신용카드 사용자 연체 예측 AI

스터디 2조 고선욱 유상준

목차

1. 주제 및 데이터 소개

1. 주제 및 데이터 소개

1.주제

신용카드 사용자 데이터를 보고 사용자의 대금 연체 정도를 예측하는 알고리즘 개발

2. 배경

신용카드사는 신용카드 신청자가 제출한 개인정보와 데이터를 활용해 신용 점수를 산정합니다.
신용카드사는 이 신용 점수를 활용해 신청자의 향후 채무 불이행과 신용카드 대급 연체 가능성을 예측합니다.

현재 많은 금융업계는 인공지능(AI)를 활용한 금융 서비스를 구현하고자 합니다.
사용자의 대금 연체 정도를 예측할 수 있는 인공지능 알고리즘을 개발해 금융업계에 제안할 수 있는 인사이트를 발굴하기 위함입니다.

1. 주제 및 데이터 소개

데이터 소개 및 변수 설명

train.csv : train 데이터 : 신용카드 사용자들의 개인 신상정보 Credit 포함, train.shape : (26457, 20)

test.csv : test 데이터 : 신용카드 사용자들의 개인 신상정보 Credit 미포함, test.shape : (10000, 19)

•gender: 성별

•car: 차량 소유 여부

•reality: 부동산 소유 여부

•child_num: 자녀 수

•income_total: 연간 소득

•income_type: 소득 분류
- •['Commercial associate', 'Working', 'State servant', 'Pensioner', 'Student']

•edu_type: 교육 수준
- •['Higher education' ,'Secondary / secondary special', 'Incomplete higher', 'Lower secondary', 'Academic degree']

•family_type: 결혼 여부
- •['Married', 'Civil marriage', 'Separated', 'Single / not married', 'Widow']

•house_type: 생활 방식
- •['Municipal apartment', 'House / apartment', 'With parents', 'Co-op apartment', 'Rented apartment', 'Office apartment']

•DAYS_BIRTH: 출생일
- •데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 하루 전에 태어났음을 의미

•DAYS_EMPLOYED: 업무 시작일
- 데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 하루 전부터 일을 시작함을 의미
- 양수 값은 고용되지 않은 상태를 의미함

•FLAG_MOBIL: 핸드폰 소유 여부

•work_phone: 업무용 전화 소유 여부

•phone: 전화 소유 여부

•email: 이메일 소유 여부

•occyp_type: 직업 유형

•family_size: 가족 규모

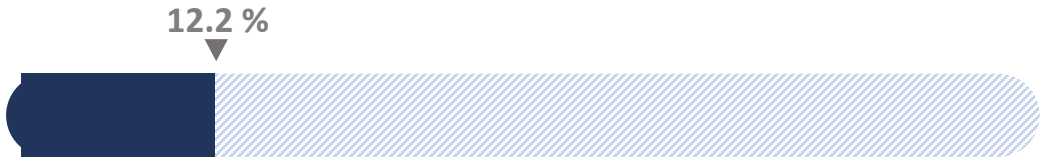
•begin_month: 신용카드 발급 월
- 데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 한 달 전에 신용카드를 발급함을 의미

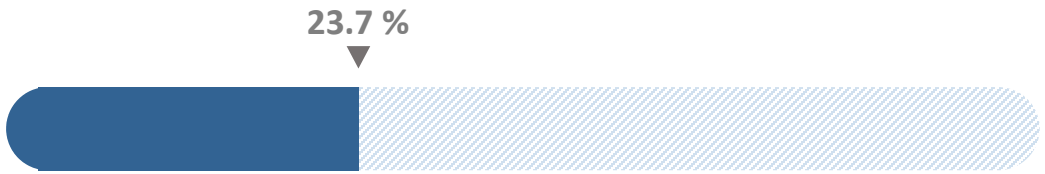
•credit: 사용자의 신용카드 대금 연체를 기준으로 한 신용도
- 낮을 수록 높은 신용의 신용카드 사용자를 의미함

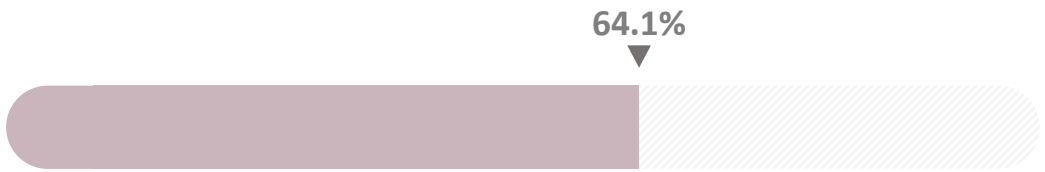
1. 주제 및 데이터 소개

타겟 변수 설명

credit: 사용자의 신용카드 대금 연체를 기준으로 한 신용도
신용도가 낮을 수록 높은 신용의 신용카드 사용자를 의미







0	전체 데이터 중신용도가0등급 인 데이터의 비율
1	전체 데이터 중신용도가1등급 인 데이터의 비율
2	전체 데이터 중신용도가2등급 인 데이터의 비율

2. EDA 및 데이터 탐색

2. EDA 및 데이터 탐색

데이터의 분포를 '신용도'에 따라 확인해보자

우리가 예측하고 싶은 것은 개인의 신용도이며 신용카드 신청자가 제출한 개인정보와 데이터를 활용해 신용 점수를 산정

```
In [7]:
# 신용등급별 범주형 컬럼의 비율을 파이 차트로 시각화 하기 위한 함수
def draw_cat_pie(column):
    f, ax = plt.subplots(1, 3, figsize=(30, 20))

    wedgeprops={'width': 0.7, 'edgecolor': 'w', 'linewidth': 5}
    colors = sns.color_palette('Set2')

    credit_0[column].value_counts().plot.pie(autopct='%.1f%%', ax=ax[0], textprops={'fontsize': 15},
                                             startangle=90, counterclock=False,
                                             colors=colors, shadow=True, wedgeprops=wedgeprops)
    ax[0].set_title('credit=0인 {} ratio'.format(column), size=20)
    ax[0].set_ylabel('')

    credit_1[column].value_counts().plot.pie(autopct='%.1f%%', ax=ax[1], textprops={'fontsize': 15},
                                             startangle=90, counterclock=False,
                                             colors=colors, shadow=True, wedgeprops=wedgeprops)
    ax[1].set_title('credit=1인 {} ratio'.format(column), size=20)
    ax[1].set_ylabel('')

    credit_2[column].value_counts().plot.pie(autopct='%.1f%%', ax=ax[2], textprops={'fontsize': 15},
                                             startangle=90, counterclock=False,
                                             colors=colors, shadow=True, wedgeprops=wedgeprops)
    ax[2].set_title('credit=2인 {} ratio'.format(column), size=20)
    ax[2].set_ylabel('')

    plt.show()
```

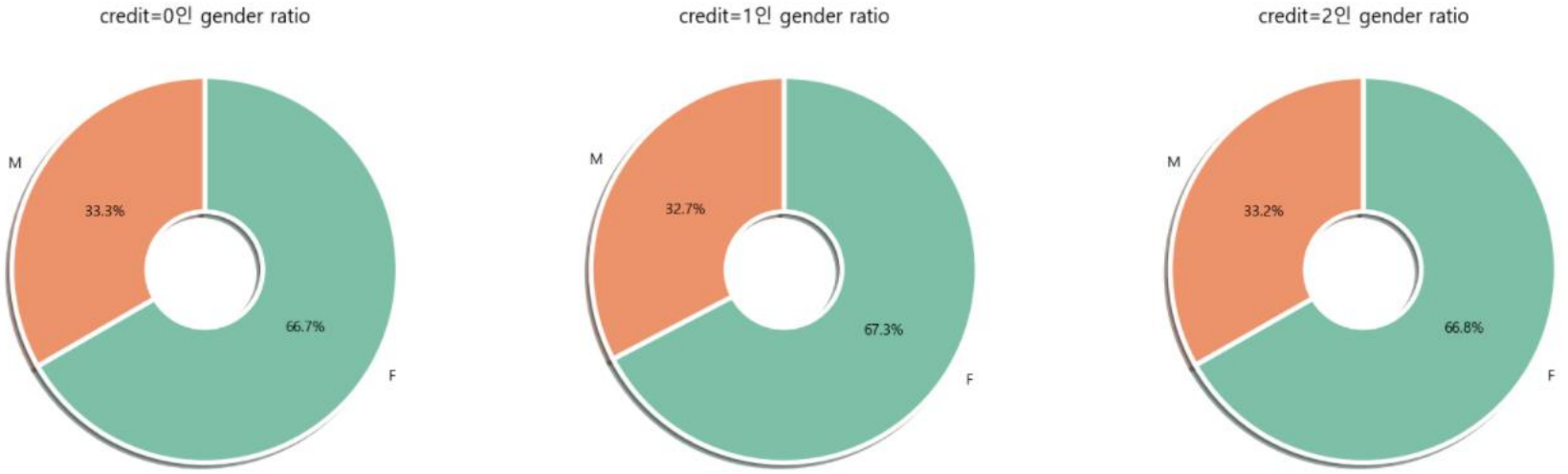
```
def draw_cat_bar(column):

    f, ax = plt.subplots(1, 3, figsize=(30, 8))


    sns.countplot(y = column,
                  data = credit_0,
                  ax = ax[0],
                  palette='Set2',
                  order = credit_0[column].value_counts().index)
    ax[0].tick_params(labelsize=15)
    ax[0].set_title('credit=0인 {} count'.format(column), size=20)
    ax[0].set_xlabel('count', size=15)
    ax[0].set_ylabel('')

    sns.countplot(y = column,
                  data = credit_1,
                  ax = ax[1],
                  palette='Set2',
                  order = credit_1[column].value_counts().index)
    ax[1].tick_params(labelsize=15)
    ax[1].set_title('credit=1인 {} count'.format(column), size=20)
    ax[1].set_xlabel('count', size=15)
    ax[1].set_ylabel('')

    sns.countplot(y = column,
                  data = credit_2,
                  ax = ax[2],
                  palette='Set2',
                  order = credit_2[column].value_counts().index)
    ax[2].tick_params(labelsize=15)
    ax[2].set_title('credit=2인 {} count'.format(column), size=20)
    ax[2].set_xlabel('count', size=15)
    ax[2].set_ylabel('')

    plt.subplots_adjust(wspace=0.6, hspace=0.3)
    plt.show()
```

2. EDA 및 데이터 탐색

신용등급 별 성별 분포

```
In [10]: ▶ draw_cat_pie('gender')
```

executed in 282ms, finished 01:03:35 2021-12-02



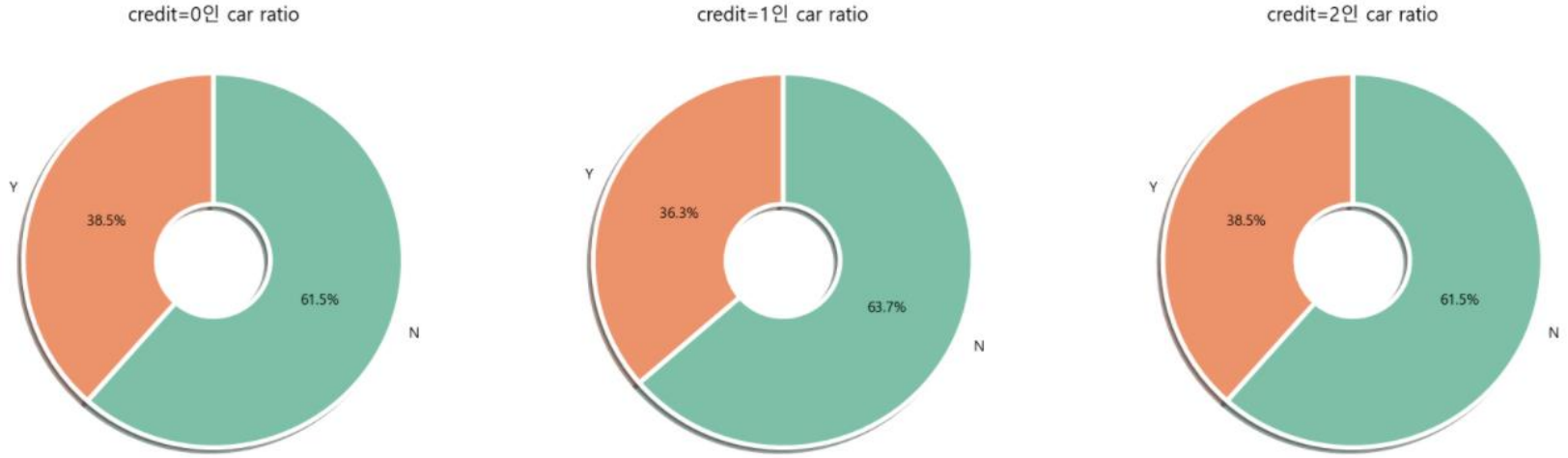
- 모든 신용등급에서 전체적으로 여성이 남성에 비해 약 2배 가량 더 많았다.

2. EDA 및 데이터 탐색

신용등급 별 차량 소유 여부 분포

```
In [11]: ▶ draw_cat_pie('car')
```

executed in 268ms, finished 01:03:35 2021-12-02



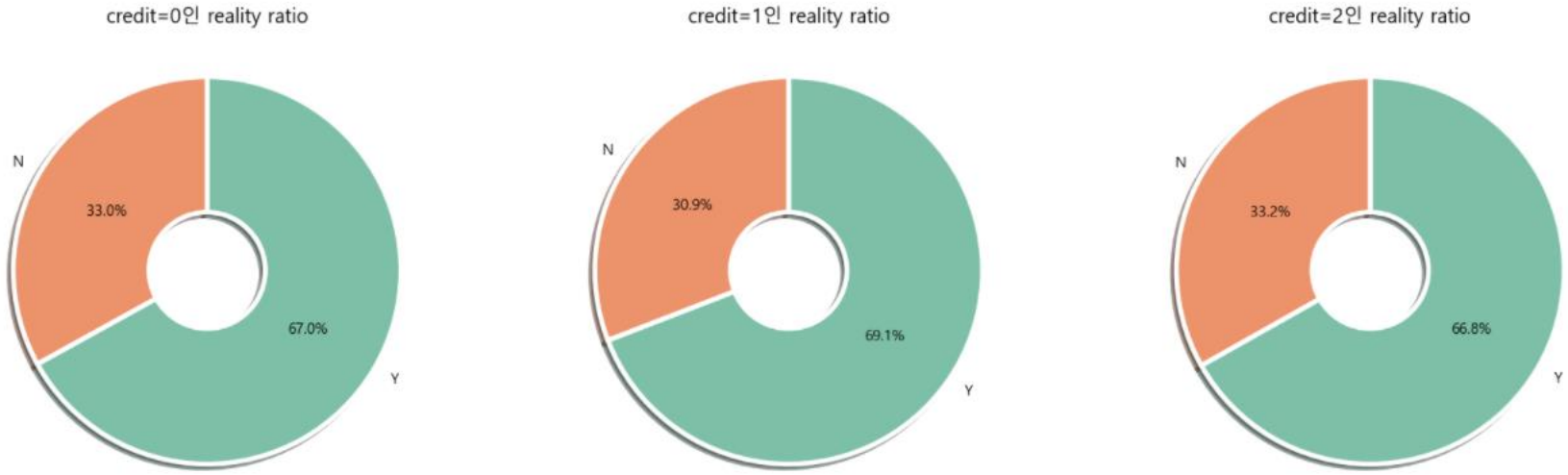
- 신용등급 여부와 상관없이 전체적으로 차량을 소유하지 않은 사람들이 차량을 소유한 사람보다 더 많았다

2. EDA 및 데이터 탐색

신용등급 별 부동산 소유 여부 분포

```
In [12]: ▶ draw_cat_pie('reality') # reality: 부동산 소유 여부
```

executed in 283ms, finished 01:03:36 2021-12-02



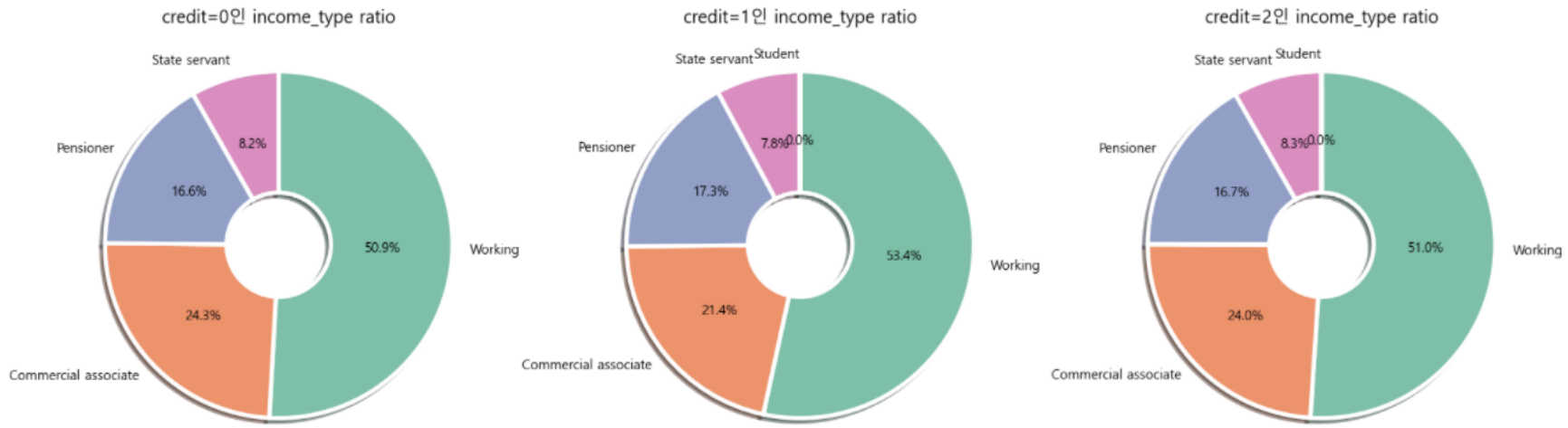
- 부동산 소유 여부도 마찬가지로 신용등급 여부와 상관없이 전체적으로 부동산을 소유하지 않은 사람보다 소유한 사람이 더 많았다.

