2022 빅콘테스트 퓨처스부문

대출고대로 분석했조

결과보고서

팀명 : 대출고대로분석했조
팀장 : 박태주(qkrxown5985@gmail.com)
팀원 : 김채성(zzdd0814@naver.com)
김채현(chaehyeongim0315@gmail.com)
김지원(kjiwon1222@gmail.com)

목차

앱 사용성 데이터를 통한
대출신청 예측분석

데이터 전처리

가. 데이터 전처리의 목표정리

데이터를 바로 사용하기에 많은 결측치와 오류값이 존재한다. 이들은 결과 예측에 오류를 생기게 하므로 제거하거나 다른 값으로 바꾸어 주어야 한다. (수정필요)

나. 3개의 데이터 확인

고객들의 대출 산청여부를 예측하기 위해서는 train데이터와 test데이터를 가지는 loan_result데이터와 고객들의 신용정보를 이용하기 위해 user_spec데이터를 사용한다.

데이터 전처리

다. 변수삭제

특정기간 학습용 데이터이기 때문에 시간과 관련된 변수들은 우리가 예측하고자 하는 is_applied 변수와 독립적이라 가정했다.
또한 데이터에 익명성이 존재하기 때문에 성별과도 독립적이라 가정했다.
그리고 loan_result에서는 product_id 자체가 상품을 나타낼 수 있고, bank_id는 상품예측에서 중복된 정보를 가진다.
그래서 user_spec 에서 gender, insert_time, company_enter_month 변수들을, loan_result에서는 loanapply_insert_time, bank_id 변수를 을 제거했다.

```
user_spec = user_spec.drop(['insert_time', 'gender','company_enter_month'],axis=1)
loan_result = loan_result.drop(['loanapply_insert_time','bank_id'],axis=1)
```
✓ 0.6s

라. 새로운 변수

분석의 편의를 위해 birth_year의 값들과 2022의 차이를 age라는 새로운 변수로 할당한다. 출생년도를 나이로 바꾸어주는 전처리 작업을 수행한다.

```
user_spec.rename(columns = {"birth_year":"age"},inplace=True)
user_spec['age'] = 2022 - user_spec['age']
```
0.9s

데이터 전처리

마. 범주형 변수 처리

employment_type, houseown_type, purpose 열에 대해 각 목록을 영어로 변환한다. purpose의 경우, 동일한 의미에 대해 영어와 한국어로 범주가 나누어져 있어서 이것들을 하나로 통합했다.

EMPLOYMENT_TYPE

```
#employment_type
print(user_spec['employment_type'].unique())
user_spec = user_spec.replace({'employment_type':'기타'}, 'ETC')
user_spec = user_spec.replace({'employment_type':'정규직'}, 'REGULAR')
user_spec = user_spec.replace({'employment_type':'계약직'}, 'COMTRACT')
user_spec = user_spec.replace({'employment_type':'일용직'}, 'DAILY')
print(user_spec['employment_type'].unique())
```

```
['기타' '정규직' '계약직' '일용직' nan]
['ETC' 'REGULAR' 'COMTRACT' 'DAILY' nan]
```

HOUSEOWN_TYPE

```
#houseown_type
print(user_spec['houseown_type'].unique())
user_spec = user_spec.replace({'houseown_type':'자가'}, 'OWN')
user_spec = user_spec.replace({'houseown_type':'기타가족소유'}, 'ETCFAMILY')
user_spec = user_spec.replace({'houseown_type':'전월세'}, 'RENT')
user_spec = user_spec.replace({'houseown_type':'배우자'}, 'SPOUSE')
print(user_spec['houseown_type'].unique())
```

```
['자가' '기타가족소유' '전월세' '배우자' nan]
['OWN' 'ETCFAMILY' 'RENT' 'SPOUSE' nan]
```

PURPOSE

```
#purpose
print(user_spec['purpose'].unique())
user_spec = user_spec.replace({'purpose':'투자'}, 'INVEST')
user_spec = user_spec.replace({'purpose':'자동차구입'}, 'BUYCAR')
user_spec = user_spec.replace({'purpose':'주택구입'}, 'BUYHOUSE')
user_spec = user_spec.replace({'purpose':'사업자금'}, 'BUSINESS')
user_spec = user_spec.replace({'purpose':'생활비'}, 'LIVING')
user_spec = user_spec.replace({'purpose':'전월세보증금'}, 'HOUSEDEPOSIT')
user_spec = user_spec.replace({'purpose':'대환대출'}, 'SWITCHLOAN')
user_spec = user_spec.replace({'purpose':'기타'}, 'ETC')
print(user_spec['purpose'].unique())
```

```
['기타' '대환대출' '생활비' '사업자금' '주택구입' '전월세보증금' '투자' 'LIVING' 'SWITCHLOAN' 'ETC'
 'INVEST' '자동차구입' 'BUSINESS' 'BUYCAR' nan 'HOUSEDEPOSIT' 'BUYHOUSE']
['ETC' 'SWITCHLOAN' 'LIVING' 'BUSINESS' 'BUYHOUSE' 'HOUSEDEPOSIT' 'INVEST'
 'BUYCAR' nan]
```

```
    loan = pd.merge(user_spec, loan_result, how='inner')
    print(loan.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13527250 entries, 0 to 13527249
Data columns (total 18 columns):
 #   Column                              Dtype
---  ------                              -----
 0   application_id                      int64
 1   user_id                             int64
 2   age                                 float64
 3   credit_score                        float64
 4   yearly_income                       float64
 5   income_type                         object
 6   employment_type                     object
 7   houseown_type                       object
 8   desired_amount                      float64
 9   purpose                             object
 10  personal_rehabilitation_yn          float64
 11  personal_rehabilitation_complete_yn float64
 12  existing_loan_cnt                   float64
 13  existing_loan_amt                   float64
 14  product_id                          int64
 15  loan_limit                          float64
 16  loan_rate                           float64
 17  is_applied                          float64
dtypes: float64(11), int64(3), object(4)
memory usage: 1.9+ GB
None
```

데이터 전처리

바. merge data

user_spec과 loan_result를 application_id키를 기준으로 병합하여 데이터 분석 및 예측에 활용할 수 있도록 하나의 셋으로 병합했다.

user_spec에서는 application_id가 고유키인데 비해, loan_result에서는 하나의 application_id 당 여러 행을 가지기 때문에, loan_result의 행을 기준으로 행별 application_id에 대응하는 user_spec의 행을 병합하게된다.

병합이 완료되면 두 개의 데이터셋 중 한쪽에만 포함되는 application_id의 행들은 모두 삭제되어 분석과 예측에 활용할 수 있는 행만 남게된다.

병합시에는 python pandas package의 merge 함수를 사용했다.

데이터 전처리

사. 결측치 처리

결측치가 존재하는 컬럼 (9개)

age, credit_score, yearly_income,
personal_rehabilitation_yn,
personal_rehabilitation_complete_yn,
existing_loan_cnt, existing_loan_amt,
loan_limit, loan_rate

(1) 개인회생여부와 납입완료여부

(2) 누락된 데이터 다른 행에서 찾기

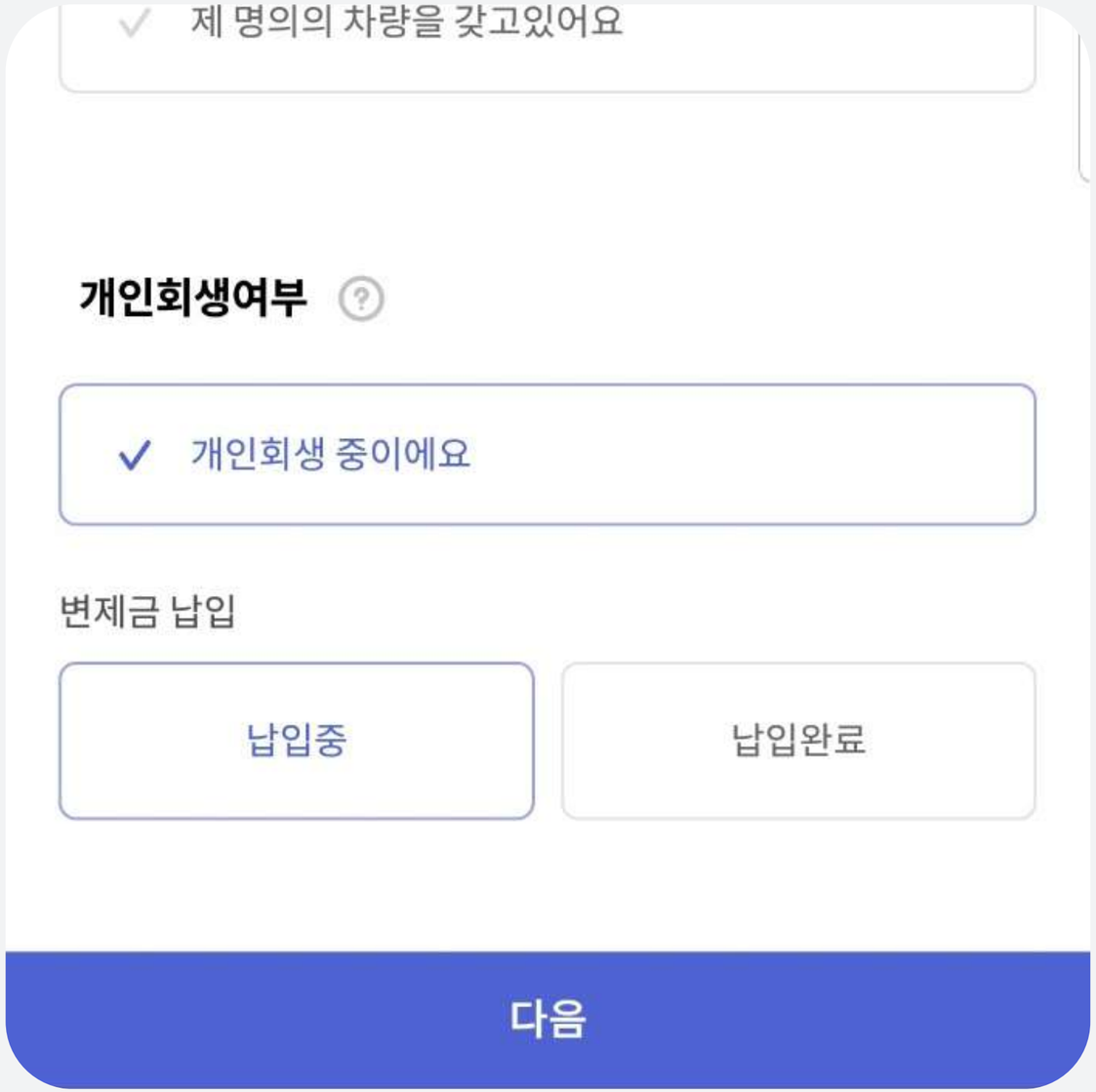
(3) 다중대입법 (MULTIPLE IMPUTATION)

(1)개인회생여부와 납입완료여부

실제 핀다 앱에서 개인회생여부(personal_rehabilitation_yn)을 선택 안 하거나, 선택하면 납입완료여부(personal_rehabilitation_complete_yn)을 납입완료, 납입중 둘 중 하나를 선택해야 다음 화면으로 넘어간다. 따라서 개인회생여부에서 개인회생을 선택하면 personal_rehabilitation_yn은 1이 되고 personal_rehabilitation_complete_yn는 반드시 0 또는 1의 값을 갖는다. 개인회생을 선택하지 않으면 저절로 personal_rehabilitation_yn은 0이 되고 personal_rehabilitation_complete_yn도 0이 된다.

Personal_rehabilitation_yn이 0이면 납입완료(1)이 뜰 수 없기에 오류라고 판단되어 is_applied가 nan값이 아닌 것을 확인하고 제거했다. 나머지는 0이나 nan값을 갖는데 nan값을 0으로 채운다.

Personal_rehabilitation_yn과 personal_rehabilitation_complete_yn 둘 다 nan값을 가지는 경우가 있지만 이는 단순 데이터 누락이라 생각하여 각각 0으로 채운다.



개인회생여부 - 아니요(0) , 납입완료여부 - 예 (1)

```python
#개인회생여부 - 0 , 납입완료여부 - 1
condition1 = (loan.personal_rehabilitation_yn == 0) & (loan.personal_rehabilitation_complete_yn == 1) & (loan.is_applied.isna())
print(loan[condition1]) #0-1인 상황에서 test데이터에는 없다.
condition2 = (loan.personal_rehabilitation_yn == 0) & (loan.personal_rehabilitation_complete_yn == 1)
print(len(loan[condition2])) #이러한 오류가 16개만 가짐--> 더욱 제거 가능
```

Python

```
Empty DataFrame
Columns: [application_id, user_id, age, credit_score, yearly_income, income_type, employment_type, houseown_type, desired_amount,
purpose, personal_rehabilitation_yn, personal_rehabilitation_complete_yn, existing_loan_cnt, existing_loan_amt, product_id,
loan_limit, loan_rate, is_applied]
Index: []
16
```

```python
remove = loan[(loan['personal_rehabilitation_yn'] == 0) & (loan['personal_rehabilitation_complete_yn'] == 1)].index
loan = loan.drop(remove)
#0-1제거
```

개인회생여부 - 0 , 납입완료여부 - NaN ; 개인회생여부 - NaN, 납입완료여부 -NaN

```
#개인회생여부 - 0 , 납입완료여부 - nan ; 개인회생여부 - nan , 납입완료여부 - nan 외 nan을 0으로 채우기
loan['personal_rehabilitation_yn'] = loan['personal_rehabilitation_yn'].fillna(0)
loan['personal_rehabilitation_complete_yn'] = loan['personal_rehabilitation_complete_yn'].fillna(0)
```

개인회생여부 (Personal_rehabilitation_yn)와 납입완료여부 (Personal_rehabilitation_complete_yn)의 결측치 처리 완료

```
print(loan.isna().sum())
```

```
application_id                       0
user_id                              0
age                             128096
credit_score                   1509275
yearly_income                        6
income_type                          0
employment_type                      0
houseown_type                        0
desired_amount                       0
purpose                              0
personal_rehabilitation_yn           0
personal_rehabilitation_complete_yn  0
existing_loan_cnt              2685708
existing_loan_amt              3890152
product_id                           0
loan_limit                        7382
loan_rate                         7382
is_applied                     3257239
dtype: int64
```

(2)누락된 데이터 다른 행에서 찾기

age, credit_score, yearly_income, existing_loan_cnt, existing_loan_amt, loan_limits, loan_rates에 대해 NaN 값을 가진 행과 동일한 user_id를 가진 행에서 해당 열의 값이 존재한다면, 그 값으로 결측치를 대체하는 방법을 수행하였다.

