



#2021_클라우드 AI 융합전문가 양성과정

모듈 프로젝트 3

3조 : 이호연, 김택완 , 신종유, 신승재



목차

#1, 주제

#2, WBS

#3, EDA

#4, MODEL

Part 1

주제 선정

주제 선정

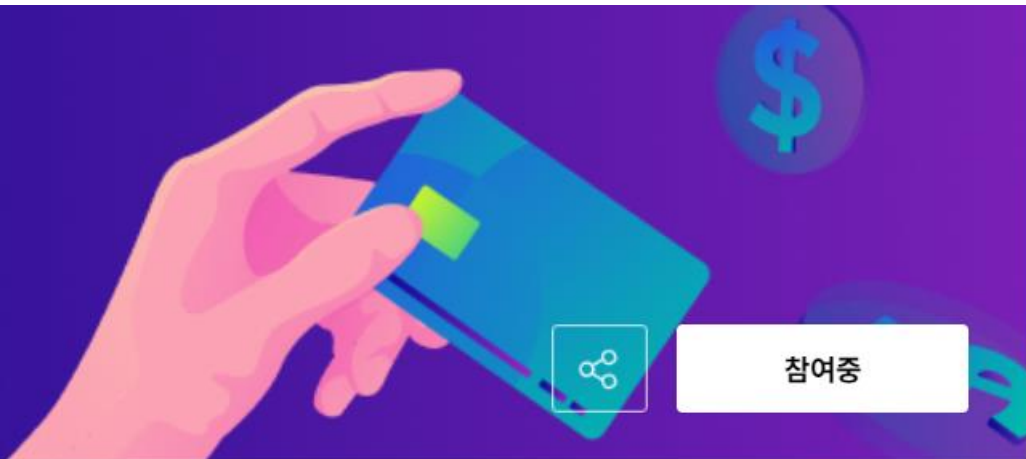
신용카드 사용자 연체 예측 AI 경진대회

월간 데이콘 14 | 금융 | 정형 | Logloss

🏆 상금 : 100만원

🕒 2021.04.05 ~ 2021.05.24 17:59 [+ Google Calendar](#)

👤 2,031명 🏠 마감



1. 주제

신용카드 사용자 데이터를 보고 사용자의 대금 연체 정도를 예측하는 알고리즘 개발

2. 배경

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

Part 2

WBS

WBS

WBS Work Breakdown Structure						
일요일	월요일	화요일	수요일	목요일	금요일	토요일
29		31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

프로젝트 시작 (Jan 8) → 주제 선정 (Jan 9) → 코드 분석 1등 (Jan 10)

역할 분담 후 개인 분석 진행 및 피드백 (Jan 13 - Jan 16) → PPT 준비 (Jan 17)

프로젝트 발표 (Jan 23)

Part 3

EDA

Data type

Column : 19.

gender, car, reality, child_num, income_total, income_type, house_type, DAYS_BIRTH, DAYS_EMPLOYED, FLAG_MOBILE, work_phone, phone, email, occyp_type, family_size, begin_month, credit

Dtypes : float64(4), int64(8), object(8),

Class : DataFrame, Entries : 26457

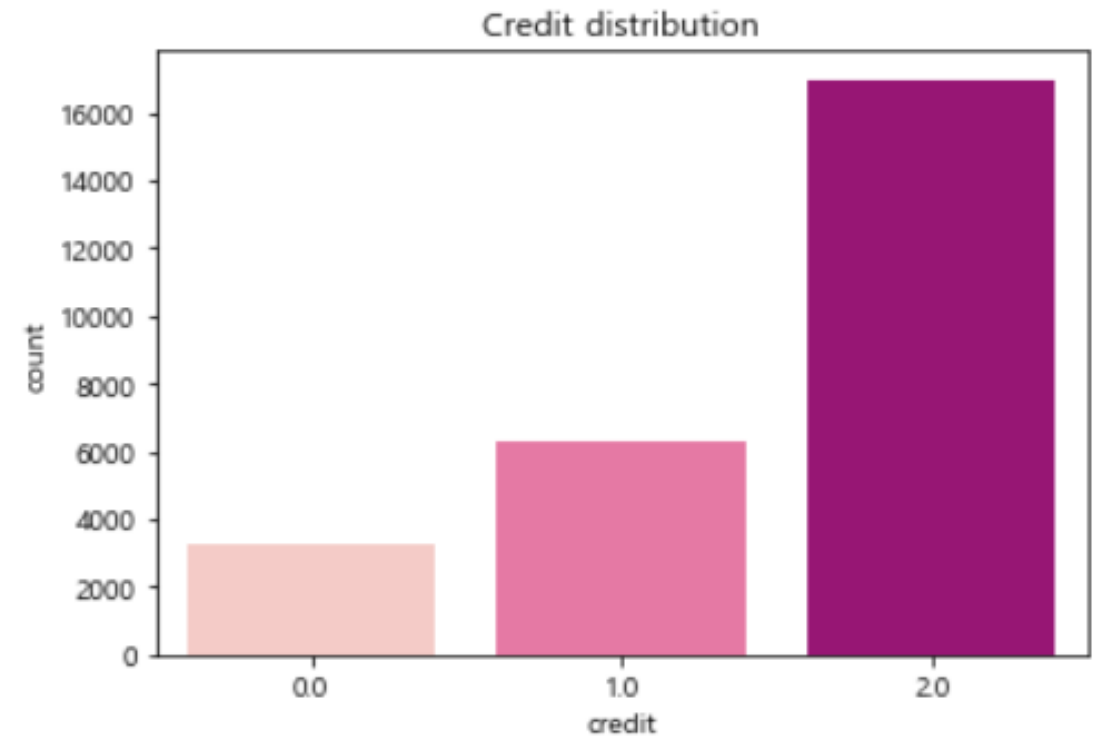
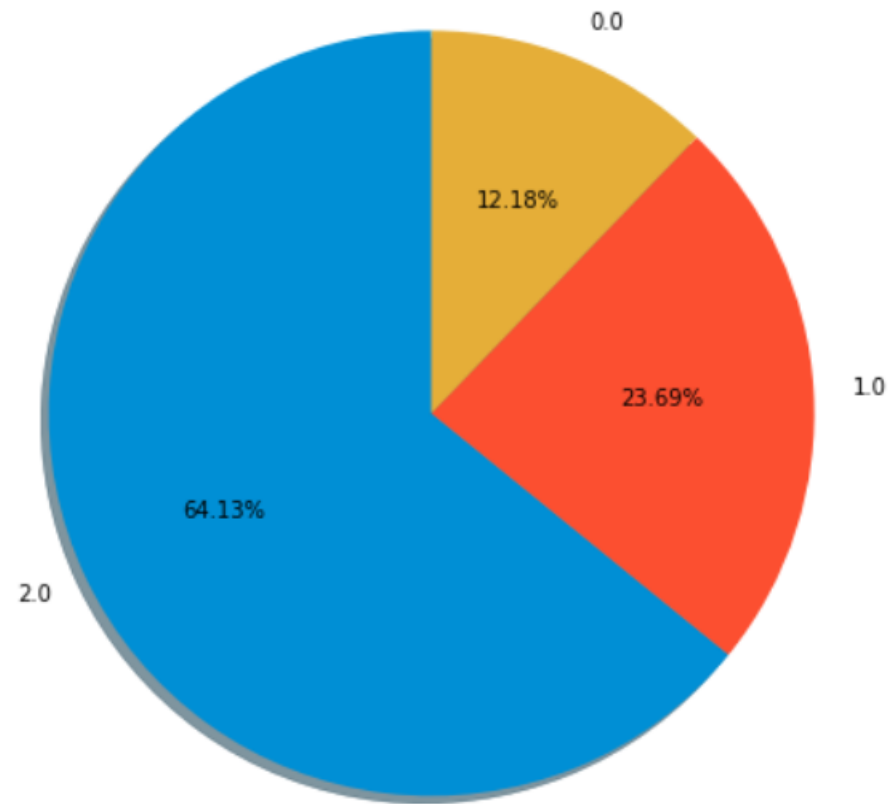
X : non credit columns