2. EDA 및 데이터 탐색

신용등급 별 소득 분류 분포

```
In [13]: ▶ draw_cat_pie('income_type')
```

executed in 328ms, finished 01:03:36 2021-12-02



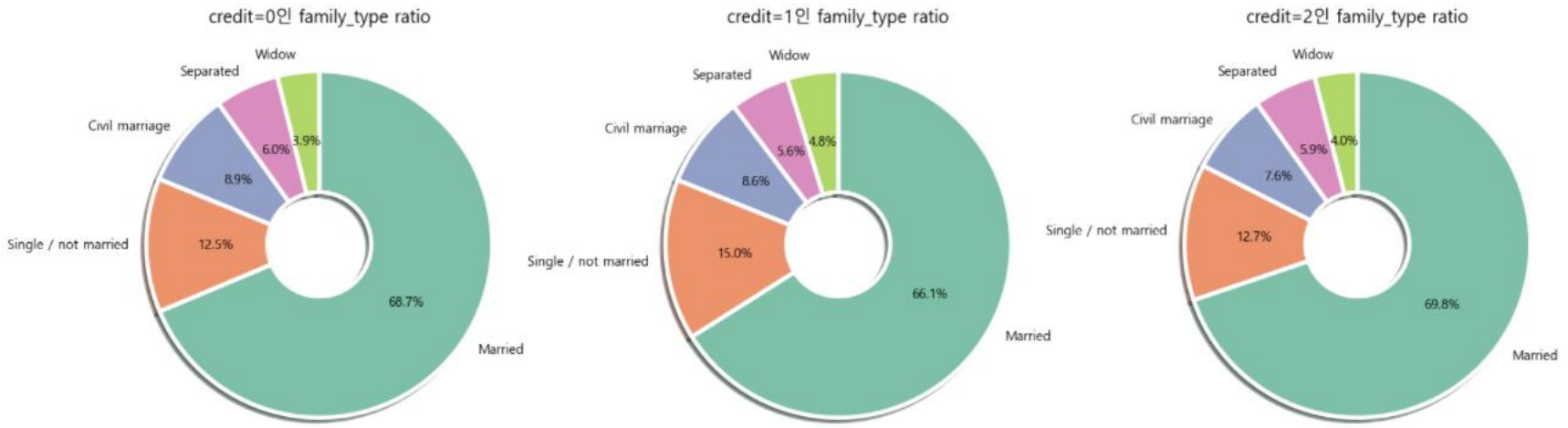
- Commercial associate는 상업 영업 사원으로 회사의 대표자를 의미한다는데 자세한 것은 추후 과정을 진행하면서 더 알아가보자.
- Pensioner은 연금 수령자이며, State servant는 주를 위해 일하는 사람으로 공무원으로 해석하면 될 것 같다.
- Working은 노동자, 즉 일반적인 회사원 혹은 근로자라고 봐야 될 것 같다.
- 신용등급별로 분포가 비슷하게 되어있으며 특이사항으로는 소득분류의 절반은 근로자이며 학생이 거의 존재하지 않는다.

2. EDA 및 데이터 탐색

신용등급 별 결혼여부 분포

```
In [18]: draw_cat_pie('family_type')
```

executed in 344ms, finished 01:03:37 2021-12-02

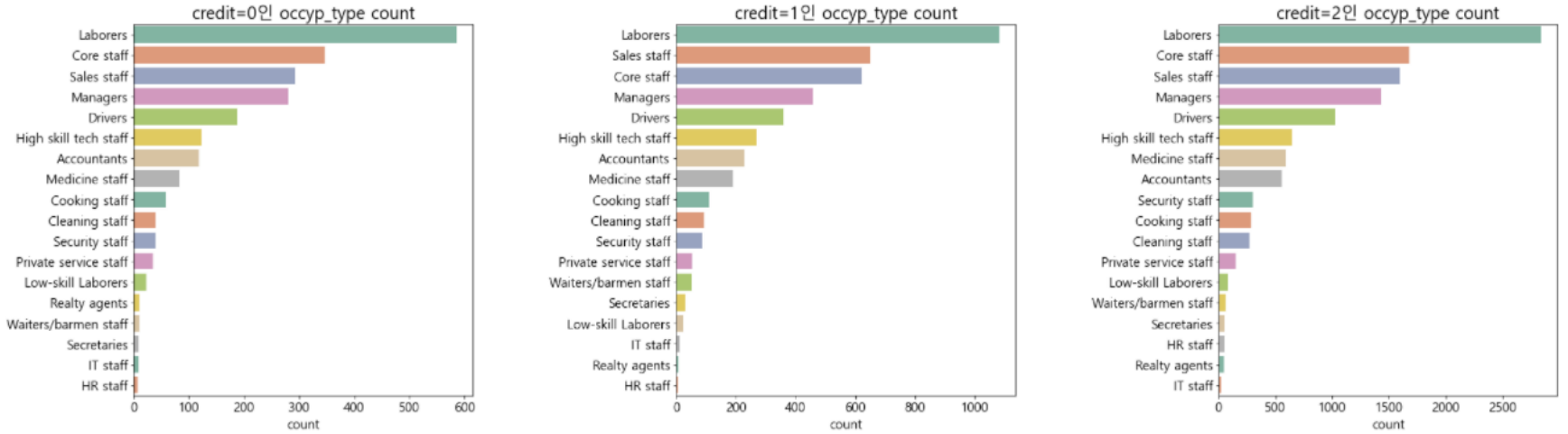


- 마찬가지로 신용등급별로 분포가 비슷하게 나타났으며, 결혼한 가정이 절반 이상을 차지한다.

2. EDA 및 데이터 탐색

신용등급 별 직업 유형 분포

In [26]:
```
draw_cat_bar('occyp_type')
```
executed in 373ms, finished 01:03:38 2021-12-02



In [27]:
```
print("Labor's ratio: {}%".format(round(train['occyp_type'].value_counts()[0] / len(train) * 100,2))) # 널값의 비율
```
executed in 14ms, finished 01:03:38 2021-12-02

```
Labor's ratio: 17.05%
```

In [28]:
```
len(train) - train['occyp_type'].value_counts().sum() # 널값 존재
```
executed in 12ms, finished 01:03:38 2021-12-02

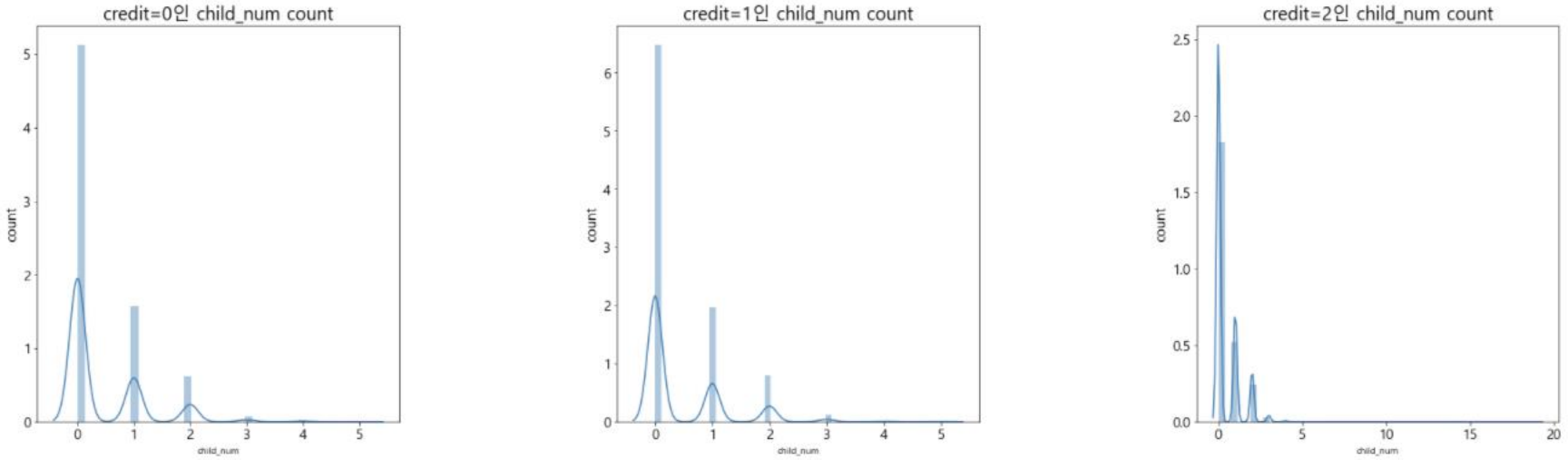
```
8171
```

2. EDA 및 데이터 탐색

신용등급 별 자녀 수의 분포

```
In [31]: draw_num_bar('child_num')
```

executed in 569ms, finished 01:03:39 2021-12-02



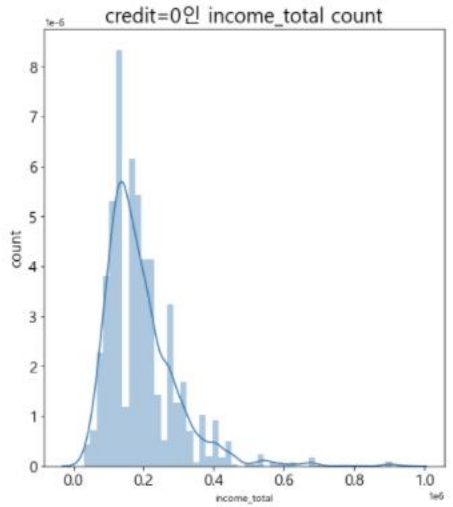
- 대체적으로 자녀가 없는 사람들이 많이 분포해있다.

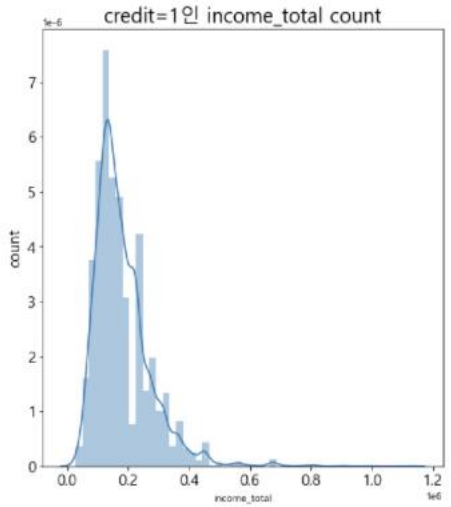
2. EDA 및 데이터 탐색

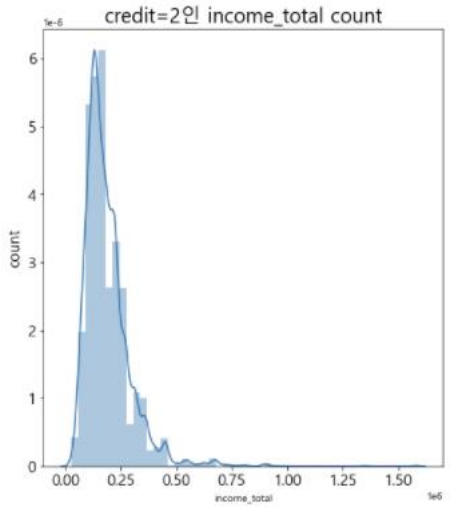
신용등급 별 연간 소득의 분포

```
In [32]: ▶ draw_num_bar('income_total')
```

executed in 582ms, finished 01:03:39 2021-12-02







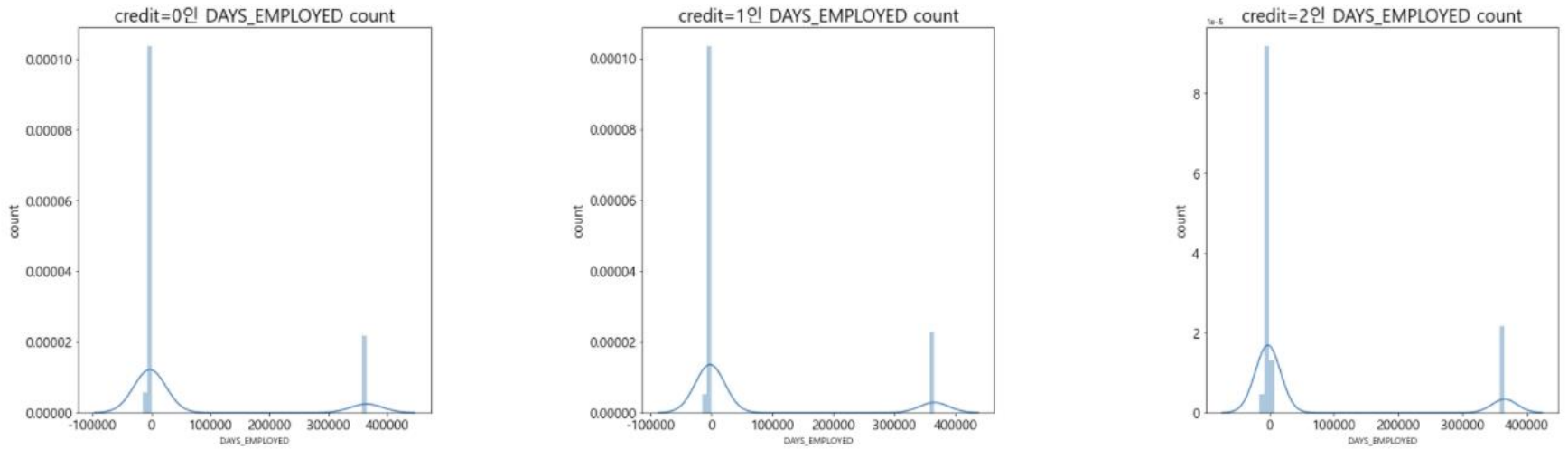
- 등급 간의 연간 소득의 분포의 차이가 거의 없었다.

2. EDA 및 데이터 탐색

신용등급 별 업무 시작일의 분포

```
In [34]: ▶ draw_num_bar('DAYS_EMPLOYED')
```

executed in 448ms, finished 01:03:40 2021-12-02



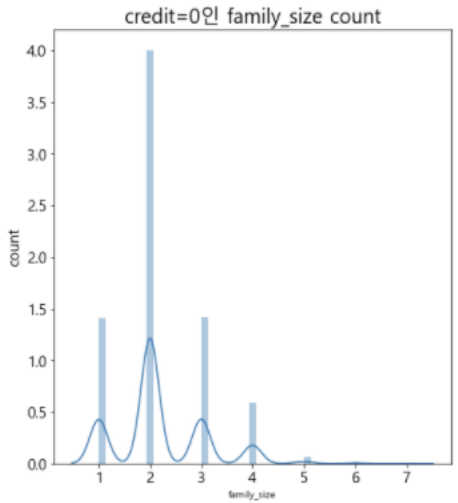
- 값이 0이라는 것은 직업이 없는 사람을 의미하며, 대체적으로 무직인 사람들이 많이 분포하였으며, 등급 간의 유의미한 분포의 차이는 없었다.