문제발견

age의 결측치 데이터를 확인하던 중 한 가지 현상을 발견했다. 바로 user의 한 application_id에 age 값이 누락된 반면, 다른 application_id에 age값이 채워져 있다는 것이다.

```
#문제발견
## age nan데이터 중 user_id가 이제 다른 데이터에서 값이 존재하는지 확인
loan1 = loan.copy()
condition1 = loan1.age.isna()
age_null_user = loan1[condition1].user_id.unique()
print(age_null_user) #age에서 nan값을 가지는 user_id
print(len(age_null_user)) #age에서 nan값을 가지는 user_id의 수: 5696명
#그중 첫번쩨: user_id가 49072인 사람이 가지는 age값을 출력
age_list = loan1['age'].loc[loan1['user_id']==49072].unique()
print(age_list) #그 결과 37이라는 값이 존재
#따라서 다른 모든 nan데이터들도 다른 곳에서 값을 가지면 그 값으로 채움
```

```
[ 49072  77317 364670 ... 433763 397914 501037]
5696
[nan 37.]
```

처리

따라서 다른 application_id에서 가지는 값으로 nan값을 채우면 될 것이다. 채운 결과, 원래 128096개 결측치에서 108227개까지 줄였다. 그러면 이 108227개를 다른 방법으로 채우면 될 것이다.

```
#1.age의 nan값을 가진 user_id가 다른 application_id에서 age값을 가지면 그 값으로 nan값을 채울 수 있다.
condition1 = loan1.age.isna()
print(len(loan1[condition1].user_id)) #age의 nan값 수 = 128096
age_null_user = loan1[condition1].user_id.unique()
print(len(age_null_user)) #user_id의 갯수: 5696명
for user1 in age_null_user:
    age_list = loan1['age'].loc[loan1['user_id']==user1].unique()
    if len(age_list) == 2: #만약 age값을 2개 가진다면, 하나는 nan, 하나는 값이기에 nan을 나머지 값으로 채운다
        loan1['age'].loc[loan1['user_id']==user1] = loan1['age'].loc[loan1['user_id']==user1].fillna(age_list[~np.isnan(age_list)

print(loan1['age'].isna().sum()) # 바꾼뒤 다시 age의 결측치 수량을 측정, 108227으로 감소됨
condition2 = loan1.age.isna()
print(len(loan1[condition2].user_id.unique())) #갯수를 세어본 결과 총 4954 명이다. 따라서 이 4954명의 age값을 따로 채워주면 된다
```

Python

```
128096
5696

C:\Users\김지원\AppData\Local\Temp\ipykernel_9976\1638112197.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  loan1['age'].loc[loan1['user_id']==user1] = loan1['age'].loc[loan1['user_id']==user1].fillna(age_list[~np.isnan(age_list)][0])

108227
4954
```

credit_score, yearly_income, existing_loan_cnt, existing_loan_amt, loan_limit, loan_rate 도 똑같이 처리

credit_score: 원래 1509275개 결측치에서 1311772개까지 줄였다. (2개 이상, 평균값으로 채움)

yearly_income: 원래 6개 결측치에서 0개까지 다 채웠다.

existing_loan_cnt: 원래 2685708개 결측치에서 2685708개로 하나도 못채웠다.

existing_loan_amt: 원래 3890152개 결측치에서 3890152개로 하나도 못채웠다.

loan_limit: 원래 7382개 결측치에서 7005개까지 줄였다.

loan_rate: 원래 7382개 결측치에서 7063개까지 줄였다.

(3) 다중대입법 (Multiple Imputation)

가능한 대체 값의 분포에서 추출된 서로 다른 값으로 결측치를 처리
한 복수의 데이터셋을 생성한 뒤, 이들 데이터셋에서 대하여
각각 분석을 수행하고, 그 결과 얻은 모수의 추정량과 표본오차를 통
합하여 하나의 분석 결과를 제시하는 방법

요약:
--결측치를 제외한 나머지 변수들을 이용해 해당 결측치를 예측한다.
--이것을 여러 번 반복을 통해 신뢰성 있는 대체 값을 넣어준다.
--Sklearn.impute.IterativeImputer를 사용한다.

MI(다중대입법)--sklearn.impute.IterativeImputer

현재 이러한 추정방식은 아직 실험단계이기에 depreciation cycle없이 예측 및 API가 변경될 수 있어 먼저 sklearn.experimental에서 enable_iterative_imputer를 import하고, sklearn.impute에서 Iterative Imputer를 import한다.

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
```

그리고 데이터에서 우선 nan값 예측에 도움을 주지 않는 application_id, user_id, desired_amount, personal_rehabilitation_yn, personal_rehabilitation_complete_yn을 제거하고, 범주형 자료인 income_type, employment_type, houseown_type, purpose를 제거하고, is_applied의 nan값은 예측해야 하는 값이기에 다중대체법으로 nan값을 채우면 안되기에 제거한다.

```
unfilled_data = loan1.copy()
#다중대입법에 도움을 주지 못하는 index, user_id나 application_id, 변주형 변수를 제거하고, is_applied는 test_data이기에 다중대입법으로 채
unfilled_data = unfilled_data.drop(['Unnamed: 0','application_id','user_id','income_type','employment_type','houseown_type','desi
```

IterativeImputer 함수는 다음 변수들이 존재한다.

estimator: 추정방법 defalt: 베이지안

max_iter: 최종라운드에 채운 값들을 return하기 전에 실행할 라운드 수, 조기종료 가능 defalt: 10

tol: 정지조건의 허용 오차 defalt: 1e-3

imputation_order: feature가 impute하는 순서 roman: 왼쪽에서 오른쪽

skip_complete: 학습데이터에 값이 있으면 따로 추정하지 않고 그 값을 채움 defalt: False

verbose: 설명의 자세한 정도 defalt:0

random_state: random seed

```
imp = IterativeImputer(verbose =2, max_iter=20, tol= 1e-10,imputation_order='roman',skip_complete=True,random_state= 10)
```

FIT를 통해 데이터들을 학습한다

```
imp.fit(unfilled_data)
```

TRANSFORM을 통해 NAN값들을 채워준다

```
fill = imp.transform(unfilled_data)
```

결과로 나온 것은 ARRAY이기에 COLUMN명을 설정해 주고 DATAFRAME으로 바꾸어 준다

```
columns_names = ["age","credit_score","yearly_income","existing_loan_cnt","existing_loan_amt","product_id","loan_limit","loan_r
filled_data = pd.DataFrame(fill, columns= columns_names)
```

처음에 지웠던 COLUMN들을 다시 추가해 준다.

```
filled_data['Unnamed: 0'] = loan1['Unnamed: 0']
filled_data['application_id'] = loan1['application_id']
filled_data['user_id'] = loan1['user_id']
filled_data['income_type'] = loan1['income_type']
filled_data['employment_type'] = loan1['employment_type']
filled_data['houseown_type'] = loan1['houseown_type']
filled_data['desired_amount'] = loan1['desired_amount']
filled_data['purpose'] = loan1['purpose']
filled_data['personal_rehabilitation_yn'] = loan1['personal_rehabilitation_yn']
filled_data['personal_rehabilitation_complete_yn'] = loan1['personal_rehabilitation_complete_yn']
filled_data["is_applied"] = loan1["is_applied"]
```

column들을 다시 원래처럼 정리해 주고,
결측치를 확인해 보니 다 채워져 있는 것을 확인할 수 있다.

따라서 데이터 전처리는 다 완성하였다.

