중 95개가 대출 신청x. 5개가 대출 신청o 라고 했을 때. 모든 데이터를 대출신청x 라고 예측하게 되면 정확도는 95 % 이지만. 이 문제에서 중요한 대출 신청o 는 예측을 잘 못하게 되는 문제 발생

### [ scale_pos_weight 조절 ]

- 불균형 데이터를 위한 하이퍼 파라미터
- 불균형 데이터 셋의 균형 유지
- Default = 1

### [ Random Under Sampling ]

- 더 많은 쪽의 데이터를 줄이는 방법 중 한 가지
- 무작위로 정상 데이터의 일부만 선택

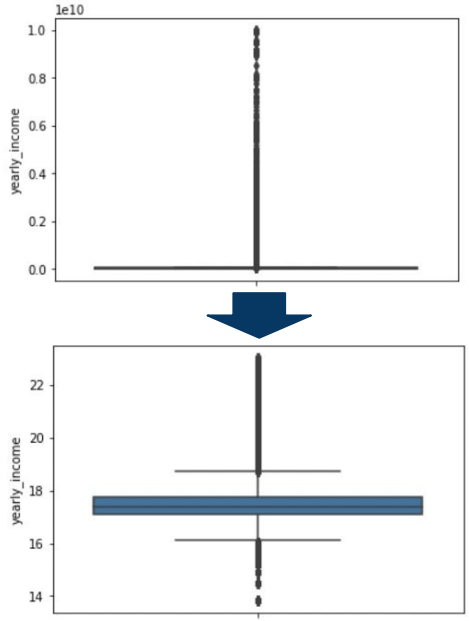
*분석 환경의 한계로 인해 데이터 손실이 있지만.
데이터의 개수를 줄여 모델링 시간을 감소시킬 수 있는 비교적 단순한 방법인 Random Under Sampling 방법 채택

**두 가지 방법을 중심으로 파라미터를 최적화 하며 모델링 진행**

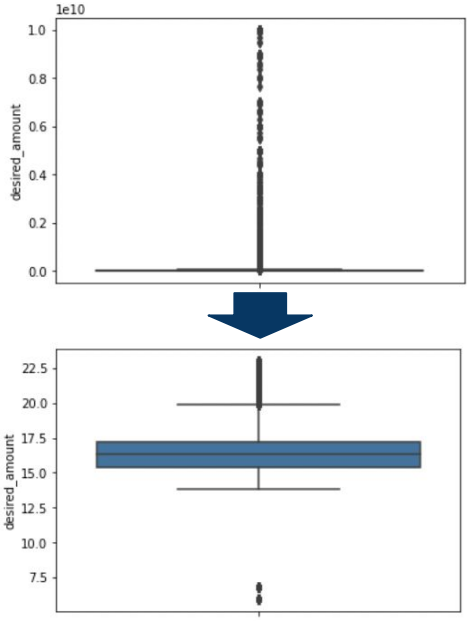
# 04. 모델링



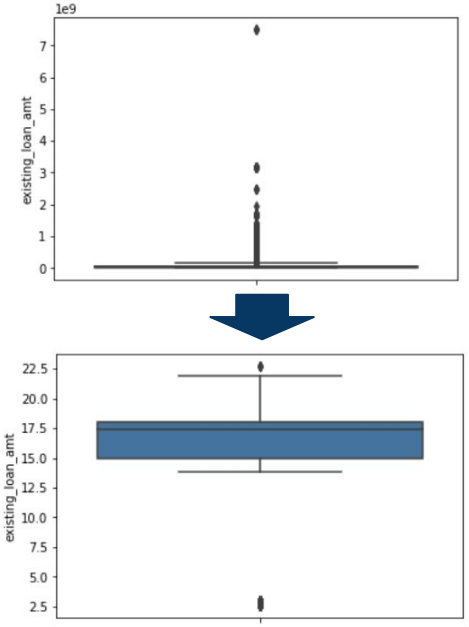
## "데이터 분포 log 변환"




[ yearly_income ]




[ desired_amount]




[ existing_loan_amt]

# 04. 모델링



## "분석 기법 정리"

### XGBoost

- 머신러닝 기법들 중 뛰어난 성능
- 과적합 규제 기능
- 병렬 CPU 환경에서의 학습
- 내장된 교차검증

### LightGBM

- 머신러닝 기법들 중 뛰어난 성능
- Leaf - wise
- GOSS

(Gradient based One Side ampling)