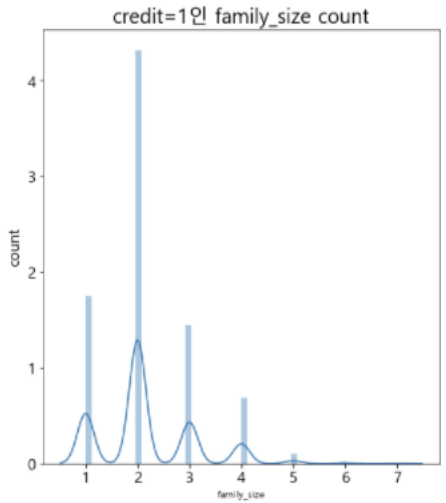
2. EDA 및 데이터 탐색

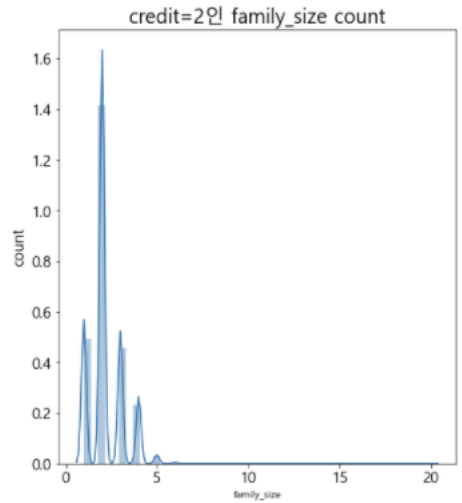
신용등급 별 가족 규모 분포

```
In [35]: ▶ draw_num_bar('family_size')
```

executed in 523ms, finished 01:03:41 2021-12-02







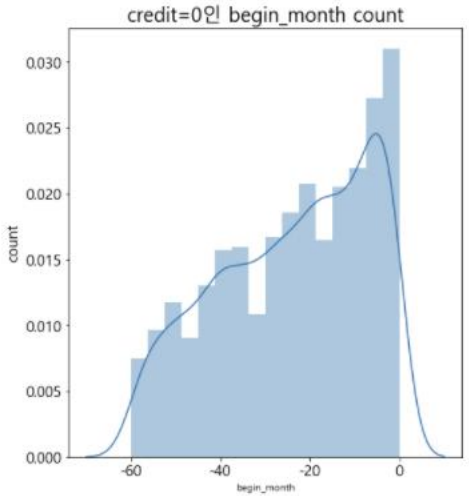
- 앞서 보았던 자녀가 없는 사람들이 많이 존재했다는 것을 미루어 보았을 때, 가족 규모가 2명이 많이 분포한다는 것을 유추할 수 있으며 시각적으로도 확인할 수 있었다.
- 또한 앞에서 확인한 결혼한 사람들이 많았다는 것도 연관되었다는 사실을 알 수 있다.
- 등급 간의 유의미한 분포의 차이는 없었다.

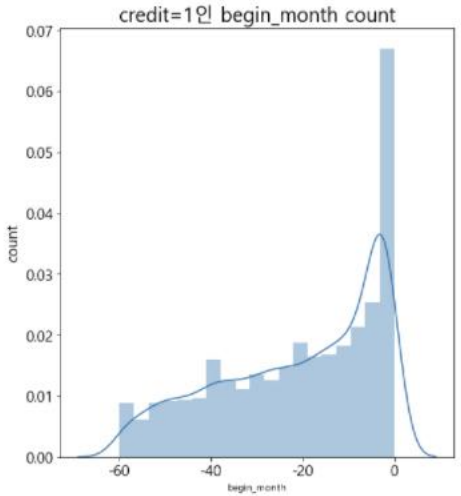
2. EDA 및 데이터 탐색

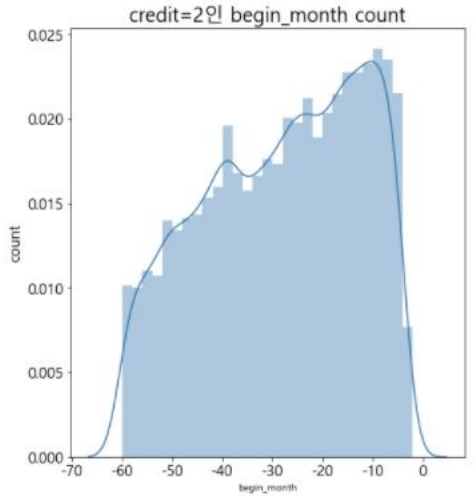
신용등급 별 신용카드 발급 월 분포

```
In [36]: ▶ draw_num_bar('begin_month')
```

executed in 403ms, finished 01:03:41 2021-12-02







- 3개의 신용등급 모두 10개월 이하로 카드발급을 사람들이 많이 분포한다는 것을 알 수 있다.

3. 데이터 전처리

3. 데이터 전처리

결측치 확인 및 처리

In [43]:
```
train.isnull().sum()
```
executed in 14ms, finished 01:03:42 2021-12-02

```
gender              0
car                 0
reality             0
child_num           0
income_total        0
income_type         0
edu_type            0
family_type         0
house_type          0
DAYS_BIRTH          0
DAYS_EMPLOYED       0
FLAG_MOBIL          0
work_phone          0
phone               0
email               0
occyp_type       8171
family_size         0
begin_month         0
credit              0
dtype: int64
```

In [45]:
```
train[train['occyp_type'].isnull()]['credit'].value_counts() # 등급별 직업유형의 값이 널값인 데이터의 분포
```
executed in 14ms, finished 01:03:42 2021-12-02

```
2.0    5266
1.0    1938
0.0     967
Name: credit, dtype: int64
```

- train 데이터의 credit 비율과 occyp_type 가 NaN인 비율이 비슷하여 drop을 시키려 하였으나 컬럼의 이름이 직업종류인 것을 볼 수 있다.
- 추후 파생변수를 생성하는데 있어 필요할지도 몰라 일단 놔두었다.

In [46]:
```
print("occyp_type's null values ratio: {}%".format(round(train['occyp_type'].isnull().sum() / len(train) * 100, 1)))
```
executed in 14ms, finished 01:03:42 2021-12-02

```
occyp_type's null values ratio: 30.9%
```

- 또한 그 비율이 30.9%로 적지 않기 때문에 어떻게 처리할지에 대해 생각해보자

In [47]:
```
train = train.fillna({'occyp_type': 'No job or No info'})
test = test.fillna({'occyp_type': 'No job or No info'})
```
executed in 14ms, finished 01:03:42 2021-12-02

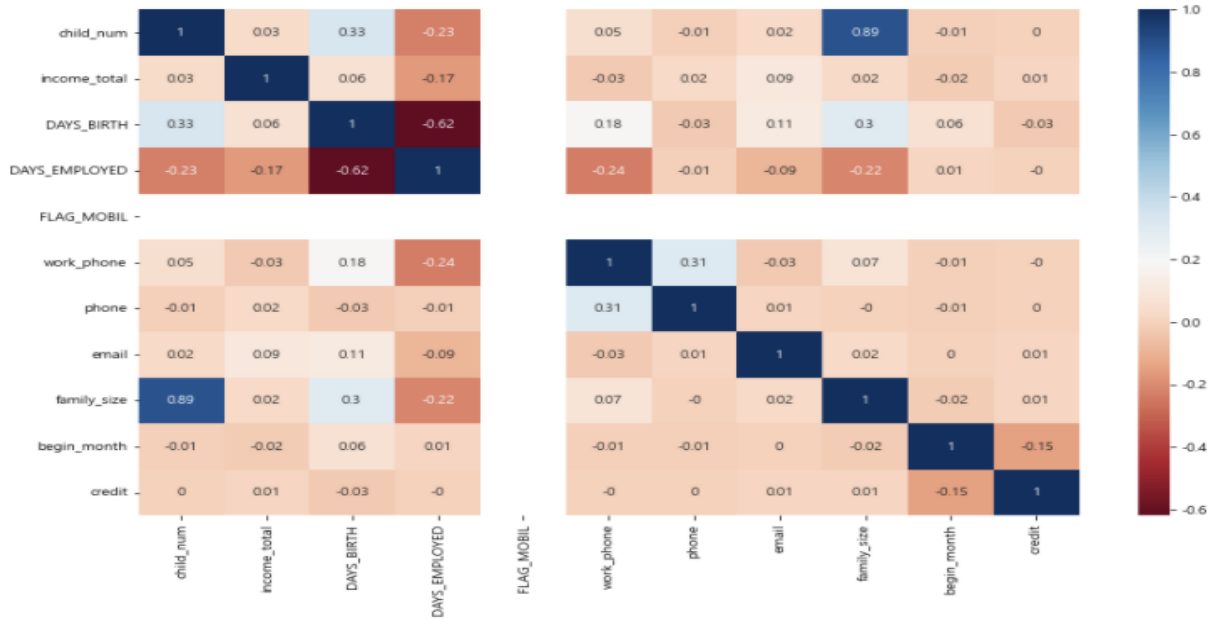
- 컬럼의 이름이 직업유형인 것으로 보아 직업이 없거나 정보가 없는 것으로 유추해야 될 것 같다.

3. 데이터 전처리

변수 간 상관관계 확인

In [48]:
```
plt.figure(figsize =(14,8))
corr = train.corr().round(2)
sns.heatmap(corr, annot=True, cmap='RdBu');
```

executed in 448ms, finished 01:03:42 2021-12-02



- flag_mobil의 값이 없어 확인해봐야겠다.
- child_num과 family_size사이에는 상관도가 0.89로 매우 높아 다중공선성이 발생할 수 있을 것 같다.
- 이 두개의 컬럼을 각각 활용하는 것보다 위에서 언급했던 것처럼 파생변수를 만드는데 활용해야 될 것 같다.

3. 데이터 전처리

필요 없는 컬럼 제거 – 'FLAG_MOBIL'

In [49]:
```
train['FLAG_MOBIL'].value_counts()
```

executed in 15ms, finished 01:03:42 2021-12-02

```
1    26457
Name: FLAG_MOBIL, dtype: int64
```

- 데이터에 있는 모든 사람들이 핸드폰을 소유하고 있으므로, 필요없는 컬럼으로 간주해야겠다.

3.3.1 필요없는 컬럼 제거

In [50]:
```
train.drop('FLAG_MOBIL', axis=1, inplace=True)
test.drop('FLAG_MOBIL', axis=1, inplace=True)
```

executed in 14ms, finished 01:03:42 2021-12-02

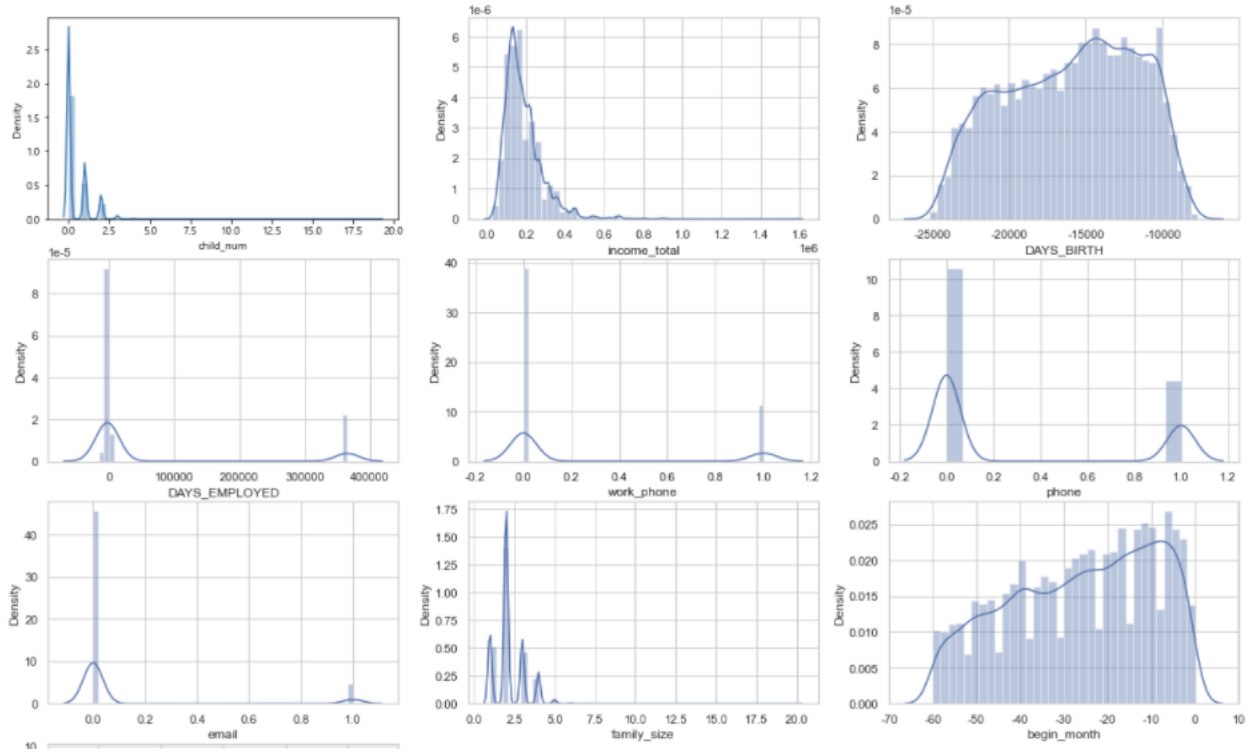
3. 데이터 전처리

수치형 데이터 전처리– 컬럼 별 분포 확인

In [54]:
```
plt.figure(figsize = (20,16))
for i, column in enumerate(numeric_vars):
    plt.subplot(4,3,i+1)
    sns.set_theme(style='whitegrid')
    sns.distplot(train[column])
plt.show()
```

executed in 2.38s, finished 01:03:45 2021-12-02

- work_phone, phone, email 같은 경우는 소유하고 있거나 소유하고 있지 않은 것으로 이미 라벨인코딩이 되어있는 상태로 보면 될 것 같다.
- 몇몇의 수치형 컬럼들은 데이터들이 한쪽으로 치우친 skew되어 있는 것이 보인다.
- 또한 음수인 값들이 존재하는 컬럼들이 있는데 **파생변수**를 만들거나 **스케일링**을 하는 데 있어 양수화로 전환할 필요가 있어 보인다.
- 정규화와 양수 변환 이전에 이상치를 먼저 확인하여 처리할게 있으면 처리해보자.

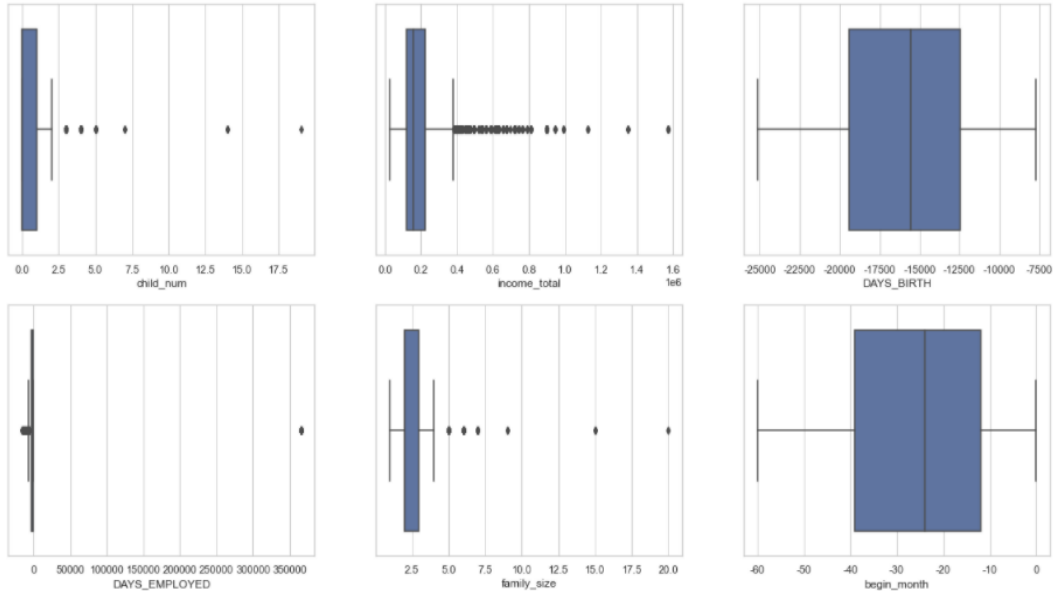


3. 데이터 전처리

수치형 데이터 전처리– 컬럼 별 이상치 확인

In [55]:
```
numeric_vars = ['child_num', 'income_total', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'family_size', 'begin_month']
```
executed in 14ms, finished 01:03:45 2021-12-02

In [56]:
```
plt.figure(figsize = (20,16))

for i, column in enumerate(numeric_vars):
    plt.subplot(3,3,i+1)
    sns.set_theme(style='whitegrid')
    sns.boxplot(train[column])
plt.show()
```
executed in 552ms, finished 01:03:45 2021-12-02



In [57]:
```
def get_outlier(df=None, column=None, weight=1.5):
    df = train[column]
    quantile_25 = np.percentile(df.values, 25)
    quantile_75 = np.percentile(df.values, 75)

    iqr = quantile_75 - quantile_25
    iqr_weight = iqr * weight
    lowest_val = quantile_25 - iqr_weight
    highest_val = quantile_75 + iqr_weight

    outlier_index = df[(df < lowest_val) | (df > highest_val)].index
    print('{}의 outlier의 수 : '.format(column) , len(outlier_index))

    return outlier_index
```
executed in 14ms, finished 01:03:45 2021-12-02

In [58]:
```
outlier_col = ['child_num', 'income_total', 'DAYS_EMPLOYED', 'family_size']
for item in outlier_col:
    get_outlier(df=train, column=item, weight=1.5)
```
executed in 14ms, finished 01:03:45 2021-12-02

```
child_num의 outlier의 수 :  369
income_total의 outlier의 수 :  1129
DAYS_EMPLOYED의 outlier의 수 :  5726
family_size의 outlier의 수 :  350
```

- 이렇게 컬럼별 outlier의 개수가 나왔다.
- 성능을 확인해가면서 이상치를 어떻게 처리할 것인지 계속해서 고려해야 될 것 같다.

