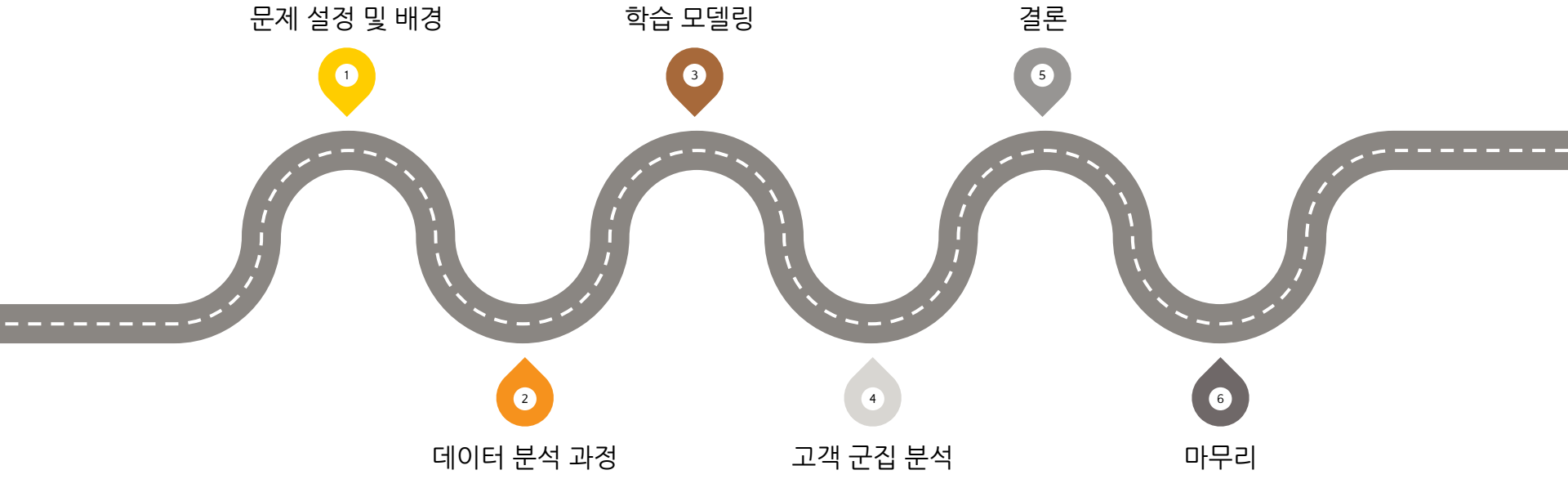
2022 빅콘테스트

대출 신청 예측 분석

Team: 보리차

황예원, 윤지환

로드맵



문제 설정 및 배경

+ 3.00 %

2022년 10월 한국 중앙은행 기준금리

Kospi 2162.97

2022년 10월 12일 기준

2022년 현재는 경제 위기 상황이다.

문제 설정 및 배경

앱 사용성 데이터를 통한 대출 신청 예측 분석

- 가명화된 데이터를 기반으로 고객의 대출상품 신청여부 예측
  (2022년 3~5월 데이터제공 / 2022년 6월 예측)

- 예측 모델을 활용한 탐색적 데이터 분석 수행

- 대출 신청,미신청 고객을 분류해 고객의 특성 분석 결과 도출

문제 설정 및 배경

금리 상승 경제 하락

대출 신청 감소

예비 대출 신청 고객을 위한 서비스 필요

데이터 분석 과정

데이터 확인 및 병합

- user_spec.shape = (1394216, 17)
- loan_result.shape = (13527363, 7)

‘application_id’ 기준 [user_spec, loan_result] inner merge

```
# loan_result, user_spec 'application id' 기준 inner merge
merge_user_loan = pd.merge(loan_result, user_spec, how='inner', on='application_id')
merge_user_loan.head()
```

	application_id	loanapply_insert_time	bank_id	product_id	loan_limit	loan_rate	is_applied	user_id	birth_year	gender	..
0	1748340	2022-06-07 13:05:41	7	191	42000000.0	13.6	NaN	430982	1996.0	1.0	
1	1748340	2022-06-07 13:05:41	25	169	24000000.0	17.9	NaN	430982	1996.0	1.0	
2	1748340	2022-06-07 13:05:41	2	7	24000000.0	18.5	NaN	430982	1996.0	1.0	
3	1748340	2022-06-07 13:05:41	4	268	29000000.0	10.8	NaN	430982	1996.0	1.0	
4	1748340	2022-06-07 13:05:41	11	118	5000000.0	16.4	NaN	430982	1996.0	1.0	

5 rows × 23 columns

merge_user_loan.shape = (13527250, 23)

데이터 분석 과정

Train, target 데이터 분리

Column ‘is_applied’ 가 NaN인 row를 target으로 분리

```
merge_train = merge_user_loan[merge_user_loan['is_applied'].notnull()]
merge_target = merge_user_loan[merge_user_loan['is_applied'].isnull()]
print(f'total length : {len(merge_user_loan)}, train length : {len(merge_train)}, target length : {len(merge_target)}')
```

```
total length : 13527250, train length : 10270011, target length : 3257239
```

데이터 분석 과정

변수 확인

변수명과 자료형을 확인

- **고객 개인 정보**
  (생일, 성별, 수입, 신용등급, 직업 등)
- **고객 대출 정보**
  (기존 대출 존재 여부, 개인 회생 여부, 대출 목적 등)

```
merge_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10270011 entries, 13301 to 13519577
Data columns (total 23 columns):
 #   Column                                Dtype
---  ------                                -----
 0   application_id                        int64
 1   loanapply_insert_time                 object
 2   bank_id                               int64
 3   product_id                            int64
 4   loan_limit                            float64
 5   loan_rate                             float64
 6   is_applied                            float64
 7   user_id                               int64
 8   birth_year                            float64
 9   gender                                float64
 10  insert_time                           object
 11  credit_score                          float64
 12  yearly_income                         float64
 13  income_type                           object
 14  company_enter_month                   float64
 15  employment_type                       object
 16  houseown_type                         object
 17  desired_amount                        float64
 18  purpose                               object
 19  personal_rehabilitation_yn            float64
 20  personal_rehabilitation_complete_yn   float64
 21  existing_loan_cnt                     float64
 22  existing_loan_amt                     float64
dtypes: float64(13), int64(4), object(6)
memory usage: 1.8+ GB
```

데이터 분석 과정

결측치 확인

```
merge_train.isnull().sum()
application_id                      0
loanapply_insert_time               0
bank_id                             0
product_id                          0
loan_limit                       5625
loan_rate                        5625
is_applied                          0
user_id                             0
birth_year                      91626
gender                          91626
insert_time                         0
credit_score                  1243812
yearly_income                       0
income_type                         0
company_enter_month            303568
employment_type                     0
houseown_type                       0
desired_amount                      0
purpose                             0
personal_rehabilitation_yn    5873229
personal_rehabilitation_complete_yn    9232232
existing_loan_cnt              2143811
existing_loan_amt              3044140
dtype: int64
```

```
merge_target.isnull().sum()
application_id                      0
loanapply_insert_time               0
bank_id                             0
product_id                          0
loan_limit                       1757
loan_rate                        1757
is_applied                    3257239
user_id                             0
birth_year                      36470
gender                          36470
insert_time                         0
credit_score                  265464
yearly_income                       6
income_type                         0
company_enter_month             96769
employment_type                     0
houseown_type                       0
desired_amount                      0
purpose                             0
personal_rehabilitation_yn      15472
personal_rehabilitation_complete_yn    2561745
existing_loan_cnt              541898
existing_loan_amt              846023
dtype: int64
```

train에서만 결측치 존재하는 컬럼	target에서도 결측치 존재하는 컬럼
user_id	loan_limit
insert_time	loan_rate
income_type	birth_year
employment_type	gender
houseown_type	credit_score
desired_amount	yearly_income
purpose	company_enter-month
	personal_rehabilitation_yn
	personal_rehabilitation_complete_yn
	existing_loan_cnt
	existing_loan_amt

데이터 분석 과정

purpose 컬럼에서 영문/한글명 처리

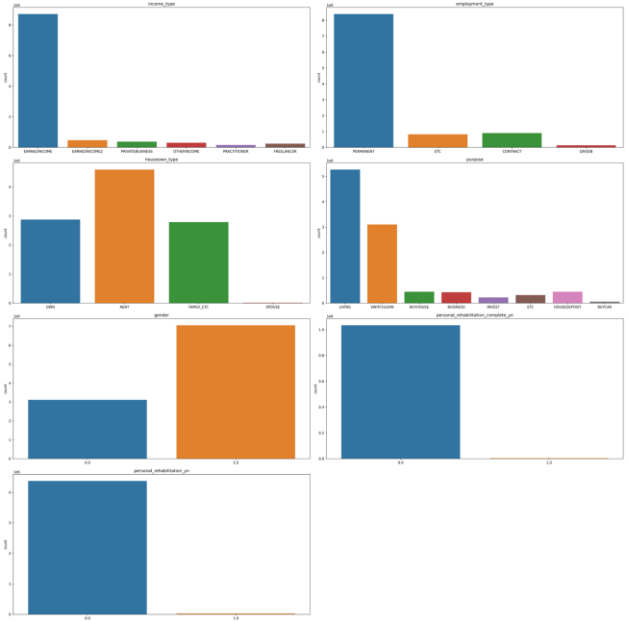
```
### purpose value counts ###
생활비              5156402
대환대출            3066047
주택구입             433081
전월세보증금          426639
사업자금             418196
기타                 303879
투자                 219684
LIVING               107532
자동차구입            55759
SWITCHLOAN            33615
HOUSEDEPOSIT          15912
BUYHOUSE              10989
BUSINESS               8513
ETC                    7374
INVEST                 3634
BUYCAR                 2755
Name: purpose, dtype: int64
```

```python
# korean value to english
def kor_to_eng(df):

    df2 = df.copy()
    df2.replace({'employment_type':'정규직'}, 'PERMANENT', inplace=True)
    df2.replace({'employment_type':'계약직'}, 'CONTRACT', inplace=True)
    df2.replace({'employment_type':'일용직'}, 'DAYJOB', inplace=True)
    df2.replace({'employment_type':'기타'}, 'ETC', inplace=True)

    df2.replace({'houseown_type':'전월세'}, 'RENT', inplace=True)
    df2.replace({'houseown_type':'자가'}, 'OWN', inplace=True)
    df2.replace({'houseown_type':'기타가족소유'}, 'FAMILY_ETC', inplace=True)
    df2.replace({'houseown_type':'배우자'}, 'SPOUSE', inplace=True)

    df2.replace({'purpose':'생활비'}, 'LIVING', inplace=True)
    df2.replace({'purpose':'대환대출'}, 'SWITCHLOAN', inplace=True)
    df2.replace({'purpose':'주택구입'}, 'BUYHOUSE', inplace=True)
    df2.replace({'purpose':'전월세보증금'}, 'HOUSEDEPOSIT', inplace=True)
    df2.replace({'purpose':'사업자금'}, 'BUSINESS', inplace=True)
    df2.replace({'purpose':'기타'}, 'ETC', inplace=True)
    df2.replace({'purpose':'투자'}, 'INVEST', inplace=True)
    df2.replace({'purpose':'자동차구입'}, 'BUYCAR', inplace=True)

    return df2
```

```
### purpose value counts ###
LIVING              5263934
SWITCHLOAN          3099662
BUYHOUSE             444070
HOUSEDEPOSIT         442551
BUSINESS             426709
ETC                  311253
INVEST               223318
BUYCAR                58514
Name: purpose, dtype: int64
```