.y : credit

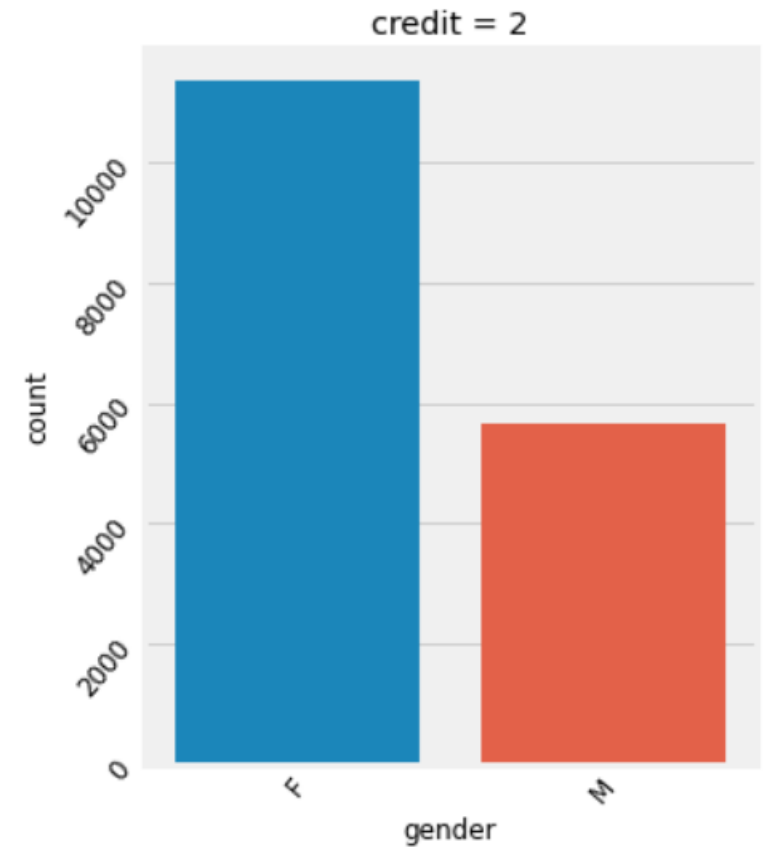
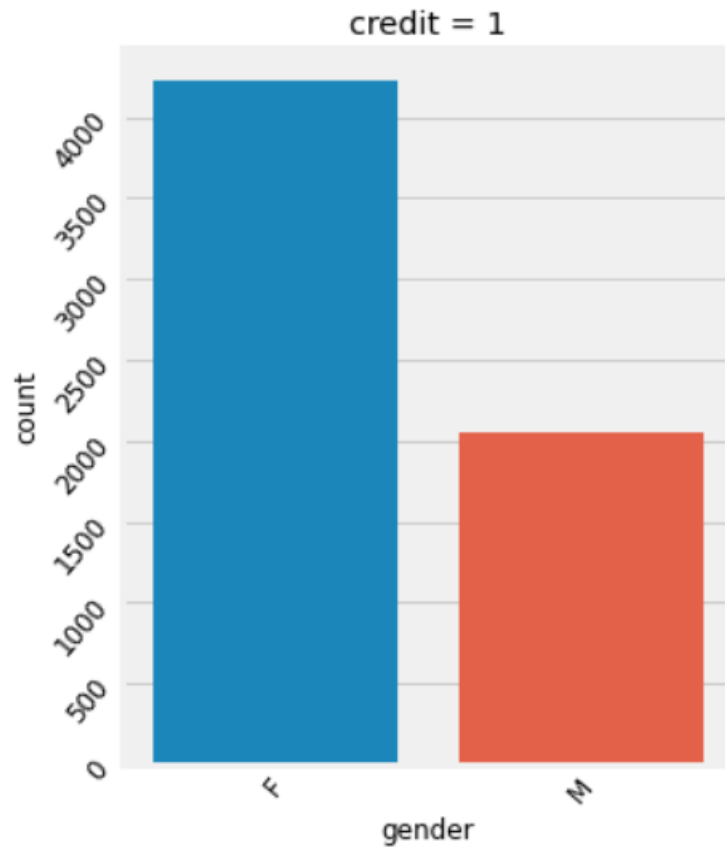
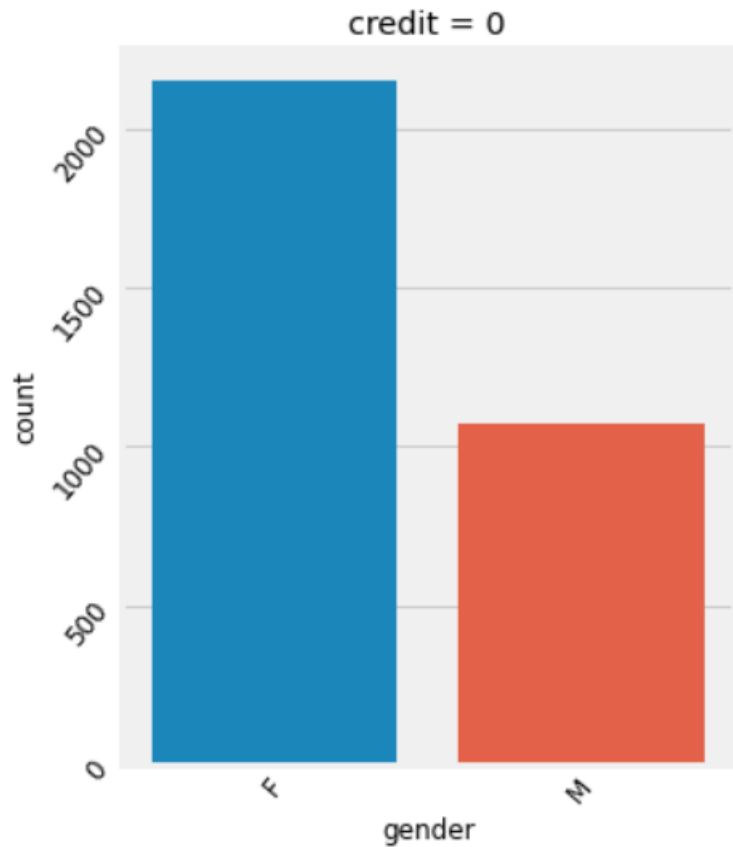
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26457 entries, 0 to 26456
Data columns (total 20 columns):
#   Column                Non-Null Count  Dtype
---  -
0   index                 26457 non-null  int64
1   gender                26457 non-null  object
2   car                   26457 non-null  object
3   reality               26457 non-null  object
4   child_num             26457 non-null  int64
5   income_total          26457 non-null  float64
6   income_type           26457 non-null  object
7   edu_type              26457 non-null  object
8   family_type           26457 non-null  object
9   house_type            26457 non-null  object
10  DAYS_BIRTH            26457 non-null  int64
11  DAYS_EMPLOYED         26457 non-null  int64
12  FLAG_MOBIL            26457 non-null  int64