3. 데이터 전처리

수치형 데이터 전처리 – 'DAYS_EMPLOYED', 'DAYS_BIRTH', 'begin_month'

3.4.3 'DAYS_EMPLOYED' 컬럼

- DAYS_EMPLOYED: 업무 시작일
  - 데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 하루 전부터 일을 시작함을 의미
  - 양수 값은 고용되지 않은 상태를 의미하므로 0으로 처리한다

In [59]:
```
train['DAYS_EMPLOYED'] = train['DAYS_EMPLOYED'].map(lambda x: 0 if x > 0 else x)
test['DAYS_EMPLOYED'] = test['DAYS_EMPLOYED'].map(lambda x: 0 if x > 0 else x)
```
executed in 14ms, finished 01:03:45 2021-12-02

3.4.4 음수값이 존재하는 컬럼 양수로 변환 - 'DAYS_BIRTH', 'begin_month', 'DAYS_EMPLOYED'

In [60]:
```
neg_vars = ['DAYS_BIRTH', 'begin_month', 'DAYS_EMPLOYED']
for var in neg_vars:
    train[var]=np.abs(train[var])
    test[var]=np.abs(test[var])
```
executed in 14ms, finished 01:03:45 2021-12-02

In [61]:
```
train.head(1)
```
executed in 13ms, finished 01:03:45 2021-12-02

“3-1

파생 변수 생성

1. 수치형 데이터
2. 값 확인 및 이상치 제거
3. 범주형 데이터

3. 데이터 전처리

파생변수 생성– 수치형 데이터

3.5 파생변수 생성 - 수치형 데이터

- 유의미하다고 생각되는 파생변수 리스트 -> 나중에 인사이트와도 연관성을 고려하여 선정함

3.5.1 'age' 변수 생성

- DAYS_BIRTH: 출생일
- 데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 하루 전에 태어났음을 의미
- 값을 사용하기 어려운 범위이므로 나이 컬럼을 만들어 볼 수 있을 것 같다.

In [62]:
```
train['DAYS_BIRTH'].describe()
```

executed in 14ms, finished 01:03:45 2021-12-02

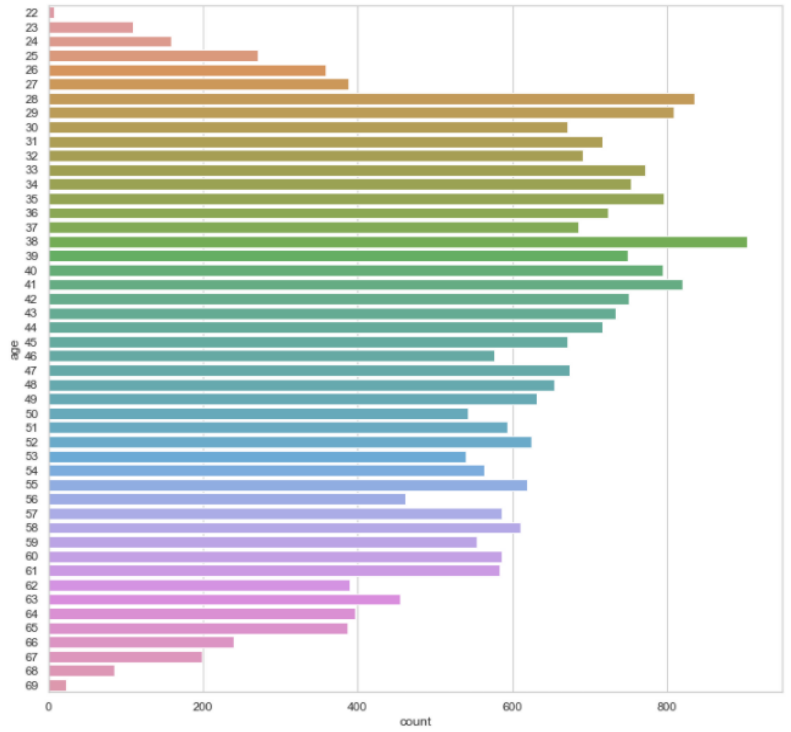
```
count    26457.000000
mean     15958.053899
std       4201.589022
min       7705.000000
25%      12446.000000
50%      15547.000000
75%      19431.000000
max      25152.000000
Name: DAYS_BIRTH, dtype: float64
```

In [63]:
```
train['age'] = train['DAYS_BIRTH'] // 365 + 1
test['age'] = test['DAYS_BIRTH'] // 365 + 1
```

executed in 14ms, finished 01:03:45 2021-12-02

In [64]:
```
plt.figure(figsize=(12,12))
sns.countplot(y=train['age']);
```

executed in 269ms, finished 01:03:45 2021-12-02



- 앞서 EDA에서 확인했듯이 20~40대사이의 분포가 가장 많다.

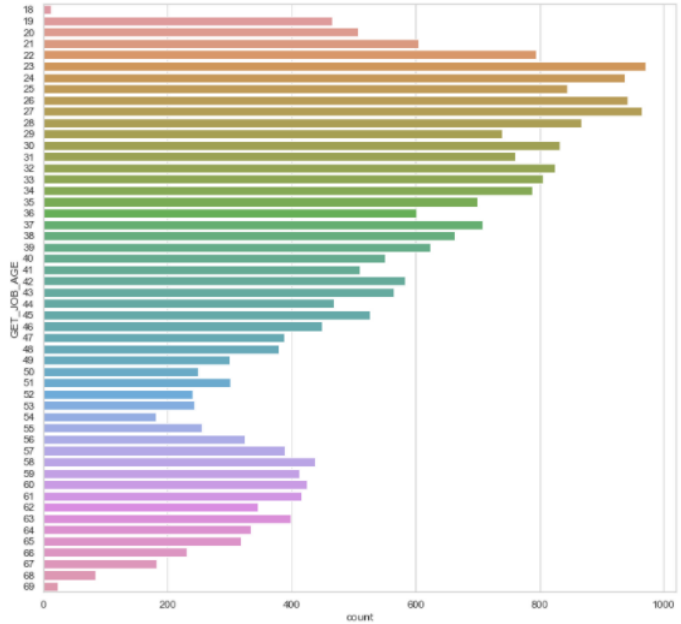
3. 데이터 전처리

파생변수 생성– 수치형 데이터

3.5.2 일을 시작한 나이 - 'GET_JOB_AGE' 변수 추가

In [65]:
```
train['GET_JOB_AGE'] = (train['DAYS_BIRTH'] - train['DAYS_EMPLOYED']) // 365 + 1
test['GET_JOB_AGE'] = (test['DAYS_BIRTH'] - test['DAYS_EMPLOYED']) // 365 + 1
```
executed in 14ms, finished 01:03:45 2021-12-02

In [66]:
```
plt.figure(figsize=(12,12))
sns.countplot(y=train['GET_JOB_AGE']);
```
executed in 358ms, finished 01:03:46 2021-12-02



- 보편적인 취업 나이인 20세 초반에서 20세 후반사이까지 가장 많이 분포한다.

3.5.3 날짜 단위 별 파생변수 생성 - 'days_employed_year', 'days_employed_month', 'days_employed_week', 'begin_year'

- DAYS_EMPLOYED: 업무 시작일
  - 데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 하루 전부터 일을 시작함을 의미
  - 양수 값은 고용되지 않은 상태를 의미함
- begin_month: 신용카드 발급 월
  - 데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 한 달 전에 신용카드를 발급함을 의미

In [67]:
```
# age생성과 마찬가지로
train['begin_year'] = train['begin_month'] // 12 + 1 # 카드 발급 연 단위를 월 단위가 12개월 안일 때를 1년 기준으로 잡음

train['DAYS_EMPLOYED_YEAR'] = train['DAYS_EMPLOYED'] // 365 + 1 # 고용 일수도 마찬가지로 반올림해줌
train['DAYS_EMPLOYED_MONTH'] = train['DAYS_EMPLOYED'] // 30 + 1
train['DAYS_EMPLOYED_WEEK'] = train['DAYS_EMPLOYED'] // 7 + 1
```
executed in 13ms, finished 01:03:46 2021-12-02

In [68]:
```
# age생성과 마찬가지로
test['begin_year'] = test['begin_month'] // 12 + 1 # 카드 발급 연 단위를 월 단위가 12개월 안일 때를 1년 기준으로 잡음

test['DAYS_EMPLOYED_YEAR'] = test['DAYS_EMPLOYED'] // 365 + 1 # 고용 일수도 마찬가지로 반올림해줌
test['DAYS_EMPLOYED_MONTH'] = test['DAYS_EMPLOYED'] // 30 + 1
test['DAYS_EMPLOYED_WEEK'] = test['DAYS_EMPLOYED'] // 7 + 1
```
executed in 14ms, finished 01:03:46 2021-12-02

3. 데이터 전처리

파생변수 생성– 수치형 데이터

3.5.4 'income total'은 현재 연차의 연간 소득을 의미, 내가 일한 기간에 대해 나누었을 때 평균 기간에 대한 소득을 구할 수 있음

In [69]:
```
train['INCOME_EMPLOYED_RATIO_DAY'] = train['income_total'] / train['DAYS_EMPLOYED'] # 일 기준
train['INCOME_EMPLOYED_RATIO_WEEK'] = train['income_total'] / train['DAYS_EMPLOYED_WEEK'] # 주 기준
train['INCOME_EMPLOYED_RATIO_MONTH'] = train['income_total'] / train['DAYS_EMPLOYED_MONTH'] # 월 기준
train['INCOME_EMPLOYED_RATIO_YEAR'] = train['income_total'] / train['DAYS_EMPLOYED_YEAR'] # 연 기준
```
executed in 14ms, finished 01:03:46 2021-12-02

In [70]:
```
test['INCOME_EMPLOYED_RATIO_DAY'] = test['income_total'] / test['DAYS_EMPLOYED'] # 일 기준
test['INCOME_EMPLOYED_RATIO_WEEK'] = test['income_total'] / test['DAYS_EMPLOYED_WEEK'] # 주 기준
test['INCOME_EMPLOYED_RATIO_MONTH'] = test['income_total'] / test['DAYS_EMPLOYED_MONTH'] # 월 기준
test['INCOME_EMPLOYED_RATIO_YEAR'] = test['income_total'] / test['DAYS_EMPLOYED_YEAR'] # 연 기준
```
executed in 14ms, finished 01:03:46 2021-12-02

3.5.5 가족수 - 자식수 -> 실질적으로 경제적 활동을 하는 사람 수

In [71]:
```
train['economical_people'] = train['family_size'] - train['child_num']
test['economical_people'] = test['family_size'] - test['child_num']
```
executed in 12ms, finished 01:03:46 2021-12-02

3.5.6 income을 가족 수 및 자식 수로 나눈 비율

- 가족 1명당 연간 소득이 소비되는 수치 확인 -> 부양 능력?이라고 해석해도 될듯하다.
- 자식 1명당 연간 소득이 소비되는 수치 확인 -> 교육비 지출 능력?

In [72]:
```
train['INCOME_fam_RATIO'] = train['income_total'] / train['family_size']
train['INCOME_child_num_RATIO'] = train['income_total'] / train['child_num']
```
executed in 14ms, finished 01:03:46 2021-12-02

In [73]:
```
test['INCOME_fam_RATIO'] = test['income_total'] / test['family_size']
test['INCOME_child_num_RATIO'] = test['income_total'] / test['child_num']
```
executed in 14ms, finished 01:03:46 2021-12-02

3.5.7 연간수입을 살아온 날에 대한 비율 -> 개인의 밥벌이 능력이라고 해석?

In [74]:
```
train['income_per_days_birth'] = train['income_total'] / train['DAYS_BIRTH']  # 일 단위
train['income_per_age'] = train['income_total'] / train['age']  # 연 단위
```
executed in 13ms, finished 01:03:46 2021-12-02

In [75]:
```
test['income_per_days_birth'] = test['income_total'] / test['DAYS_BIRTH']  # 일 단위
test['income_per_age'] = test['income_total'] / test['age']  # 연 단위
```
executed in 14ms, finished 01:03:46 2021-12-02

3. 데이터 전처리

파생변수 생성– 값 확인 및 이상치 제거

In [76]:
```
new_col = ['age', 'GET_JOB_AGE', 'begin_year', 'DAYS_EMPLOYED_YEAR', 'DAYS_EMPLOYED_MONTH', 'DAYS_EMPLOYED_WEEK',
           'INCOME_EMPLOYED_RATIO_DAY', 'INCOME_EMPLOYED_RATIO_WEEK',
          'INCOME_EMPLOYED_RATIO_MONTH', 'INCOME_EMPLOYED_RATIO_YEAR',
          'INCOME_fam_RATIO', 'INCOME_child_num_RATIO', 'economical_people',
           'income_per_days_birth', 'income_per_age']
train[new_col].head()
```

executed in 29ms, finished 01:03:46 2021-12-02

index	age	GET_JOB_AGE	begin_year	DAYS_EMPLOYED_YEAR	DAYS_EMPLOYED_MONTH	DAYS_EMPLOYED_WEEK	INCOME_E
0	39	26	1.0	13	157	673	43.002761
1	32	27	1.0	5	52	221	160.714286
2	53	41	2.0	13	148	634	101.488498
3	42	36	4.0	6	70	299	96.797323
4	42	36	3.0	6	71	301	74.821853

- INCOME_child_num_RATIO에서 inf 값들이 보인다
- 아마 자식이 없는 사람들이 대다수 분포했으므로 분모가 0으로 나누어져서 이런 결과가 나온 것 같다.

In [78]:
```
train.loc[train['INCOME_child_num_RATIO'] == np.inf, 'INCOME_child_num_RATIO'] = 0
test.loc[test['INCOME_child_num_RATIO'] == np.inf, 'INCOME_child_num_RATIO'] = 0
```

executed in 14ms, finished 01:03:46 2021-12-02

- 자식이 없으면 교육비 지출을 하지 않을 것으로 판단되어 0으로 대체

In [79]:
```
train['INCOME_EMPLOYED_RATIO_DAY'].value_counts()
```

executed in 14ms, finished 01:03:46 2021-12-02

```
inf          4438
91.836735      39
72.862694      26
116.666667     24
100.000000     24
               ...
91.911765       1
26.020155       1
92.879257       1
12.556679       1
79.960513       1
Name: INCOME_EMPLOYED_RATIO_DAY, Length: 6181, dtype: int64
```

3. 데이터 전처리

파생변수 생성– 값 확인 및 이상치 제거

In [79]:
```
train['INCOME_EMPLOYED_RATIO_DAY'].value_counts()
```

executed in 14ms, finished 01:03:46 2021-12-02

```
inf          4438
91.836735      39
72.862694      26
116.666667     24
100.000000     24
               ...
91.911765       1
26.020155       1
92.879257       1
12.556679       1
79.960513       1
Name: INCOME_EMPLOYED_RATIO_DAY, Length: 6181, dtype: int64
```

- 여기도 inf값이 많이 잡힌다
- 무직인 사람들, days_employed의 값이 0으로 대체되어졌던 것들이 분모로 사용된 것 같다. 다시 확인해보자.