데이터 EDA

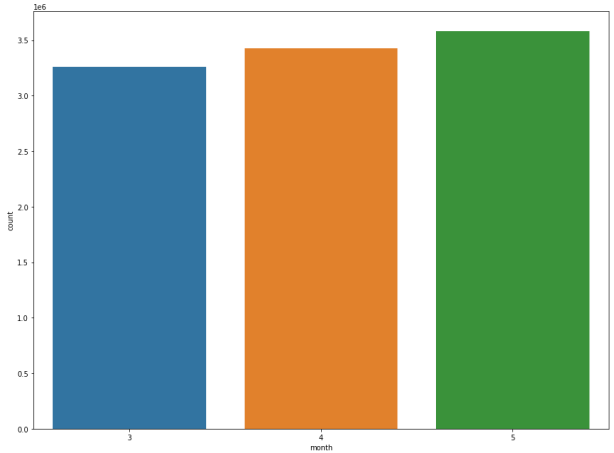
전체 데이터 카테고리형 변수 분포 시각화



데이터 EDA

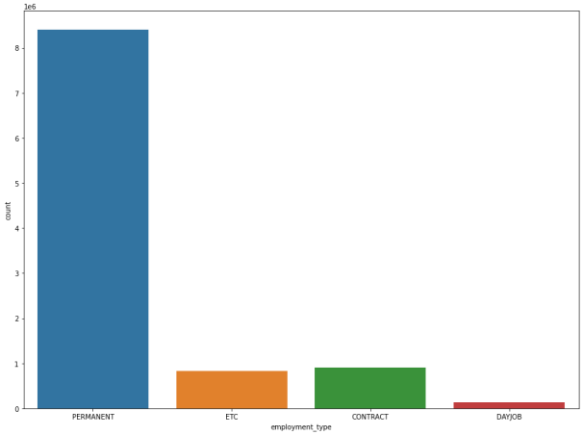
월별 대출 신청 수

3월 ~ 5월까지 꾸준히 증가



4대 보험, 정규직 시각화

예상대로 직장 4대 보험 가입자, 정규직의 비율이 압도적으로 높음

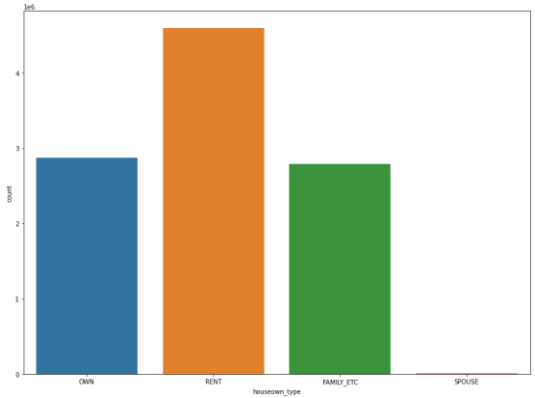


데이터 EDA

주택 보유 여부 시각화

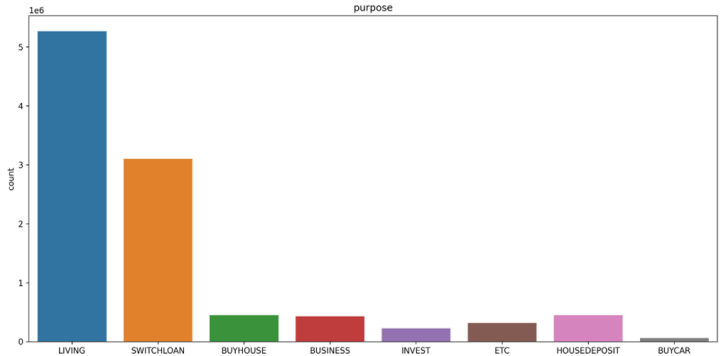
소유보다 대여를 하는 사람의 비율이 많음

배우자는 가족에 포함되므로 spouse 변수 family_etc에 포함 고려



대출 목적 시각화

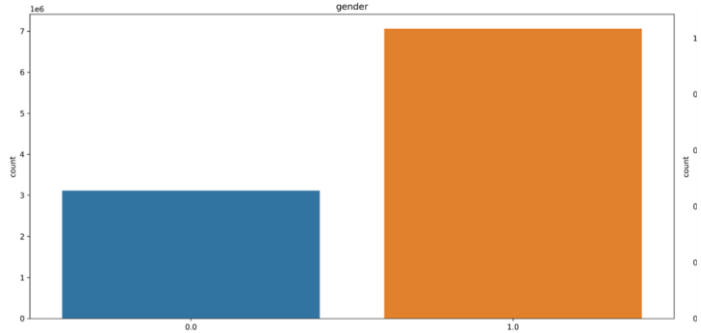
생활비 목적이 가장 많았으며, 대환 대출이 그 뒤를 이음



데이터 EDA

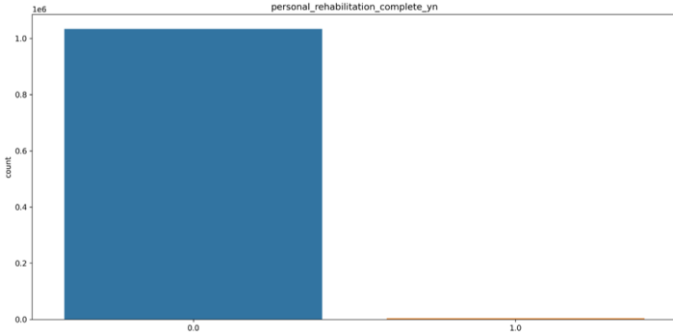
성별 시각화

여성이 더 많음



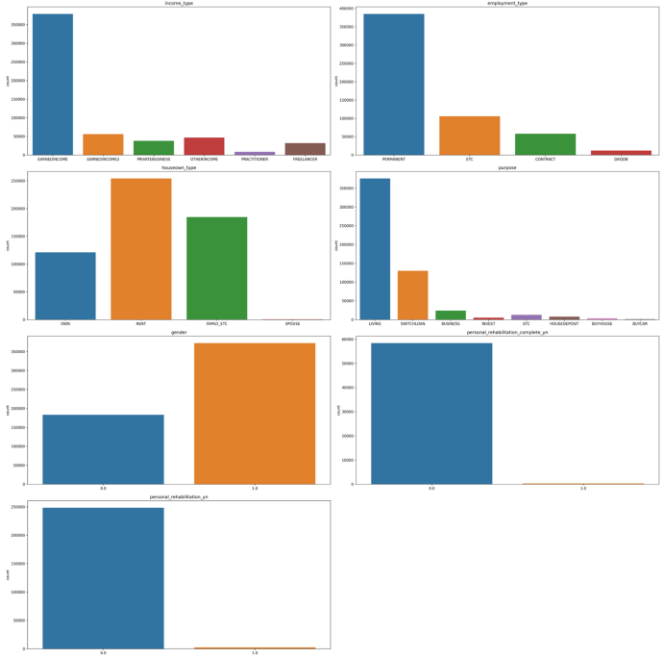
개인 회생자

personal_rehabilitation_complete_yn, personal_rehabilitation_yn은 모두 개인회생자에 관련된 변수 -> 하나로 합치기



데이터 EDA

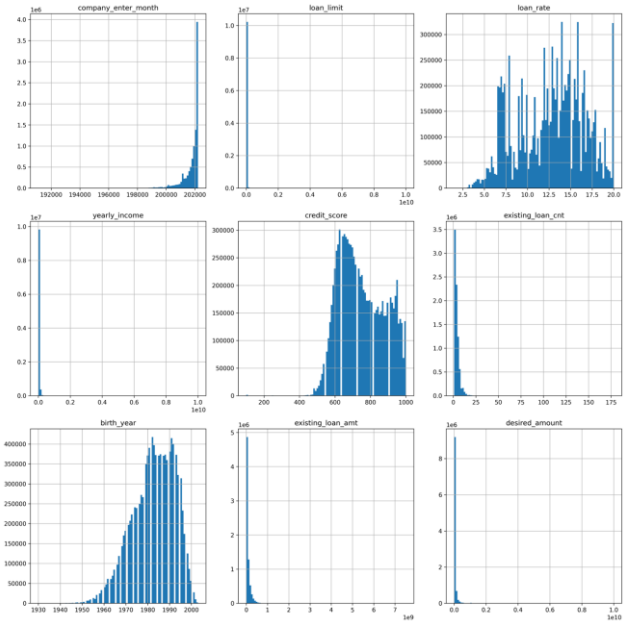
대출을 신청한 데이터 (is_applied = 1) 카테고리형 변수 분포 시각화



- 전체적인 분포 차이는 그렇게 크지 않음
- 정규직보다 다른 근로 형태의 비중이 조금 더 높아짐
- 생활비 목적의 대출이 더 많아짐

데이터 EDA

전체 데이터 수치형 변수 분포 시각화



- loan_limit, yearly_income, desired_amount : 이상치 존재 가능성
- credit_score : 특정 데이터가 매우 낮은 수치 보임
- existing_loan_cnt, existing_loan_amt : 기대출수/금액이라는 비슷한 특성

데이터 EDA

월별 특성 파악

‘Insert_time’ 피처를 통해 월별 (3,4,5) 로 데이터를 나누어 분석

```
# 월별 데이터프레임
merge_train_3 = merge_train.loc[merge_train['insert_month'] == 3]
merge_train_4 = merge_train.loc[merge_train['insert_month'] == 4]
merge_train_5 = merge_train.loc[merge_train['insert_month'] == 5]
```

```
len(merge_train_3), len(merge_train_4), len(merge_train_5) # 극단적 차이 X. 증가 추세
```

```
(3262692, 3427273, 3579958)
```

전체 신청 비율 -> 이용자수와 함께 증가

```
merge_train_3['is_applied'].sum(), merge_train_4['is_applied'].sum(), merge_train_5['is_applied'].sum() # 극단적 차이 X. 증가 추세
```

```
(173145.0, 183419.0, 203770.0)
```

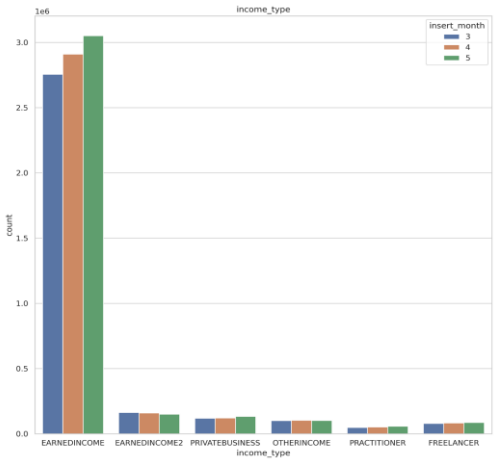
```
merge_train_3['is_applied'].sum() / len(merge_train_3), merge_train_4['is_applied'].sum() / len(merge_train_4), merge_train_5['is_applied'].sum() / len(merge_train_5) # 극단적 차이 X. 증가 추세
```