```
#보기 좋게 column들을 다시 원래처럼 정리해 준다.
filled_data = filled_data[['Unnamed: 0','application_id','us
print(filled_data.isna().sum())
#이제 결측치를 확인해 보니 다 채워져 있는 것을 확인할 수 있다.
```

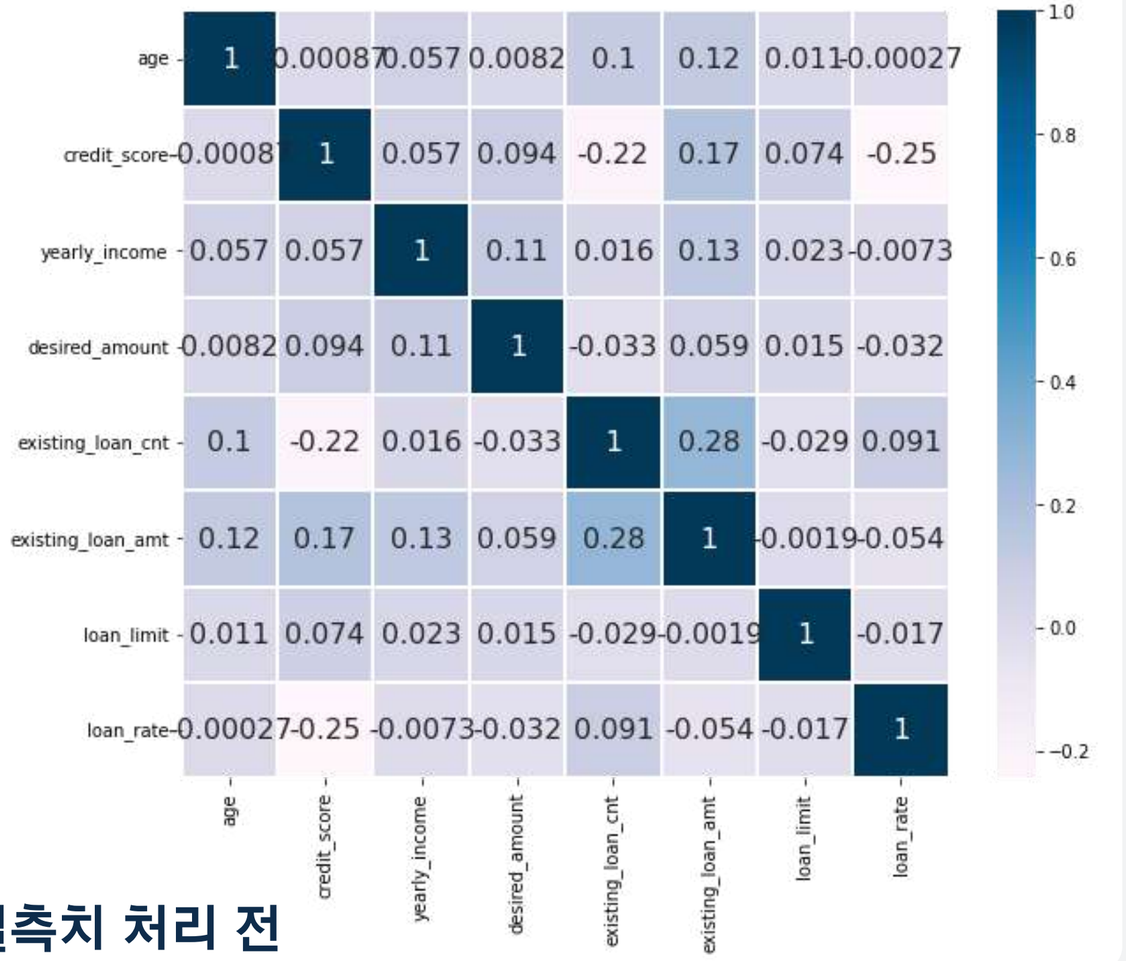
```
Unnamed: 0                              0
application_id                          0
user_id                                 0
age                                     0
credit_score                            0
yearly_income                           0
income_type                             0
employment_type                         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
product_id                              0
loan_limit                              0
loan_rate                               0
is_applied                        3257239
dtype: int64
```

EDA

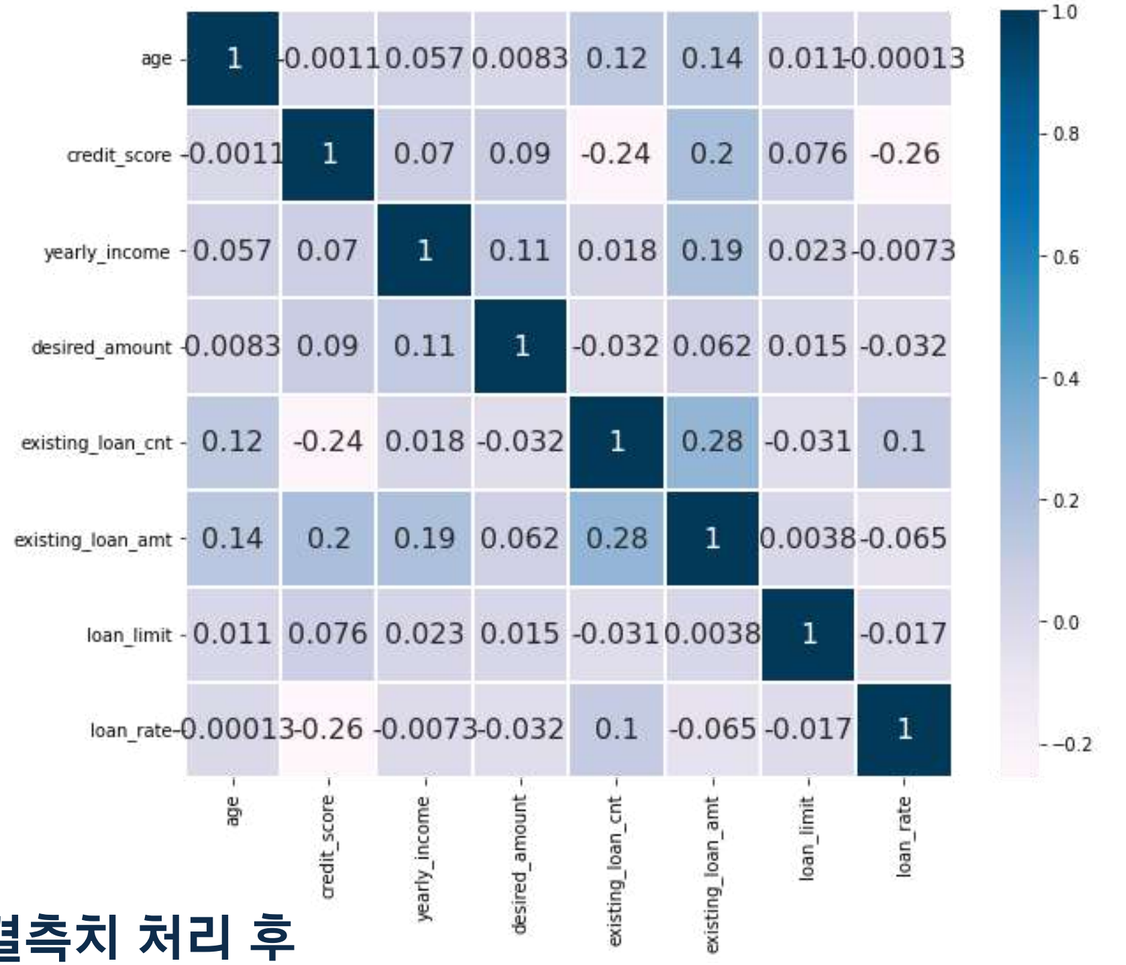
EDA 목표

- EDA를 통해 주어진 user_spec, loan_result, log_data의 데이터를 탐색해보면서 전체적인 데이터의 구조를 파악한다.
- 데이터에 담긴 많은 변수들이 서로 어떤 관계가 있는지 탐색해본다.
- 타겟 변수인 is_applied와 각 변수별 관계를 시각화를 통해 분석해본다.
- EDA를 통해 주어진 많은 양의 데이터에 대한 이해도를 높이고, 변수의 의미를 파악하는 것에 중점을 둔다.
- 각 문제에서 요구되는 목표를 달성하기 위한 변수 선택 및 적절한 통계 모형을 EDA 결과를 참고하여 선택한다.

결측치 처리 전 VS 결측치 처리 후



결측치 처리 전



결측치 처리 후

- 결측치 처리 전과 결측치 처리 후의 변수 간 상관관계가 큰 차이가 없다.
- 따라서 선택한 결측치 처리 방법이 좋은 방법이라고 할 수 있다!

VIF: 결측치 처리 전

	VIF Factor	features
0	0.829180	age
1	0.432026	credit_score
2	1.033645	yearly_income
3	1.021013	desired_amount
4	1.109474	existing_loan_cnt
5	1.157090	existing_loan_amt
6	1.004319	loan_limit
7	0.765226	loan_rate

결측치 처리 전

- VIF를 통해 변수 간 상관관계가 있는지 확인한다.
- 선택한 컬럼: age, credit_score, yearly_income, desired_amount, existing_loan_cnt, existing_loan_amt, loan_limit, loan_rate (범주형 변수 및 이분변수 컬럼은 코드 실행을 위해 제외)
- VIF가 10이 넘으면 다중공선성 문제가 있다고 판단한다. VIF 결과를 보면 각 변수별 VIF Factor가 모두 낮은 수치를 나타내어 변수 간 다중공선성 문제는 없다고 판단하였다.

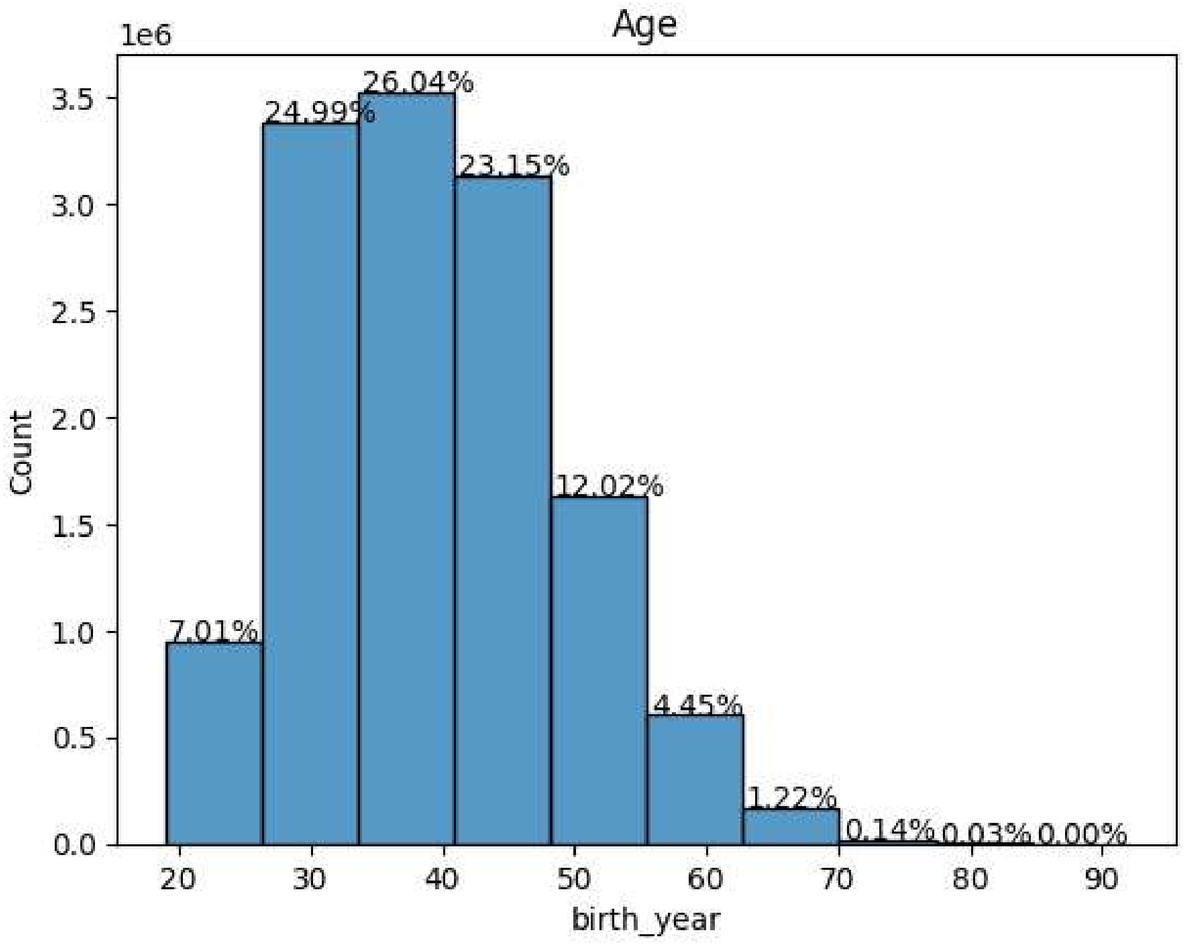
VIF: 결측치 처리 후

	VIF Factor	features
0	0.846550	age
1	0.405027	credit_score
2	1.049346	yearly_income
3	1.020755	desired_amount
4	1.127592	existing_loan_cnt
5	1.198426	existing_loan_amt
6	1.006239	loan_limit
7	0.780801	loan_rate

결측치 처리 후

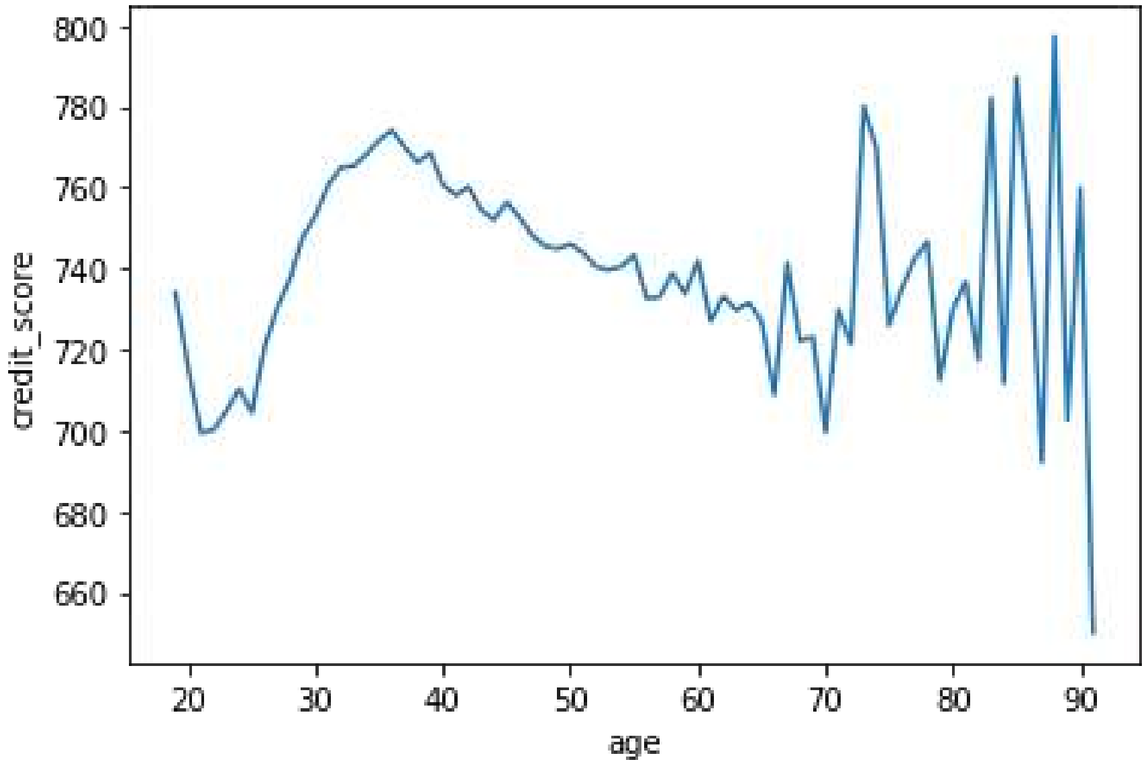
- 결측치 처리 후의 VIF를 실행하였다.
- 선택한 컬럼: age, credit_score, yearly_income, desired_amount, existing_loan_cnt, existing_loan_amt, loan_limit, loan_rate (범주형 변수 및 이분변수 컬럼은 코드 실행을 위해 제외)
- VIF 결과를 보면 각 변수별 VIF Factor가 모두 낮은 수치를 나타내어 변수 간 다중공선성 문제는 없다고 판단하였다.

Age



- user_spec과 loan_result를 merge한 데이터에서, 19XX와 같이 표현된 birth_year 컬럼을 2022년을 기준으로 계산을 통해 현재 나이를 구하여 birth_year컬럼의 값을 대치하였다.
- 각 나이대가 전체 데이터에서 얼마만큼의 비율을 차지하는지 시각화를 통해 살펴보았다.
- 40대 > 30대 > 50대가 가장 높은 비율을 보였다.

Age



- age를 기준으로 groupby를 한 후 각 나이별 credit_score의 평균을 시각화하였다.
- 20대에서 신용점수가 낮고, 30대에서 다시 신용 점수가 높아졌다가, 70대 전까지 감소하는 경향을 살펴볼 수 있었다.
- 70대 이후로는 신용점수 평균이 나이마다 차이가 크게 나는 것으로 보인다.

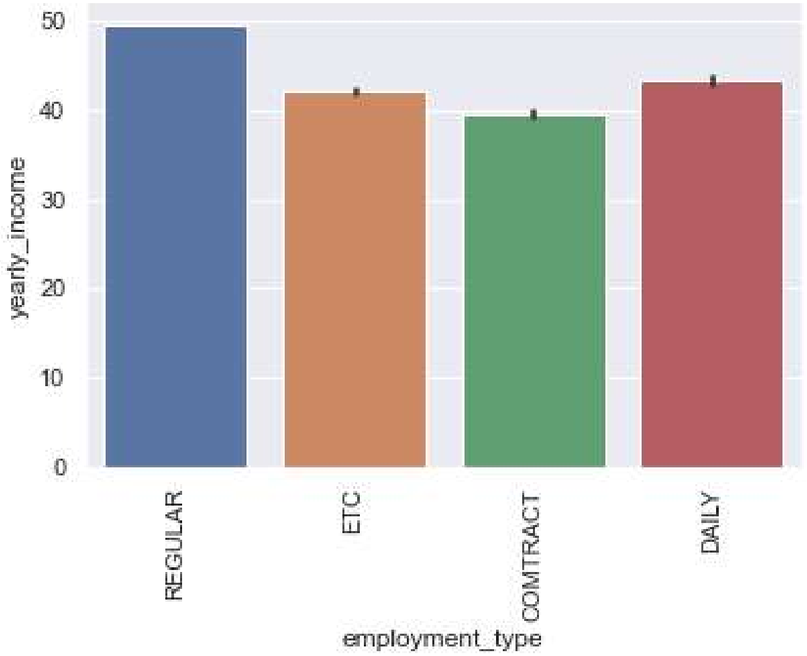
결측치 처리 전 변수 간 관계 파악

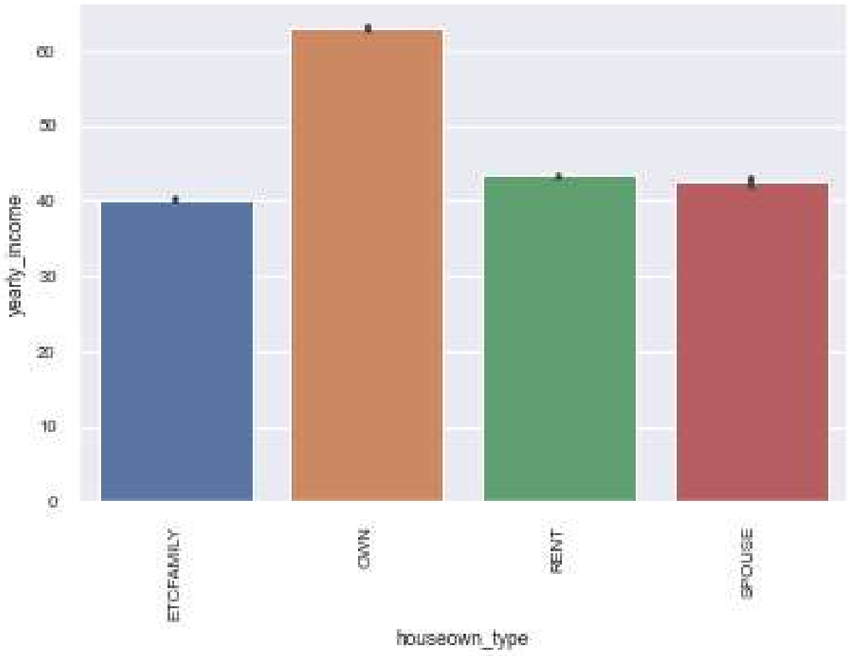
```
data.dtypes
```

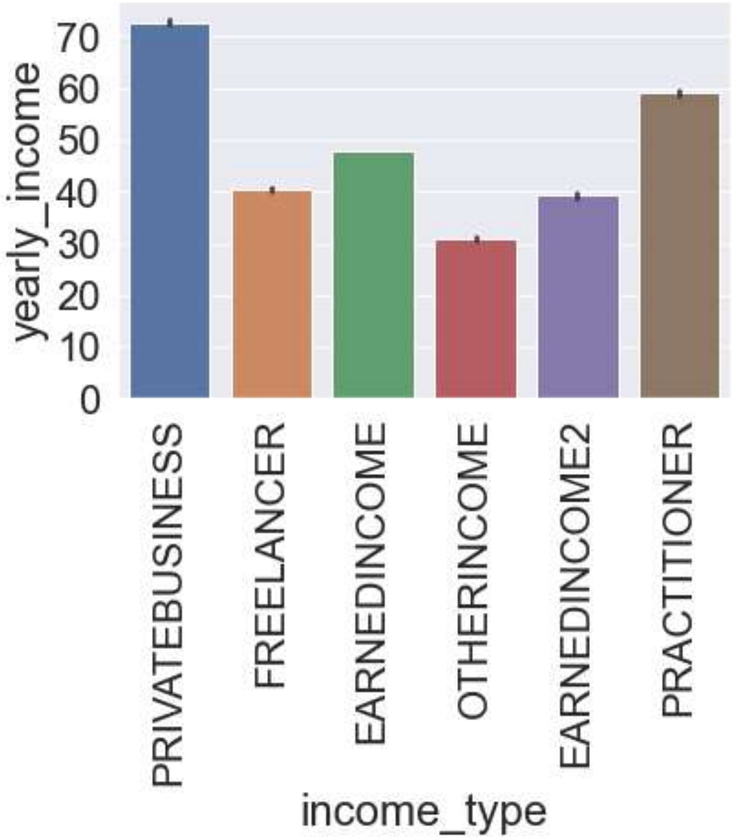
```
Unnamed: 0                          int64