13  work_phone            26457 non-null  int64
14  phone                 26457 non-null  int64
15  email                 26457 non-null  int64
16  occyp_type            18286 non-null  object
17  family_size           26457 non-null  float64
18  begin_month           26457 non-null  float64
19  credit                26457 non-null  float64
dtypes: float64(4), int64(8), object(8)
memory usage: 4.0+ MB
```

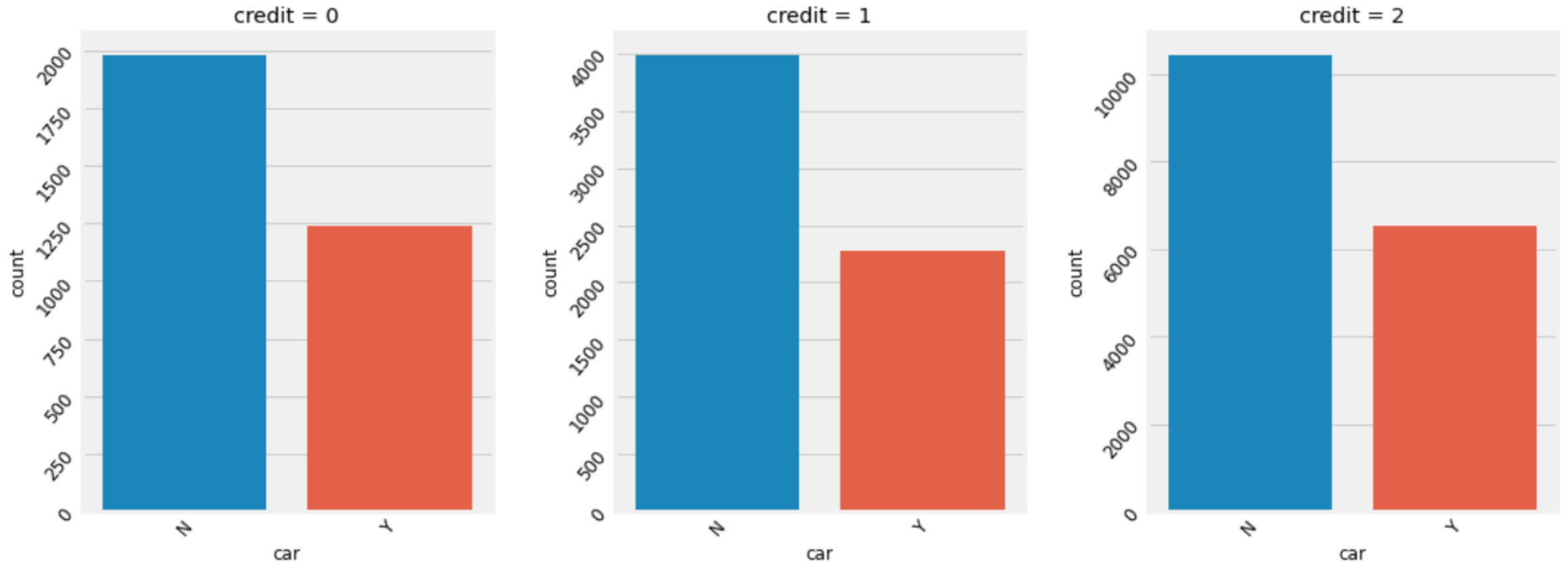

Credit ratio



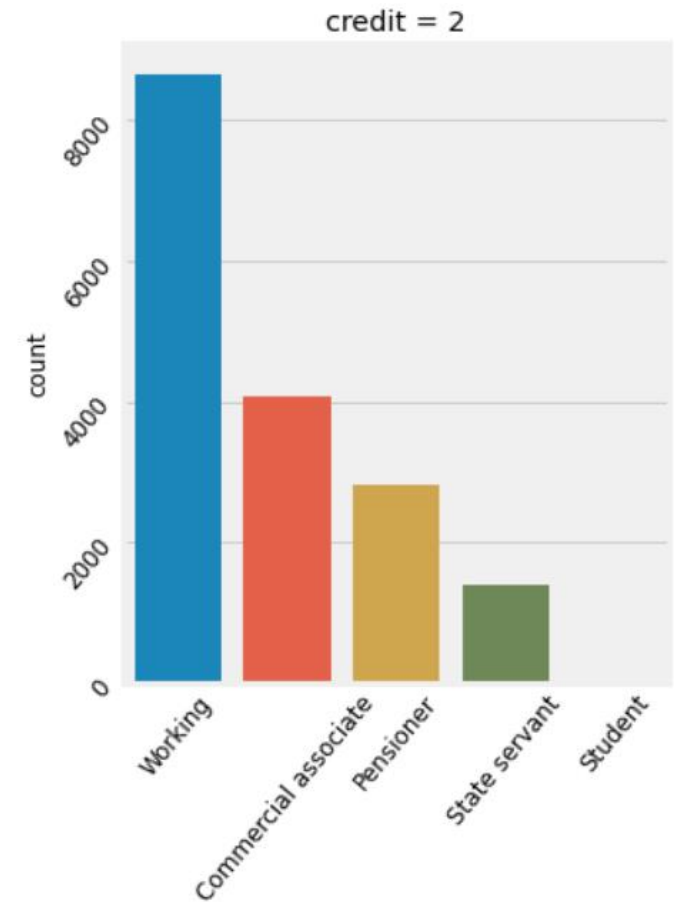
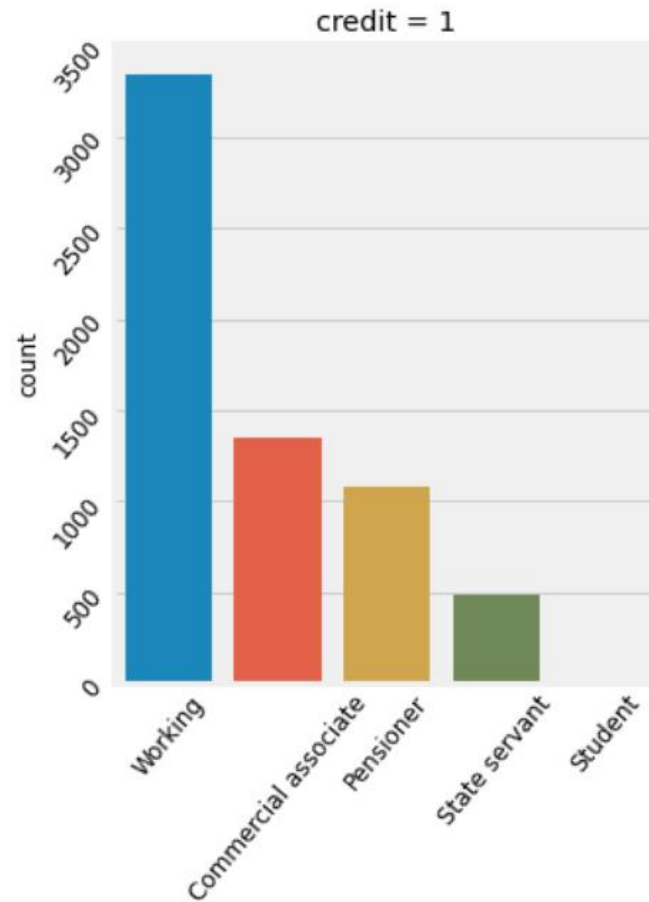
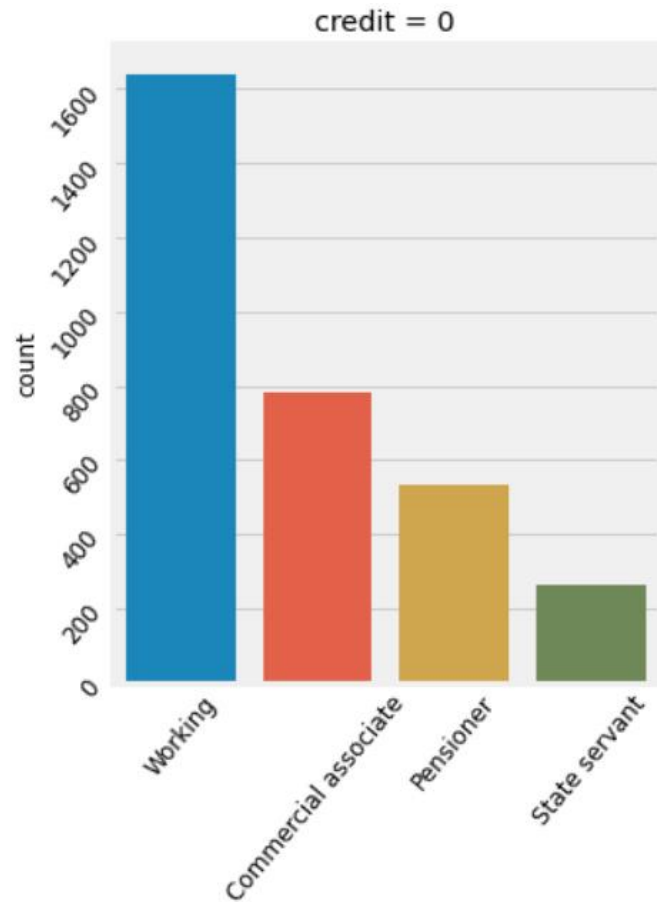
Gender ratio



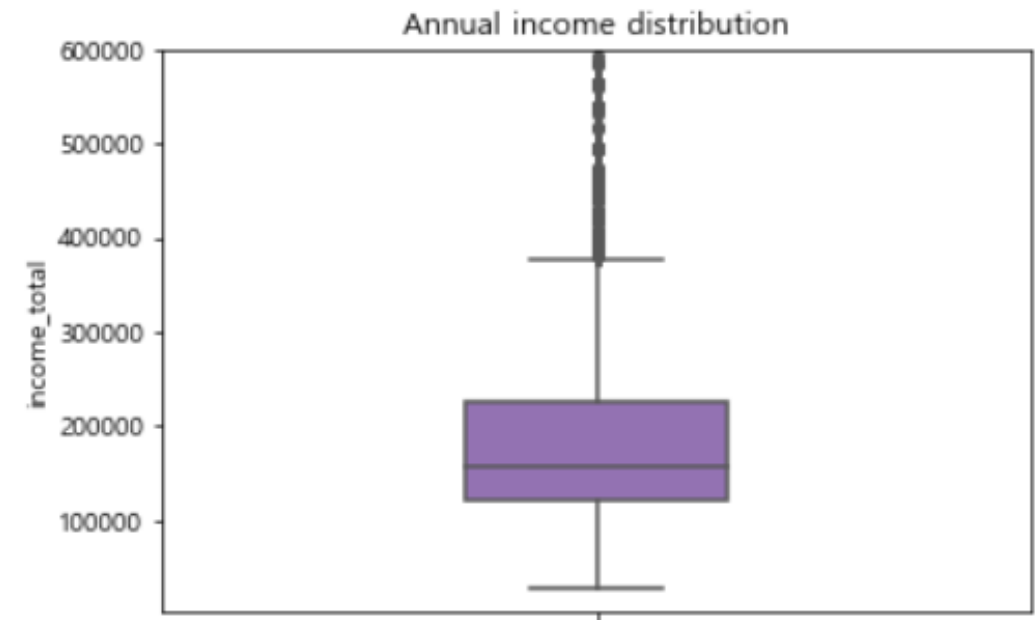
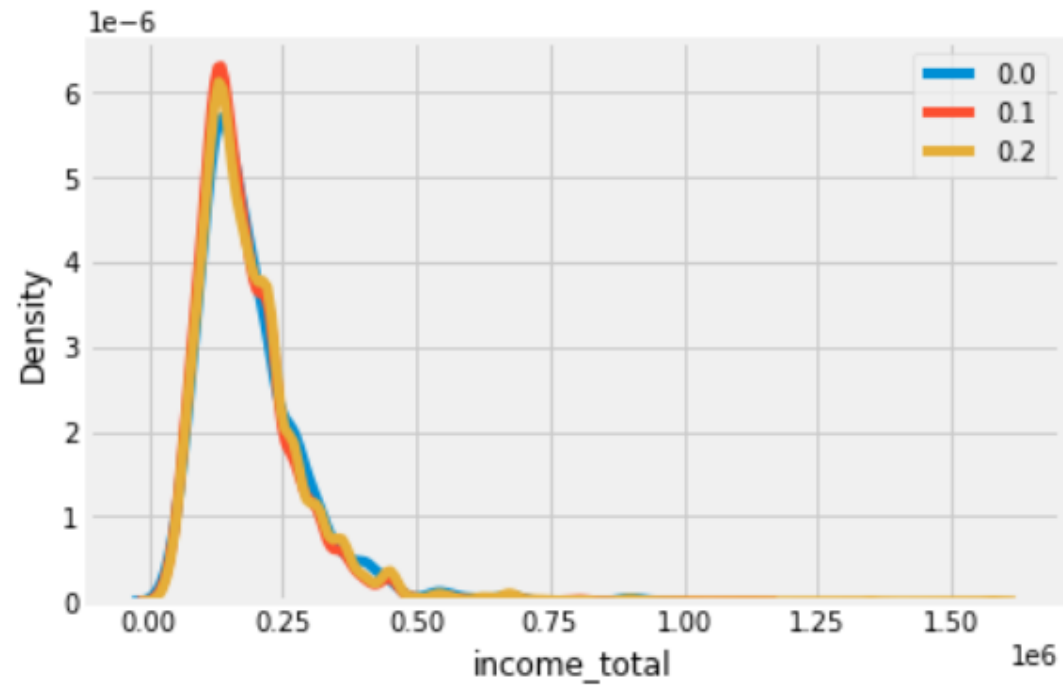
Car ratio



Income type



Income total



Part 4

MODEL

1

XGBoost

2

LGBM

3

RF

4

Catboost

XGBoost

XGBoost

1	사전 작업
2	데이터 상관 관계 분석
3	전처리
4	학습 데이터 생성
5	모델 학습
6	모델 테스트
7	k_fold 교차검증

1. 사전 작업