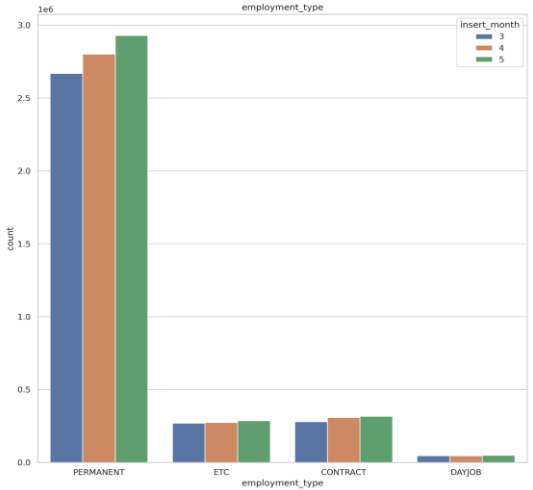
```
(0.053068141277202996, 0.053517475847415716, 0.056919662185980956)
```

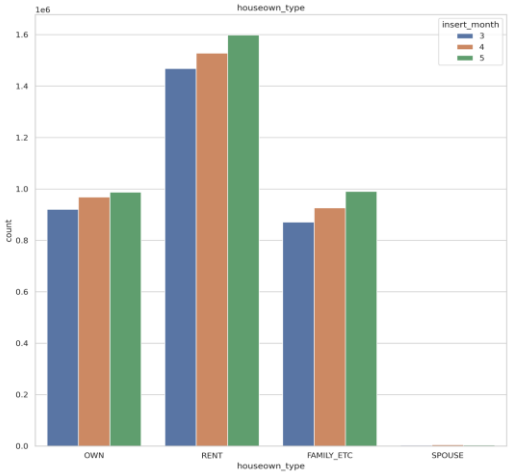
데이터 EDA

월별 특성 파악

월별 변수 평균치 비교 -> 유의미한 차이 존재 X



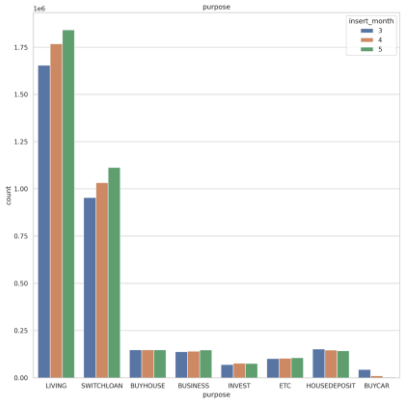


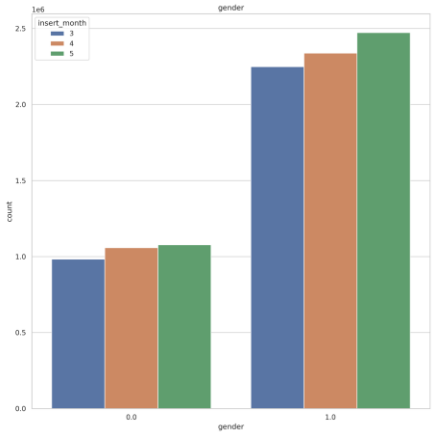


데이터 EDA

월별 특성 파악

월별 변수 평균치 비교 -> 유의미한 차이 존재 X





데이터 EDA

시간대별 특성 파악

하루의 시간을 4구간으로 나누어 (0-6시, 6-12시, 12-18시, 18-24시) 전체 이용자수와 이용자수 대비 신청자수 비율 확인

```
[58] # 시간대별 데이터프레임
     merge_train_0_6 = merge_train.loc[(merge_train['insert_hour'] >= 0) & (merge_train['insert_hour'] < 6)]
     merge_train_6_12 = merge_train.loc[(merge_train['insert_hour'] >= 6) & (merge_train['insert_hour'] < 12)]
     merge_train_12_18 = merge_train.loc[(merge_train['insert_hour'] >= 12) & (merge_train['insert_hour'] < 18)]
     merge_train_18_24 = merge_train.loc[(merge_train['insert_hour'] >= 18) & (merge_train['insert_hour'] < 24)]

[59] len(merge_train_0_6), len(merge_train_6_12), len(merge_train_12_18), len(merge_train_18_24)

     (708010, 3192819, 3984847, 2384247)
```

데이터 EDA

시간대별 특성 파악

```
merge_train_0_6['is_applied'].sum(), merge_train_6_12['is_applied'].sum(), merge_train_12_18['is_applied'].sum(), merge_train_18_24['is_applied'].sum()
```

```
(47582.0, 192677.0, 218655.0, 101420.0)
```

```
merge_train_0_6['is_applied'].sum() / len(merge_train_0_6), merge_train_6_12['is_applied'].sum() / len(merge_train_6_12), merge_train_12_18['is_applied
```

```
(0.06720526546235223,
 0.0603469849058152 1,
 0.0548716174046331,
 0.0425375391056379 6)
```

시간대별로 유의미한 이용자수 차이와 신청자 비율의 차이를 확인 가능

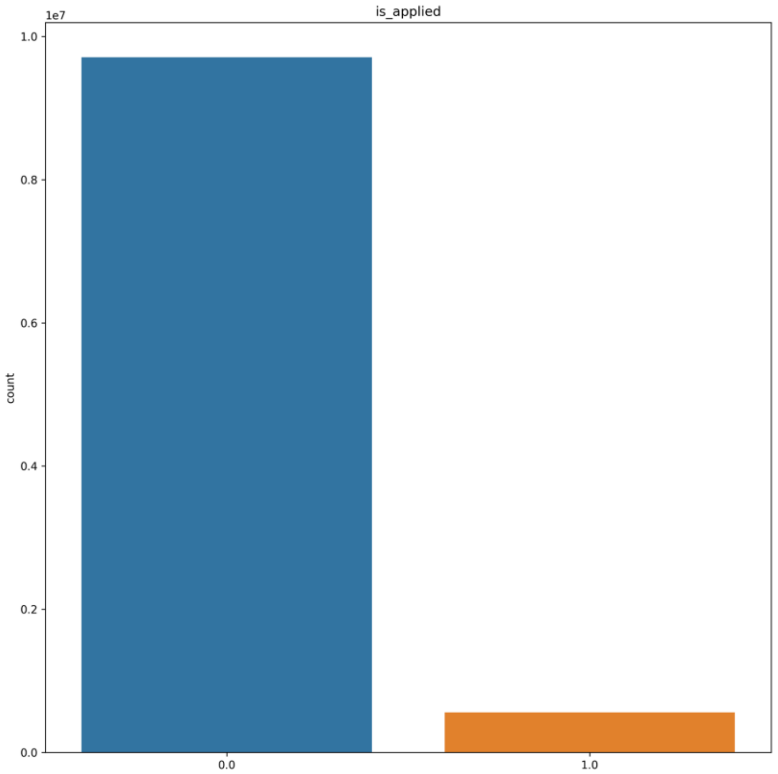
-> 이용 시간대를 나타내는 변수 'hour' 삽입

데이터 EDA

Target 시각화

target이 굉장히 불균형함을 알 수 있음

-> 균형 랜덤포레스트 (BalancedRandomforest) 활용해보기



데이터 EDA

컬럼별 분석 노트

파일명		
테이블설명		
No.	컬럼ID	분석노트
1	application_id	
2	user_id	target data에서는 Nan값인 경우 없으므로 Nan인 데이터 drop
3	birth_year	다른 대출 기록으로 Nan값 filling, 나머지는 중앙값 채우기 -> 연도 대신 나이로 변환해 'Age' 피처 추가
4	gender	다른 대출 기록으로 NaN값 filling, 나머지는 0으로 채우기
5	insert_time	target data에서는 Nan값인 경우 없으므로 최근 시간으로 filling -> 'hour' 피처 추가 -> 시간대별로 binning (0,6,12,18,24)
6	credit_score	다른 대출 기록으로 Nan값 filling, 나머지 결측치는 중앙값 채우기 ->올크레딧 (KCB) 신용점수에 따른 신용등급을 기준으로 binning을 진행한다.
7	yearly_income	다른 대출 기록으로 Nan값 filling
8	income_type	4대보험 가입자/ 그외로 나눌것
9	company_enter_month	현재 시간으로 결측치 채우기 -> 재직일수로 변환해서 'day' 피처 추가
10	employment_type	정규직/그 외로 나눌것
11	houseown_type	자가 소유/그외로 나눌것
12	desired_amount	
13	purpose	사업/투자','집, 차구매','보증금','주택대출', '생활비', '대환대출', '기타' 로 나눌 것
14	personal_rehabilitation_yn	개인회생자 - 납입 미완 (0), 개인회생자 - 납입 완 (1), 기타 - (2) 라벨링 진행 null인것들중 credit_score_cut 3 미만인 것들 - 0 6 미만인 것들 - 1 나머지는 2로 라벨링
15	personal_rehabilitation_complete_yn	
16	existing_loan_cnt	0으로 filling -> 둘은 공통적인 특성을 가진 변수이고, 대출 기록보다 남아있는 대출 금액이 중요 existing_loan_cnt는 제거
17	existing_loan_amt	

정리 내용을 토대로 결측치 제거, 피처 생성 및 변형

결측치 처리 - user_spec (1)

User_id : user_id가 null인 경우는 train data에만 존재하고, 자세한 분석이 힘드므로 drop

Birth_year : fill_from_others(by='user_id'), 나머지는 중앙값으로 결측치 보간. 또한 생년을 현재 나이로 변환한 'Age' 피처 추가

Gender : fill_from_others(by='user_id'), 나머지는 0으로 채웠다.