application_id                      int64
user_id                             int64
age                               float64
credit_score                      float64
yearly_income                     float64
income_type                        object
employment_type                    object
houseown_type                      object
desired_amount                    float64
purpose                            object
personal_rehabilitation_yn        float64
personal_rehabilitation_complete_yn    float64
existing_loan_cnt                 float64
existing_loan_amt                 float64
product_id                          int64
loan_limit                        float64
loan_rate                         float64
is_applied                        float64
```

- user_spec과 loan_result를 merge한 데이터에서 object 타입에 해당하는 것은 income_type, employment_type, houseown_type, purpose 변수로 확인되었다.
- 이후 범주형 변수들과 다른 컬럼의 관계를 조사하였다.

변수 간 관계 파악 (1): yearly_income을 중심으로

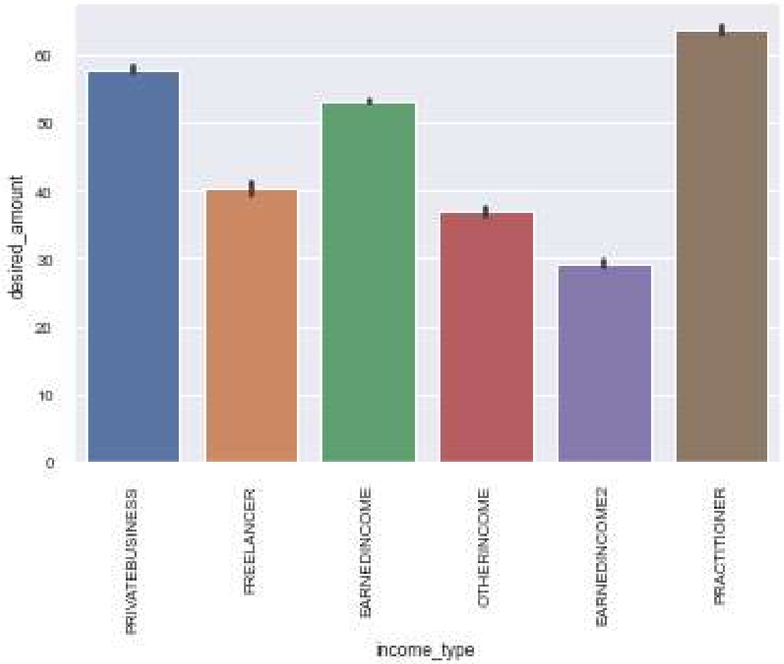


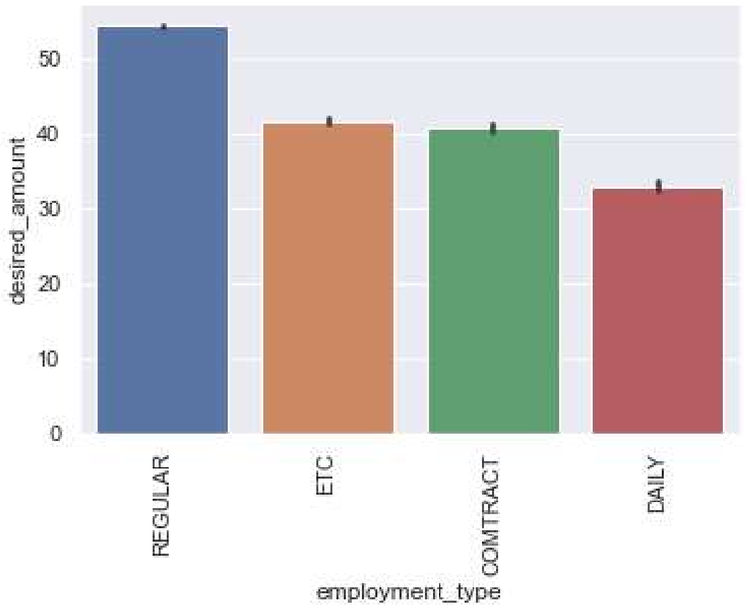


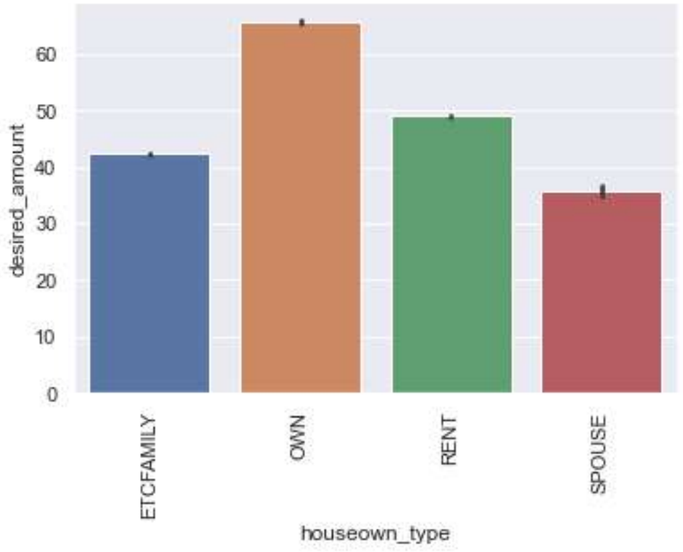


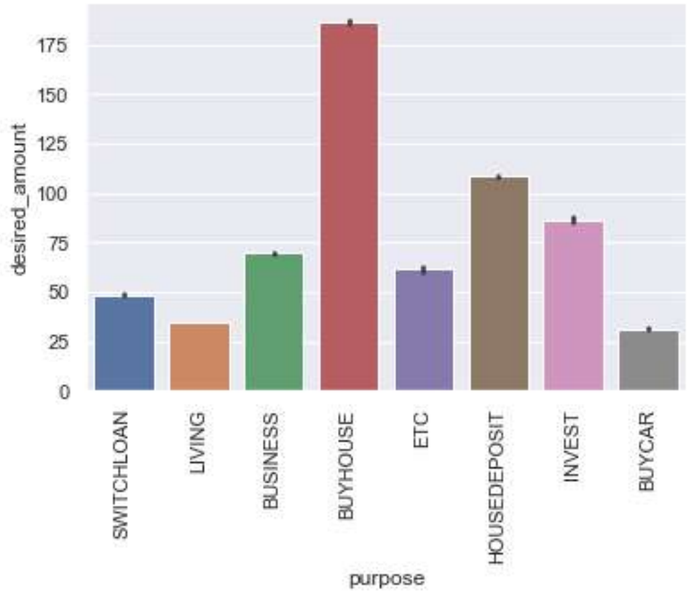
1. employment_type: REGULAR(정규직)에서 평균적으로 yearly_income이 가장 높았다.
2. houseown_type: OWN(자가)에서 평균적으로 yearly_income이 가장 높았다.
3. income_type : PRIVATEBUSINESS에서 평균적으로 yearly_income이 가장 높았다.

변수 간 관계 파악 (2): desired_amount를 중심으로



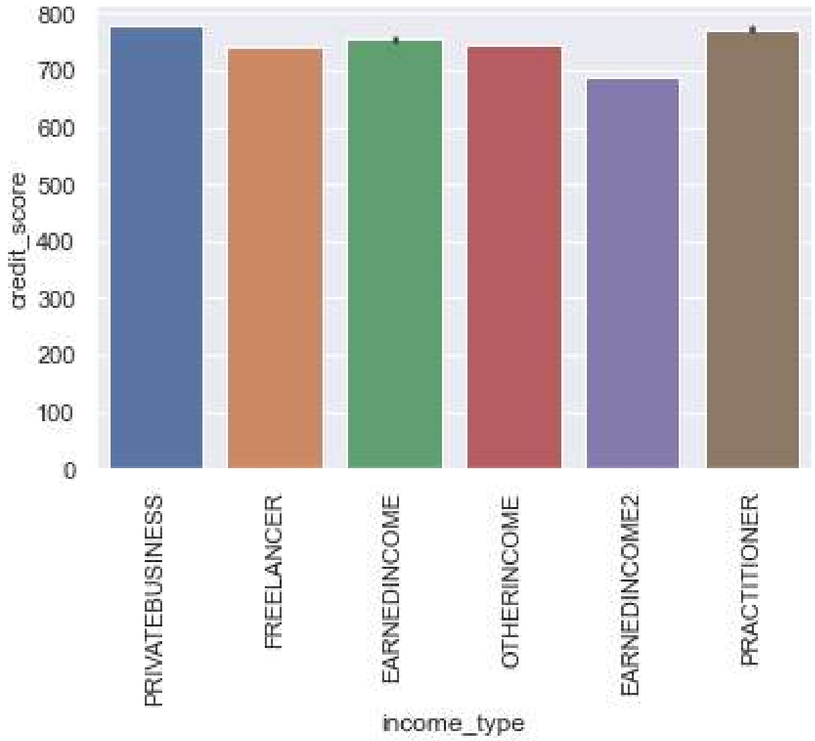


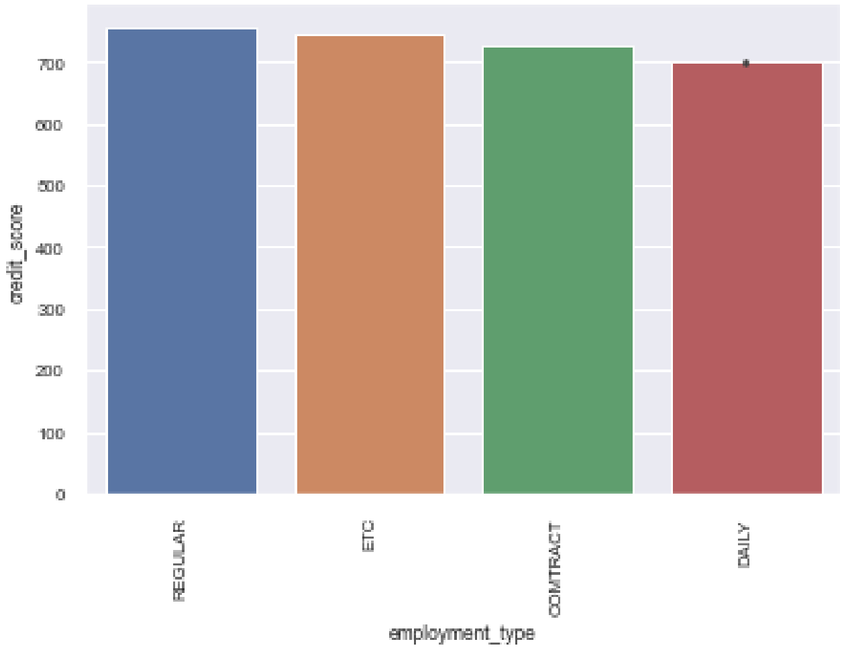


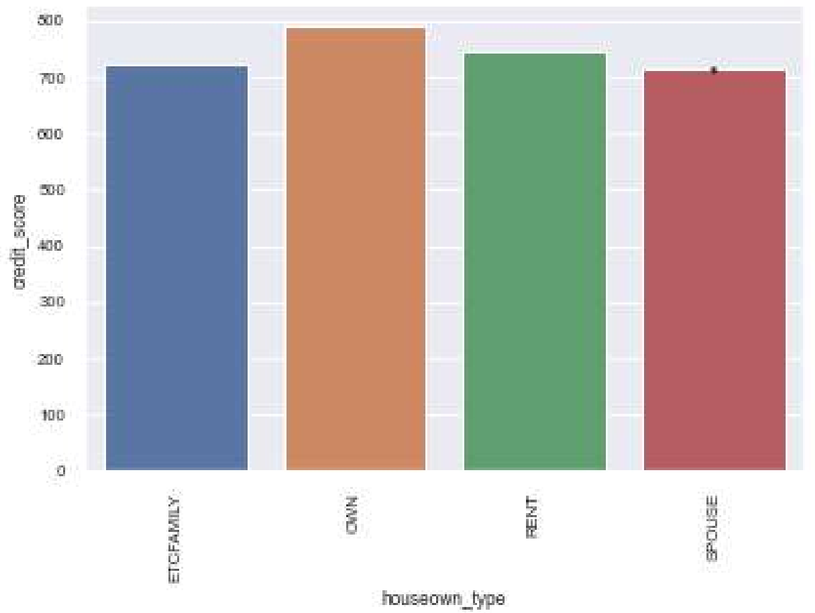


1. income_type: desired_amount는 평균적으로 PRACTITIONER에서 높았다.
2. employment_type: desired_amount는 평균적으로 REGULAR(정규직)에서 높았다.
3. houseown_type: desired_amount는 평균적으로 OWN(자가)에서 높았다.
4. puspose: desired_amount는 평균적으로 BUYHOUSE일 때 매우 높았다.

변수 간 관계 파악 (3): credit_score를 중심으로



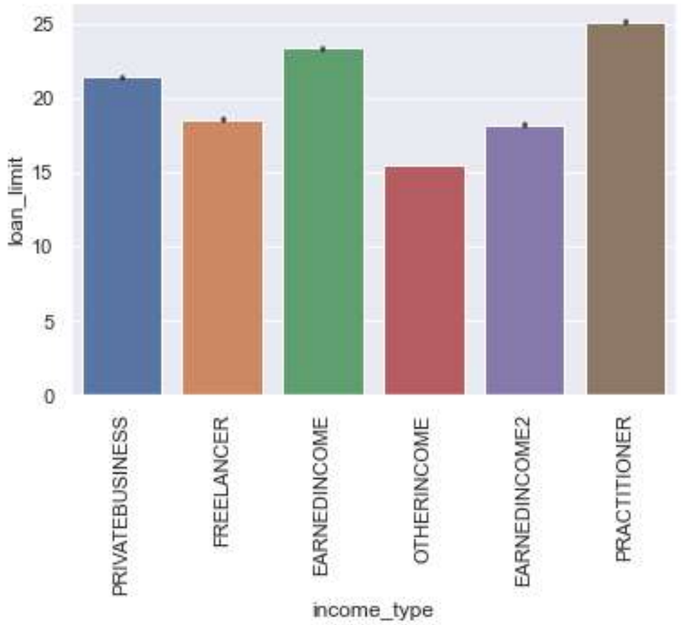


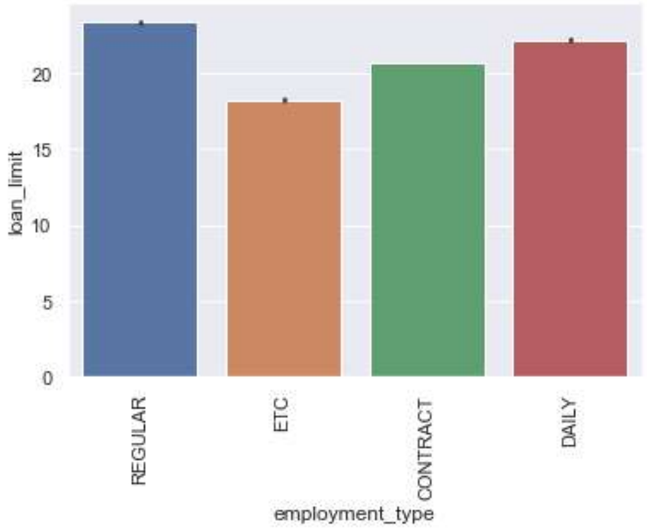


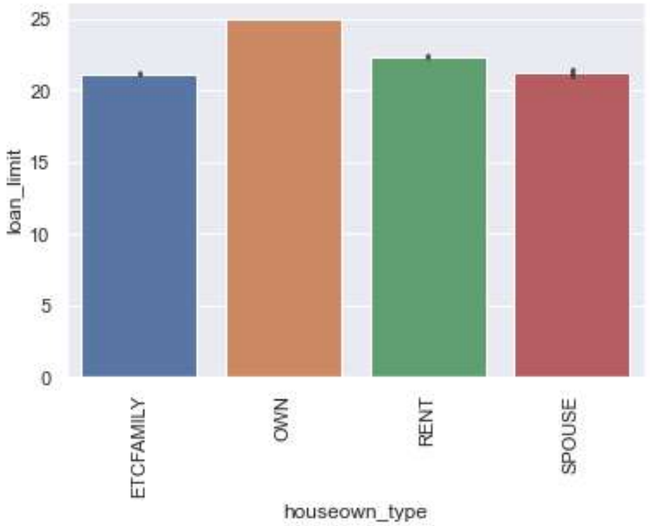
1. income_type: credit_score는 평균적으로 PRACTITIONER에서 높았다.
2. employment_type: credit_score는 평균적으로 REGULAR(정규직)에서 높았다.
3. houseown_type: credit_score는 평균적으로 OWN(자가)에서 높았다.

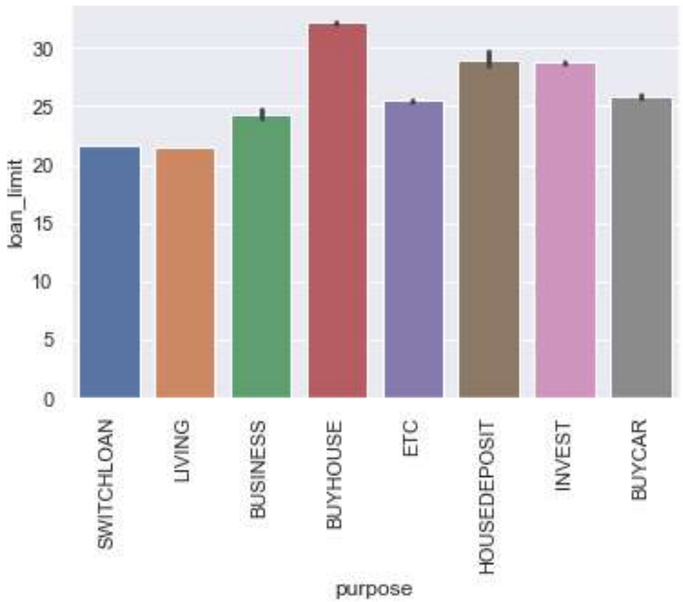
- 1, 2, 3번의 그래프 모두 큰 차이 없이 평균 700대로 비슷했다.

변수 간 관계 파악 (4): loan_limit을 중심으로





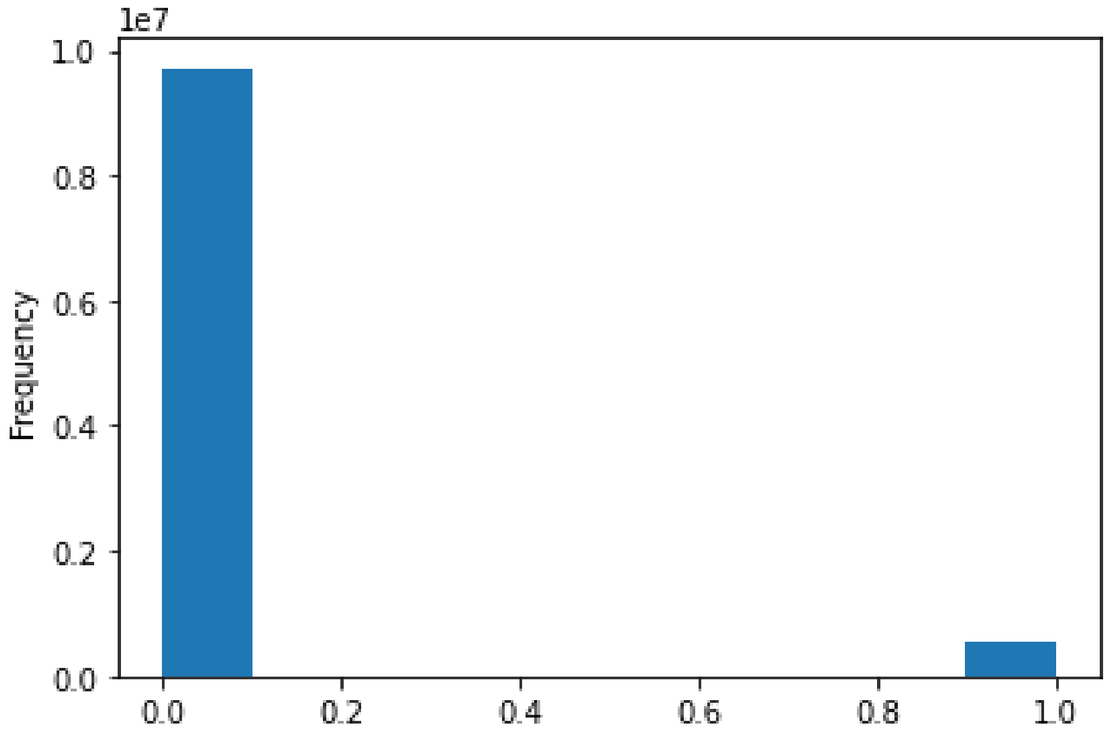




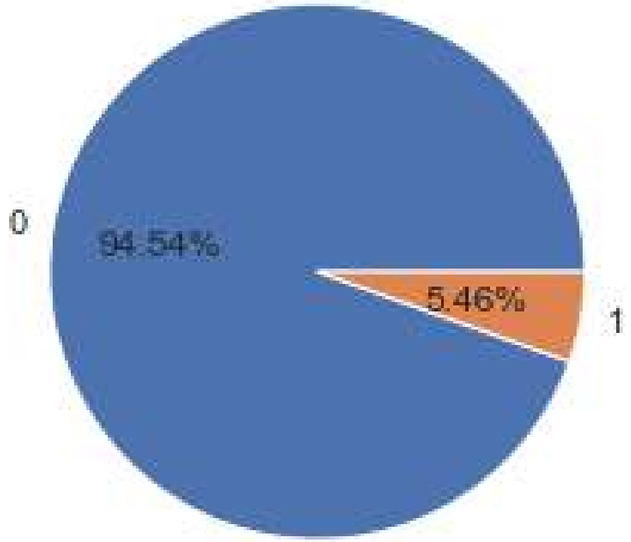
1. income_type: loan_limit은 평균적으로 PRACTITIONER에서 높았다.
2. employment_type: loan_limit은 평균적으로 REGULAR(정규직)에서 높았다.
3. houseown_type: loan_limit은 평균적으로 OWN(자가)에서 높았다.
4. purpose: loan_limit은 평균적으로 BUYHOUSE에서 높았다.

- 범주형 변수들에 따라 평균을 비교해본 결과 모든 연속형 변수(yearly_income, credit_score, desired_amount, loan_limit)에서 정규직, 자가, BUYHOUSE가 가장 높은 수치를 보였다.

is_applied 컬럼 살펴보기



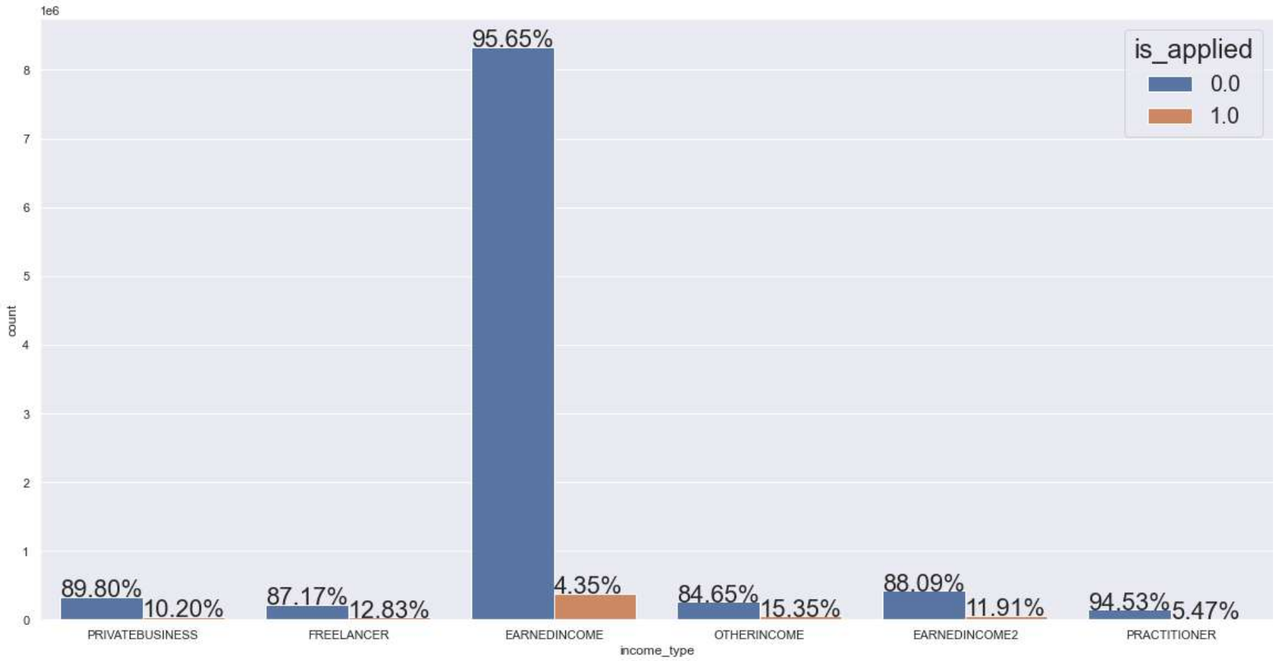
loan_result is_applied 컬럼의 0과 1 개수



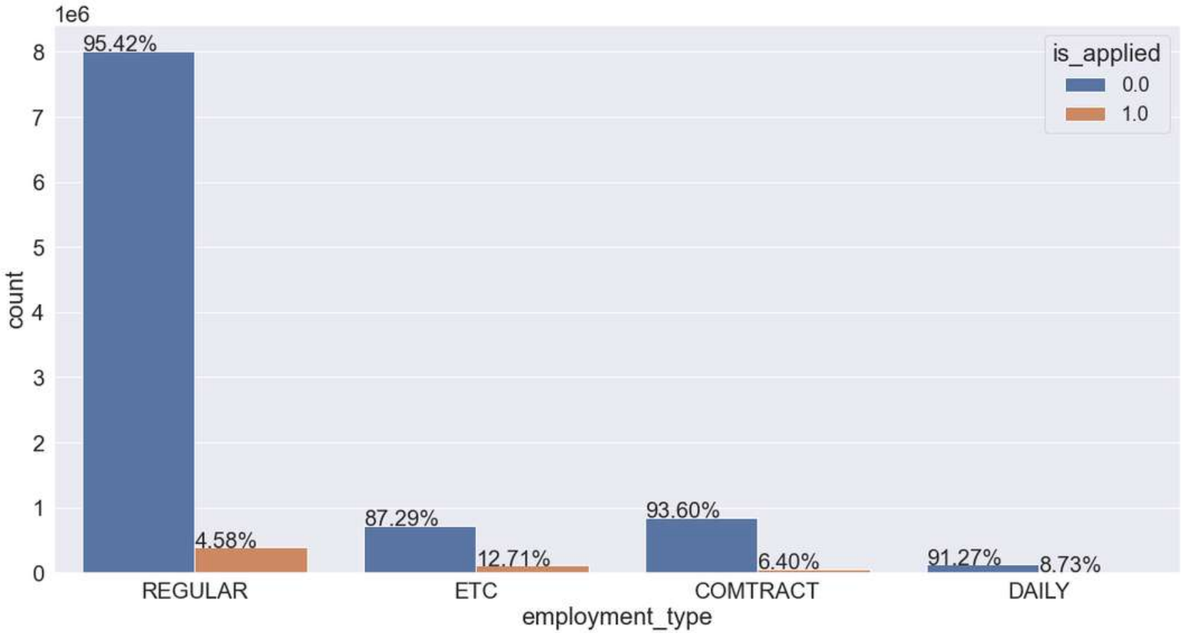
결측치 처리 전 user_spec과 loan_result를 merge 한 데이터에서 is_applied 컬럼의 0과 1 비율

- is_applied 컬럼의 0과 1의 개수를 시각화하면, 1에 비해 0이 훨씬 더 많은 빈도를 나타내고 있다.
- 타겟 변수인 is_applied의 0과 1의 비율이 큰 차이가 난다는 특징을 유의하며 향후 과정을 진행한다.