- EFB

(Exclusive Feature Buning)

### Bayesian-Optimization

- 불필요한 하이퍼 파라미터 반복 탐색을 줄여
- 보다 빠르게 최적 하이퍼 파라미터를 찾을 수 있는 최적화 방법

+

### CatBoost

- 다른 GBM에 비해 overfitting이 적음
- 범주형 변수에 대해 특정 인코딩 방식으로 인하여 모델의 정확도와 속도가 높음
- encoding 작업을 하지 않고도 그대로 모델의 input으로 사용 가능

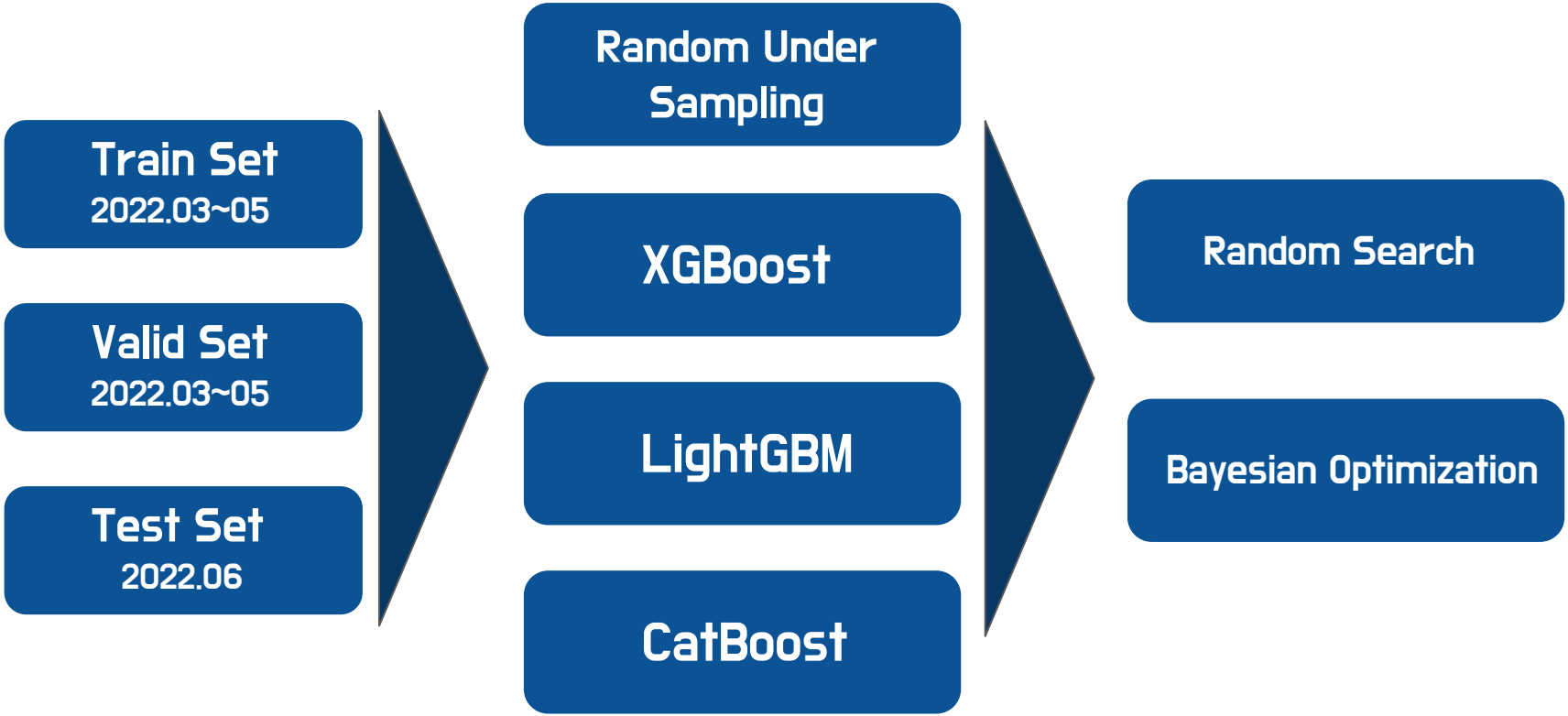
### Random Search

- Grid Search에 비해 불필요한 반복 수행 횟수를 줄여. 더 빠르게 최적해를 찾을 수 있는 최적화 방법

〈데이터 전처리: user_spec 부분에 작성한 자료〉

# 04. 모델링




Train Set
2022.03~05
Valid Set
2022.03~05
Test Set
2022.06
Random Under Sampling
XGBoost
LightGBM
CatBoost
Random Search
Bayesian Optimization