```
index          0
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
```

사전 데이터 처리

1. null 값 탐색
2. occyp_type Column의 결측치 'NaN' 대입
3. 'object' 형식의 Columns 카테고리화(인코딩)

2. 데이터 상관 관계 분석

다중공선성 분석

데이터 값을 Min Max Scaler로 변환

[] 다중공선성

	index	gender	car	reality	child_num	income_total	income_type	edu_type	family_type	house_type	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL	work_phone	phone	email	occyp_type	family_size	begin_month	credit
0	0.000000	0.0	0.0	0.0	0.000000	0.113372	0.00	0.00	0.00	0.0	0.644982	0.028885	0.0	0.0	0.0	0.0	0.000000	0.052632	0.900000	0.5
1	0.000038	0.0	0.0	1.0	0.052632	0.142442	0.00	0.25	0.25	0.2	0.789362	0.037204	0.0	0.0	0.0	1.0	0.055556	0.105263	0.916667	0.5
2	0.000076	1.0	1.0	1.0	0.000000	0.273256	0.25	0.00	0.00	0.2	0.347624	0.029607	0.0	0.0	1.0	0.0	0.111111	0.052632	0.633333	1.0
3	0.000113	0.0	0.0	1.0	0.000000	0.113372	0.00	0.25	0.00	0.2	0.576833	0.035755	0.0	0.0	1.0	0.0	0.166667	0.052632	0.383333	0.0
4	0.000151	0.0	1.0	1.0	0.000000	0.084302	0.50	0.00	0.00	0.2	0.579756	0.035721	0.0	0.0	0.0	0.0	0.111111	0.052632	0.566667	1.0
...
26452	0.999849	0.0	0.0	0.0	0.105263	0.127907	0.50	0.25	0.00	0.2	0.749298	0.036038	0.0	0.0	0.0	0.0	0.277778	0.157895	0.966667	0.5
26453	0.999887	0.0	0.0	1.0	0.052632	0.098837	0.25	0.00	0.50	0.2	0.565197	0.034749	0.0	0.0	0.0	0.0	0.000000	0.052632	0.216667	1.0
26454	0.999924	0.0	1.0	0.0	0.000000	0.171512	0.25	0.25	0.25	0.4	0.863759	0.035957	0.0	0.0	0.0	0.0	0.277778	0.052632	0.583333	1.0
26455	0.999962	1.0	0.0	1.0	0.000000	0.093023	0.25	0.50	0.75	0.2	0.860148	0.040965	0.0	0.0	0.0	0.0	0.055556	0.000000	0.016667	1.0
26456	1.000000	0.0	0.0	0.0	0.000000	0.034884	0.25	0.25	0.25	0.2	0.319998	0.038587	0.0	0.0	0.0	0.0	0.555556	0.052632	0.850000	1.0

26457 rows × 20 columns

2. 데이터 상관 관계 분석

VIF Factor	
0	3.875224
1	1.839528
2	1.956880
3	3.144501
4	18.547735
5	3.786377
6	7.874293
7	3.080559
8	6.202781
9	4.507897
10	9.478061
11	5.392946
12	NaN
13	1.593469
14	1.579259
15	1.136802
16	2.027587
17	52.536673
18	5.094965
19	5.365185

VIF Factor가 10을 초과할 경우 독립변수간에 강한 상관관계 존재

Column 4 : child_num

Column 17 : family_size

➤ child_num은 family_size에 종속되는 변수이므로 drop

3. 전처리

```

index          26457
gender          2
car             2
reality         2
income_total    249
income_type     5
edu_type        5
family_type     5
house_type      6
DAYS_BIRTH      6621
DAYS_EMPLOYED   3470
FLAG_MOBIL      1
work_phone      2
phone           2
email           2
occyp_type      19
family_size     10
begin_month     61
credit          3
dtype: int64

```

카테고리 요소의 개수를 측정

요소 개수가 한개이거나 열개를 초과한 카테고리 전처리

전처리 List

1. occyp_type
2. FLAG_MOBIL
3. DAYS_BIRTH
4. DAYS_EMPLOYED
5. begin_month

3. 전처리

1. occyp_type

무직(NaN) -> 0
유직(not Nan) -> 1

2. FLAG_MOBIL

모든 값이 1이므로 drop

3. DAYS_BIRTH/
DAYS_EMPLOYED/
begin_month

DAYS는 365/
month는 12로 나누고
내림 연산

∴ 각각 AGE/
WORK_YEARS/
MEMBERSHIP_YEARS
변환

3. 전처리

전처리한 카테고리 요소들의 개수

index	26457
gender	2
car	2
reality	2
income_total	249
income_type	5
edu_type	5
family_type	5
house_type	6
work_phone	2
phone	2
email	2
occyp_type	2
family_size	10
credit	3
AGE	48
WORK_YEARS	44
MEMBERSHIP_YEARS	6
dtype:	int64

4. 학습 데이터 생성

```
[35] from sklearn.model_selection import train_test_split

[36] X_train, X_val, y_train, y_val = train_test_split(train_x, train_y,
                                                    stratify = train_y,
                                                    test_size = 0.25,
                                                    random_state = 10086
                                                    )

[37] print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)

(19842, 17) (6615, 17) (19842,) (6615,)
```

학습 데이터 생성

모델 학습에 이용될 X_train, y_train과
검증에 사용될 X_val, y_val 생성

train_y의 요소별 비율을 고려하기 위하여
stratify = train_y 추가

75%를 학습에 사용, 25%를 검증에 사용함

원활한 복원을 위해 random_state 지정

5. 모델 학습

```
from xgboost import XGBClassifier
xgb_clf = XGBClassifier(n_estimators=30)

params = {'max_depth': [23, 25, 27],
          'min_child_weight': [1],
          'colsample_bytree': [0.125, 0.25, 0.5]}

xgboost = GridSearchCV(xgb_clf, param_grid=params)
```

```
xgboost.fit(
    X_train,
    y_train,
    eval_set=[(X_val, y_val)],
    eval_metric="mlogloss"
)
```

```
xgboost.best_params_
```

```
{'colsample_bytree': 0.5, 'max_depth': 25, 'min_child_weight': 1}
```

GridSerchCV

GridSerchCV를 이용하여 최적의 하이퍼파라미터 값을 찾음

학습

모델을 학습하고 베스트 하이퍼파라미터 값을 출력

6. 모델 테스트

X_val에 대한 예측 확률

y_proba = xgboost.predict_proba(X_val)