변수와 is_applied의 관계 (1)

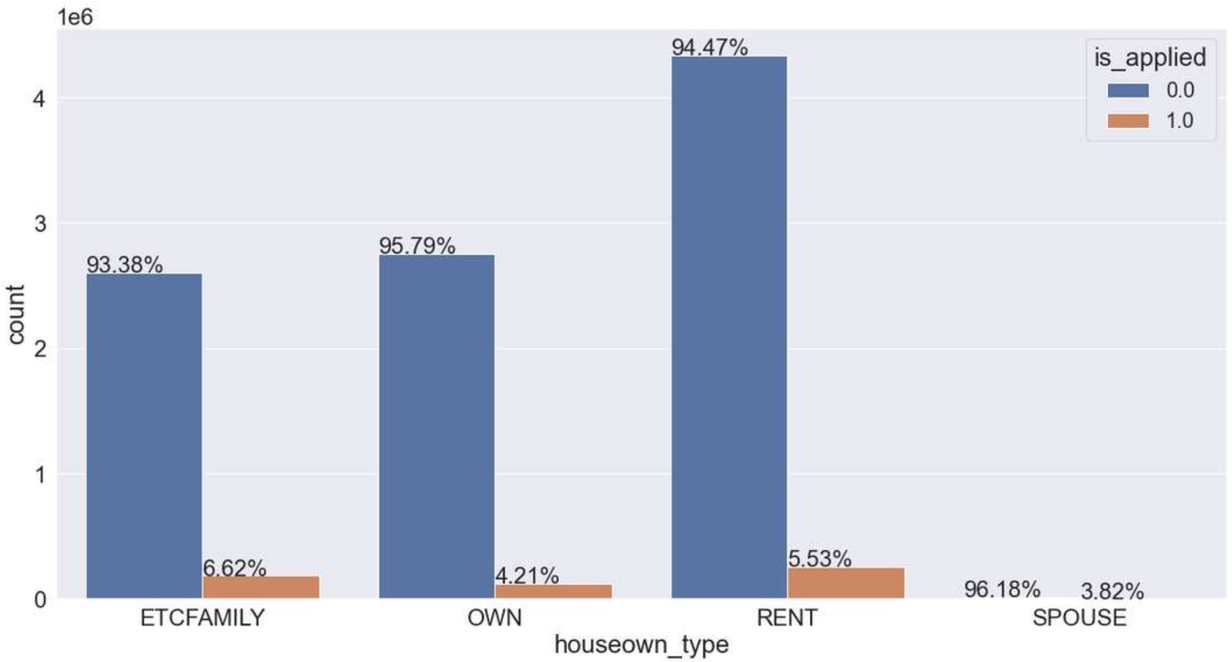


- user_spec과 loan_result를 결측치 처리 전 inner로 merge한 후 살펴보았다.
- income_type과 is_applied의 관계를 확인하였다.
- EARNEDINCOME의 개수가 가장 많았다.
- is_applied가 1인 비율은 OTHERINCOME에서 가장 높았다.

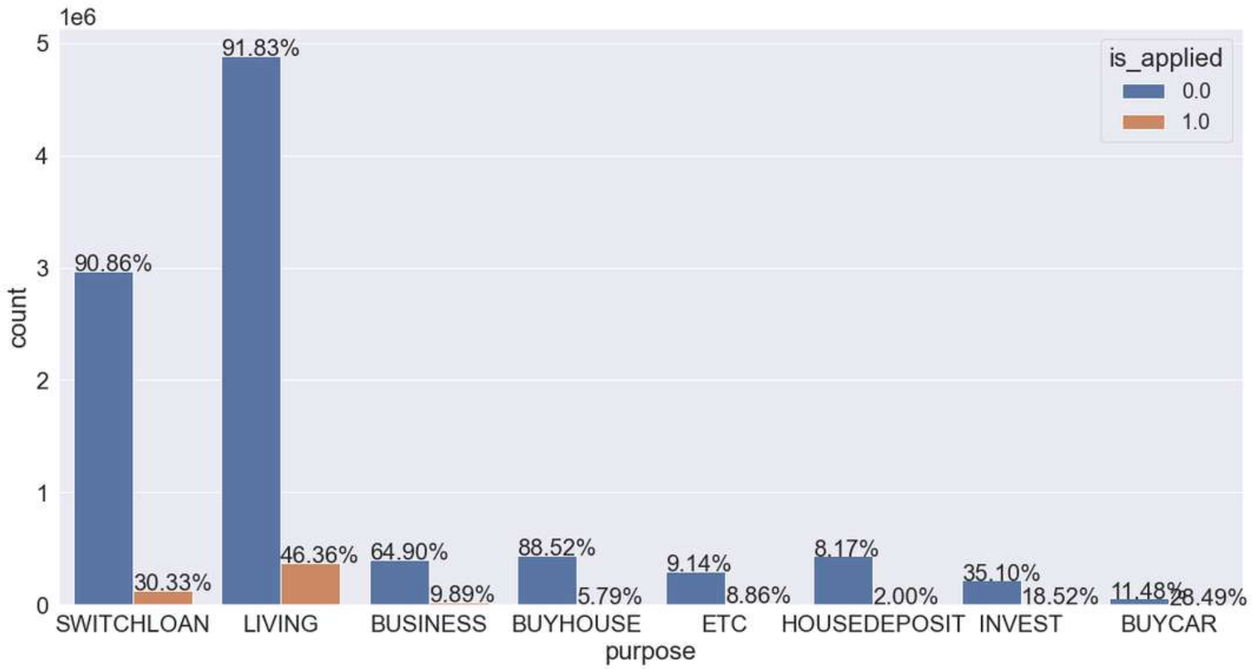


- employment_type별 is_applied의 관계를 확인하였다.
- REGULAR(정규직)의 개수가 가장 많았다.
- is_applied가 1인 비율은 ETC(기타)에서 가장 높았다.

변수와 is_applied의 관계 (2)

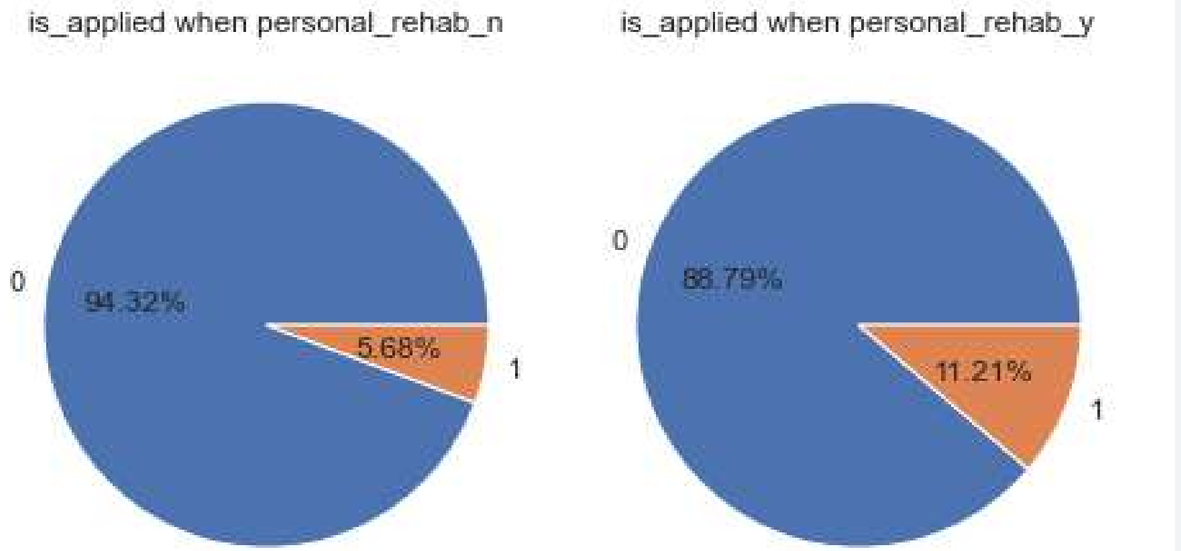


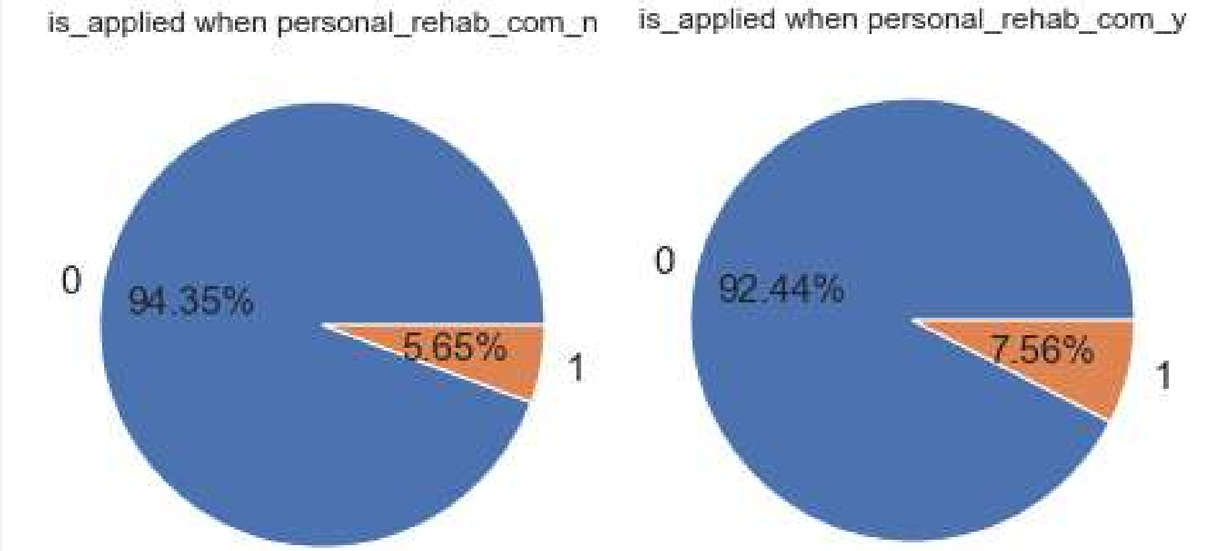
- houseown_type과 is_applied의 관계를 확인하였다.
- RENT(전월세) 개수가 가장 많았다.
- ETCFAMILY(기타 가족)에서 is_applied가 1인 비율이 가장 높았다.



- purpose와 is_applied의 관계를 확인하였다.
- LIVING(생활비)의 개수가 가장 많았다.
- is_applied가 1인 비율은 LIVING에서 가장 높았다.

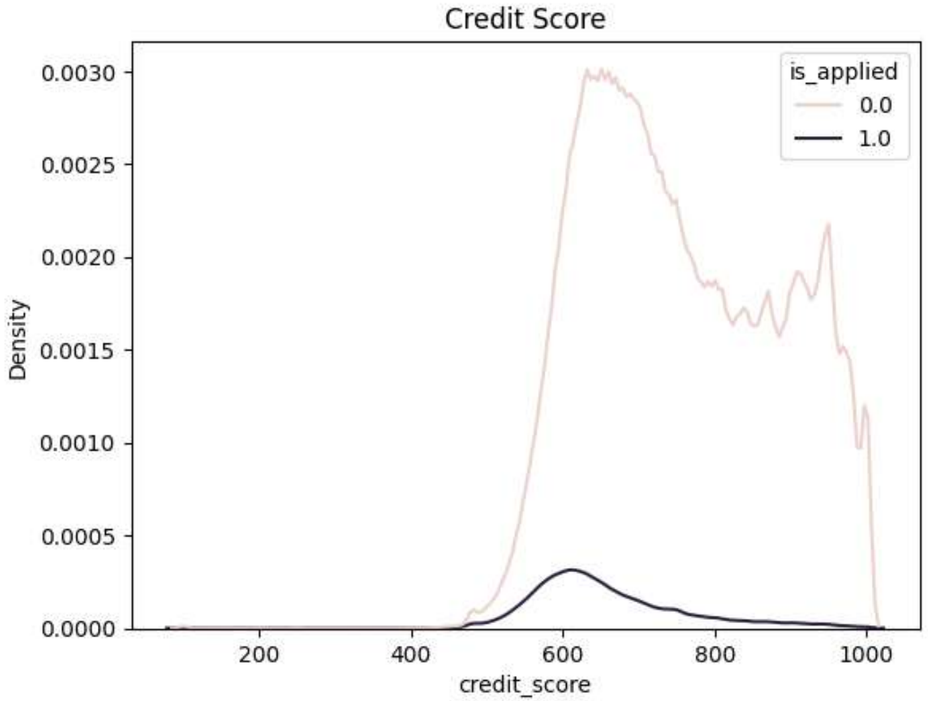
변수와 is_applied의 관계 (3)

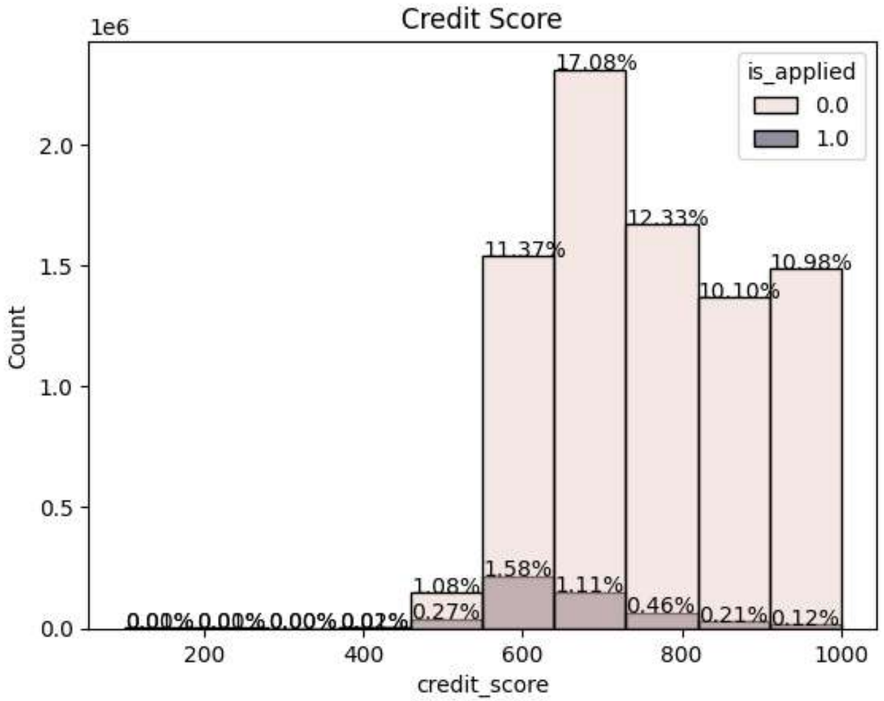




1. 개인회생 여부가 0 또는1일 때, 각각 is_applied에서 0과 1의 비율이 어떻게 되는지 알아보았다.
    - [rehab_n = 개인회생 여부가 0]인 경우보다 [rehab_y = 개인회생 여부가 1]일 때 is_applied가 1인 비율이 더 높았다.
2. 마찬가지로 개인회생완료 여부가 0 또는1일 때, 각각 is_applied에서 0과 1의 비율이 어떻게 되는지 알아보았다.
    - [rehab_com_n = 개인회생완료 여부가 0]인 경우보다 [rehab_com_y = 개인회생완료 여부가 1]일 때 is_applied가 1인 비율이 더 높았다.

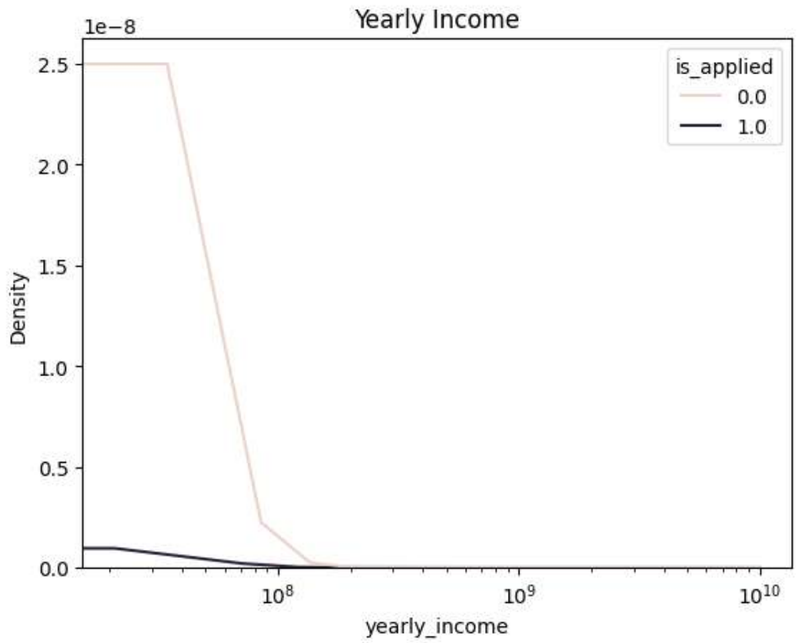
변수와 is_applied의 관계 (4)

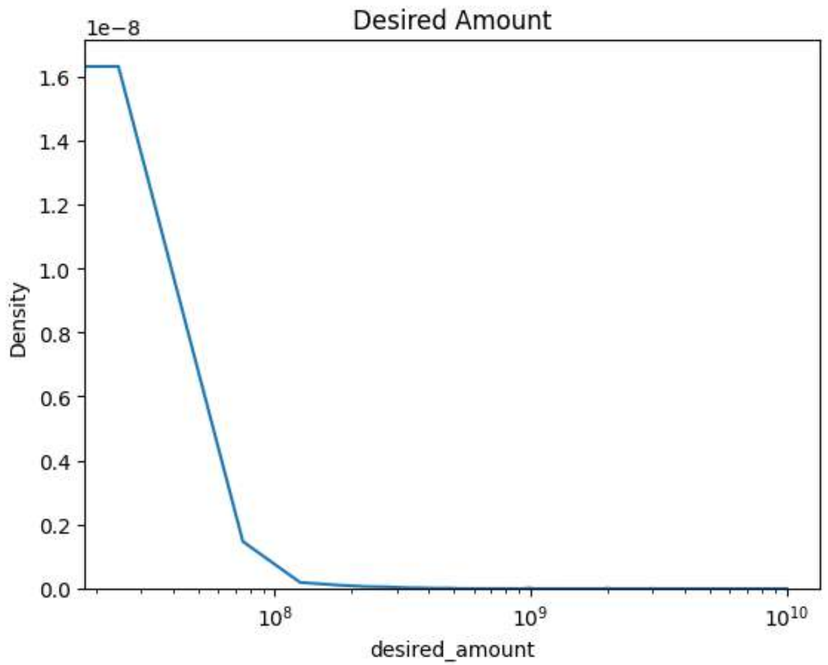


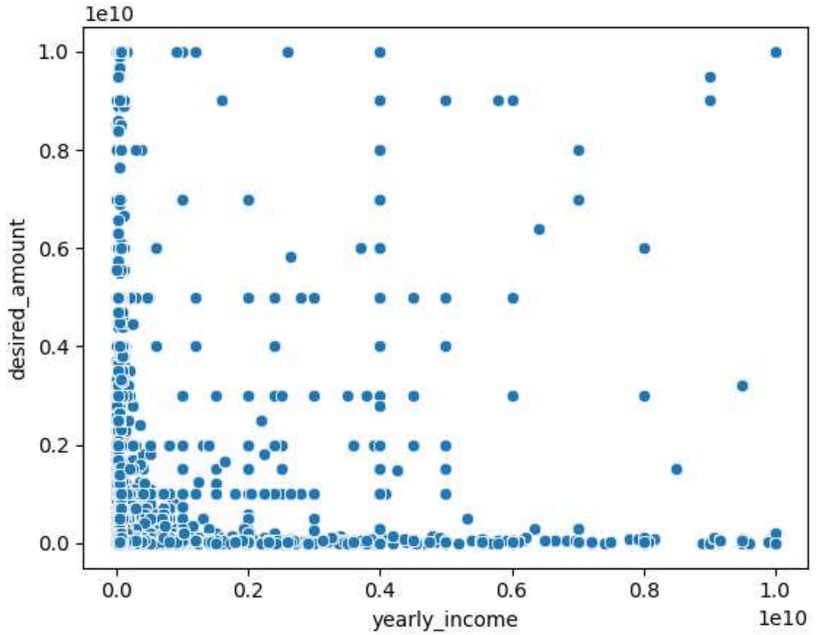


- credit_score의 분포와 is_applied 분포를 함께 시각화하였다.
- 시각화 결과를 보면, is_applied가 0일 때는 대략 700점대 정도에서 credit_score의 비율이 가장 높다는 것을 알 수 있었다.
- is_applied가 1일 때는 대략 600점대 정도에서 credit_score가 가장 높은 비율을 보였다.

변수와 is_applied의 관계 (5)







- 첫 번째 그래프는 yearly_income과 is_applied, 두 번째 그래프는 desired_amount 변수를 시각화하여 전체적인 분포를 살펴보았다.
- x축의 scale은 log로 지정하였다.
- yearly_income과 desired_amount의 관계를 산점도를 통해 시각화한 결과, 유의미한 패턴은 보이지 않았다.

EDA 결과

- EDA를 통해 데이터의 각 변수별 전반적인 특징, 결측치, 형태 등 기본적인 정보를 파악할 수 있었다.
- 결측치 처리 전 및 결측치 처리 후 데이터에서 변수 간 연관성이 존재하는지 상관관계 및 VIF를 통해 확인하였다. 유의미한 상관관계나 다중공선성 문제는 없는 것으로 보았다.
- 주어진 데이터의 is_applied 컬럼에서 0과 1이 얼마만큼 존재하는지 조사해 본 결과 0이 1보다 훨씬 많이 존재한다는 것을 확인하였다.
- 범주형 데이터에 속하는 것들과 is_applied 컬럼의 관계를 시각화하여 각 범주형 변수별로 is_applied가 1인 비율이 해당 컬럼의 어떤 유형에서 가장 높은 지 알아보았다.
- 개인회생 및 개인회생완료 컬럼과 is_applied의 관계를 살펴보았다. 개인회생 및 개인회생완료 컬럼별 회생 여부에 따른 is_applied의 0과 1의 비율을 구해보고, 회생 여부에 따른 비율이 얼마만큼 차이가 나는지 확인해보았다.
- 향후 진행되는 모델링 및 문제 해결 과정에서 is_applied 변수와 각 변수 간의 특징, user_spec 특징 등을 고려하여 어떤 변수들을 중점적으로 활용할지 결정한다.

예측 모델링

가. 예측 모델링의 목표

나. 변수 선택

다. 데이터 분리

라. Resampling

마. Preprocessing

바. Modeling

- choosing models & hyperparameters with Bayesian Optimization
- Voting Classification
- threshold

사. 결과 예측 및 분석

예측 모델링의 목표

본 데이터의 모델링 목표는 'loan_result.csv'의 is_applied열의 값(0 또는 1)을 채우는 것이다.
즉, 어플 사용자들의 개인정보 및 신용정보 등을 변수로 하여 대출신청여부를 판정하는 예측 모델을 만들고자 한다.
데이터의 특성과 모델 학습시 발생할 수 있는 문제점 등을 고려하여 변수 선택, 전처리, resampling, 모델 선택 및 최적화 방법을 선택할 것이다.

과정	변수 선택	데이터 분리	Resampling
Preprocessing	모델 학습, 튜닝	VotingClassifier	threshold 조절

변수 선택

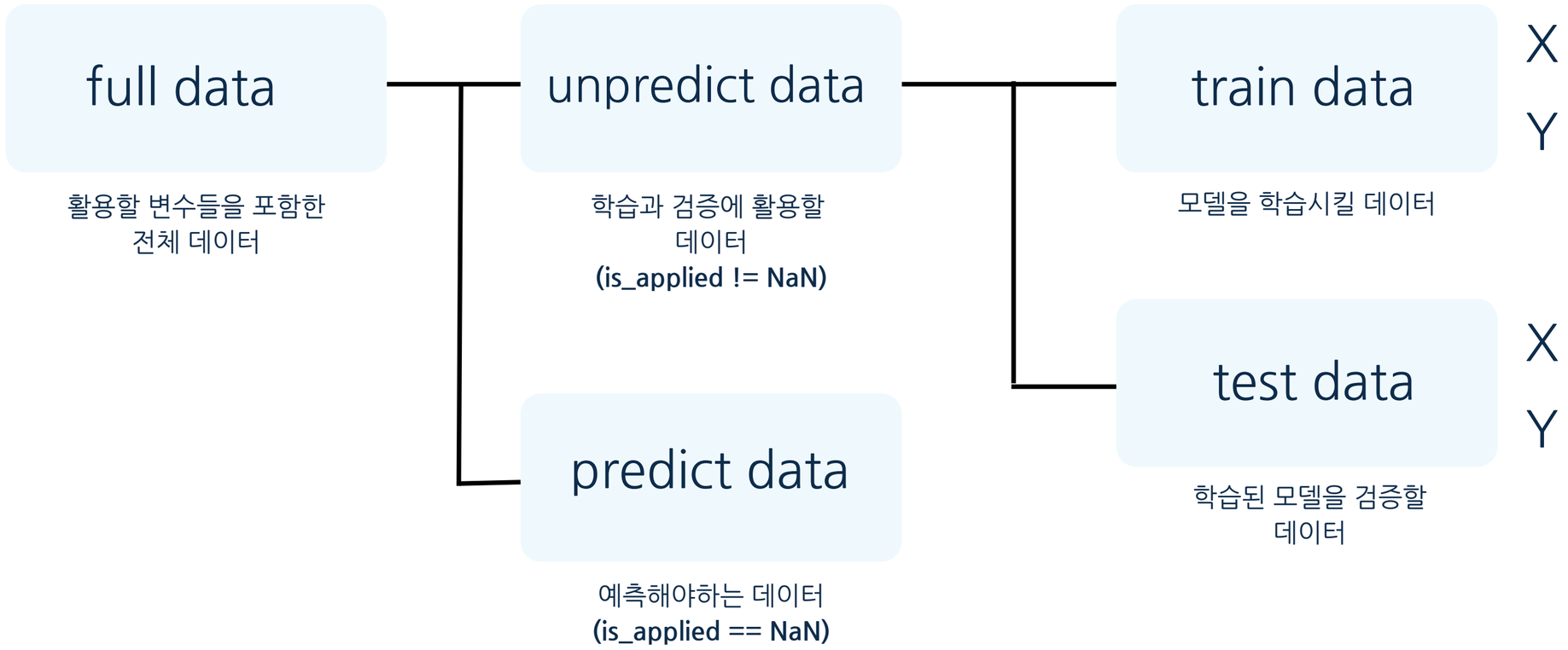
독립변수로 사용한 변수:

- credit_score
- desired_amount
- loan_limit
- loan_rate
- product_id
- (purpose 중) living