In [80]:
```
train.loc[train['DAYS_EMPLOYED'] == 0,['occyp_type', 'age', 'income_total', 'DAYS_EMPLOYED']]
test.loc[test['DAYS_EMPLOYED'] == 0,['occyp_type', 'age', 'income_total', 'DAYS_EMPLOYED']]
```

executed in 14ms, finished 01:03:46 2021-12-02

index	occyp_type	age	income_total	DAYS_EMPLOYED
26457	No job or No info	61	112500.0	0
26464	No job or No info	56	141750.0	0
26467	No job or No info	58	90000.0	0
26470	No job or No info	59	90000.0	0
26471	No job or No info	63	202500.0	0
...	...	...	...	...
36425	No job or No info	56	135000.0	0
36438	No job or No info	65	135000.0	0
36445	No job or No info	54	117000.0	0
36449	No job or No info	63	180000.0	0
36451	No job or No info	65	131400.0	0

1697 rows × 4 columns

- age와 같이 확인해보니 대부분 직장에서 은퇴한 나이대이다.
- 또한 현재 일을하고 있지 않아서 일한 일수에 대한 데이터가 반영되지 않았을 수 도 있다.
- 단순 유추해보았을 때 연간소득의 근원이 직접 일하면서 받는 돈이 아닌 다른 방식(연금)으로 지급되는 것 같다.
- 연간소득을 일한기간으로 나누어 기간 당 소득을 구하는게 목적이였으므로 이 경우에는 0으로 대체해주어야겠다.

In [81]:
```
train.loc[train['INCOME_EMPLOYED_RATIO_DAY'] == np.inf, 'INCOME_EMPLOYED_RATIO_DAY'] = 0
test.loc[test['INCOME_EMPLOYED_RATIO_DAY'] == np.inf, 'INCOME_EMPLOYED_RATIO_DAY'] = 0
```

executed in 14ms, finished 01:03:46 2021-12-02

3. 데이터 전처리

파생변수 생성– 값 확인 및 이상치 제거

In [82]:
```
train['economical_people'].value_counts()
```
executed in 10ms, finished 01:03:46 2021-12-02

```
 2.0    20331
 1.0     6120
 0.0        5
-1.0        1
Name: economical_people, dtype: int64
```

- 분명 'economical_people'은 (가족 전체 구성원의 수 - 자식의 수)의 값인데 0은 부모가 없거나 or 잘못된 값, -1은 아예 잘못된 값인 것 같다. 확인해보자.

In [83]:
```
train.loc[train['economical_people'] <= 0,] # 잘못된 값 삭제
```
executed in 29ms, finished 01:03:46 2021-12-02

index	gender	car	reality	child_num	income_total	income_type	edu_type	family_type	house_type	DAYS_BIRTH	...	DAYS_EMPLOYED_WEEK	INCOME
5825	M	Y	Y	1	450000.0	Commercial associate	Secondary / secondary special	Single / not married	House / apartment	18173	...	97	663.71681
14900	M	Y	N	2	225000.0	Working	Secondary / secondary special	Married	House / apartment	14776	...	317	101.7179(
16110	F	N	Y	1	108000.0	Working	Secondary / secondary special	Single / not married	House / apartment	12723	...	162	95.40636(

In [84]:
```
train = train[train['economical_people'] >= 1]
test = test[test['economical_people'] >= 1]
train['economical_people'].value_counts()
```
executed in 14ms, finished 01:03:46 2021-12-02

```
2.0    20331
1.0     6120
Name: economical_people, dtype: int64
```

3. 데이터 전처리

파생변수 생성– 범주형 데이터

3.6 파생변수 생성 - 범주형 데이터

- 유의미하다고 생각되는 파생변수 리스트 -> 나중에 인사이트와도 연관성을 고려하여 선정함

3.6.1 자녀가 있는 가정 vs 없는 가정

- 데이터 출처가 중국
- 중국에선 자녀의 제한을 두는 법률이 존재
  - ex) 실제 자녀가 3명 이상임에도 법적으로 3명 이하로 신고했을 가능성이있다.
  - 그렇다면, 자녀의 수는 신뢰할만한 데이터가 아니다.
- 자녀가 있는데 없다고 거짓 신고할 가능성은 현저히 낮다는 가정하에, 자녀의 유무에 따른 차이가 있을까 싶어 파생변수를 만들었다.

```
train['child'] = 0
test['child'] = 0
```

executed in 14ms, finished 01:03:46 2021-12-02

```
train.loc[train.child_num >= 1,'child'] = 1
test.loc[test.child_num >= 1,'child'] = 1
train.child.value_counts()
```

executed in 14ms, finished 01:03:46 2021-12-02

```
0    18340
1     8111
Name: child, dtype: int64
```

3. 데이터 전처리

파생변수 생성– 범주형 데이터

3.6.2 맞벌이 부부 vs 외벌이 부부 컬럼 생성 (economical_people 이용)

- https://prism45.tistory.com/270
- 맞벌이 부부는 외벌이 부부보다 수입이 많기에 과소비가 많다는 근거가 있는 자료

In [87]:
```
# 맞벌이 부부 = 1, 외벌이 부부 = 0
train['dual_income'] = train['economical_people'] - 1
test['dual_income'] = test['economical_people'] - 1
```

executed in 14ms, finished 01:03:46 2021-12-02

In [88]:
```
train['dual_income'].value_counts()
```

executed in 14ms, finished 01:03:46 2021-12-02

```
1.0    20331
0.0     6120
Name: dual_income, dtype: int64
```

3. 데이터 전처리

파생변수 생성– 범주형 데이터

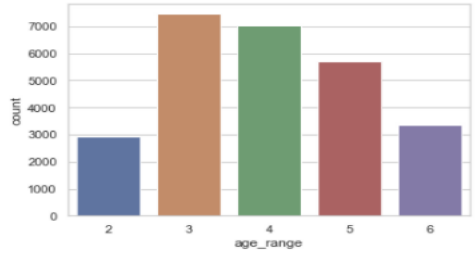
3.6.3 'age_range' 세대별 컬럼 생성 - 20대, 30대...60대

In [89]:
```
train['age_range'] = train['age']
test['age_range'] = test['age']
```
executed in 14ms, finished 01:03:46 2021-12-02

In [90]:
```
train.loc[(train['age_range'] < 30) & (train['age_range'] >= 20),'age_range']= 2
train.loc[(train['age_range'] < 40) & (train['age_range'] >= 30),'age_range']= 3
train.loc[(train['age_range'] < 50) & (train['age_range'] >= 40),'age_range']= 4
train.loc[(train['age_range'] < 60) & (train['age_range'] >= 50),'age_range']= 5
train.loc[(train['age_range'] < 70) & (train['age_range'] >= 60),'age_range']= 6
```
executed in 14ms, finished 01:03:46 2021-12-02

In [91]:
```
test.loc[(test['age_range'] < 30) & (test['age_range'] >= 20),'age_range']= 2
test.loc[(test['age_range'] < 40) & (test['age_range'] >= 30),'age_range']= 3
test.loc[(test['age_range'] < 50) & (test['age_range'] >= 40),'age_range']= 4
test.loc[(test['age_range'] < 60) & (test['age_range'] >= 50),'age_range']= 5
test.loc[(test['age_range'] < 70) & (test['age_range'] >= 60),'age_range']= 6
```
executed in 14ms, finished 01:03:46 2021-12-02

In [92]:
```
sns.countplot(x=train['age_range']);
```
executed in 88ms, finished 01:03:46 2021-12-02



“3-2

Feature engineering

1. Feature Scaling

2. Feature Encoding

3. 데이터 전처리

Feature engineering – Scaling

3.7 수치형 데이터 feature scaling

- 추후 범주형 데이터 전처리와 스케일링의 편의성을 위해 컬럼을 확실하게 구분해놔야 할 것 같다.

In [98]:
```
len(train.columns)
```
executed in 14ms, finished 22:26:49 2021-11-30

```
36
```

In [98]:
```
cat_vars = ['gender', 'car', 'reality', 'income_type', 'edu_type', 'family_type', 'house_type', 'occyp_type'] # 라벨링을 진행해야 되는 범주형
labeled_vars = ['phone', 'work_phone', 'email', 'child', 'dual_income', 'age_range'] # 라벨링이 끝난 범주형 변수
numeric_vars = ['income_total', 'economical_people', 'family_size', 'DAYS_BIRTH', 'child_num', 'DAYS_EMPLOYED', 'DAYS_EMPLOYED_WEEK', 'DAY
                'DAYS_EMPLOYED_YEAR', 'begin_month', 'begin_year', 'age', 'GET_JOB_AGE', 'INCOME_EMPLOYED_RATIO_DAY', 'INCOME_EMPLOYED_RAT
                'INCOME_EMPLOYED_RATIO_MONTH', 'INCOME_EMPLOYED_RATIO_YEAR', 'INCOME_fam_RATIO', 'INCOME_child_num_RATIO',
                'income_per_days_birth', 'income_per_age'] # 스케일링을 진행해야 되는 수치형 변수
target_var = ['credit'] # 타겟 변수
```
executed in 6ms, finished 01:04:35 2021-12-02

In [100]:
```
len(cat_vars) + len(labeled_vars) + len(numeric_vars) + len(target_var)
```
executed in 14ms, finished 22:26:49 2021-11-30

```
36
```

In [101]:
```
train[numeric_vars].describe()
```
executed in 59ms, finished 22:26:49 2021-11-30

click to expand output; double click to hide output

	income_total	economical_people	family_size	DAYS_BIRTH	child_num	DAYS_EMPLOYED
count	2.645100e+04	26451.000000	26451.000000	26451.000000	26451.000000	26451.000000
			2.197119	15958.384976	0.428490	2198.712034
std	1.018741e+05	0.421717	0.916643	4201.877382	0.747307	2370.361309
min	2.700000e+04	1.000000	1.000000	7705.000000	0.000000	0.000000
25%	1.215000e+05	2.000000	2.000000	12446.000000	0.000000	407.000000
50%	1.575000e+05	2.000000	2.000000	15552.000000	0.000000	1539.000000
75%	2.250000e+05	2.000000	3.000000	19431.000000	1.000000	3153.000000
max	1.575000e+06	2.000000	20.000000	25152.000000	19.000000	15713.000000

8 rows × 21 columns

- 보다시피 수치형 컬럼별로 값들의 편차가 제각각이다
- 일단 값이 제일 큰 income_total의 컬럼은 log scale을 통해 진행하고 분포를 보자.

3. 데이터 전처리

Feature engineering – Scaling

3.7.1 'income_total'을 로그 변환 했을 때의 분포 확인

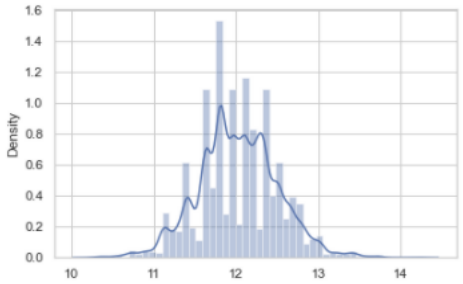
In [102]:
```
import copy
```
executed in 14ms, finished 22:26:49 2021-11-30

In [103]:
```
train_copy = copy.deepcopy(train) # 수치형 변수는 스케일링을 수시로 바꿔서 지표를 확인해야 하므로 카피를 만들겠다.
test_copy = copy.deepcopy(test)
```
executed in 14ms, finished 22:26:49 2021-11-30

In [104]:
```
amount_log = np.log1p(train_copy['income_total'])
train_copy['income_total'] = amount_log
```
executed in 14ms, finished 22:26:49 2021-11-30

In [105]:
```
amount_log = np.log1p(test_copy['income_total'])
test_copy['income_total'] = amount_log
```
executed in 13ms, finished 22:26:49 2021-11-30

In [106]:
```
sns.distplot(x=train_copy['income_total']);
```
executed in 283ms, finished 22:26:49 2021-11-30



3. 데이터 전처리

Feature engineering – Scaling

3.7.2 'income_total'로부터 파생된 변수들 중 mean값이 큰 변수들에 대한 로그 변환 했을 때의 분포 확인

In [107]:
```
train[numeric_vars].columns
```
executed in 14ms, finished 22:26:49 2021-11-30

In [108]:
```
# 컬럼의 평균값이 2만이 넘는 income_total로 부터 파생된 변수
big_mean_cols = ['INCOME_EMPLOYED_RATIO_WEEK', 'INCOME_EMPLOYED_RATIO_MONTH', 'INCOME_EMPLOYED_RATIO_YEAR',
                 'INCOME_fam_RATIO', 'INCOME_child_num_RATIO']
```
executed in 15ms, finished 22:26:49 2021-11-30

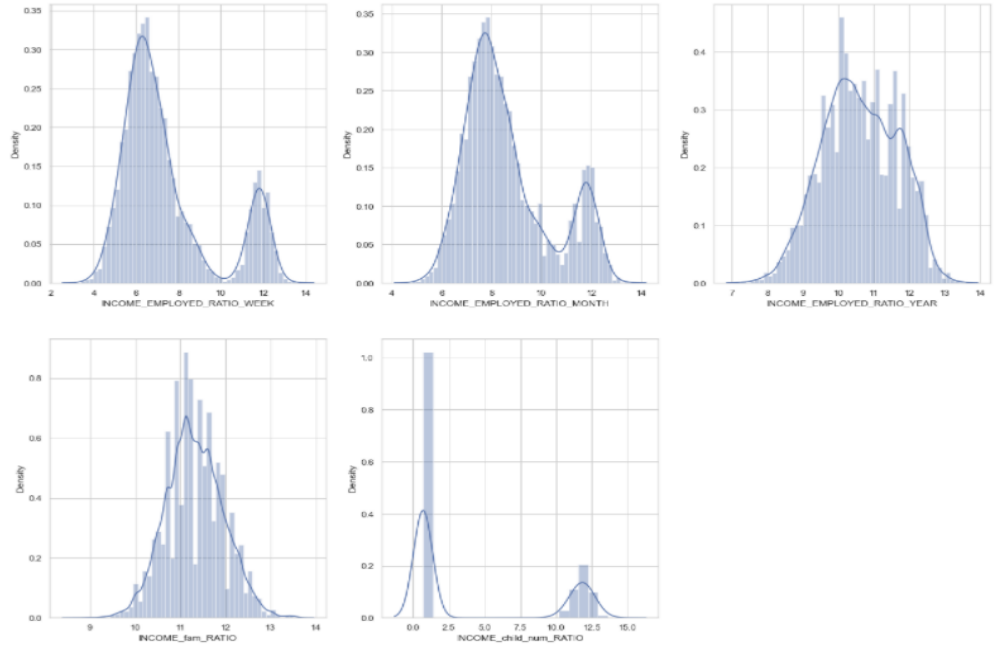
In [109]:
```
for col in train_copy[big_mean_cols].columns:
    amount_log = np.log1p(train_copy[col] + 1)
    train_copy[col] = amount_log
```
executed in 14ms, finished 22:26:49 2021-11-30

In [110]:
```
for col in test_copy[big_mean_cols].columns:
    amount_log = np.log1p(test_copy[col] + 1)
    test_copy[col] = amount_log
```
executed in 14ms, finished 22:26:49 2021-11-30

In [111]:
```
train_copy[big_mean_cols].head()
```
executed in 14ms, finished 22:26:49 2021-11-30

index	INCOME_EMPLOYED_RATIO_WEEK	INCOME_EMPLOYED_RATIO_MONTH	INCOME_EMPLOYED_RATIO_YEAR	INCOME_fam_RATIO	INCOME_child_num_RATIO
0	5.713375	7.163799	9.653674	11.525368	0.693147
1	7.022787	8.468342	10.809768	11.320578	12.419174
2	6.567768	8.020448	10.452111	12.323865	0.693147
3	6.521000	7.970691	10.426795	11.525368	0.693147
4	6.263885	7.705402	10.175497	11.274059	0.693147

In [112]:
```
plt.figure(figsize = (20,16))
for i, column in enumerate(train_copy[big_mean_cols].columns):
    plt.subplot(2,3,i+1)
    sns.set_theme(style='whitegrid')
    sns.distplot(train_copy[column])
plt.show()
```
executed in 1.24s, finished 22:26:50 2021-11-30



- 비교적 고른 것 같지만 그래프로 보았을 때, 몇몇 변수들은 아직 분포가 고르지 않은 느낌이다.
- 정규화가 확실한지는 더 살펴봐야겠다.