# 04. 모델링



## "XGBoost"

### [ scale_pos_weight 조절 ]

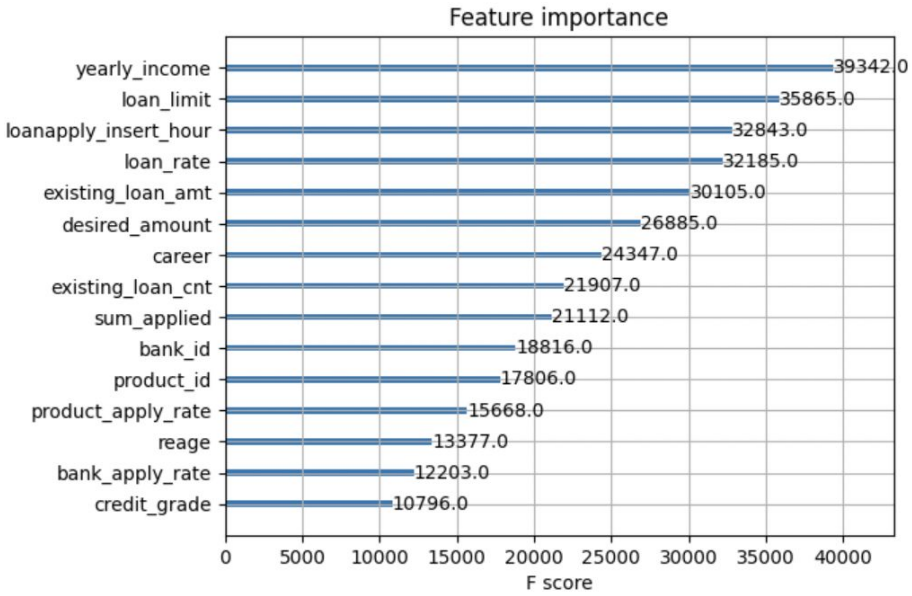
```
xgb_param = XGBClassifier(n_estimators=1000, learning_rate=0.2,
                          max_depth=14, random_state = 2022,
                          colsample_bytree = 0.75,
                          min_child_weight = 0.5, scale_pos_weight = 5)
xgb_param.fit(X_train, y_train)
xgb_param_preds = xgb_param.predict(X_val)
```

### [ Random Under Smapling ]

```
rus = RandomUnderSampler(random_state= 42,sampling_strategy= {0:600000, 1:392199 })
X_rus, y_rus = rus.fit_resample(X_train, y_train)
xgb_rus = xgb.XGBClassifier(silent=False, booster='gbtree',learning_rate=0.3,
                            colsample_bytree = 0.4, subsample = 0.8,
                            objective='binary:logistic', n_estimators=100,
                            max_depth=10, gamma=0.25,
                            seed=777)

rus_pred = xgb_rus.fit(X_rus, y_rus).predict(X_val)
print(classification_report(y_val, rus_pred))
```

### [ Feature importance top 15]



# 04. 모델링



"LightGBM"

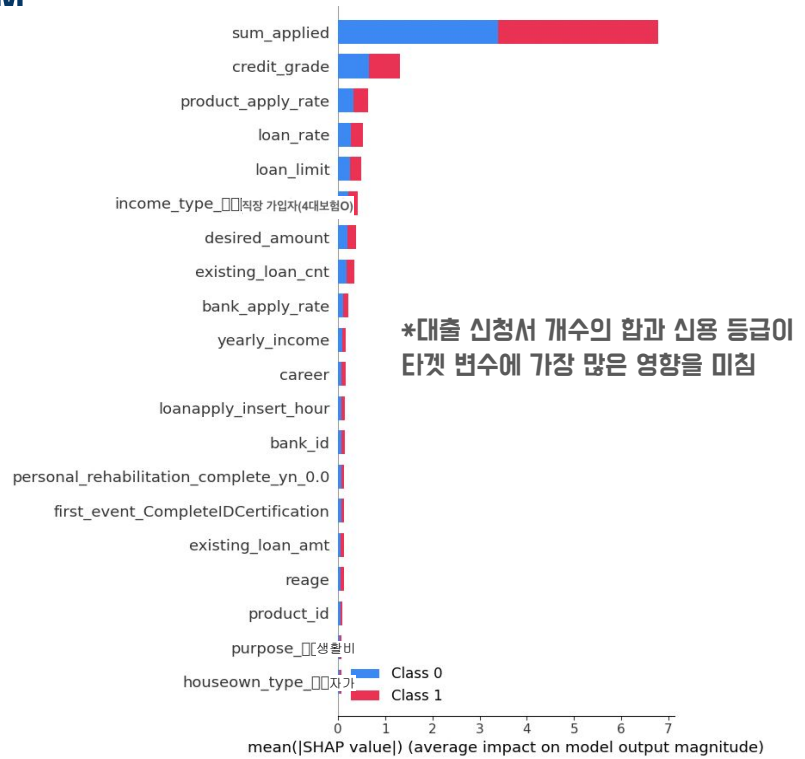
## [ scale_pos_weight 조절 ]

```
lgb_param = LGBMClassifier(scale_pos_weight = 4, n_estimators = 800,
                           objective = 'binary', metric='f1',
                           num_leaves = 72 ,
                           min_child_samples = 12, max_depth = 25,
                           learning_rate = 0.3)
lgb_param.fit(X_train, y_train)
lgb_param_preds = lgb_param.predict(X_val)
```