Insert_time : target data에서는 null값인 경우 없으므로 최근 시간으로 결측치 채운 후 시간대별로 구간을 나눈 변수 (0-6, 6-12, 12-18, 18-24)를 추가

Credit_score : 금융 대출 기록에 따라 신용점수가 바뀌는 것을 고려하여 fill_from_others() 함수 대신 중앙값으로 채운 다음 올크레딧(KCB) 기준에 따라 신용등급으로 구간 나누기

Yearly_income : fill_from_others(by='user_id'), 나머지는 중앙값으로 채운 다음 skewed data를 정규분포에 가깝게 만들기 위해 log scaling을 진행

Income_type : 4대 보험 가입자 / 그외로 나누기

데이터 전처리

결측치 처리 - user_spec (2), loan_result

Company_enter_month : 현재 시간으로 결측치를 채운 후, str 자료형을 datetime 자료형으로 변환한 다음 재직일수로 환산해서 변수로 추가

Employment_type : 정규직/그 외로 나눌 것

Houseown_type : 자가 소유/그 외로 나눌 것

Desired_amount : skewed distribution. Log scaling 진행

Purpose : '사업/투자','집, 차구매 ','보증금','주택대출', '생활비', '대환대출', '기타' 로 나눌 것

Personal_rehabilitation : 개인 회생자 중 납입이 미완료인 경우는 0, 납입이 완료인 경우는 1, 나머지는 2로 라벨링

+) null값인 것들 중 credit_score_cut이 3 미만인 것들은 0, 6 미만인 것들은 1 나머지는 2로 라벨링

Existing_loan_cnt : column drop

Exisiting_loan_amt : 0으로 filling

loan_rate, loan_limit : fill_from_others(by='product_id'), 나머지는 중앙값

데이터 전처리

이상치 확인

yearly_income max인 경우 : 2명의 유저 user_id = [169742, 580362]

신청하지 않았으므로 데이터 중요도 낮고, 극단적인 수치로 예측 모델 훈련에 악영향 끼침 -> row drop

```
merge_train.loc[merge_train['yearly_income'] == merge_train['yearly_income'].max()].groupby('user_id').mean()
```

user_id	application_id	bank_id	product_id	loan_limit	loan_rate	is_applied	birth_year	gender	credit_score	yearly_income	co
169742	922944.586667	33.613333	133.866667	5.562667e+07	8.949333	0.0	1981.0	0.0	1000.0	1.000000e+10	
580362	553804.545455	11.000000	128.454545	1.518182e+07	17.100000	0.0	1992.0	1.0	810.0	1.000000e+10	

데이터 전처리

이상치 확인

loan_limit max인 경우 : 1개의 상품 (product_id = 246)

어떠한 경우 이 상품을 신청했는지 확인

```
merge_train.loc[(merge_train['loan_limit'] == merge_train['loan_limit'].max()) & (merge_train['is_applied'] == 1)]
```

	application_id	loanapply_insert_time	bank_id	product_id	loan_limit	loan_rate	is_applied	user_id	birth_year	gender	...	income_type	company_enter_month	employment_type	houseown_type	de
100861	639784	2022-03-01 14:33:29	43	246	1.000000e+10	9.9	1.0	449938	1968.0	1.0	...	EARNEDINCOME	200504.0	PERMANENT	OWN	
650919	1934129	2022-04-04 23:50:05	43	246	1.000000e+10	9.9	1.0	796150	1990.0	1.0	...	EARNEDINCOME	201403.0	PERMANENT	FAMILY_ETC	
848261	54458	2022-03-10 19:48:54	43	246	1.000000e+10	9.9	1.0	550872	1972.0	1.0	...	EARNEDINCOME	200705.0	PERMANENT	RENT	
5080738	497413	2022-03-23 08:49:06	43	246	1.000000e+10	9.9	1.0	226244	1977.0	0.0	...	EARNEDINCOME	201702.0	PERMANENT	OWN	
7978549	503070	2022-04-05 00:12:23	43	246	1.000000e+10	9.9	1.0	796150	1990.0	1.0	...	EARNEDINCOME	201403.0	PERMANENT	FAMILY_ETC	
9306646	2026384	2022-03-22 17:41:25	43	246	1.000000e+10	9.9	1.0	847796	1991.0	1.0	...	EARNEDINCOME	201406.0	PERMANENT	OWN	
13333269	850504	2022-03-02 21:00:48	43	246	1.000000e+10	9.9	1.0	449938	1968.0	1.0	...	EARNEDINCOME	200504.0	PERMANENT	OWN	

원하는 금액, 목적과 맞지 않는 상품 신청한 경우 (application_id = 1934129, 503070) drop

데이터 전처리

Log_data count 변수 추가

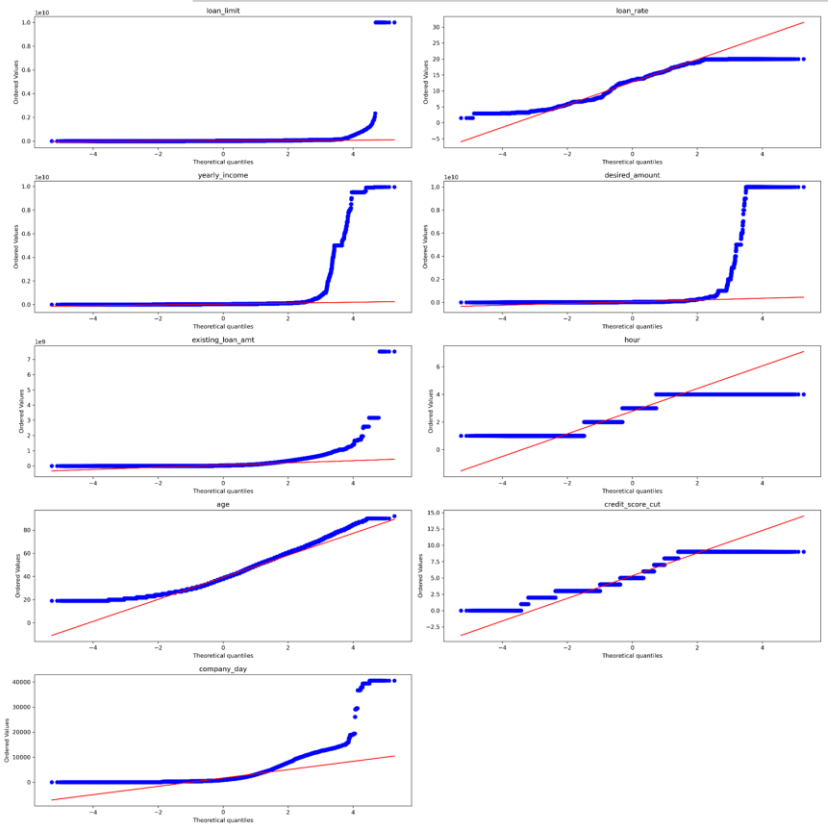
- ‘user_id’ 별로 로그 데이터의 행동들을 카운트한 변수 추가 -> 결측치 평균값으로 채우기
- 로그인 횟수나 단순한 인증 횟수는 큰 영향을 주지 않음
- 중요변수로 판단되는 ['StartLoanApply’, 'EndLoanApply''GetCreditInfo', 'UseDSRCalc', 'UseLoanManage', 'UsePrepayCalc'] 만 선택해 삽입

데이터 전처리

QQ plot

QQ plot을 통해 numerical value들의 분포가 정규 분포와 어느정도 차이나는지 확인

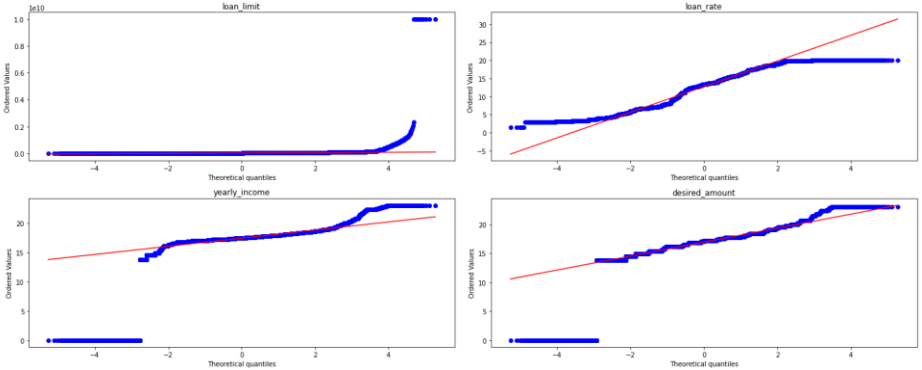
yearly_income,
desired_amount,
existing_loan_amt 변수에 대한 **log scaling** 진행

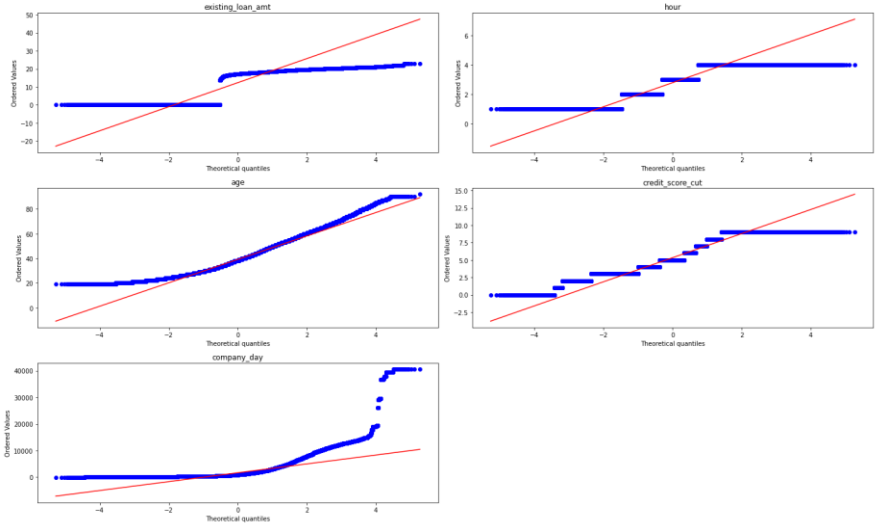


데이터 전처리

QQ plot

Log scaling 이후 QQ plot





데이터 전처리

Scaler

Standardscaler, minmaxscaler, robustscaler로 스케일링 진행 후 성능이 가장 좋았던 Standardscaler 를 사용

Encoding

순서가 없는 카테고리형 변수 -> One-Hot-encoder 를 사용

분석 노트에 써두었던대로 중요도 떨어지는 변수 합치기

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.preprocessing import OneHotEncoder

num_pipeline = Pipeline([
    ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
    ('onehotencoding', OneHotEncoder(sparse=False))
])


preprocessor = ColumnTransformer([
        ("num", num_pipeline, num_col),
        ("log_cnt", num_pipeline, log_col),
        ("cat", cat_pipeline, cat_col)
])

preprocessor
```

데이터 전처리

```
train_prepared['cat_EARNED_INCOME'] = train_prepared['onehotencoding_x0_EARNEDINCOME']
train_prepared['cat_ETC_INCOME'] = train_prepared['onehotencoding_x0_EARNEDINCOME2'] + train_prepared['onehotencoding_x0_FREELANCER'] + train_prepared['onehotencoding_x0_OTHE

target_prepared['cat_EARNED_INCOME'] = target_prepared['onehotencoding_x0_EARNEDINCOME']
target_prepared['cat_ETC_INCOME'] = target_prepared['onehotencoding_x0_EARNEDINCOME2'] + target_prepared['onehotencoding_x0_FREELANCER'] + target_prepared['onehotencoding_x0_
```

```
train_prepared['cat_PERMANENT'] = train_prepared['onehotencoding_x1_PERMANENT']
train_prepared['cat_NOT_PERMANENT'] = train_prepared['onehotencoding_x1_CONTRACT'] + train_prepared['onehotencoding_x1_DAYJOB'] + train_prepared['onehotencoding_x1_ETC']

target_prepared['cat_PERMANENT'] = target_prepared['onehotencoding_x1_PERMANENT']
target_prepared['cat_NOT_PERMANENT'] = target_prepared['onehotencoding_x1_CONTRACT'] + target_prepared['onehotencoding_x1_DAYJOB'] + target_prepared['onehotencoding_x1_ETC']
```

```
train_prepared['cat_OWN'] = train_prepared['onehotencoding_x2_OWN']; train_prepared['cat_RENT'] = train_prepared['onehotencoding_x2_RENT']
train_prepared['cat_ETC_HOUSE'] = train_prepared['onehotencoding_x2_SPOUSE'] + train_prepared['onehotencoding_x2_FAMILY_ETC']

target_prepared['cat_OWN'] = target_prepared['onehotencoding_x2_OWN']; target_prepared['cat_RENT'] = target_prepared['onehotencoding_x2_RENT']
target_prepared['cat_ETC_HOUSE'] = target_prepared['onehotencoding_x2_SPOUSE'] + target_prepared['onehotencoding_x2_FAMILY_ETC']
```