```
array([[0.17081177, 0.20069501, 0.62849325],
       [0.21783052, 0.11223634, 0.66993314],
       [0.20400596, 0.3489636 , 0.44703043],
       ...,
       [0.07142152, 0.20591855, 0.72265995],
       [0.19699238, 0.23138784, 0.5716198 ],
       [0.09103429, 0.18831943, 0.72064626]], dtype=float32)
```

y_val

36	2.0
3464	2.0
2640	2.0
22500	1.0
17275	1.0
...	...
6806	1.0
23096	2.0
7756	2.0
23331	2.0
25457	2.0

Name: credit, Length: 6615, dtype: float64



y_val 결과값 정답지

원핫 인코딩

y_val_onehot = pd.get_dummies(y_val)

	0.0	1.0	2.0
3	1	0	0
8	0	0	1
9	0	0	1
14	0	0	1
17	0	0	1
...
26424	0	1	0
26429	0	1	0
26432	0	1	0
26453	0	0	1
26455	0	0	1

5291 rows × 3 columns

결과 예측 및 log_loss

학습한 모델로 결과를 예측하고 log_loss를 평가

각 항목의 예측 값인 y_proba와
y_val을 onehot인코딩한 y_val_onehot을 비교

```
from sklearn.metrics import log_loss
```

```
# 두개가 얼마나 비슷한지 비교하여 성능 평가
log_loss(y_val_onehot, y_proba)
```

0.7807776693123046

7. k_fold 교차검증

```
from sklearn.model_selection import StratifiedKFold
```

```
# fold를 5개로 나누고 앞의 4개의 fold로 학습과 검증을하고 뒤의 1개로 테스트 진행
folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=55)
```

K_fold를 이용한 교차검증

Fold를 나누어 학습과 검증을 하고 테스트를 진행

결과값을 sub에 넣어서 제출

log_loss 변화

데이터 전처리 이전 이후 변화

0.8055361371
0.7976671843



0.7657359723
0.7551075554

```
# sub : test_x에 대한 예측값
# 0으로 된 매트릭스 sub 선언하고, 매트릭스에 각 fold를 돌 때의 요소별 예측값을 더해 채움
outcomes = []
sub = np.zeros((test_x.shape[0],3))
for n_fold, (train_index, val_index) in enumerate(folds.split(train_x, train_y)):
    X_train, X_val = train_x.iloc[train_index], train_x.iloc[val_index]
    y_train, y_val = train_y.iloc[train_index], train_y.iloc[val_index]

    xgb_clf = XGBClassifier(n_estimators=30)
    params = {'max_depth':[19, 21, 23],
              'min_child_weight':[1],
              'colsample_bytree':[0.25, 0.5, 0.75]}
    xgboost = GridSearchCV(xgb_clf, param_grid=params)

    xgboost.fit(
        X_train,
        y_train,
        eval_set=[(X_val, y_val)],
        eval_metric="mlogloss"
    )

    print(xgboost.best_params_)

    predictions = xgboost.predict_proba(X_val)
    y_val_onehot = pd.get_dummies(y_val)
    sub += xgboost.predict_proba(test_x)

    logloss = log_loss(y_val_onehot, predictions)
    outcomes.append(logloss)
    print(f"FOLD {n_fold} : logloss:{logloss}")

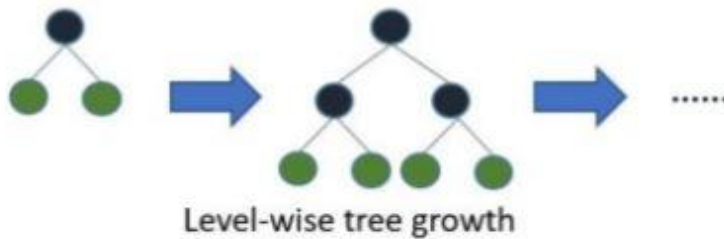
sub = sub / 5

np.mean(outcomes)
```

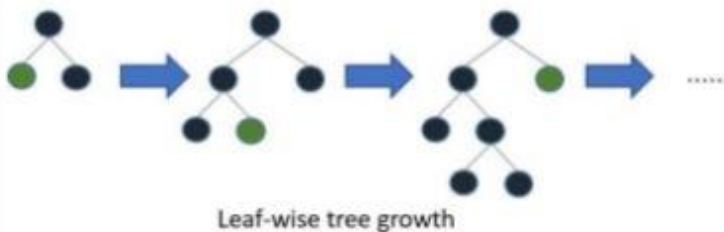
Light BGM

Part 4 Light BGM

XGBoost:



LightGBM:



Leaf-wise

Information Gain : 속성을 선택함으로써 데이터를 더 잘 구분할 수 있는지에 대한 지표

Information Gain을 기준으로 가지를 뺄어나가 학습하는 방식

GOSS(Gradient-based One-Side Sampling)

학습이 안된 데이터는 남겨두고, 학습이 잘된 데이터는 무작위 샘플링을 진행

XGBoost vs Light BGM

Light BGM은 XGBoost에 비해 학습속도가 빠르고 성능이 높을 수 있으나 데이터가 적을 경우 overfitting될 가능성이 큼.

Light BGM

1	사전 작업
2	데이터 상관 관계 분석
3	전처리
4	학습 데이터 생성
5	모델 학습
6	모델 테스트
7	k_fold 교차검증

1~4. XGBoost와 동일한 Features 사용

1. 사전 작업

2. 데이터 분석

3. 전처리

4. 데이터 생성

5. 모델 학습

```
from lightgbm import LGBMClassifier
lightgbm_clf = LGBMClassifier()
lightgbm_clf

params = {'n_estimators': [10, 50, 75, 100, 200],
          'max_depth': [5, 7, 9, 11],
          'min_child_weight': [1, 3],
          'colsample_bytree': [0.01, 0.03, 0.05]}

lightgbm = GridSearchCV(lightgbm_clf, param_grid=params)
```

```
lightgbm.fit(
    X_train,
    y_train,
    eval_set=[(X_train, y_train), (X_val, y_val)],
    eval_metric="logloss"
)
```

```
lightgbm.best_params_

{'colsample_bytree': 0.05,
 'max_depth': 15,
 'min_child_weight': 1,
 'n_estimators': 200}
```

GridSerchCV

GridSerchCV를 이용하여 최적의 하이퍼파라미터 값을 찾음

XGBoost에 비해 학습속도가 빨라 여러가지 파라미터를 실험

학습

모델을 학습하고 베스트 하이퍼파라미터 값을 출력

6. 모델 테스트

```
# X_val에 대한 예측 확률
y_proba = lightgbm.predict_proba(X_val)
```

```
array([[0.10140014, 0.22787187, 0.67072798],
       [0.05873841, 0.24584956, 0.69541203],
       [0.14248679, 0.19884103, 0.65867219],
       ...,
       [0.19486026, 0.22292694, 0.5822128 ],
       [0.11836376, 0.19526743, 0.68636881],
       [0.11504073, 0.20085901, 0.68410025]])
```

```
y_val
36      2.0
3464    2.0
2640    2.0
22500   1.0
17275   1.0
...
6806    1.0
23096   2.0
7756    2.0
23331   2.0
25457   2.0
Name: credit, Length: 6615, dtype: float64
```



```
# y_val 결과값 정답지
# 원핫 인코딩
y_val_onehot = pd.get_dummies(y_val)
```

```
      0.0  1.0  2.0
3      1   0   0
8      0   0   1
9      0   0   1
14     0   0   1
17     0   0   1
...     ...   ...
26424  0   1   0
26429  0   1   0
26432  0   1   0
26453  0   0   1
26455  0   0   1
```

5291 rows × 3 columns

결과 예측 및 log_loss

학습한 모델로 결과를 예측하고 log_loss를 평가

각 항목의 예측 값인 y_proba와
y_val을 onehot인코딩한 y_val_onehot을 비교

```
from sklearn.metrics import log_loss
```

```
# 두개가 얼마나 비슷한지 비교하여 성능 평가
log_loss(y_val_onehot, y_proba)
```

0.8570083697619332

7. k_fold 교차검증

```
from sklearn.model_selection import StratifiedKFold
```

```
# fold를 5개로 나누고 앞의 4개의 fold로 학습과 검증을하고 뒤의 1개로 테스트 진행
folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=55)
```

K_fold를 이용한 교차검증

Fold를 나누어 학습과 검증을 하고 테스트를 진행

결과값을 sub에 넣어서 제출

log_loss 변화

데이터 전처리 이전 이후 변화

0.9003776134
0.8975383195



0.854715292
0.8456383261

```
# sub : test_x에 대한 예측값
# 0으로 된 매트릭스 sub 선언하고, 매트릭스에 각 fold를 돌 때의 요소별 예측값을 더해 채움
outcomes = []
sub = np.zeros((test_x.shape[0],3))
for n_fold, (train_index, val_index) in enumerate(folds.split(train_x, train_y)):
    X_train, X_val = train_x.iloc[train_index], train_x.iloc[val_index]
    y_train, y_val = train_y.iloc[train_index], train_y.iloc[val_index]

    lightgbm_clf = LGBMClassifier(n_estimators=100)
    params = {'max_depth':[19, 21, 23],
              'min_child_weight':[1],
              'colsample_bytree':[0.25, 0.5, 0.75]}
    gridcv = GridSearchCV(lightgbm_clf, param_grid=params)

    gridcv.fit(X_train, y_train)

    print(gridcv.best_params_)
    predictions = lightgbm.predict_proba(X_val)
    y_val_onehot = pd.get_dummies(y_val)

    sub += lightgbm.predict_proba(test_x)

    logloss = log_loss(y_val_onehot, predictions)
    outcomes.append(logloss)
    print(f"FOLD {n_fold} : logloss:{logloss}")

sub = sub / 5

np.mean(outcomes)
```

RF

```
path = '/content/drive/MyDrive/모듈프로젝트3_3조/김택완/DACON_Credit/'
train = pd.read_csv(path + 'train.csv')
test = pd.read_csv(path + 'test.csv')
```