## [ Random Under Smapling ]

```
rus = RandomUnderSampler(random_state= 42,sampling_strategy= {0:600000, 1:392199 })
X_rus, y_rus = rus.fit_resample(X_train, y_train)
lgb_param2 = LGBMClassifier(n_estimators = 1500, num_leaves = 110,
                            objective = 'binary', metric='f1',
                            min_child_samples = 12, max_depth = 28,
                            learning_rate = 0.35)
lgb_param2.fit(X_rus, y_rus)
lgb_param_preds2 = lgb_param2.predict(X_val)
```

## [ SHAP value top 20]




*대출 신청서 개수의 합과 신용 등급이 타겟 변수에 가장 많은 영향을 미침

# 04. 모델링



## "Catboost"

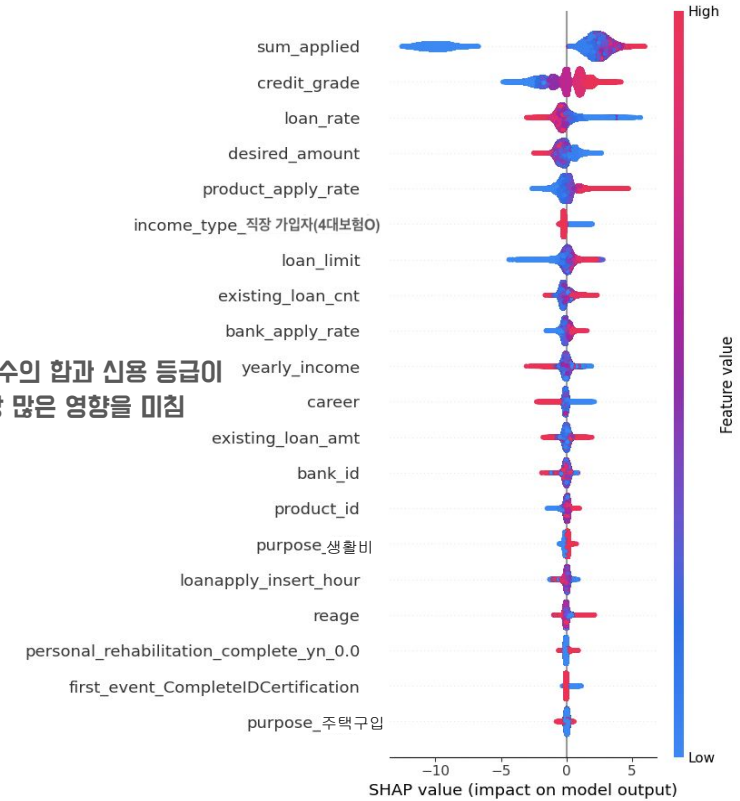
### [ scale_pos_weight 조절 ]

```
cb_rus = CatBoostClassifier(n_estimators=1000,
                            learning_rate=0.3,
                            max_depth=10,
                            scale_pos_weight = 5,
                            random_state = 42)
cb_rus.fit(X_train, y_train)
cb_rus_preds= cb_rus.predict(X_val)
```