3. 데이터 전처리

Feature engineering – Scaling

Kolmogorov-Smirnov test: 누적 확률분포를 이용하여 표본의 확률분포와 모집단의 확률분포 간의 유사성 확인

3.7.3 Kolmogorov-Smirnov 테스트를 통한 정규성 검증

In [113]:
```
from scipy.stats import kstest

result = kstest(train_copy['income_total'], 'norm')
result
```

executed in 13ms, finished 22:26:50 2021-11-30

```
KstestResult(statistic=1.0, pvalue=0.0)
```

In [114]:
```
result = []
for col in train_copy[big_mean_cols].columns:
    value = kstest(train_copy[col], 'norm')
    result.append(value)
result
```

executed in 29ms, finished 22:26:50 2021-11-30

```
[KstestResult(statistic=0.9996381358446139, pvalue=0.0),
 KstestResult(statistic=0.9999991247057177, pvalue=0.0),
 KstestResult(statistic=0.9999999999998004, pvalue=0.0),
 KstestResult(statistic=1.0, pvalue=0.0),
 KstestResult(statistic=0.7558914042144173, pvalue=0.0)]
```

- 값이 모두 0이 나온다...확실하게 수치형 변수를 모두 포함해서 StandardScaler를 적용해보자.

3. 데이터 전처리

Feature engineering – Scaling

3.7.4 Standard Scaling 적용

In [115]:
```
from sklearn.preprocessing import StandardScaler

for col in train_copy[numeric_vars].columns:
    scaler = StandardScaler()
    value = scaler.fit_transform(train_copy[col].values.reshape(-1, 1))
    train_copy[col] = value
```

executed in 119ms, finished 22:26:51 2021-11-30

In [116]:
```
for col in test_copy[numeric_vars].columns:
    scaler = StandardScaler()
    value = scaler.fit_transform(test_copy[col].values.reshape(-1, 1))
    test_copy[col] = value
```

executed in 29ms, finished 22:26:51 2021-11-30

In [117]:
```
train_copy[numeric_vars].head()
```

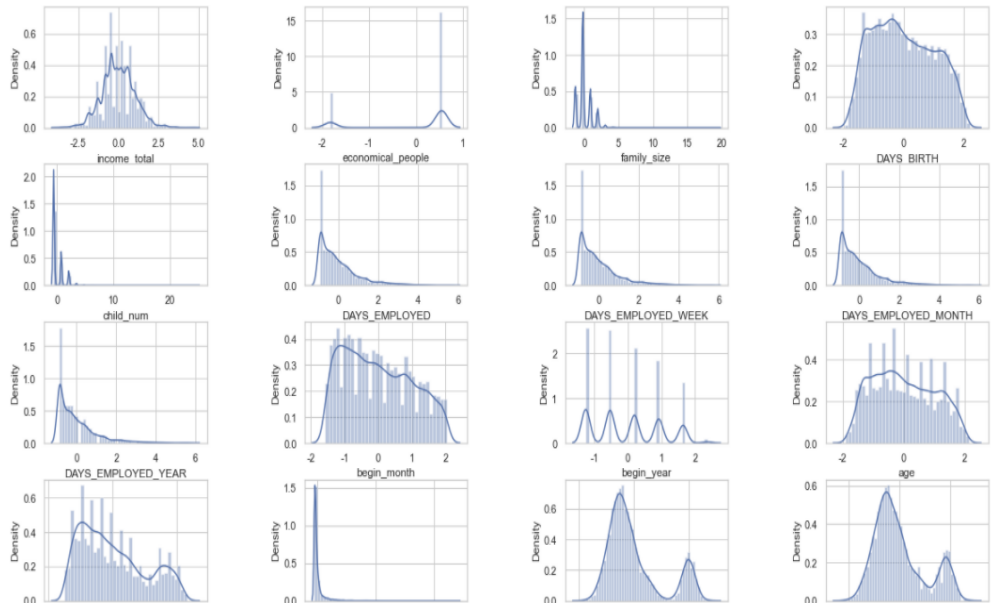
executed in 29ms, finished 22:26:51 2021-11-30

index	income_total	economical_people	family_size	DAYS_BIRTH	child_num	DAYS_EMPLOYED	DAYS_E
0	0.408550	0.548651	-0.215049	-0.490120	-0.573390	1.059052	1.05820
1	0.825905	0.548651	0.875909	-1.089625	0.764773	-0.277900	-0.27690
2	2.069291	0.548651	-0.215049	0.744590	-0.573390	0.943034	0.94300
3	0.408550	0.548651	-0.215049	-0.207146	-0.573390	-0.045020	-0.04651
4	-0.114134	0.548651	-0.215049	-0.219284	-0.573390	-0.039536	-0.04060

5 rows × 21 columns

In [119]:
```
# 분포 확인
plt.figure(figsize = (20,16))
for i, column in enumerate(train_copy[numeric_vars].columns):
    plt.subplot(6,4,i+1)
    sns.set_theme(style='whitegrid')
    sns.distplot(train_copy[column])
plt.subplots_adjust(wspace=0.6, hspace=0.3)
plt.show()
```

executed in 4.68s, finished 22:26:55 2021-11-30



3. 데이터 전처리

Feature engineering – Scaling

In [120]:
```
result = {}
for col in train_copy[numeric_vars].columns:
    value = kstest(train_copy[col], 'norm')
    result[col] = value.pvalue

for k,v in result.items():
    print('{col}의 p-value: {p_value}'.format(col=k, p_value=v))
```

executed in 329ms, finished 22:26:56 2021-11-30

```
income_total의 p-value: 4.81917375978407e-95
economical_people의 p-value: 0.0
family_size의 p-value: 0.0
DAYS_BIRTH의 p-value: 4.970614086079955e-72
child_num의 p-value: 0.0
DAYS_EMPLOYED의 p-value: 0.0
DAYS_EMPLOYED_WEEK의 p-value: 0.0
DAYS_EMPLOYED_MONTH의 p-value: 0.0
DAYS_EMPLOYED_YEAR의 p-value: 0.0
begin_month의 p-value: 4.884307970977191e-154
begin_year의 p-value: 0.0
age의 p-value: 4.4320464974326124e-113
GET_JOB_AGE의 p-value: 5.460388833246204e-239
INCOME_EMPLOYED_RATIO_DAY의 p-value: 0.0
INCOME_EMPLOYED_RATIO_WEEK의 p-value: 0.0
INCOME_EMPLOYED_RATIO_MONTH의 p-value: 0.0
INCOME_EMPLOYED_RATIO_YEAR의 p-value: 1.3361676332358522e-63
INCOME_fam_RATIO의 p-value: 2.9171484608029973e-65
INCOME_child_num_RATIO의 p-value: 0.0
income_per_days_birth의 p-value: 2.1534313512663427e-266
income_per_age의 p-value: 6.42232200855886e-282
```

- 기존부터 편향된 데이터가 존재하는 컬럼에 대해서는 수치가 변하지 않는 것 같다.
- 일단 여기까지 진행하고 모델링을 해봐야될 것 같다.

3. 데이터 전처리

Feature engineering – Encoding

3.8 범주형 데이터 feature encoding

In [122]:
```
train[cat_vars].info()
```
executed in 29ms, finished 22:26:56 2021-11-30

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26451 entries, 0 to 26456
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   gender       26451 non-null  object
 1   car          26451 non-null  object
 2   reality      26451 non-null  object
 3   income_type  26451 non-null  object
 4   edu_type     26451 non-null  object
 5   family_type  26451 non-null  object
 6   house_type   26451 non-null  object
 7   occyp_type   26451 non-null  object
dtypes: object(8)
memory usage: 2.8+ MB
```

In [123]:
```
train[cat_vars].head()
```
executed in 14ms, finished 22:26:56 2021-11-30

index	gender	car	reality	income_type	edu_type	family_type	house_type	occyp_type
0	F	N	N	Commercial associate	Higher education	Married	Municipal apartment	No job or No info
1	F	N	Y	Commercial associate	Secondary / secondary special	Civil marriage	House / apartment	Laborers
2	M	Y	Y	Working	Higher education	Married	House / apartment	Managers
3	F	N	Y	Commercial associate	Secondary / secondary special	Married	House / apartment	Sales staff
4	F	Y	Y	State servant	Higher education	Married	House / apartment	Managers

- edu_type만 unique한 값들이 level이 있고 나머지 컬럼들은 그냥 각기 다른 동등한 level의 값들이다.

3. 데이터 전처리

Feature engineering – Encoding

예측이 아닌 분류 계열의 ML알고리즘을 적용하기 때문에 One-hot encoding대신 label encoding을 진행

3.8.1 Binary인 변수들을 일단 Labelencoder를 통해 숫자로 변환

In [124]:

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
bin_col = ['gender', 'car', 'reality']
for target_col in bin_col:
    display(train[[target_col]].head())
    encoder.fit(train[target_col])
    display(encoder.transform(train[target_col]))
    train[target_col] = encoder.transform(train[target_col])
```

executed in 44ms, finished 22:26:56 2021-11-30

index	gender
0	F
1	F
2	M
3	F
4	F

```
array([0, 0, 1, ..., 0, 1, 0])
```

3.8.3 나머지 범주형 변수들을 ordinal encoding

In [129]:

```
from category_encoders import OrdinalEncoder

remain_cols = ['income_type', 'family_type', 'house_type', 'occyp_type']
for target_col in remain_cols:
    encoder = OrdinalEncoder(cols=target_col)
    display(train[[target_col]].head())
    encoder.fit(train[target_col])
    display(encoder.transform(train[target_col]))
    train[target_col] = encoder.transform(train[target_col])
```

executed in 194ms, finished 22:26:57 2021-11-30

0	Commercial associate
1	Commercial associate
2	Working
3	Commercial associate
4	State servant

index	income_type
0	1
1	1
2	2
3	1
4	3

3. 데이터 전처리

Feature engineering – Encoding

3.8.4 'edu_type'은 순위적 특성이 있으므로 mapping encoding을 통해 따로 스케일링

In [131]:

```
print('edu_type 종류 : ',list(train.edu_type.unique()))
display(train[['edu_type']].head())
edu_order = {
    'Lower secondary' : 0, # 중학교 미만
    'Secondary / secondary special' : 1, #중학교
    'Incomplete higher' : 2, # 고등학교 중퇴
    'Higher education' : 3, # 고등학교 졸업
     'Academic degree' : 4 # 학사 이상
}
train.edu_type = train.edu_type.map(edu_order)
display(train[['edu_type']].head())
```

executed in 29ms, finished 22:26:57 2021-11-30

```
edu_type 종류 :  ['Higher education', 'Secondary / secondary special', 'Incomplete higher', 'Lower secondary', 'Academic degree']
```

index	edu_type
0	Higher education
1	Secondary / secondary special
2	Higher education
3	Secondary / secondary special
4	Higher education

index	edu_type
0	3
1	1
2	3
3	1
4	3

4. 모델링

“4-1

분석 기법 결정

1. 데이터 활용 방안
2. 목표

1. 데이터 활용 방안

문제점 : Test data set에 label 인 “credit”변수가 존재하지 않는다.

- **Y_test가 주어지지 않은 데이터**

- **최초 주어진 train data 내부에서 자체 평가 데이터를 생성 (validation data)**
  - **train : validation = 8 : 2 의 비율로 나누고, “val” 이라고 지칭**

- **Label 변수인 “Credit” 변수의 심각한 불균형**
  - **0,1의 비율이 낮고, 2의 비율이 높음**

2. 목표

주된 목표 : “성능의 최대화”

- **log_loss를 최소화 하는 최적의 모델 탐색**
  - **1. 모델 간 비교**
  - **2. oversampling 적극 활용**
    - **SMOTE 방법**
  - **3. 나누어진 train data를 k-fold로 나누어 cross validation 진행,**
    모델의 최적 하이퍼파라미터를 탐색.

“4-2
GridSearch + 모델별 validation set 예측 성능 지표

1. RandomForest
2. XGBoost
3. LGBM
4. CatBoost

0. GridSearchCV

4가지 모델의 하이퍼파라미터 최적화.

- GridSearchCV를 이용

<LGBM 예시>

```
from sklearn.model_selection import GridSearchCV

param_grid = [ {
    'n_estimators' : [800],
    'max_depth' : [15,20,25],
    'num_leaves' : [64,128,256],
    'learning_rate' : [0.05],
    'min_data_in_leaf' : [20,30],
}]

lgbm = LGBMClassifier(objective = 'multiclass', metric = 'multi_logloss', n_jobs = -1)

GridSearch = GridSearchCV(estimator = lgbm, param_grid=param_grid,
                          cv = 5, scoring='neg_log_loss')

GridSearch.fit(X_train,y_train, early_stopping_rounds = 10, eval_set = [(X_val, y_val)])
```

1. RandomForest

Best Model : RandomForestClassifier(max_depth=20, n_estimators=400)

Accuracy, Confusion Matrix, Precision, Recall

<oversampling X>

```
==================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.49      0.16      0.24       644
         1.0       0.67      0.39      0.49      1254
         2.0       0.73      0.93      0.82      3393

    accuracy                           0.71      5291
   macro avg       0.63      0.49      0.52      5291
weighted avg       0.68      0.71      0.67      5291

[[ 103   75  466]
 [  41  487  726]
 [  65  169 3159]]
==================================================
log_loss :  0.7194832644924728
accuracy_score :  0.7085617085617085
```

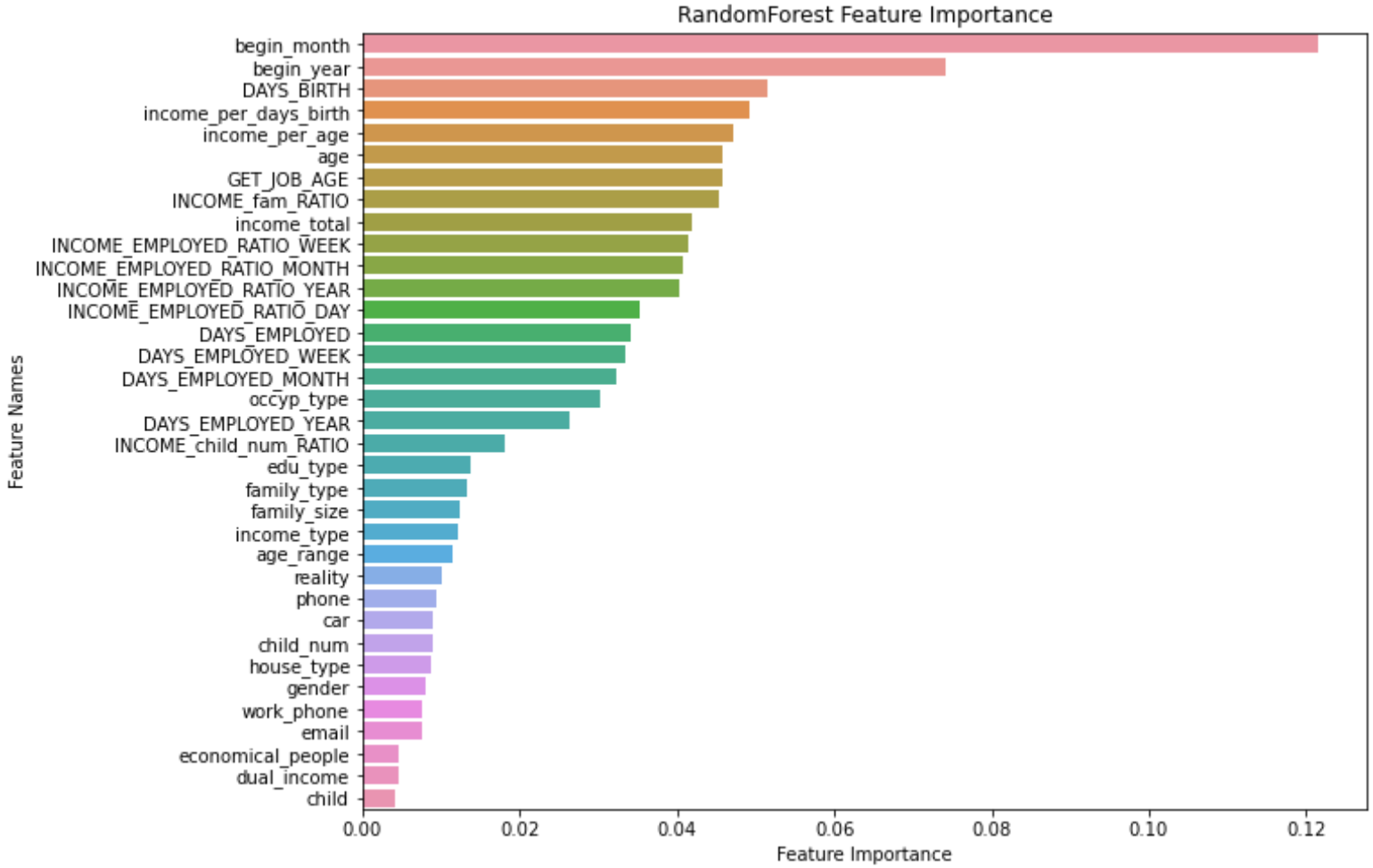
<oversampling O>

```
==================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.83      0.80      0.81       644
         1.0       0.86      0.87      0.87      1254
         2.0       0.93      0.93      0.93      3393

    accuracy                           0.90      5291
   macro avg       0.87      0.87      0.87      5291
weighted avg       0.90      0.90      0.90      5291

[[ 516   31   97]
 [  30 1093  131]
 [  79  142 3172]]
==================================================
log_loss :  0.4333109052992005
accuracy_score :  0.9036099036099036
```