```
train.fillna('NaN', inplace=True)
test.fillna('NaN', inplace=True)
```

```
train = train[(train['family_size'] <= 7)]
train = train.reset_index(drop=True)
```

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

```
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)
```

```
feats = ['DAYS_BIRTH', 'begin_month', 'DAYS_EMPLOYED']
for feat in feats:
    train[feat] = np.abs(train[feat])
    test[feat] = np.abs(test[feat])
```

데이터 전처리

1. Train, Test 데이터의 null 값을 채워주고
2. family_size 가 7이상 데이터 삭제
3. 유의미 하지않은 컬럼(index, mobil(휴대폰소지) 삭제
4. 고용일수가 양수면 고용되지 않은상태를 모두 0으로 맞춤
5. 수집일 기준으로 역셈으로 되어있는 컬럼들 절대값 처리

```

for df in [train, test]:
    # before_EMPLOYED: 고용되기 전까지의 일수
    df['before_EMPLOYED'] = df['DAYS_BIRTH'] - df['DAYS_EMPLOYED']
    df['income_total_beforEMP_ratio'] = df['income_total'] / df['before_EMPLOYED']
    df['before_EMPLOYED_m'] = np.floor(df['before_EMPLOYED'] / 30) - ((np.floor(df['before_EMPLOYED'] / 30) / 12).astype(int) * 12)
    df['before_EMPLOYED_w'] = np.floor(df['before_EMPLOYED'] / 7) - ((np.floor(df['before_EMPLOYED'] / 7) / 4).astype(int) * 4)

    #DAYS_BIRTH 파생변수- Age(나이), 태어난 월, 태어난 주(출생연도의 n주차)
    df['Age'] = df['DAYS_BIRTH'] // 365
    df['DAYS_BIRTH_m'] = np.floor(df['DAYS_BIRTH'] / 30) - ((np.floor(df['DAYS_BIRTH'] / 30) / 12).astype(int) * 12)
    df['DAYS_BIRTH_w'] = np.floor(df['DAYS_BIRTH'] / 7) - ((np.floor(df['DAYS_BIRTH'] / 7) / 4).astype(int) * 4)

    #DAYS_EMPLOYED_m 파생변수- EMPLOYED(근속연수), DAYS_EMPLOYED_m(고용된 달), DAYS_EMPLOYED_w(고용된 주(고용연도의 n주차))
    df['EMPLOYED'] = df['DAYS_EMPLOYED'] // 365
    df['DAYS_EMPLOYED_m'] = np.floor(df['DAYS_EMPLOYED'] / 30) - ((np.floor(df['DAYS_EMPLOYED'] / 30) / 12).astype(int) * 12)
    df['DAYS_EMPLOYED_w'] = np.floor(df['DAYS_EMPLOYED'] / 7) - ((np.floor(df['DAYS_EMPLOYED'] / 7) / 4).astype(int) * 4)

    #ability: 소득/(살아온 일수+ 근무일수)
    df['ability'] = df['income_total'] / (df['DAYS_BIRTH'] + df['DAYS_EMPLOYED'])

    #income_mean: 소득/ 가족 수
    df['income_mean'] = df['income_total'] / df['family_size']

```

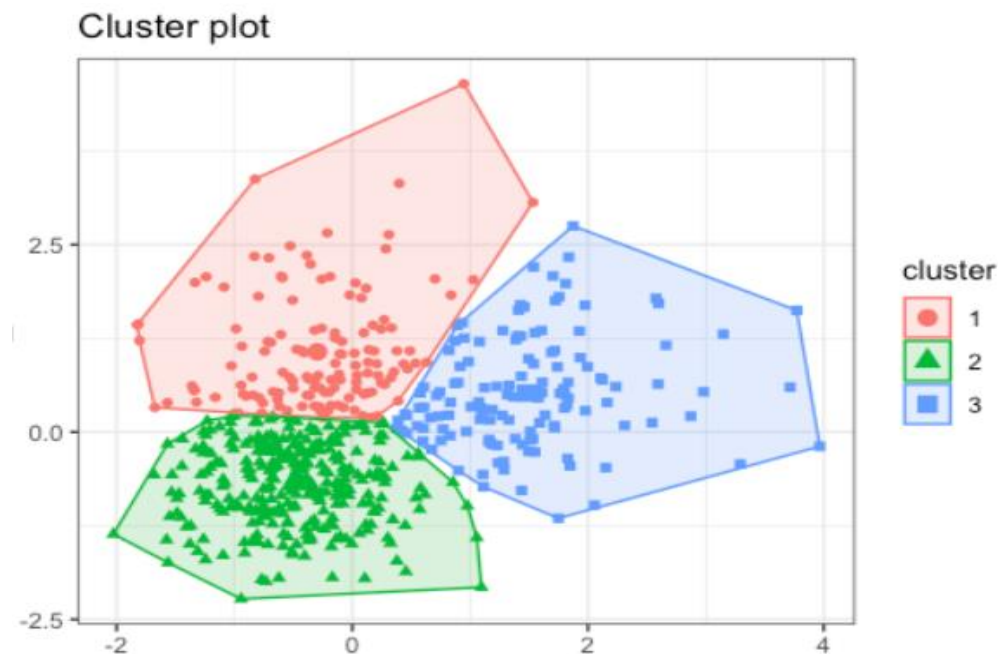
파생 변수 생성

1. 고용되기 전까지 일수
2. 나이
3. 근속연수
4. 고소득 판별 변수
5. 가족부양 능력 변수
6. KMEANS

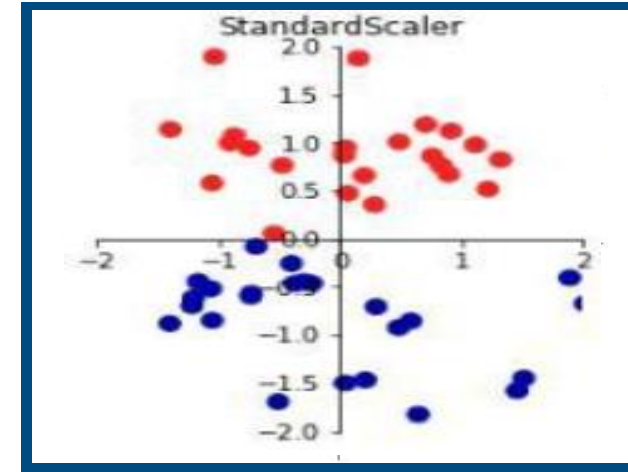
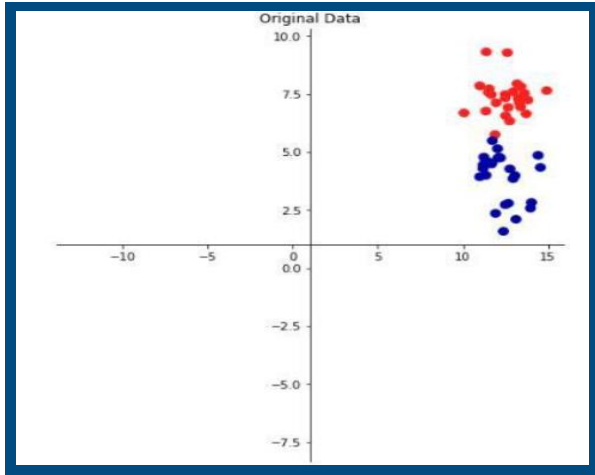
K-MEANS

K – means clustering 알고리즘은 비지도 학습으로
특성이 비슷한 데이터를 군집시켜주는 군집화 알고리즘

K-MEANS로 간단한 파생 변수 생성



```
kmeans_train = train.drop(['credit'], axis=1)
kmeans = KMeans(n_clusters=36, random_state=42).fit(kmeans_train)
train['cluster'] = kmeans.predict(kmeans_train)
test['cluster'] = kmeans.predict(test)
```



```
scaler = StandardScaler()  
train[numerical_feats] = scaler.fit_transform(train[numerical_feats])  
test[numerical_feats] = scaler.transform(test[numerical_feats])
```

Standard Scaler

각 **Feature**의 데이터 평균을 0, 분산을 1로 변경해 모든 특성이 같은 스케일을 갖게 합니다.