```
train_prepared['cat_BUSINESS_INVEST'] = train_prepared['onehotencoding_x3_BUSINESS'] + train_prepared['onehotencoding_x3_INVEST']
train_prepared['cat_BUY'] = train_prepared['onehotencoding_x3_BUYCAR'] + train_prepared['onehotencoding_x3_BUYHOUSE']
train_prepared['cat_HOUSEDEPOSIT'] = train_prepared['onehotencoding_x3_HOUSEDEPOSIT']

target_prepared['cat_BUSINESS_INVEST'] = target_prepared['onehotencoding_x3_BUSINESS'] + target_prepared['onehotencoding_x3_INVEST']
target_prepared['cat_BUY'] = target_prepared['onehotencoding_x3_BUYCAR'] + target_prepared['onehotencoding_x3_BUYHOUSE']
target_prepared['cat_HOUSEDEPOSIT'] = target_prepared['onehotencoding_x3_HOUSEDEPOSIT']
```

데이터 전처리

최종 피처 선택

- 타깃 값과의 상관계수 확인 -> 상관계수가 낮은 피처들이 많이 보임
- 학습의 효율을 위해 **|상관계수| < 0.03**인 column drop

```
corr_target.loc[abs(corr_target) >= 0.03]

cat_SWITCHLOAN          -0.036555
cat_HOUSEDEPOSIT        -0.034760
num_hour                -0.031741
cat_NOT_PERMANENT        0.081406
num_company_day         -0.036276
num_loan_rate           -0.069533
cat_ETC_INCOME           0.115619
cat_PERMANENT           -0.081406
cat_REHAB_ETC           -0.047960
cat_LIVING               0.075977
num_existing_loan_amt    0.031185
num_credit_score_cut    -0.158473
cat_EARNED_INCOME       -0.115619
log_cnt_EndLoanApply     0.039005
num_desired_amount      -0.123646
cat_OWN                 -0.034234
cat_BUY                 -0.044733
cat_REHAB_COMP           0.048067
cat_ETC_HOUSE            0.031171
num_yearly_income       -0.038972
is_applied               1.000000
Name: is_applied, dtype: float64
```

학습-검증용 데이터 분리

StratifiedShuffleSplit (계층적 데이터 샘플링)

```
1
2 from sklearn.model_selection import StratifiedShuffleSplit
3
4 def split_data(data, label=label, test_size = 0.02, random_state = 42):
5     split = StratifiedShuffleSplit(n_splits=5, test_size = test_size, random_state=random_state)
6
7     for train_index, valid_index in split.split(data, label):
8         train_data = data.loc[train_index]
9         train_label = label.loc[train_index]
10        valid_data = data.loc[valid_index]
11        valid_label = label.loc[valid_index]
12
13    return train_data, train_label, valid_data, valid_label
14 train_data, train_label, valid_data, valid_label = split_data(train, label)
15 train_data
```

베이스라인 모델

- **선형 분류 모델**

SVC(kernel=[‘rbf’, ’poly’])

- **트리 기반 모델**

RandomforestClassifier
LightGBM

3개의 모델을 베이스라인 모델로 선정, 학습 및 성능평가 진행

SVC

Kernel : rbf

- Kernel 중 가장 많이 사용하는 rbf 를 사용
- 많은 양의 데이터로 학습을 하다보니 학습 속도 저하와 RAM 문제 발생

```
estimator = SVC(kernel = 'rbf',C = 10, gamma = 0.1)
n_estimators = 3
n_jobs = 1
svc_model = BaggingClassifier(base_estimator=estimator,
                              n_estimators=n_estimators,
                              max_samples=1./n_estimators,
                              n_jobs=n_jobs)
svc_model.fit(X_train,y_train)
```

SVC

Kernel : poly

Rbf 커널에서의 문제를 해결하고자 다항식의 형태인 poly 커널 사용

```
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier

estimator = SVC(kernel = 'poly', gamma = 0.1)
n_estimators = 3
n_jobs = 1
svc_model = BaggingClassifier(base_estimator=estimator,
                              n_estimators=n_estimators,
                              max_samples=1./n_estimators,
                              n_jobs=n_jobs)
svc_model.fit(X_train,y_train)
```

SVC

모델 학습 결과

Target 불균형 문제로 인해 Accuracy 는 높게 나왔지만 F1 score 는 아주 낮게 나온 것을 확인

metric	score
accuracy	0.9453666082
precision	0.2602739726
recall	0.000241985812
f1 score	0.0004835220766

BalancedRamdomForest

모델 설명

샘플링 기법을 제공하는 imblearn 에서 제안한 랜덤포레스트 모델

```
from imblearn.ensemble import BalancedRandomForestClassifier

bal_clf= BalancedRandomForestClassifier(random_state=42)

bal_clf.fit(train_data, train_label)
```

선정 이유

타깃 불균형 문제에 영향을 덜 받기 위해

BalancedRamdomForest

모델 튜닝

GridSearchCV 를 사용하여 모델 튜닝 진행

```
from sklearn.model_selection import GridSearchCV

rf_tuned_parameters = [{ 'n_estimators' : [100],
           'min_samples_leaf' : [1, 2, 4, 8 ],
           'min_samples_split' : [2, 4, 8]
            }
]

rf_tuned = GridSearchCV(BalancedRandomForestClassifier(random_state=42),rf_tuned_parameters,cv = 2,scoring="f1",verbose=2,refit=True)
rf_tuned.fit(train_data,train_label)
```

BalancedRamdomForest

모델 성능

왼쪽 : Train 오른쪽 : valid

	metric	score
0	accuracy	0.819461
1	precision	0.227932
2	recall	0.967206
3	f1 score	0.368924

	metric	score
0	accuracy	0.811754
1	precision	0.220530
2	recall	0.966717
3	f1 score	0.359134

모델 학습 결과

Accuracy 는 낮아졌지만 F1 score 를 보면 성능이 향상된 것을 확인할 수 있다.

LightGBM

모델 설명

Gradient Boosting 프레워크로 Tree 기반 학습 알고리즘

선정 이유

Light GBM은 큰 사이즈의 데이터를 다룰 수 있고 실행시킬 때 적은 메모리를 차지함

LightGBM

```
import lightgbm as lgb
from sklearn.metrics import f1_score
from bayes_opt import BayesianOptimization

n_folds = 10

d_train = lgb.Dataset(train_data, label=train_label.values, free_raw_data=False)
d_valid = lgb.Dataset(valid_data, label=valid_label.values, free_raw_data=False, reference=d_train)

# simplify
baseline_lgbm = lgb.LGBMClassifier(random_state=42)
baseline_lgbm.fit(train_data, train_label)
```

LightGBM

모델 성능

왼쪽 : Train 오른쪽 : valid

	metric	score
0	accuracy	0.945537
1	precision	0.537694
2	recall	0.012745
3	f1 score	0.024899

	metric	score
0	accuracy	0.945540
1	precision	0.539157
2	recall	0.012778
3	f1 score	0.024964

모델 학습 결과

아무것도 설정하지 않은 베이스라인에 서는 좋지 않은 성능 보임

-> 하이퍼파라미터 튜닝을 통한 수정

하이퍼파라미터 튜닝

Optuna 라이브러리를 이용해 각 베이스라인 모델들의 하이퍼파라미터를 튜닝

```
!pip install optuna
```

```
import optuna
from optuna import Trial
from optuna.samplers import TPESampler
from sklearn.model_selection import cross_val_score
```

RandomForest

```
def rf(trial : Trial) -> float:
  params_rf = {
    "max_depth" : trial.suggest_int('max_depth', 100, 500),
    "max_leaf_nodes" : trial.suggest_int('max_leaf_nodes', 100, 1000),
    "n_estimators" :  trial.suggest_int('n_estimators', 500, 1000),
    "verbosity": -1,
    "seed" : 42,
  }
  rf_clf = RandomForestClassifier(**params_rf,n_jobs=-1,random_state=42)

  rf_clf.fit(train_data, train_label)
  score = cross_val_score(rf_clf, train_data, train_label, cv=5, scoring="f1")
  f1_mean = score.mean()

  return f1_mean
```

LightGBM

```
import lightgbm as lgb
from sklearn.metrics import f1_score
from bayes_opt import BayesianOptimization

n_folds = 10

d_train = lgb.Dataset(train_data, label=train_label.values, free_raw_data=False)
d_valid = lgb.Dataset(valid_data, label=valid_label.values, free_raw_data=False, reference=d_train)
```

```
param_test ={'num_leaves': (6,45),
             'min_child_weight': (1e-3, 1e3), #0.001 ~ 1000
             'subsample': (0.5,1),
             'colsample_bytree': (0.2,1),
             'reg_alpha': (0, 100),
             'reg_lambda': (0, 100),
             'learning_rate': (0.01,0.2),
             'max_depth': (3,8)} #파라미터 서칭 범위 설정

def lgb_f1_score(y_hat, data):
    y_true = data.get_label().astype(int)
    # y_hat = (y_hat >= 0.5)
    y_hat = np.round(y_hat).astype(int)
    return 'f1', f1_score(y_true, y_hat), True #f1_score metric 만들어줌

def lgb_cv(num_leaves,min_child_weight,subsample,colsample_bytree,reg_alpha,reg_lambda,max_depth,learning_
    params = {'num_iterations' : 5000,
              'early_stopping_rounds':100,
              'n_jobs':-1,
              'metric' : 'custom', #metric custom으로 f1 score 지정
              }
    params['num_leaves'] = int(round(num_leaves))
    params['min_child_weight'] = min_child_weight
    params['subsample'] = subsample
    params['colsample_bytree'] = colsample_bytree
    params['reg_alpha'] = reg_alpha
    params['reg_lambda'] = reg_lambda
    params['learning_rate'] = learning_rate
    params['max_depth']= int(round(max_depth)) #param_test의 파라미터 넣을 예정
    lgbcv = lgb.cv(params, d_train, nfold=n_folds, seed=42, stratified=True, verbose_eval = 200,
                   metrics=['None'], feval=lgb_f1_score)