```
full_data['living'] = [1 if p == 'LIVING' else 0 for p in full_data['purpose']]
full_data = full_data.drop(labels='purpose',axis=1)
```

purpose 중 'living'
변수 처리 방식

데이터 분리하기


full data
활용할 변수들을 포함한
전체 데이터
unpredict data
학습과 검증에 활용할
데이터
(is_applied != NaN)
predict data
예측해야하는 데이터
(is_applied == NaN)
train data
X
Y
모델을 학습시킬 데이터
test data
X
Y
학습된 모델을 검증할
데이터

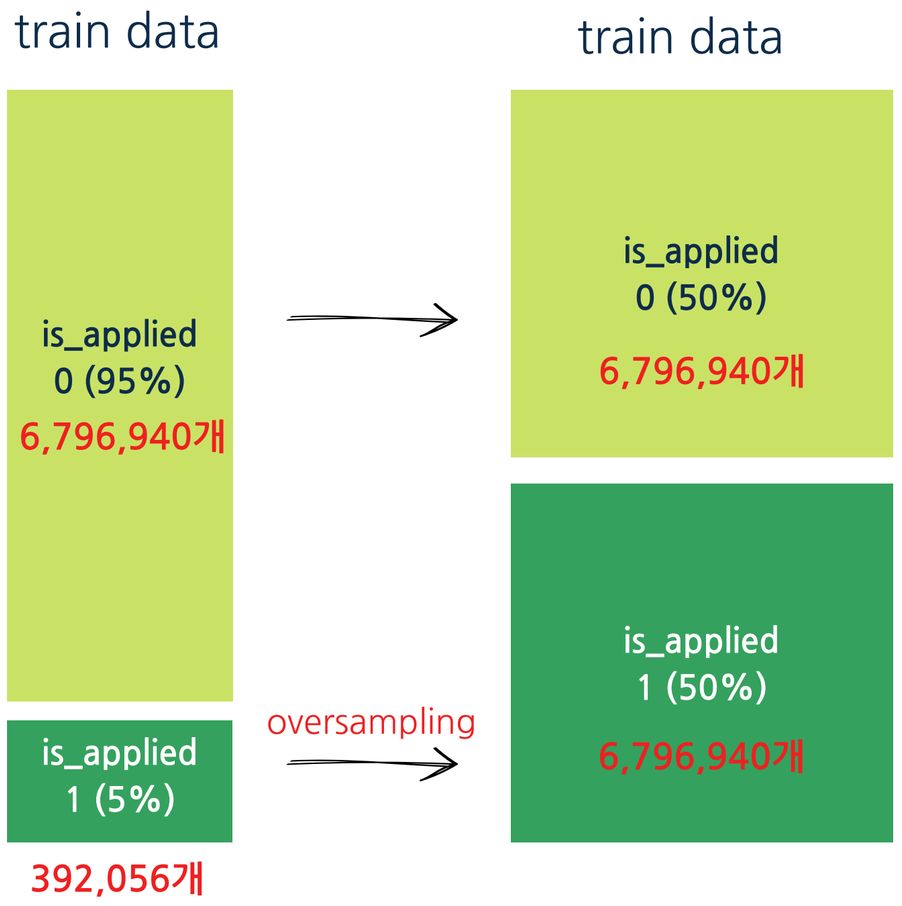
Resampling data

목적

train, test data에서 타겟 변수인 is_applied의 0, 1 비율이 약 95:5로, 한쪽으로 치우진 것을 알 수 있다.
이러한 imbalance data의 경우, 틀린 예측이 많더라도 accuracy는 높게 나오는 오류가 나타날 수 있다. 이와 같은 오류를 줄이기 위해 Oversampling을 통해 데이터의 비율을 조정하고자 한다.

대상

train data 에서 is_applied == 1



Resampling data