1. RandomForest

Best Model : RandomForestClassifier(max_depth=20, n_estimators=400)

Feature Importance



2. XGBoost

Best Model : XGBClassifier(learning_rate=0.2, max_depth=10, objective='multi:softprob')

Accuracy, Confusion Matrix, Precision, Recall

<oversampling X>

```
==================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.56      0.16      0.25       644
         1.0       0.69      0.39      0.50      1254
         2.0       0.73      0.95      0.82      3393

    accuracy                           0.72      5291
   macro avg       0.66      0.50      0.52      5291
weighted avg       0.70      0.72      0.68      5291

[[ 106   78  460]
 [  40  485  729]
 [  43  138 3212]]
==================================================
log_loss :  0.7194221976093111
accuracy_score :  0.7187677187677187
```

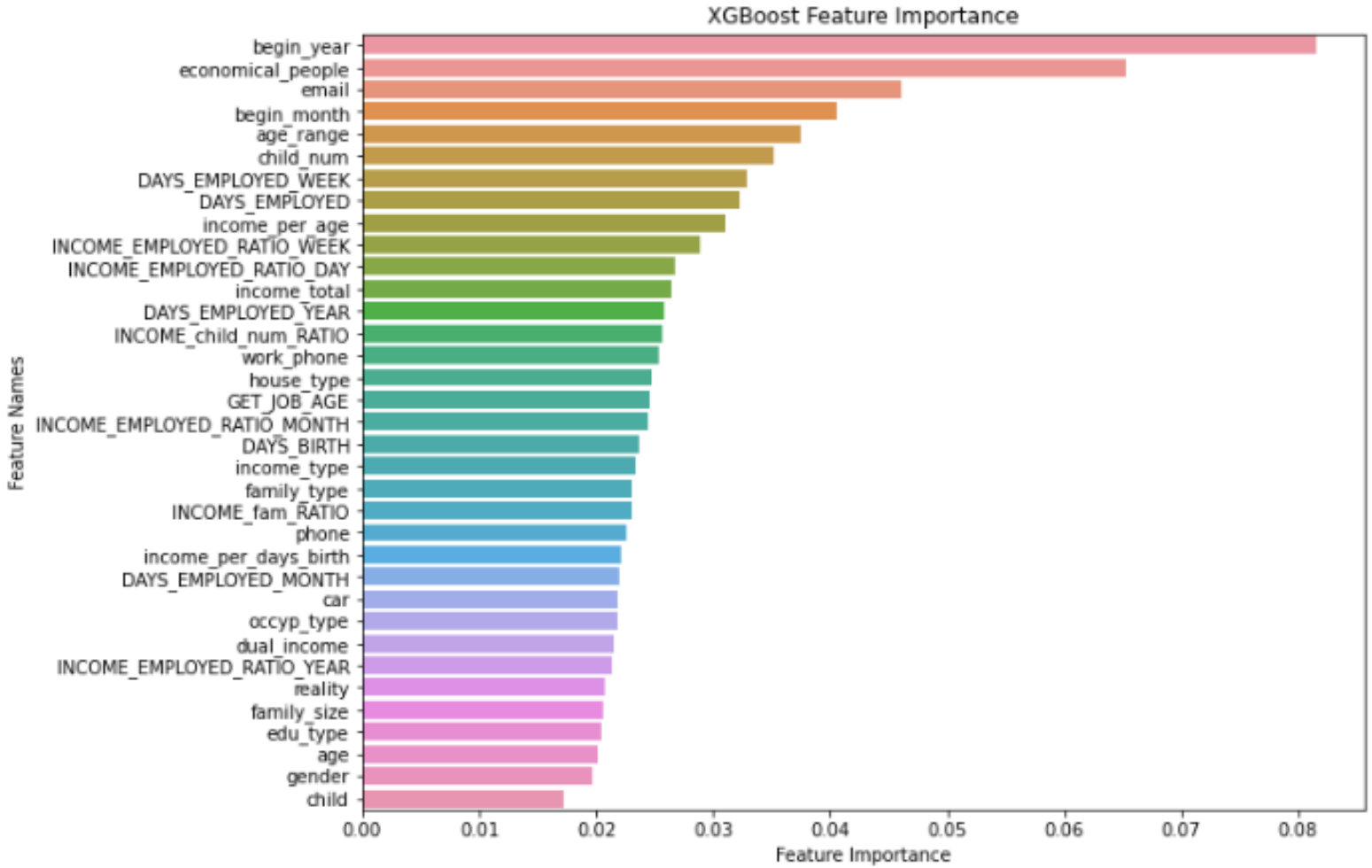
<oversampling O>

```
==================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.86      0.60      0.70       644
         1.0       0.89      0.69      0.77      1254
         2.0       0.85      0.97      0.90      3393

    accuracy                           0.86      5291
   macro avg       0.86      0.75      0.79      5291
weighted avg       0.86      0.86      0.85      5291

[[ 385   35  224]
 [  25  862  367]
 [  40   75 3278]]
==================================================
log_loss :  0.433073006306529
accuracy_score :  0.8552258552258553
```

2. XGBoost

Best Model : XGBClassifier(learning_rate=0.2, max_depth=10, objective='multi:softprob')

Feature Importance



3. LGBM

Best Model : LGBMClassifier(learning_rate=0.05, max_depth=20, ….

Accuracy, Confusion Matrix, Precision, Recall

<oversampling X>

```
================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.57      0.13      0.21       644
         1.0       0.70      0.37      0.49      1254
         2.0       0.72      0.96      0.82      3393

    accuracy                           0.72      5291
   macro avg       0.66      0.49      0.51      5291
weighted avg       0.70      0.72      0.67      5291

[[  82   83  479]
 [  29  470  755]
 [  34  117 3242]]
================================================
log_loss :  0.7170347348687243
accuracy_score :  0.7170667170667171
```

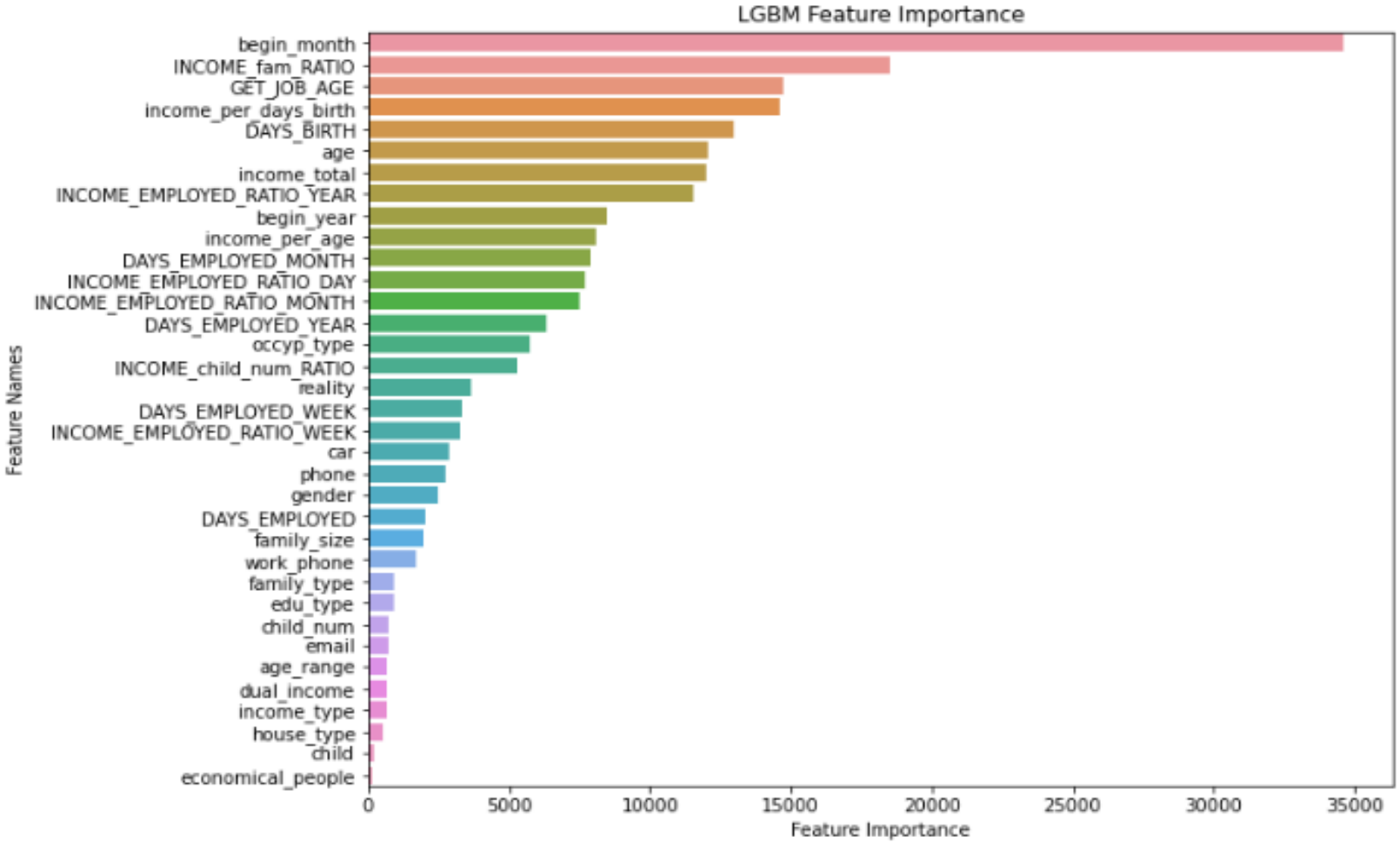
<oversampling O>

```
================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.86      0.71      0.78       644
         1.0       0.89      0.81      0.85      1254
         2.0       0.90      0.96      0.93      3393

    accuracy                           0.89      5291
   macro avg       0.88      0.83      0.85      5291
weighted avg       0.89      0.89      0.89      5291

[[ 459   34  151]
 [  34 1013  207]
 [  41   92 3260]]
================================================
log_loss :  0.33594254081238784
accuracy_score :  0.8943488943488943
```

3. LGBM

Best Model : LGBMClassifier(learning_rate=0.05, max_depth=20, ….

Feature Importance



4. CatBoost

Best Model : Algorithm Choice

Accuracy, Confusion Matrix, Precision, Recall

<oversampling X>

```
================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.60      0.05      0.09       644
         1.0       0.72      0.31      0.43      1254
         2.0       0.71      0.98      0.82      3393

    accuracy                           0.71      5291
   macro avg       0.67      0.44      0.45      5291
weighted avg       0.70      0.71      0.64      5291

[[  31   84  529]
 [  13  383  858]
 [   8   66 3319]]
================================================
log_loss :  0.7503044117969756
accuracy_score :  0.7055377055377056
```

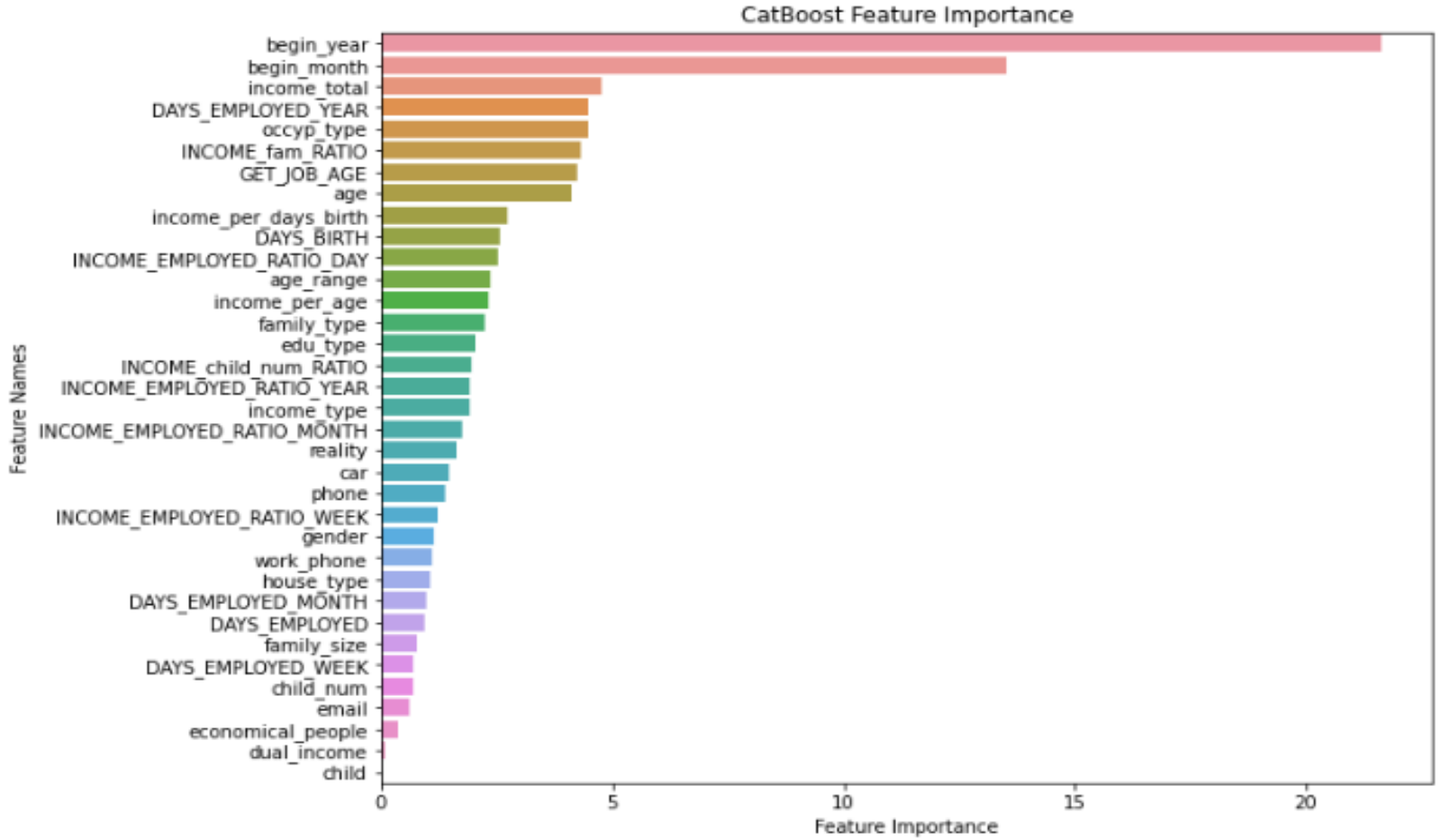
<oversampling O>

```
================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.63      0.30      0.41       644
         1.0       0.76      0.46      0.57      1254
         2.0       0.75      0.94      0.84      3393

    accuracy                           0.75      5291
   macro avg       0.71      0.57      0.61      5291
weighted avg       0.74      0.75      0.72      5291

[[ 196   59  389]
 [  30  579  645]
 [  86  124 3183]]
================================================
log_loss :  0.663499854961278
accuracy_score :  0.7480627480627481
```

4. CatBoost

Best Model : Algorithm Choice

Feature Importance



Results

4가지 모델의 공통점 (Accuracy_results)

1. Oversampling을 했을 때, 0과 1 class의 예측성공 비율이 월등히 나아진다.
   - recall, precision ..

2. Oversampling을 했을 때, 모델 평가 지표로 정했던 log_loss가 낮아진다.

3. Oversampling을 했을 때, 전체 예측 Accuracy가 높아진다.

Results

4가지 모델의 Feature Importance 해석

1. Begin Month, Begin Year 등 신용카드를 발급받은 시점이 중요 feature이다.

2. 만들어낸 파생변수가 상위 순위에 몇몇 보인다.
   - ex) 일을 시작한 나이, 실질적 경제활동 인원 수..

“4-3 Optuna + 모델별 validation set 예측 성능 지표

1. RandomForest
2. XGBoost
3. LGBM
4. CatBoost

0. Optuna

4가지 모델의 하이퍼파라미터 최적화.