    return max(lgbcv['f1-mean'])
```

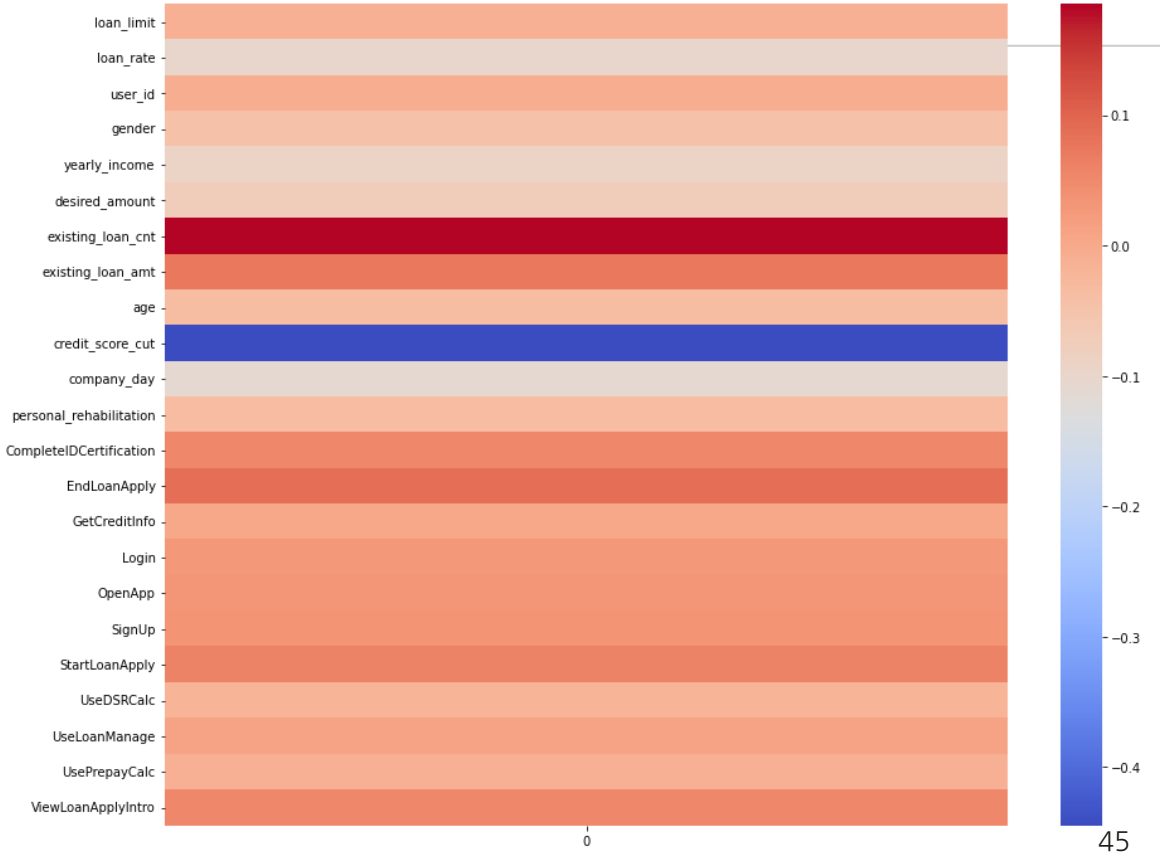
군집 분석

다음은 Target 인 ‘is_applied’ 와 다른 변수사이의 상관관계를 표현한 히트맵이다.

여기서 보면 기대출수가 가장 양의 상관관계를 가지는 것을 볼 수 있고

신용 등급이 가장 강한 음의 상관관계를 가지는 것을 확인할 수 있다.

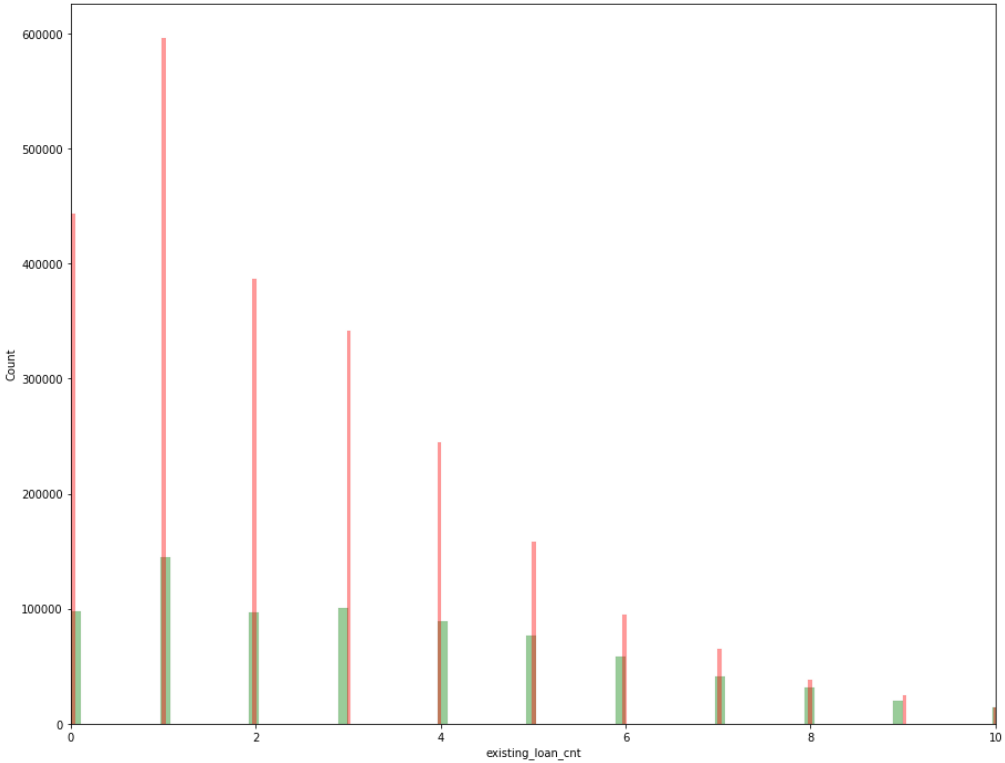
따라서 앞으로 군집분석을 신용등급에 따라 나눠서 진행할 계획이다.



군집 분석

Existing_loan 분석

Exisiting_loan_cnt (기대출 수) 를 확인해보면 대부분의 사용자가
대출을 1개이상 가지고 있는 것을 확인할 수 있다.
그리고 그중에서 대출을 1개 가지고 있는 고객이 가장 많은 것을 확인할 수 있다.
이 점을 고려하여 대출을 가지고 있는 이용자에게 좋은 대출 상품을 많이 광고하는 것을 효과적일 것이다.



군집 분석

신용 등급을 기준의 군집화 진행

Imblearn RamdomForest 를 통해 예측한 test 를 분석

고신용자 : 1 ~ 3 등급
중신용자 : 4 ~ 6 등급
저신용자 : 7 ~ 10 등급
(올크레딧 기준)

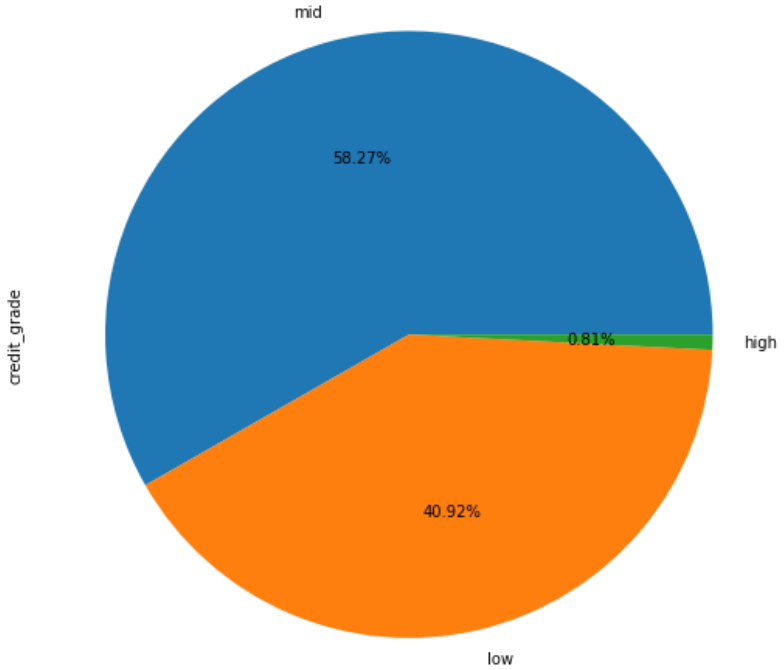
신용등급 점수표

등급	나이스	올크레딧 (KCB)
1	900 ~ 1000점	942 ~ 1000점
2	870 ~ 899점	891 ~ 941점
3	840 ~ 869점	832 ~ 890점
4	805 ~ 839점	768 ~ 831점
5	750 ~ 804점	698 ~ 767점
6	665 ~ 749점	630 ~ 697점
7	600 ~ 664점	530 ~ 629점
8	515 ~ 599점	454 ~ 529점
9	445 ~ 514점	335 ~ 453점
10	0 ~ 444점	0점 ~ 334점

군집 분석

신용 등급을 기준의 군집화 진행

Test 데이터를 확인하면
중신용자가 대출 신청비율이 가장 많고
그 다음 저 신용자,
마지막으로 고신용자의 경우 아주 적은 것을
확인할 수 있다.