```
from sklearn.utils import resample

def up_resampling(small, num):
    resample_data = resample(small, replace=True, n_samples=num, random_state=42)
    return resample_data
```

- **up_resampling** 함수 정의:

개수가 적은 쪽의 데이터를 설정한 샘플 개수(n_samples)에 맞추어 resampling

```
num_sample = len(train_data_0)
resample_train_data = pd.concat([train_data_0, up_resampling(train_data_1,num_sample)])
resample_train_data = shuffle(resample_train_data)
print(len(resample_train_data))
```

- is_applied == 1인 데이터들의 개수가 적기 때문에, is_applied == 0 인 데이터 개수에 맞게 oversampling 진행하여 데이터의 비율을 50:50으로 맞춤

Preprocessing

- **Categorical Encoding**

목적

데이터 내의 범주형 변수를 기계가 이해할 수 있는 숫자의 형태로 바꾼다. 더미변수에 0과 1을 할당한다.

대상

product_id
(숫자 형식의 변수지만, 그 숫자에 의미가 있지 않은 범주형 변수이기 때문에 encoding)

방법: binary encoding

범주형 변수의 값들을 정수로 변환하고,
정수를 이진수로 나타낸 후,
이진수의 자릿수를 분리해 각 컬럼으로 만드는 인코딩 방법.

log_2 (카테고리수) 만큼의 컬럼이 생성된다.

-> product_id: 177개
-> binary encoding 후 product_id_0 ~ product_id_7까지 8개의 열로 인코딩됨.

=> one-hot encoding 등 다른 방식보다 생성되는 컬럼수가 적고, 변수 특성에 적합하여 binary encoding을 선택함.

Preprocessing

- **Scaling**

목적

숫자형 변수들이 변수마다 사이즈가 다르면 모델이 잘못된 결과를 도출할 수 있기 때문에,
Scaler를 이용해 숫자의 범위를 일정하게 맞춘다.

대상

숫자형 변수: 'credit_score','loan_limit','loan_rate','desired_amount'

방법: StandardScaler

평균이 0 , 분산이 1인 정규분포로 표준화시키는 Scaler

모델 학습

개요

머신러닝 기법인 앙상블 모델 Classifier를 활용해 타겟 변수를 예측하였다.
VotingClassifier에 대입할 모델을 선정하기 위해, 다양한 분류 모델을 설정해 데이터를 학습시켰다.
각 모델별로 적절한 hyperparameter를 찾기 위해, Bayesian Optimization 기법을 활용했다.
F1 score가 가장 높게 나오는 모델과 각 모델에 대입할 hyperparameter를 선정한 후,
VotingClassifier에 적용시켜 타겟 변수의 각 범주에 속할 확률을 계산하는 과정을 기반으로 최종적인 모델을 학습시켰다.
마지막으로, 최종모델에 대해 0과 1을 판별할 threshold를 조절하여, F1 score가 어떻게 변하는지 관찰한 후 적절한 threshold를 선정했다.

Classifier 후보 선정

Classifier 학습 및 하이퍼파라미터 튜닝

VotingClassifier

threshold 조정

Training models with Bayesian Optimization

Bayesian Optimization

Hyperparameter optimization기법으로써의 Bayesian optimization은
어느 입력값을 받는 함수를 최댓값으로 만드는 최적의 입력값을 찾는 것을 목적으로 한다.

목적 함수와 하이퍼파라미터 쌍을 대상으로 대체 모델을 만들고, 순차적으로 하이퍼파라미터를 업데이트하여 최적의 조합을 탐색하는 방법이다.

데이터 관측치를 반복적으로 관찰하면서 사후확률분포가 개선되고 결국 목적 함수의 최대값을 반환하는 조합을 반환하는데, Grid search나 Random search 같은 방법들보다 소요 시간이 짧다.

아래 모델들을 포함해 10개 가량의 분류 모델을 default hyperparameter로 학습시켜보고,
그 중 성능이 양호한 모델들을 대상으로
하이퍼파라미터 최적화를 진행했다.

탐색 모델

LGBMClassifier, XGBoostClassifier,
ExtraTreesClassifier, BaggingClassifier
HistGradientBoostingClassifier,
RandomForestClassifier

-> **굵은 글씨인 모델들**은 최적화 결과 가장 좋은 성능을 보인 모델들이며, F1 score 최대 0.24~0.28 의 성능을 보였다.

Training models with Bayesian Optimization

- **Example**

```
# LightGBMClassifier
from lightgbm import LGBMClassifier

def LGBM_cv(num_leaves, max_depth, learning_rate, n_estimators, min_child_samples):
    lgbm = LGBMClassifier(
        num_leaves = int(num_leaves),
        max_depth = int(max_depth),
        learning_rate = learning_rate,
        n_estimators = int(n_estimators),
        min_child_samples = int(min_child_samples)
    )

    lgbm.fit(resample_X, resample_Y)

    y_pred= lgbm.predict(X_test)
    f1 = f1_score(Y_test, y_pred)

    return f1
```

최적화할 모델의 함수 정의:
조정할 hyperparameter들을 설정한 모델을 세팅하고,
train data를 대입해 학습시킨 후
test data를 대입해 예측값을 구하고,
예측 결과에 대한 모델의 F1 score를 반환하는 함수

```
lgbm_pbounds = {
    'num_leaves' : (20,50),
    'max_depth' : (3,12),
    'learning_rate' : (0.01, 0.3),
    'n_estimators' : (50,150),
    'min_child_samples' : (12,30)
}

lgbm_bo=BayesianOptimization(f=LGBM_cv, pbounds=lgbm_pbounds, verbose=2, random_state=1)

lgbm_bo.maximize(n_iter=10, acq='ei', xi=0.01)

print(lgbm_bo.max)
```

최적화할 하이퍼파라미터 구간 설정

BayesianOptimization method 실행:
bayesian optimization으로 해당 모델의 F1 score를
최대화시키는 하이퍼파라미터의 조합 찾기

VotingClassifier

Voting Classifier

앙상블 모델 중 하나로, 다양한 기본 모델을 학습시킨 후 각 모델의 결과를 집계하는 것을 기반으로 예측 결과를 도출하는 분류 모델.

- Hard voting: 모델들의 결과를 집계해 가장 표를 많이 얻은 class를 최종 예측값으로 결정
- Soft voting: 기본 모델들이 class의 확률을 예측할 수 있을 때, 예측을 평균내어 최종 예측값 결정

앞선 최적화과정으로 선정한 모델들과 하이퍼 파라미터 값들을 사용해 기본 모델을 설정하고, VotingClassifier에 대입하여 hard voting, soft voting을 진행하고 학습시킴.

(LGBMClassifier, XGBClassifier, HistGradientClassifier 사용)

```
# hard voting
eclf1 = VotingClassifier(estimators=[
    ('lgbm', clf1), ('xgb', clf2), ('hgb', clf3)], voting='hard')
eclf1 = eclf1.fit(resample_X, resample_Y)
y_pred = eclf1.predict(X_test)
```