- Optuna를 이용

<XGBoost 예시>

```
def objective_xgb(trial: Trial) -> float:
    params_xgb = {
        "random_state": 42,
        "learning_rate": trial.suggest_discrete_uniform('learning_rate', 0.01, 0.1, 0.01),
        "n_estimators": trial.suggest_int('n_estimators', 0, 1000),
        "objective": "multiclass",
        "metric": "multi_logloss",
        "reg_alpha": trial.suggest_float("reg_alpha", 1e-8, 3e-5),
        "reg_lambda": trial.suggest_float("reg_lambda", 1e-8, 9e-2),
        "max_depth": trial.suggest_int("max_depth", 1, 20),
        "colsample_bytree": trial.suggest_discrete_uniform('colsample_bytree', 0.5, 0.9, 0.1),
        "subsample": trial.suggest_discrete_uniform('subsample', 0.5, 0.9, 0.1),
    }
```

원하는 Parameter의 범위를 설정해
range(start,end,step)의 형식으로 입력 할 수 있다.

```
Best Score: 0.691593056951268
Best trial {'learning_rate': 0.02, 'n_estimators': 292, 'reg_alpha': 1.0997191680377813e-05, 'reg_lambda': 0.041046304018833
39, 'max_depth': 16, 'colsample_bytree': 0.5, 'subsample': 0.7}
```

1. RandomForest

< Optuna Parameters >

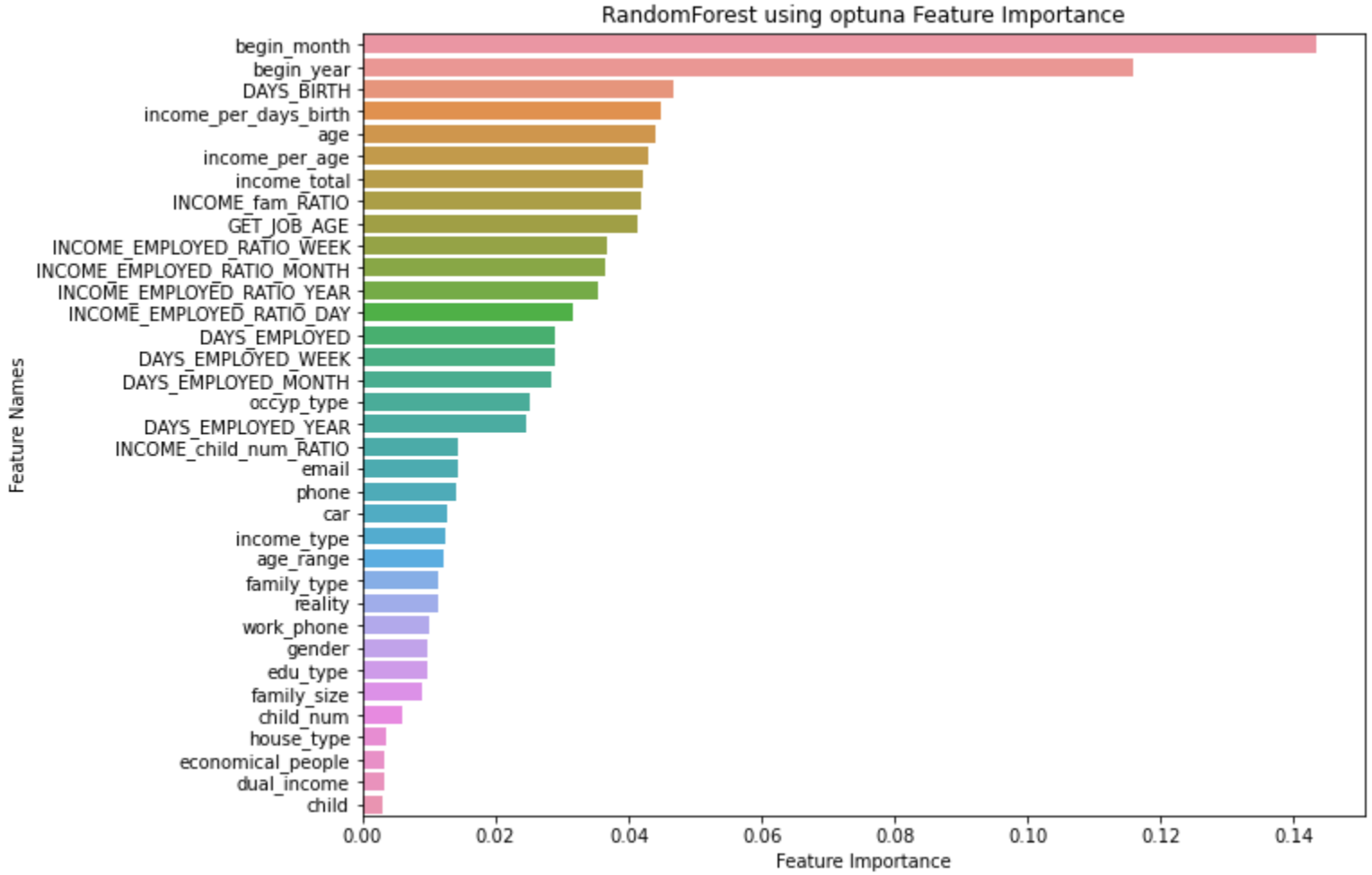
'n_estimators' : 483,
'max_depth' : 40,
'min_samples_split' : 30,
'min_samples_leaf' : 31

< Accuracy Results (Over Sampling) >

```
==================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.49      0.44      0.47       644
         1.0       0.67      0.54      0.60      1254
         2.0       0.79      0.86      0.82      3393

    accuracy                           0.73      5291
   macro avg       0.65      0.62      0.63      5291
weighted avg       0.73      0.73      0.73      5291

[[ 284   88  272]
 [  64  682  508]
 [ 229  242 2922]]
==================================================
log_loss :  0.8151452758697733
accuracy_score :  0.7348327348327348
```

1. RandomForest

< Feature Importance >



2. XGBoost

< Optuna Parameters >

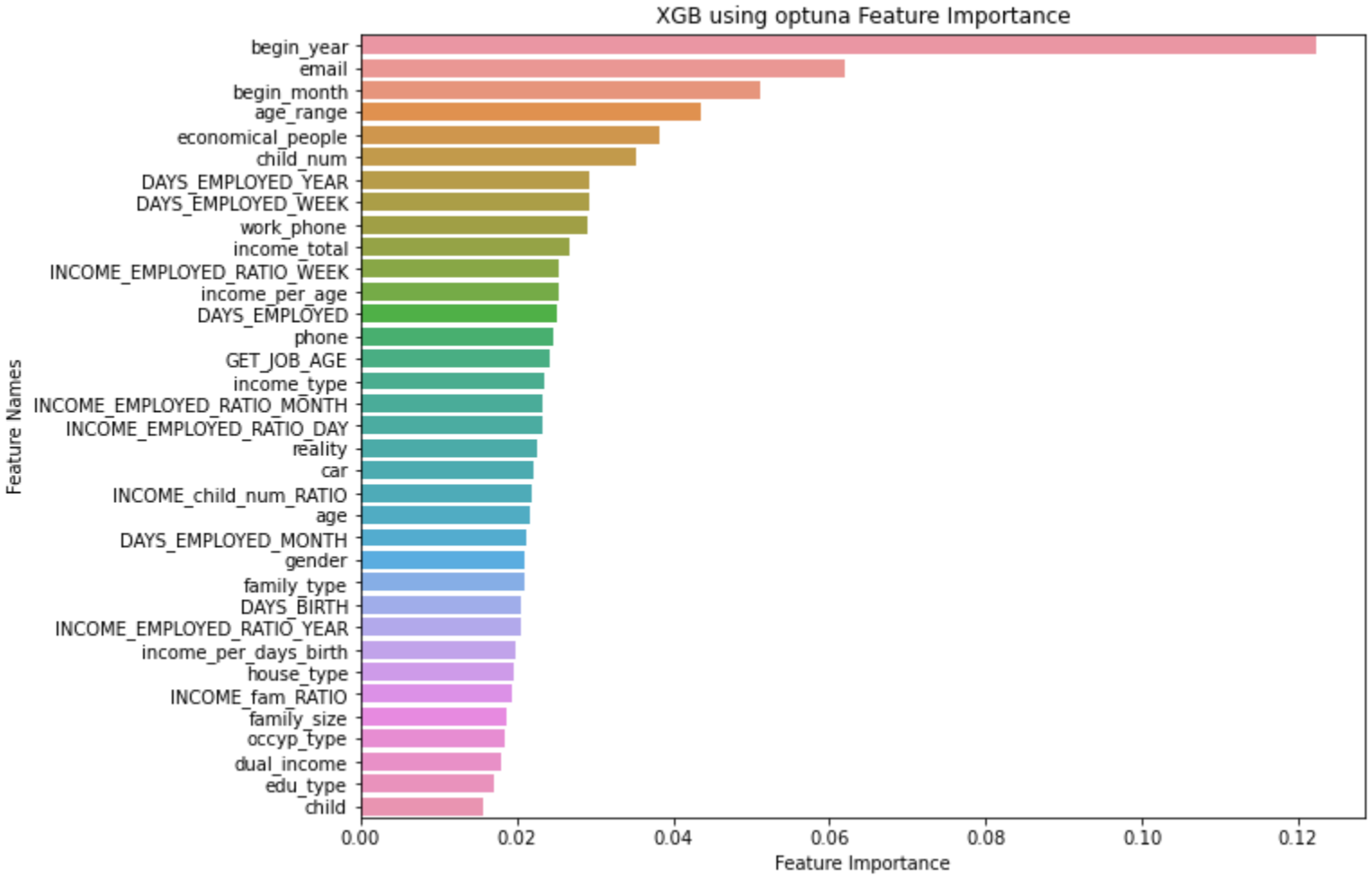
'learning_rate' : 0.02,
'n_estimators' : 292,
'max_depth': 16,
'colsample_bytree': 0.5,
'subsample': 0.7

< Accuracy Results (Over Sampling)>

```
================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.78      0.32      0.45       644
         1.0       0.85      0.48      0.61      1254
         2.0       0.77      0.97      0.86      3393

    accuracy                           0.78      5291
   macro avg       0.80      0.59      0.64      5291
weighted avg       0.79      0.78      0.75      5291

[[ 207   50  387]
 [  23  606  625]
 [  37   61 3295]]
================================================
log_loss :  0.6129271461014243
accuracy_score :  0.7764127764127764
```

2. XGBoost

< Feature Importance >



3. LGBM

< Optuna Parameters >

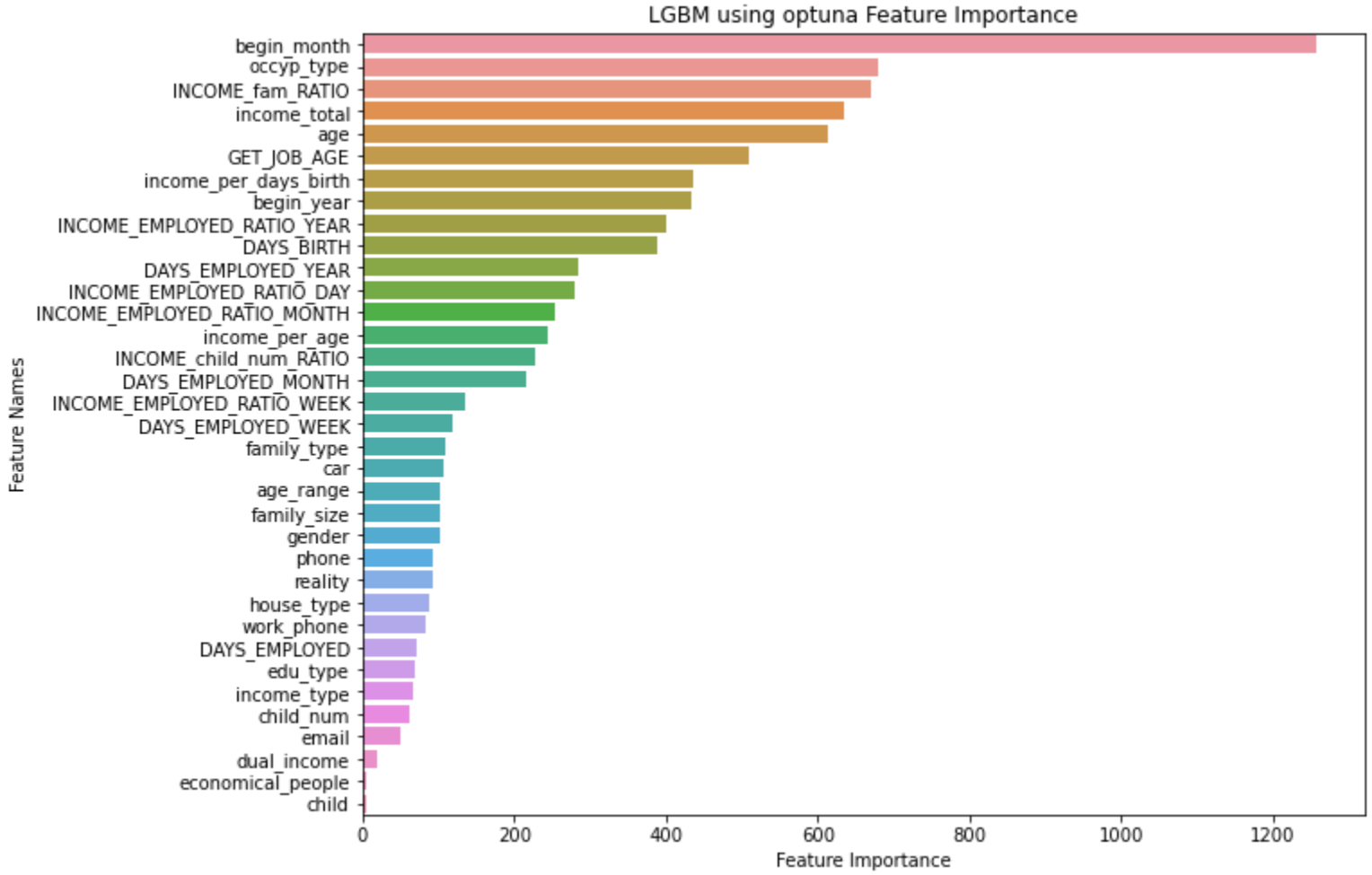
'max_depth': 20,
'num_leaves': 214,
'colsample_bytree': 0.5274034664069657,
'subsample': 0.42727747704497043,
'subsample_freq': 2,
'min_child_samples': 34,
'max_bin': 357

< Accuracy Results (Over Sampling)>

```
==================================================
Validataion set 예측 성능 평가
              precision    recall  f1-score   support

         0.0       0.72      0.18      0.29       644
         1.0       0.79      0.37      0.50      1254
         2.0       0.73      0.97      0.83      3393

    accuracy                           0.73      5291
   macro avg       0.75      0.51      0.54      5291
weighted avg       0.74      0.73      0.69      5291

[[ 117   58  469]
 [  19  462  773]
 [  26   67 3300]]
==================================================
log_loss :  0.7069757067872396
accuracy_score :  0.7331317331317331
```

3. LGBM

< Feature Importance >



Results

1. GridSearch를 했을 때 보다는, 0과 1 class 예측 성공률이 떨어진다.
   - recall, precision ..

2. GridSearch를 했을 때 보다, log_loss가 높아진다.

3. GridSearch 보다 정해주는 Parameters의 수가 많아 진다.

4. 실제 Test set으로 DACON 홈페이지에서 실행 결과, GridSearch보다 좋은 logloss를 보여주는 결과도 있었다

5. 결론

“5-1

결론

1. 분석 결과 해석

분석 결과 해석

1. **Begin Month, Begin Year 등 신용카드를 발급받은 시점이 중요 feature이다.**
   - 신용도에 영향을 주는 변수는 시간 변수일 가능성이 크다
   - 머신러닝 모델 특성상, 단위 당 확률변화 등의 영향은 알 수 없다.

2. **만들어낸 파생변수가 모델에 잘 녹아 들어갔음을 확인했다.**
   - Ex) 일을 시작한 나이, 실질적 경제활동 인원 수..

3. **Oversampling이 불균형한 데이터를 보완해주었다. (validation set 한정)**
   - 불균형한 라벨의 비율을 가진 데이터에서 예측 성능이 상당히 많이 향상되었다.

“5-2

한계점

1. 시간적한계
2. 성능적한계

한계점

1. 성능적 한계

- y_test가 주어지지 않아서, DACON 사이트에 1일 3회 제출을 해 test_score를 확인 할 수 있는 한계점이 존재하여 원하는 만큼의 test_score 확보에 실패한점이 아쉬웠다.

2. 시간적 한계

- GridSearch 혹은 Optuna로 하이퍼파라미터를 적합하는 시간소요가 커서, 여러 조합을 시도하지 못한 것이 아쉬웠다.

감사합니다 :)