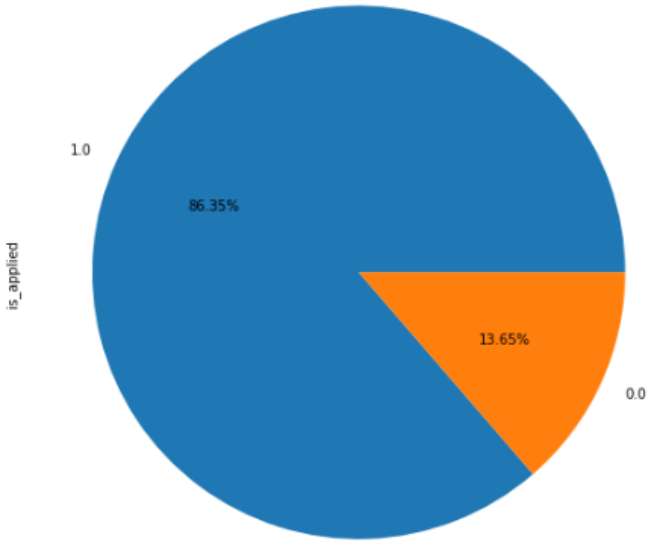


군집 분석

고신용자 대출 승인 비율

App 을 사용하는 수는 가장 적지만
실제 대출 신청에 따른 승인 비율이 고신용자 전체 수 대비 87% 로 상당히 높은 것을 알 수 있다.

따라서 사용하는 인원은 적지만 실제 대출 승인으로 가는 고객이 많다.



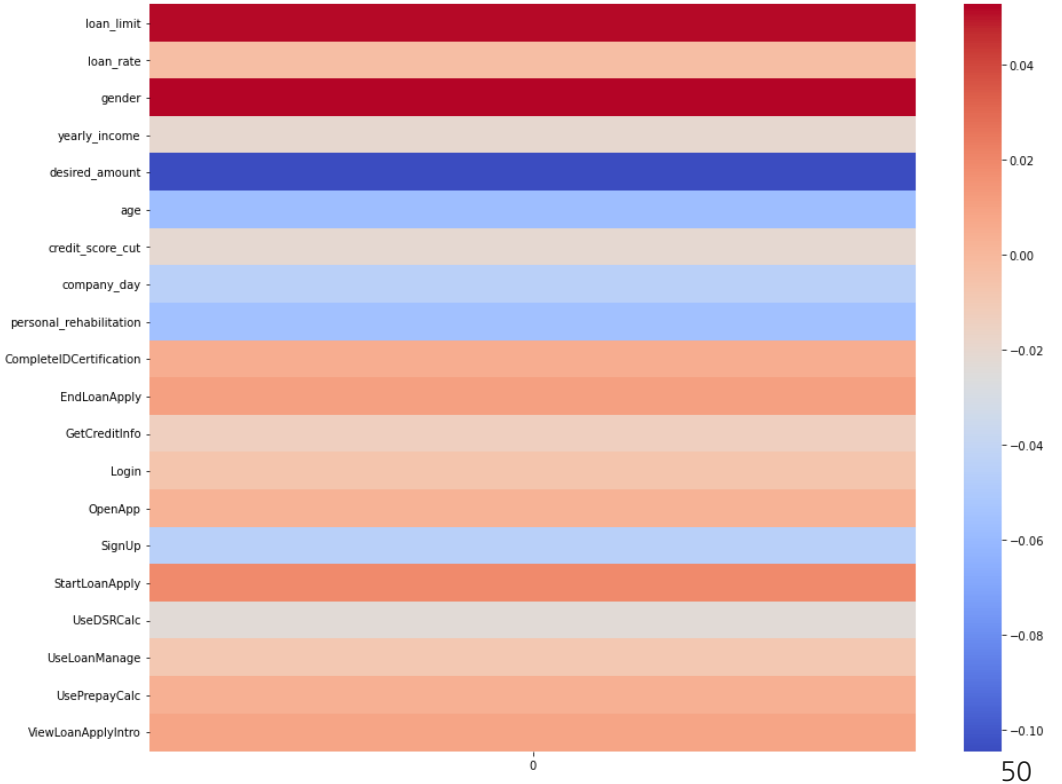
군집 분석

고신용자 상관관계

Target 인 'is_applied' 와의 상관관계를 히트맵으로 표현한 결과이다.

고객 정보 측면
양의 상관관계 : 성별, 대출 한도
음의 상관관계 : 나이 대출희망금액, 개인회생여부

앱 사용성 측면
양의 상관관계 : 한도 조회
음의 상관관계 : 회원가입



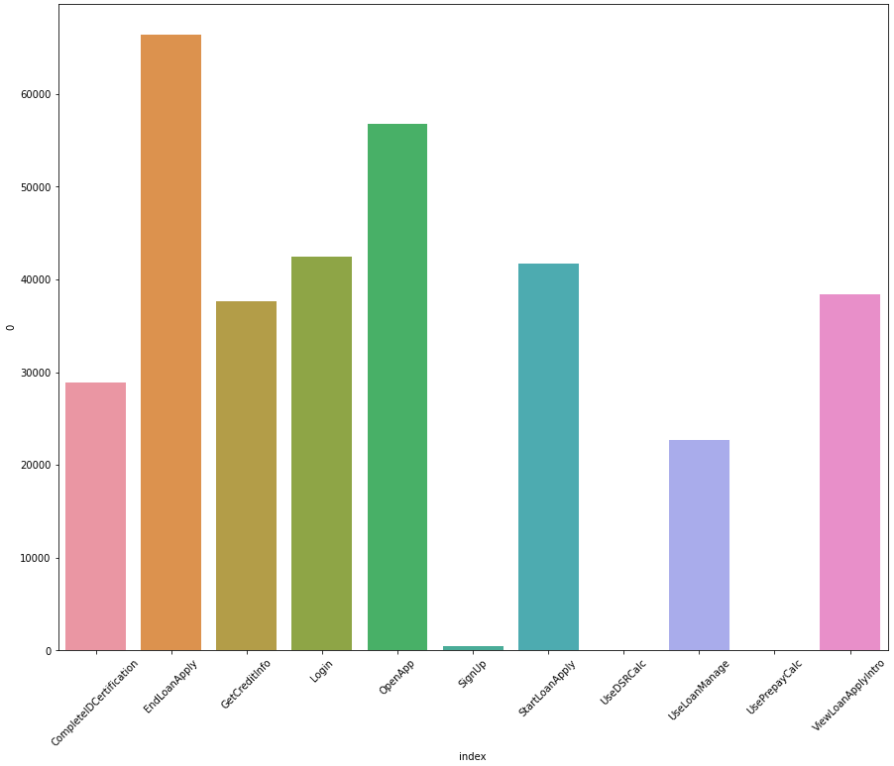
군집 분석

고신용자 앱 사용성 EDA

고신용자의 경우 대출 신청의 비율이 높기 때문에
고신용자 고객이 대출을 조회하도록 해야한다.

위의 히트맵을 보면 앱사용성 데이터 중에서
한도 조회에 대한 상관관계가 가장 높고
한도조회 결과를 확인한 로그 가장 많으므로
한도조회 부분에서 고객에게 좋은 정보를 제공 해야 한다.

앱 측면에서 대출 승인 비율이 높은 고신용자를 위한 시작 이벤트를 추진하며 앱을 설치하도록 만들어야 한다.

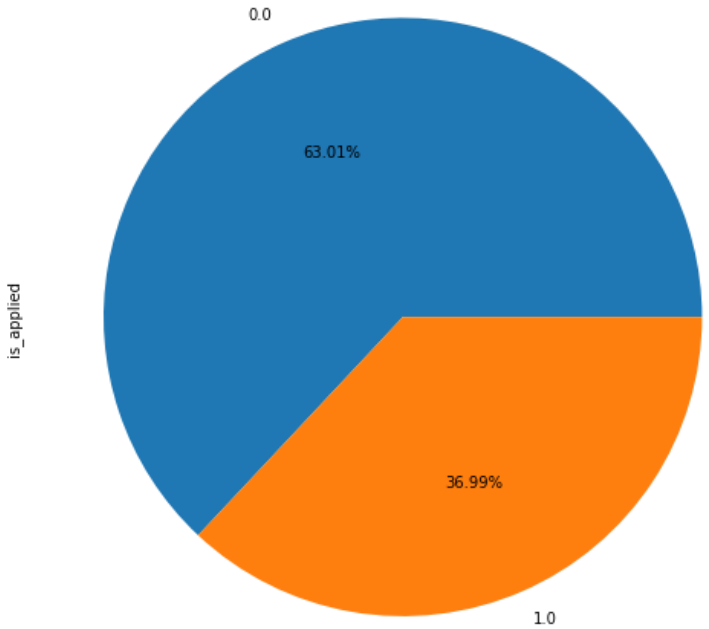


군집 분석

중신용자

App 을 사용하는 수는 가장 많은 가방 활발한 군집이다.

대출 조회 대비 신청 비율을 보면 대출 신청한 경우와 신청하지 않은 비율이 6 : 4 인 것을 볼 수 있다.



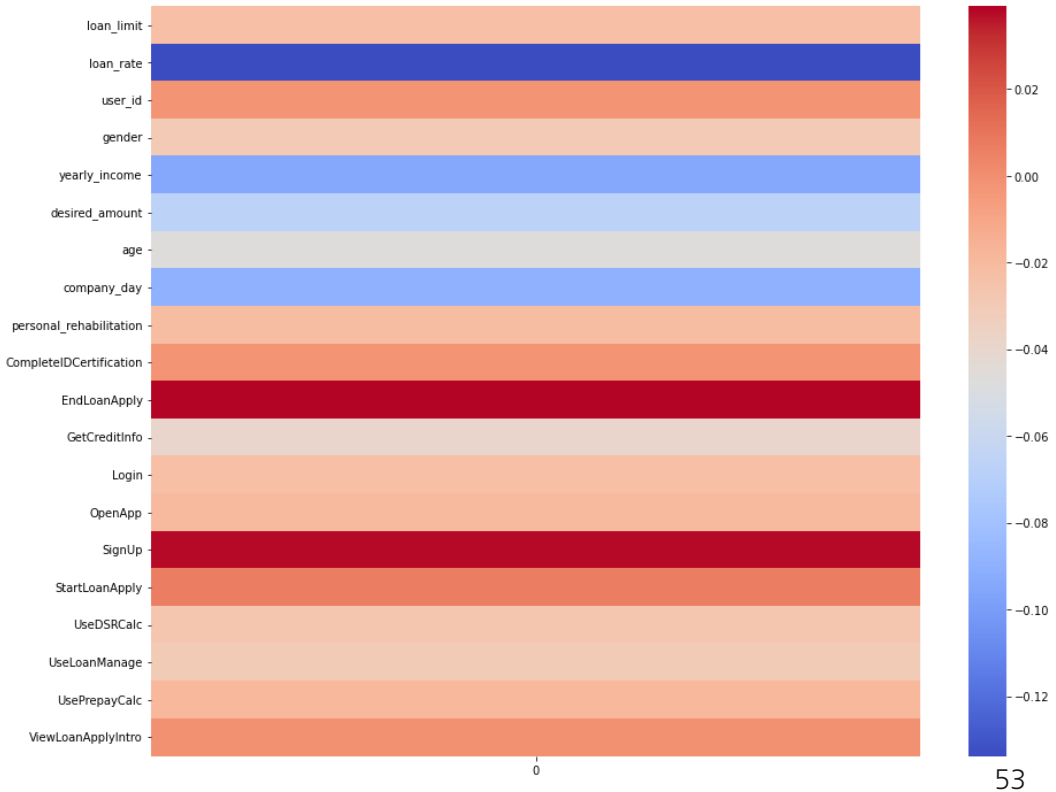
군집 분석

중신용자 타겟 상관관계

타겟과의 상관관계

고객 정보 측면
양의 상관관계 : 개인회생여부, 대출한도
음의 상관관계 : 대출 금리, 연봉, 근무일 수
앱 사용성 측면
양의 상관관계 : 한도 조회, 회원가입
음의 상관관계 : 회원가입

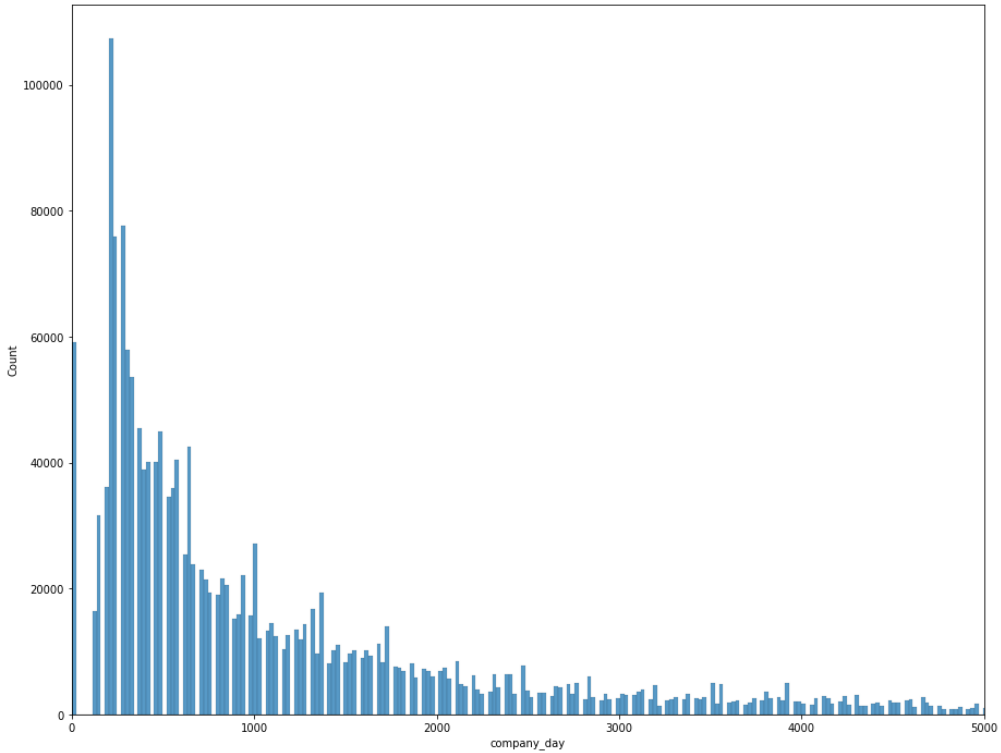
고신용자와 다르게 연봉, 근무일 수가 중요한 변수가 등장한 것을 확인