```
1 from sklearn.model_selection import GridSearchCV
2 from sklearn.model_selection import train_test_split
3
4 params = { 'n_estimators' : [10, 20, 50, 100, 200],
5           'max_depth' : [6, 8, 10, 12, 14, 16],
6           'min_samples_leaf' : [8, 12, 16],
7           'min_samples_split' : [8, 16, 20]
8         }
9 X_train, X_val, y_train, y_val = train_test_split(X, pd.DataFrame(y),
10                                                stratify=y, test_size=0.25,
11                                                random_state = 10086)
12
13 # RandomForestClassifier 객체 생성 후 GridSearchCV 수행
14 rf_clf = RandomForestClassifier(random_state = 0, n_jobs = None)
15 grid_cv = GridSearchCV(rf_clf, param_grid = params, cv = 3, n_jobs = None)
16 grid_cv.fit(X_train, y_train)
17
18 print('최적 하이퍼 파라미터: ', grid_cv.best_params_)
19 print('최고 예측 정확도: {:.4f}'.format(grid_cv.best_score_))
20
```

최적 하이퍼 파라미터: {'max_depth': 16, 'min_samples_leaf': 8, 'min_samples_split': 8, 'n_estimators': 20}
최고 예측 정확도: 0.6583

최적 하이퍼 파라미터

GridSearchCV 를 활용

RF의 하이퍼 파라미터인

N_estimators(트리 수), max_depth(깊이), 등

최고 정확도를 보여주는 파라미터를 산출

```

skfold = StratifiedKFold(n_splits=n_fold, shuffle=True, random_state=seed)
folds=[]
for train_idx, valid_idx in skfold.split(X, y):
    folds.append((train_idx, valid_idx))

cat_pred = np.zeros((X.shape[0], n_class))
cat_pred_test = np.zeros((X_test.shape[0], n_class))

for fold in range(n_fold):
    print(f'##n----- Fold {fold} -----##n')
    train_idx, valid_idx = folds[fold]
    X_train, X_valid, y_train, y_valid = X.iloc[train_idx], X.iloc[valid_idx], y[train_idx], y[valid_idx]

    model_cat = RandomForestClassifier(n_estimators= 20, max_depth= 16, min_samples_leaf= 8,
                                     min_samples_split=8, verbose=0, random_state=0, n_jobs= None)
    model_cat.fit(X_train, y_train)

    cat_pred[valid_idx] = model_cat.predict_proba(X_valid)
    cat_pred_test += model_cat.predict_proba(X_test) / n_fold
    print(f'CV Log Loss Score: {log_loss(y_valid, cat_pred[valid_idx]):.6f}')

print(f'##tLog Loss: {log_loss(y, cat_pred):.6f}')

```

Kfold 교차검증

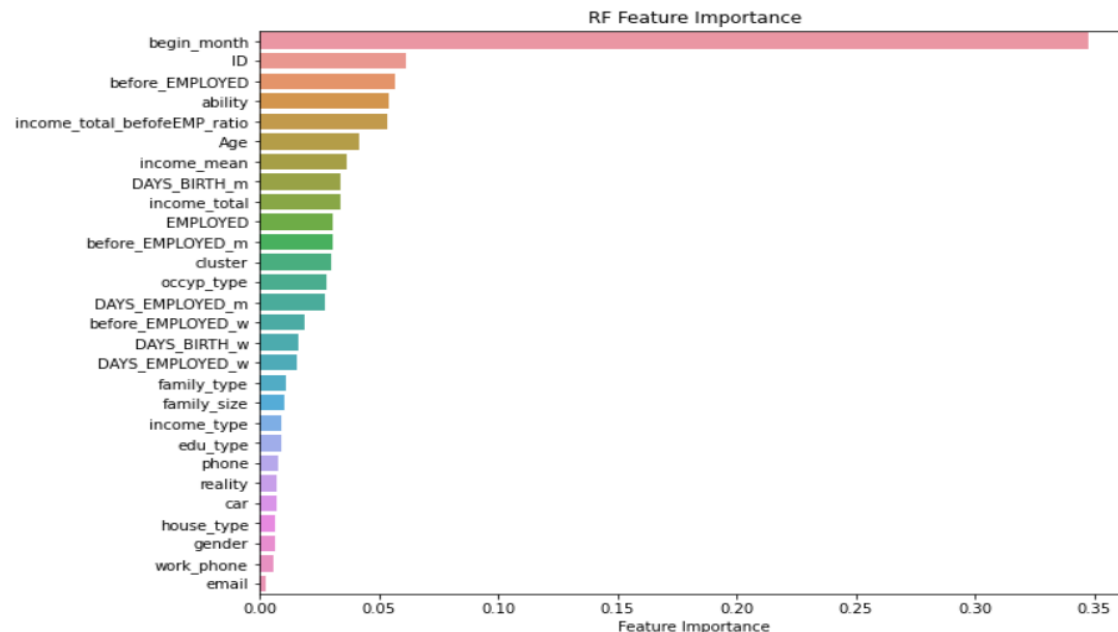
최적 하이퍼 파라미터를 K-fold 교차검증으로

Log loss 값을 산출 및 결과 값 도출

RF – Feature Importance

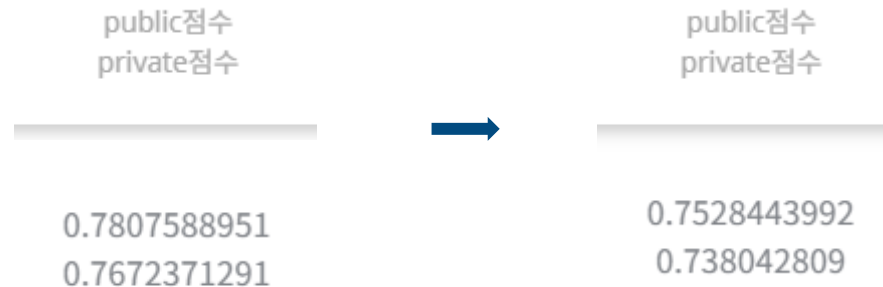
각 Feature의 중요도 시각화

Begin_month(카드 발급 일) 컬럼이 영향력이 높음



제출 결과

파생변수 생성, 최적 하이퍼 파라미터로 log loss값을 최대한 0.73까지 줄였지만 더 이상 불가능 모델적 한계로 보임.



Catboost

Catboost.

그래디언트 부스팅

Catboost는 XGBoost, Light GBM과 같은 그래디언트 부스팅 모델이다. 기존의 그래디언트 부스팅 알고리즘에서 오버피팅(과적합) 문제를 해결하는 내부 알고리즘이 추가된 업그레이드 버전이다. 모든 데이터의 잔차를 일괄 계산하는 기존 알고리즘과 달리 하나의 잔차만 계산하고 이후 다른 데이터 잔차를 예측하는 **Ordered Boosting** 방식으로 작동한다. 범주형 변수가 많은 데이터를 모델링 할 때 사용하기 적합하기 때문에 신용카드 연체 예측 데이터에서 가장 높은 정확도를 나타낸 모델이다.

Catboost

Catboost를 설치한 후

CatboostClassifier, Pool을 import해 사용하였다.

그 외 데이터 처리 라이브러리는 다른 모델과 동일하다.

```
!pip install catboost
!pip install category_encoders

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings, random
warnings.filterwarnings(action='ignore')

from sklearn.metrics import log_loss
from sklearn.preprocessing import StandardScaler
from category_encoders.ordinal import OrdinalEncoder
from sklearn.model_selection import StratifiedKFold

from sklearn.cluster import KMeans
from catboost import CatBoostClassifier, Pool
```

데이터 전처리

```
#IQR을 이용한 이상치 탐지
train_copy = train.copy()

def remove_outlier_test(t_cp, column):
    fraud_column_data = t_cp[column]
    quan_25 = np.percentile(fraud_column_data, 25)
    quan_75 = np.percentile(fraud_column_data, 75)

    iqr = quan_75 - quan_25
    iqr = iqr * 1.5
    lowest = quan_25 - iqr
    highest = quan_75 + iqr
    outlier_index = fraud_column_data[
        (fraud_column_data < lowest) | (fraud_column_data > highest)]
    print(len(outlier_index))
```

IQR 이상치 제거

IQR 사분위법을 이용해 이상치를 제거합니다.

```
train.fillna('NaN', inplace=True)
test.fillna('NaN', inplace=True)

import missingno as msno
msno.bar(df=train.iloc[:, :], color=(0, 0, 0))
#직업 유형 변수에서 결측치가 발견된다

# credit 형 변환
train = train.astype({'credit': 'category'})
print(train.dtypes)

train.fillna('NaN', inplace=True)
test.fillna('NaN', inplace=True)
```

NULL값 처리

Msno.bar로 결측치를 그래프로 확인하고 fillna를 이용해 결측치를 채웁니다.