### [ Random Under Smapling ]

```
rus = RandomUnderSampler(random_state= 42, sampling_strategy= {0:600000, 1:392199 })
X_rus, y_rus = rus.fit_resample(X_train, y_train)
cb_rus = CatBoostClassifier(n_estimators=1000, learning_rate=0.3,
                            max_depth=10, random_state = 42)
cb_rus.fit(X_rus, y_rus)
cb_rus_preds= cb_rus.predict(X_val)
```

### [ SHAP value top 20]




*대출 신청서 개수의 합과 신용 등급이 타겟 변수에 가장 많은 영향을 미침

# 05. 모델 비교 및 선정

# 04. 모델 비교 및 선정



"F1 score"

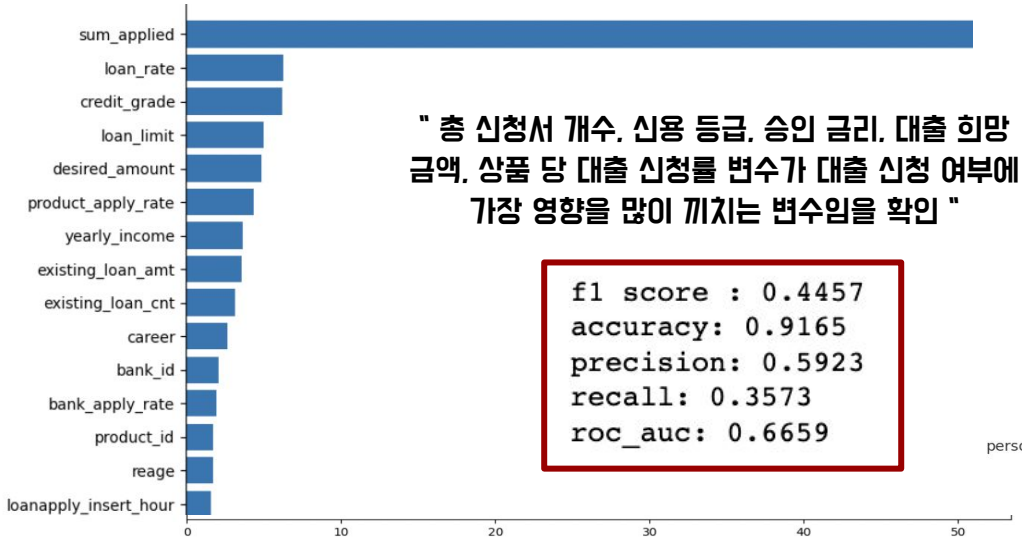
| | scale_pos_weight 조절 | Random Under Sampling |
|---|---|---|
| XGBoost | Train: 0.4426<br>Valid: 0.4115 | Train: 0.4012<br>Valid: 0.3598 |
| LightGBM | Train: 0.4592<br>Valid: 0.4263 | Train: 0.4303<br>Valid: 0.3630 |
| CatBoost Best! | Train: 0.5458<br>Valid: 0.4457 | Train: 0.4158<br>Valid: 0.3715 |

# 04. 모델 비교 및 선정



## "CatBoost (scale_pos_weight 조절) 로 최종 모델 선정"

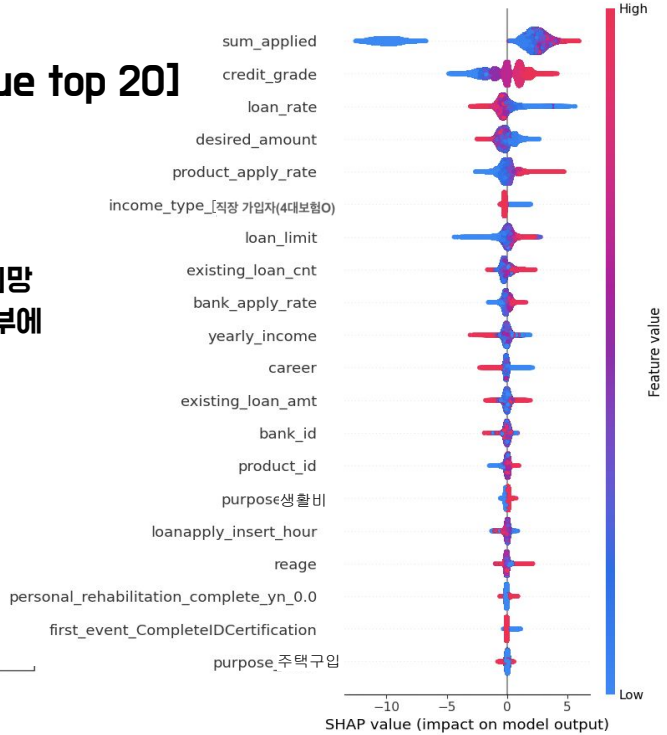
[ Feature importance top 15 ]