군집 분석

중신용자 근무일수

대부분 3년 이하의 근무 경력을 가진 사람들이 많았다.
경력이 적은 사람들이 많다.



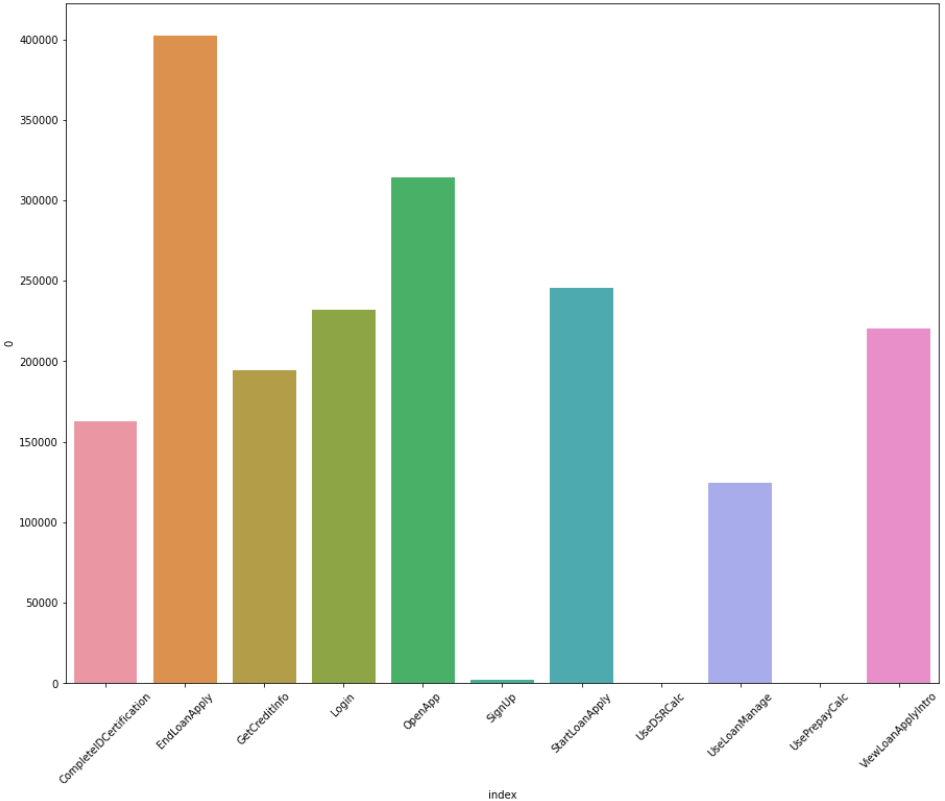
군집 분석

중신용자 앱 사용성 EDA

오른쪽은 대출 신청 고객과 대출 미신청 고객의 앱 사용 log 데이터의 차이점을 시각화한 것이다.

그래프를 보면 App 을 열어본 횟수와 한도조회를 한 결과가 가장 큰 차이를 보이는 것을 알 수있다.

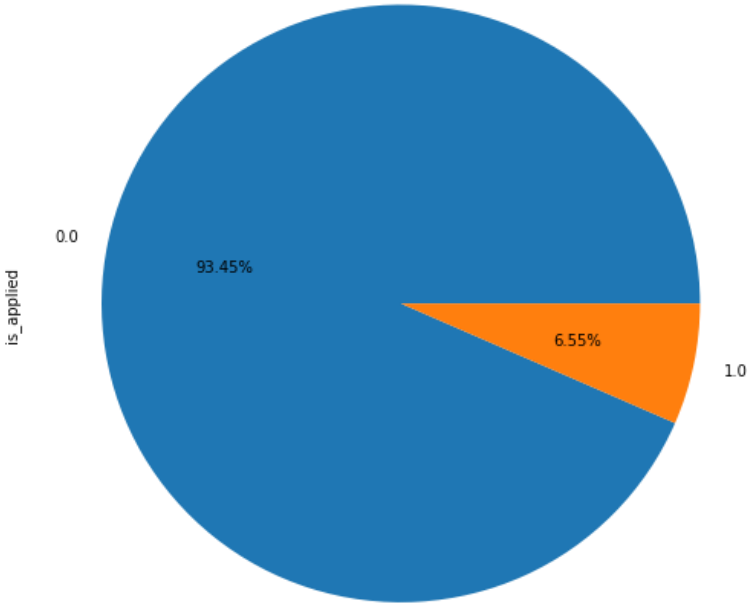
이로써 중신용자의 경우에는 한도 조회 결과를 보는 파트와 App 을 열었을 때 광고를 하는 것이 효과적으로 보인다.



군집 분석

저신용자 승인 비율

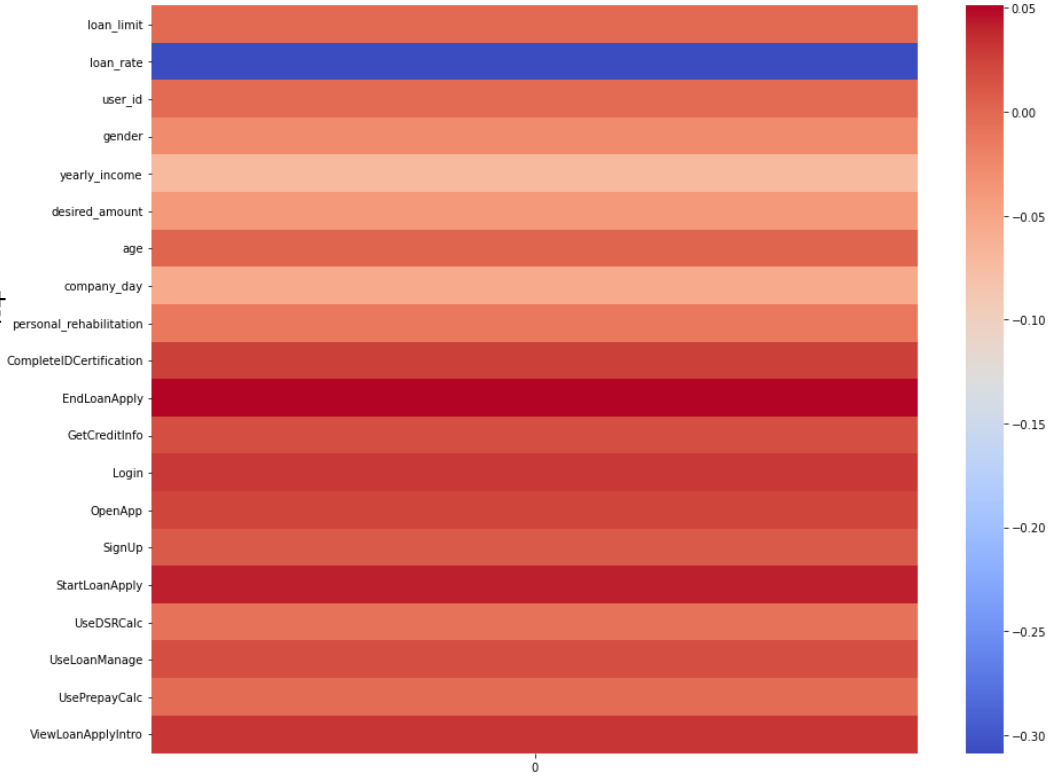
App 을 사용하는 수는 두 번째로 많지만
실제 대출 신청 대비 승인되는 비율이 6.5 퍼센트로
매우 적은 것을 확인할 수 있다.



군집 분석

저신용자 Target 상관관계

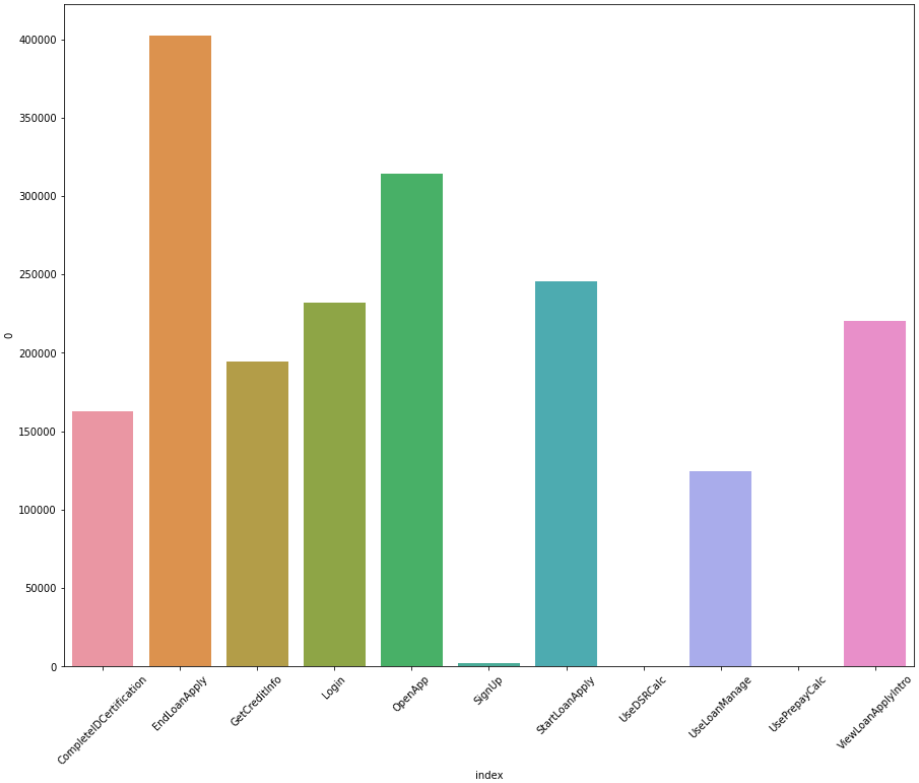
상관관계 분석 결과 loan_rate 금리를 제외한 다른 모든 변수가 양의 상관관계를 가지는 것을 알 수 있다.



군집 분석

저신용자 앱 사용성 분석

중신용자 분석과 동일하게 대출 신청 여부에 따른 차이를 시각화한 자료이다.
분석 결과 중신용자의 분석 결과와 비슷한 것을 확인할 수 있다.



저신용자 분석 결과

저신용자의 경우 대출 신청 대비 승인 비율이 매우 낮기 때문에
저신용자를 타겟으로 하는 마케팅의 경우는 효율이 낮을 것으로 예상된다.

단 사용자가 많기 때문에 대출 신청이 승인되지 않은 고객을 위한 대출 상품을 준비한다면 큰 효과를 얻을 것으로 예상된다.

아쉬운 점

많은 양의 데이터를 통해 학습을 진행하다보니 메모리와 컴퓨터 자원의 문제가 많이 발생하였습니다.
컴퓨터 성능의 문제로 모델 학습이나 기법을 적용하는데 오랜 시간이 걸린 것이 아쉽습니다.

Thanks!

You can find me at

- yoon98619@gmail.com
- hwyewon@gmail.com