```
# soft voting
eclf2 = VotingClassifier(estimators=[
    ('lgbm', clf1), ('xgb', clf2), ('hgb', clf3)], voting='soft')
eclf2 = eclf2.fit(resample_X, resample_Y)
y_pred = eclf2.predict(X_test)
```

hard, soft 각각 0.256, 0.277 가량의 F1 score 도출

Threshold 조정

Threshold

이진 분류 상황에서, 0과 1 중 어느 것인지 판단하는 임계값으로, 확률로 표현된다.
default = 0.5 인 상황에서는, predict_proba(1로 판단할 확률)이 0.5 이상일 때 1값을 예측한다.
threshold를 낮추면 recall이 커지고, threshold를 높이면 precision이 커진다.

활용 방법

- train을 완료한 모델에 대해 test data의 predict_proba를 계산하고, 이 확률을 설정한 threshold와 비교해 최종적으로 반환할 binary 값을 도출한다.

- 다양한 threshold 값에 대해 classification_report를 확인하여 F1 score, precision, recall의 변화를 확인한다.

적용되는 부분

1. Voting Classifier에 대입하기 전, 개별 모델에 적용해 F1 score의 최댓값 파악
2. Soft Voting Classifier 모델에 적용해 최종적으로 반환되는 F1 score 결정

적용

- threshold 조정 전 precision과 recall을 확인해보았을 때, 주로 recall이 precision에 비해 높게 나왔다. 따라서 threshold를 0.5 보다 크게 설정하여 precision을 늘리는 방향으로 적용해보았다.

Threshold 조정

- **Example**

```
y_proba = eclf2.predict_proba(X_test)[:,1]
for threshold in np.arange(0.5, 0.8, 0.05):
  y_proba_th = y_proba >= threshold
  print(threshold)
  print(classification_report(Y_test, y_proba_th))
  print("-"*50)
```

```
0.7500000000000002
              precision    recall  f1-score   support

         0.0       0.97      0.94      0.95   2912720
         1.0       0.28      0.43      0.34    168279

    accuracy                           0.91   3080999
   macro avg       0.62      0.68      0.65   3080999
weighted avg       0.93      0.91      0.92   3080999
```

학습한 voting classifier 모델에서
0.5 ~ 0.8 까지 0.05단위로 threshold 설정하여
성능 변화 관찰
predict_proba > threshold 일 때 예측값 1

threshold = 0.75일 때
f1-score = 0.34로 가장 높았음.
threshold가 0.75보다 작을 때 ~ 0.75일 때까지
f1 score 증가했다가 0.75를 넘어가면 다시 감소

```
confusion_matrix(Y_test, y_proba_th)
```

```
array([[2795951,  116769],
       [ 112466,   55813]])
```

결괏값 예측 및 분석

학습을 완료한 최종 모델을 바탕으로, 앞선 데이터 분리 과정에서 분리한 predict data (is_applied == NaN) 의 is_applied를 예측했다.

예측한 is_applied 0, 1 비율

```
print(len(merge_fin.loc[merge_fin['is_applied'] == 1]))
print(len(merge_fin.loc[merge_fin['is_applied'] == 0]))
```

```
327821
2927373
```

군집 분석

가. 군집분석의 목표

나. 데이터 처리

다. K-MEANS++

라. PCA를 이용한 시각화

마. 통계 기법을 이용한 결과 분석

군집분석의 목표

user_spec에 있는 데이터의 특징에 따라 유저별로 앱 안에서 사용하고자 하는 서비스가 상이할 것이다. 그래서 데이터를 활용해 군집분석을 실시하고, 그 결과를 통계적으로 분석해 각 군집에 알맞는 서비스를 제공하도록 창의적인 메시지를 전달하고자 한다.

데이터 처리

log_data.csv 에서는 [user_id, event] 변수를 가져온다. event에 존재하는 [CompleteIDCertification, EndLoanApply, GetCreditInfo, Login, OpenApp, SignUp, StartLoanApply, UseDSRCalc, UseLoanManage, UsePrepayCalc, ViewLoanApplyIntro]를 one-hot-encoding을 통해 각각 하나의 변수로 만들어 준다.

groupby('user_id').sum()을 통해 각 user_id마다 각각의 서비스를 총 몇 회 사용하였는지 구한다.

user_spec.csv 에 존재하는 ['user_id','age','credit_score','yearly_income','desired_amount','income_type','houseown_type','purpose','employment_type'] 변수들을 불러오고, 범주형 변수들은 one-hot-encoding을 실시한다.

groupby('user_id').mean()을 통해 각 user_id마다 평균을 이용해 하나의 case로 만들어준다.

그리고 두 log와 user 데이터를 user_id를 기준으로 통합한다.

군집 모델

먼저 모델에 변수를 넣기 전에 scaling과정을 거치고, KMeans++ 모델을 사용하여 총 3개의 군집을 분류했다.

결과로는 0,1,2번 군집이 다음과 같이 나왔다.

0번 군집 : 34347개
1번 군집 : 790개
2번 군집 : 55개

```
# 스케일링 과정을 거쳐준다.
numerical_preprocessor = StandardScaler()
robust_preprocessor = RobustScaler()

preprocessor = ColumnTransformer([
    ('standard_scaler', numerical_preprocessor, numeric_col),
    ('robust_scaler', robust_preprocessor, robust_col)
    ], remainder = 'passthrough')

col_names = merge.columns
X = preprocessor.fit_transform(merge.iloc[:,1:])
X = pd.DataFrame(X, columns=col_names[1:])
```

```
# KMeans clustering을 사용하고, 군집은 총 3개로 분리한다.
# random_state=1234 는 군집 결과의 재생산성을 위해 바꾸지 않는다.
kmeans = KMeans(n_clusters=3, random_state=1234)
kmeans.fit(X)
clusters = kmeans.predict(X)

# 각 군집의 정보를 데이터프레임에 추가해준다.
X["Cluster"] = clusters
```

시각화

군집화 결과를 시각화 하기 전에 데이터에 존재하는 변수가 많기 때문에 먼저 PCA를 통해 차원축소를 시켜준다. 그 후 PC들을 이용해 좌표로 시각화 한다.

아래의 코드 결과를 보면 3개의 PCs로 전체 데이터의 약89% 정도를 설명할 수 있으므로, PCA를 이용한 시각화 방법이 사용 가능하다고 할 수 있다.

```
#PCA with one principal component
pca_1d = PCA(n_components=1)
#PCA with two principal components
pca_2d = PCA(n_components=2)
#PCA with three principal components
pca_3d = PCA(n_components=3)
```

```
# 1D PCA 학습
PCs_1d = pd.DataFrame(pca_1d.fit_transform(X.drop(["Cluster"], axis=1)))

# 2D PCA 학습
PCs_2d = pd.DataFrame(pca_2d.fit_transform(X.drop(["Cluster"], axis=1)))

# 3D PCA 학습
PCs_3d = pd.DataFrame(pca_3d.fit_transform(X.drop(["Cluster"], axis=1)))

PCs_1d.columns = ["PC1_1d"]
PCs_2d.columns = ["PC1_2d", "PC2_2d"]
PCs_3d.columns = ["PC1_3d", "PC2_3d", "PC3_3d"]
```

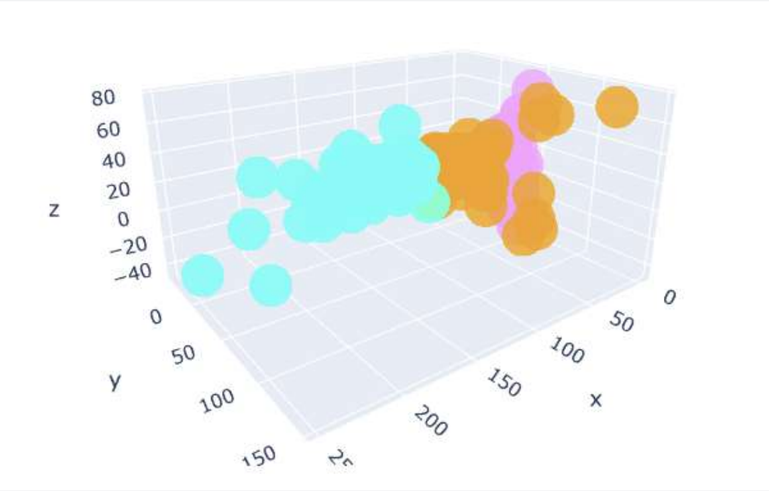
```
print(pca_3d.explained_variance_ratio_)
print(pca_3d.explained_variance_ratio_.sum())
# 3개의 PC가 전체 데이터의 약 89%를 설명가능하니, PCA를 활용해 시각화를 하는 방법을 이용하는데 문제가 없다.
```

```
[0.62190816 0.19297364 0.07374229]
0.8886240887918866
```

시각화

PCs를 이용하여 좌표에 점들을 찍어 보았을 때 군집별로 분류되어 있는 걸 확인할 수 있다.

특히 PC1을 따라서 군집들이 나누어지는 경계가 생기는 것을 확인 할 수 있고, 이는 PC1이 원본 데이터의 분산을 62% 설명하는 것을 미루어 보면 당연한 결과이다.



결과 분석

1. 군집별 평균을 알아본다.

Cluster	age	credit_score	yearly_income	desired_amount
0	33.747722	624.119535	4.138879e+07	3.134329e+07
1	40.526582	578.284567	4.332979e+07	3.123046e+07
2	41.727273	585.014421	1.126283e+08	6.867566e+07

각 군집별로 평균에 차이가 보이지만, 단순 평균이기 때문에 이 차이들이 통계적으로 유의미 한지는 확신할 수 없기 때문에 분산분석과 사후분석을 통해 더 자세하게 분석을 실시한다.

Cluster	CompleteIDCertification	EndLoanApply	GetCreditInfo	Login	OpenApp	SignUp	StartLoanApply	UseDSRCalc	UseLoanManage	UsePrepayCalc	ViewLoanApplyIntro
0	5.273794	11.612543	7.418756	0.462253	9.279413	0.087111	7.576702	0.052115	6.082161	0.085219	7.144409
1	14.726582	35.968354	26.026582	25.902532	34.175949	0.349367	21.970886	0.045570	16.973418	0.059494	20.510127
2	50.672727	143.618182	80.745455	109.927273	137.618182	1.418182	81.563636	0.054545	53.890909	0.309091	76.618182

평균적으로 보아도 event 변수들에서는 0<1<2 군집으로 갈 수록 평균적으로 시행한 횟수들이 많은 것을 확인 할 수 있다.

결과 분석

2. ANOVA를 이용해 통계분석을 실시해, 군집 별 평균 차이를 통계적으로 분석한다.

```
df = pd.DataFrame(merge_copy, columns=['age','Cluster'])

model = ols('age ~ C(Cluster)', df).fit()
print(anova_lm(model))
```

```
                df        sum_sq       mean_sq           F         PR(>F)
C(Cluster)     2.0  3.885088e+04  19425.439015  252.957151  8.401045e-110
Residual   35189.0  2.702283e+06     76.793397         NaN            NaN
```

'age'를 예시로 들었을 때, F통계량이 252.957이 나오고 p-value가 0.05보다 작은 것을 확인할 수 있다. 이는 군집 간에 age 변수의 평균을 비교해 보았을 때 통계적으로 유의미한 차이가 있다는 것을 뜻한다. 하지만 0,1,2 중 어떤 군집 간에 유의미한 차이가 있는 지는 확신할 수 없으니 '사후분석'을 통해 구체화를 실시한다.

결과 분석

3. 더 정확한 비교를 위해 사후분석을 실시한다.

```
# 4.3 사후분석
comp = MultiComparison(merge_copy.age, merge_copy.Cluster)

result = comp.allpairtest(scipy.stats.ttest_ind, method='bonf')
print(result[0])

hsd = pairwise_tukeyhsd(merge_copy['age'], merge_copy['Cluster'], alpha=0.05)
hsd.summary()

# 군집 0과1, 0과2 사이에는 age의 차이가 유의미하지만, 군집 1과2 사이에는 age차이가 유의미하다고 볼 수 없다.
```

```
Test Multiple Comparison ttest_ind
FWER=0.05 method=bonf
alphacSidak=0.02, alphacBonf=0.017
=============================================
group1 group2   stat    pval  pval_corr reject
---------------------------------------------
     0      1 -21.5007    0.0       0.0   True
     0      2  -6.7949    0.0       0.0   True
     1      2  -0.7813 0.4348       1.0  False
---------------------------------------------
```

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
0	1	6.7789	0.001	6.0398	7.518	True
0	2	7.9796	0.001	5.2078	10.7513	True
1	2	1.2007	0.5815	-1.6636	4.065	False

사후분석의 결과를 살펴보면, 군집 0과 1 사이의 검정에서는 reject이 true이니 두 군집 사이의 age평균 차이가 통계적으로 유의미하다고 할 수 있다. 0과2 사이도 똑같이 유의미하다고 할 수 있지만, 군집 1과2 사이는 reject이 false이기 때문에 두 군집의 평균 차이는 통계적으로 유의미하다고 할 수 없다.

age 이외에도 credit_score, yearly_income, desired_amount와 같은 변수들도 이와 같은 분석 과정을 거쳤다.

event 변수 분석

1. UserDSRCalc, UsePrepayCalc을 제외한 모든 변수에서는 reject가 모두 true이니, 3그룹간의 유의미한 차이가 존재한다고 할 수 있다. 이는 두 변수를 제외한 변수들에서는 0<1<2번 그룹 순으로 이용빈도가 높다고 해석할 수 있다.
2. UseDSRCalc에서는 3그룹간의 통계적 차이를 찾을 수 없다.
3. UsePrepayCalc에서 0과1번 그룹은 통계적으로 차이가 없지만, 0과2, 1과2번 그룹간에는 통계적 차이가 보이므로, 2번 그룹이 0과1번 그룹에 비해 UsePrepayCalc을 많이 이용한다고 해석할 수 있다.

```
hsd = pairwise_tukeyhsd(merge_copy['EndLoanApply'], merge_copy['Cluster'], alpha=0.05)
hsd.summary()
```

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
0	1	24.3558	0.001	22.5676	26.1441	True
0	2	132.0056	0.001	125.2996	138.7117	True
1	2	107.6498	0.001	100.7198	114.5799	True

```
hsd = pairwise_tukeyhsd(merge_copy['UseDSRCalc'], merge_copy['Cluster'], alpha=0.05)
hsd.summary()
```

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
0	1	-0.0065	0.8676	-0.0384	0.0253	False
0	2	0.0024	0.9	-0.1169	0.1217	False
1	2	0.009	0.9	-0.1143	0.1323	False

```
hsd = pairwise_tukeyhsd(merge_copy['UsePrepayCalc'], merge_copy['Cluster'], alpha=0.05)
hsd.summary()

## 0,1번 그룹에 비해서, 2번 그룹이 UsePrepayCalc을 많이 이용한다.
```

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
0	1	-0.0257	0.342	-0.0688	0.0174	False
0	2	0.2239	0.0033	0.0623	0.3855	True
1	2	0.2496	0.0013	0.0826	0.4166	True

결과분석

분석결과들을 종합해 보았을 때
군집 0은 age 작고, credit score가 높은 편이며 모든 event를 적게 사용한다.
군집 1은 군집 0과 군집2 사이의 특성들을 가진다.
군집 2은 yearly_income이 높은 편이며 모든 event들을 많이 사용하는 heavy user라고 할 수 있다.

Cluster	age	credit_score	yearly_income	desired_amount
0	33.747722	624.119535	4.138879e+07	3.134329e+07
1	40.526582	578.284567	4.332979e+07	3.123046e+07
2	41.727273	585.014421	1.126283e+08	6.867566e+07

Cluster	CompleteIDCertification	EndLoanApply	GetCreditInfo
0	5.273794	11.612543	7.418756
1	14.726582	35.968354	26.026582
2	50.672727	143.618182	80.745455

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
0	1	1940995.5642	0.7681	-4991367.7348	8873358.8633	False
0	2	71239465.2228	0.001	45242477.9413	97236452.5044	True
1	2	69298469.6586	0.001	42433250.3726	96163688.9446	True

```
hsd = pairwise_tukeyhsd(merge_copy['EndLoanApply'], merge_copy['Cluster'], alpha=0.05)
hsd.summary()
```

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
0	1	24.3558	0.001	22.5676	26.1441	True
0	2	132.0056	0.001	125.2996	138.7117	True
1	2	107.6498	0.001	100.7198	114.5799	True

대출고대로분석했조

감사합니다!