" 총 신청서 개수, 신용 등급, 승인 금리, 대출 희망 금액, 상품 당 대출 신청률 변수가 대출 신청 여부에 가장 영향을 많이 끼치는 변수임을 확인 "

```
f1 score : 0.4457
accuracy: 0.9165
precision: 0.5923
recall: 0.3573
roc_auc: 0.6659
```

[ SHAP value top 20]



# 06. 활용 방안 및 기대 효과

# 06. 활용 방안 및 기대 효과



## "고객에게 알맞은 상품제시 및 대출 서비스 수준 향상"

**[ 대출서비스 수준 향상 ]**

: 위 데이터 분석 과정과 결과를 활용하여, 기존 'finda'의 대출서비스의 질적인 향상을 불러올수 있다.

· 대출을 제공하는 공급자는 대출을 신청하는 소비자의 니즈를 파악할수 있다.

· 소비자들을 군집으로 분석하여 새로운 소비자가 들어왔을때, 어떠한 군집에 속하는지 파악하고, 매력적인 상품추천 및 상품조건 제시한다.

· 이는 더 많은 대출 제공자들을 도모하며, 'finda'를 통한 대출을 제공하는 기업은 정보를 통하여 높은 경쟁력을 제고한다.

· 더 많은 소비자들에게 보다 나은 서비스를 제공하여, 새로운 고객들을 유입시킴으로써 기업내 수익과 운영에 이로운 효과를 기대한다.

# 06. 활용 방안 및 기대 효과



## " 고객들의 대출 여부 사전 파악 가능 "

**고객들의 대출 여부 사전 파악**

: 대출을 신청하기 전에 앱 행동 로그 패턴 혹은 유저들의 스펙 데이터를 통해,
어떤 특성을 가진 군집에 속하는 유저인지를 사전에 파악하여 그에 맞는 마케팅 전략 수립을 가능하게 한다.

: 모델링을 통해 구한 '변수 중요도'를 통해 어떤 변수가 특히 대출 여부에 영향을 크게 미치는지 파악하여,
보다 편하고 직관적인 고객 파악을 가능하게 한다.

-> 중도 이탈을 줄이기 위한 전략

· 대출을 하지 않을 것 같은 특성을 띄는 유저에게는 유저의 이동경로에 대한 분석을 진행 한 후, funnel(앱 이용 단계)별 추가적인 분석을 진행하여 상품가입까지 가장 확률이 높은 경로로 유저을 유도하는 전략을 세우고, 유저 맞춤 마케팅을 실시한다.

-> 고객의 만족도를 높이기 위한 전략

· 대출을 할 것 같은 특성을 띄는 유저에게는 Lock-in 전략을 사용하여, 대출상품 신청이후의 신용도 관리나,
양질의 대출이후 맞춤 서비스를 제공함으로써, 대출상품 이용 이후, 유저의 만족도를 높이고,
이를 통해 일반 유저가 우수 고객이 될 수 있도록 한다.

# 06. 활용 방안 및 기대 효과



## "고객 대출 예측 분석을 통한 예산안의 효율화"

- 'finda'는 주로 앱 기반으로 사용되는 플랫폼으로 모바일 광고를 많이 사용하는데, 위의 분석을 바탕으로 광고비용의 효율화.
- 앱 중도 이탈 사용자나 새로운 고객을 전략적으로 공략하여, 광고를 개편하는 작업을 통해 'finda'내 대출 신청 고객수를 늘림.
- '무료 신용보험 서비스'나 '대출 관리 서비스'의 질적인 발전을 통하여 사용자의 대출 위험성을 낮출수 있음.

**최근 가계부채의 증가로 인한 민간소비 약화 문제를 완화시키고,**
**대한민국의 성장 잠재력을 키울수 있는 사회적인 문제의 해결을 도모함**

2022빅콘테스트 X finda

# 감사합니다.

Team. 우 걱정마세요
김혜현 임성수 황성아

# ⟨Reference⟩


- Tianqi Chen, & Carlos Guestrin. (2016). *XGBoost: A Scalable Tree Boosting System*.
- Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, …, Tie-Yan Liu. (2017). *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*.
- 김정화. (2018). *k-prototypes 군집분석에 관한 고찰*.