```
train.drop(['index', 'FLAG_MOBILE'], axis=1, inplace=True)
test.drop(['index', 'FLAG_MOBILE'], axis=1, inplace=True)

train['DAYS_EMPLOYED'] = train['DAYS_EMPLOYED'].replace(
    {'DAYS_EMPLOYED': 0})
test['DAYS_EMPLOYED'] = test['DAYS_EMPLOYED'].replace(
    {'DAYS_EMPLOYED': 0})

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)
```

불필요한 컬럼 제거

핸드폰 유무 등 값이 굉장히 편향되어있고 불필요한 컬럼을 제거합니다.

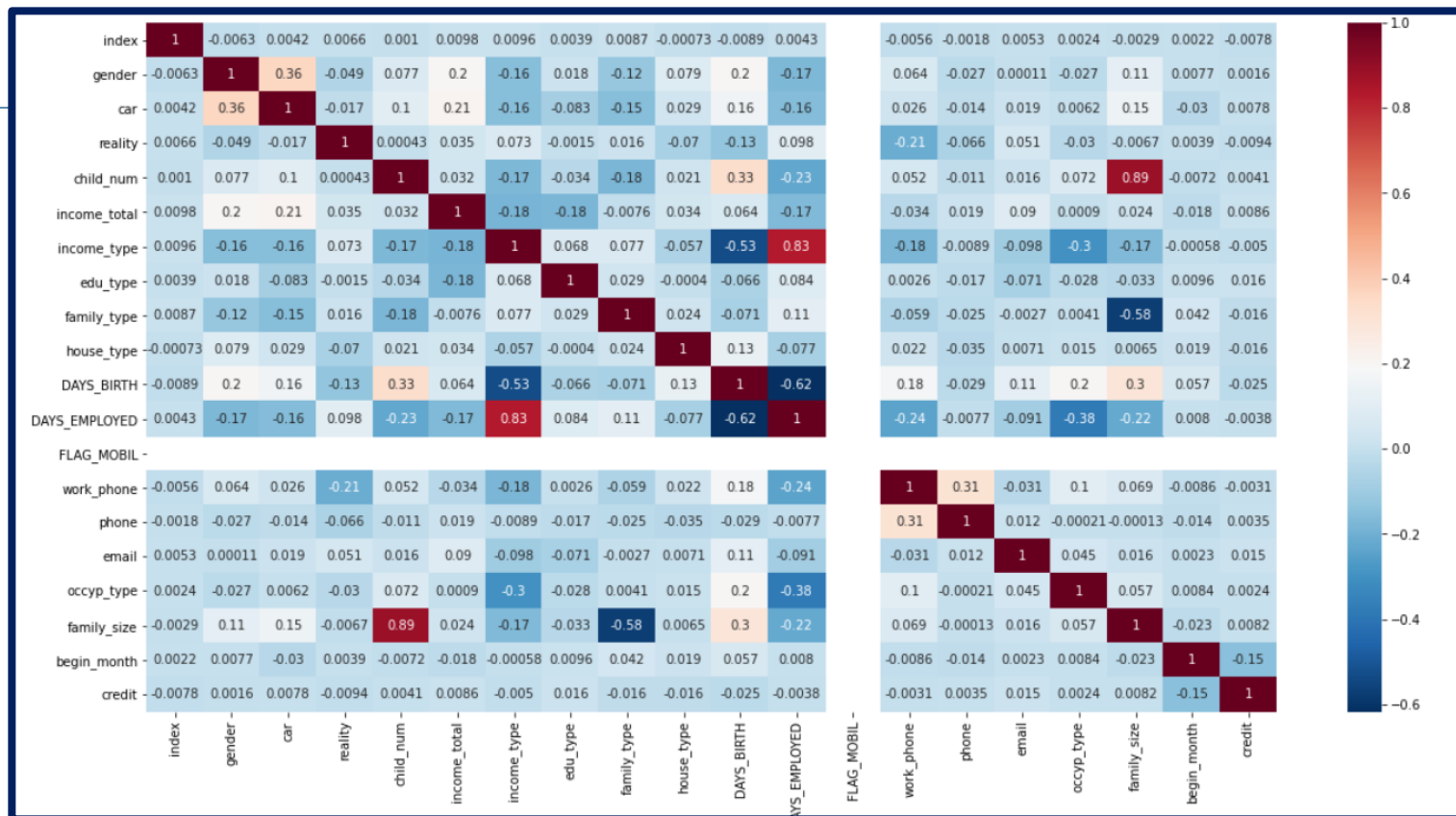
Part 4 피어슨 상관계수

상관계수

컬럼간 상관관계와
Credit 컬럼과의 상관계수를
확인한 수 상관계수가 낮아
연관관계가 없는 컬럼을
Drop합니다.

```
train.corr()["credit"]
```

```
index      -0.007841
gender      0.001562
car         0.007761
reality     -0.009387
child_num   0.004081
income_total 0.008555
income_type -0.005049
edu_type    0.015541
family_type -0.015513
house_type  -0.016111
DAYS_BIRTH  -0.025187
DAYS_EMPLOYED -0.003798
FLAG_MOBIL  NaN
work_phone  -0.003134
phone       0.003452
email       0.014812
occyp_type  0.002361
family_size 0.008227
begin_month -0.147477
credit      1.000000
Name: credit, dtype: float64
```



차원 축소

PCA 주성분 분석을 통해 변수 차원 축소를 시도
컴포넌트 4로 실시 후 확인해보자
주성분 네개가 y값을 설명하는 비율이 43% 가량으로 나와서 하나의 주성분이 Y값을 대부분 설명하는 경우는 아님을 확인

```
from sklearn.preprocessing import StandardScaler # 표준화 패키지 라이브러리
x = train.drop(['credit'], axis=1).values # 독립변인들의 value값만 추출
y = train['credit'].values # 종속변인 추출

x = StandardScaler().fit_transform(x) # x객체에 x를 표준화한 데이터를 저장

from sklearn.decomposition import PCA
pca = PCA(n_components=4) # 주성분을 몇개로 할지 결정
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data=principalComponents, columns = ['principal component1', 'principal component2',
# 주성분으로 이루어진 데이터 프레임 구성
```

principalDf.head()A

	principal component1	principal component2	principal component3	principal component4
0	0.309469	-0.544165	0.202381	0.472131
1	1.382695	0.059118	-0.236717	2.298145
2	0.604310	-0.306563	-2.688433	-1.756660
3	0.206211	-0.556902	0.695876	0.080649
4	-0.265785	0.311887	-1.075756	0.300466

pca.explained_variance_ratio_

array([0.17864448, 0.10119741, 0.08159349, 0.07399762])

sum(pca.explained_variance_ratio_)

0.4354330012939758

```

for df in [train, test]:
    # before_EMPLOYED: 고용되기 전까지의 일수
    df['before_EMPLOYED'] = df['DAYS_BIRTH'] - df['DAYS_EMPLOYED']
    df['income_total_befoeEMP_ratio'] = df['income_total'] / df['before_EMPLOYED']
    df['before_EMPLOYED_m'] = np.floor(df['before_EMPLOYED'] / 30) - ((np.floor(df['before_EMPLOYED'] / 30) - 1) * 30)
    df['before_EMPLOYED_w'] = np.floor(df['before_EMPLOYED'] / 7) - ((np.floor(df['before_EMPLOYED'] / 7) - 1) * 7)

    #DAYS_BIRTH 파생변수- Age(나이), 태어난 월, 태어난 주(출생연도의 n주차)
    df['Age'] = df['DAYS_BIRTH'] // 365
    df['DAYS_BIRTH_m'] = np.floor(df['DAYS_BIRTH'] / 30) - ((np.floor(df['DAYS_BIRTH'] / 30) - 1) * 30)
    df['DAYS_BIRTH_w'] = np.floor(df['DAYS_BIRTH'] / 7) - ((np.floor(df['DAYS_BIRTH'] / 7) - 1) * 7)

    #DAYS_EMPLOYED_m 파생변수- EMPLOYED(근속연수), DAYS_EMPLOYED_m(고용된 달), DAYS_EMPLOYED_w(고용된 주)
    df['EMPLOYED'] = df['DAYS_EMPLOYED'] // 365
    df['DAYS_EMPLOYED_m'] = np.floor(df['DAYS_EMPLOYED'] / 30) - ((np.floor(df['DAYS_EMPLOYED'] / 30) - 1) * 30)
    df['DAYS_EMPLOYED_w'] = np.floor(df['DAYS_EMPLOYED'] / 7) - ((np.floor(df['DAYS_EMPLOYED'] / 7) - 1) * 7)

    #ability: 소득/(살아온 일수+ 근무일수)
    df['ability'] = df['income_total'] / (df['DAYS_BIRTH'] + df['DAYS_EMPLOYED'])

    #income_mean: 소득/ 가족 수
    df['income_mean'] = df['income_total'] / df['family_size']

```

파생 변수

Before_Employed
Income_total
Income_mean
Age
Days_birth_m
Days_birth_w
Employed
Ability

파생 변수 생성


```
#ID 생성: 각 컬럼의 값들을 더해서 고유한 사람을 파악(*한 사람이 여러 개 카드를 만들 가능성을 고려해 begin_month는 제외함)
df['ID'] = ''
df['child_num'].astype(str) + '_' + df['income_total'].astype(str) + '_' + df['DAYS_BIRTH'].astype(str) + '_' + df['DAYS_EMPLOYED'].astype(str) + '_' + df['work_phone'].astype(str) + '_' + df['phone'].astype(str) + '_' + df['email'].astype(str) + '_' + df['family_size'].astype(str) + '_' + df['gender'].astype(str) + '_' + df['car'].astype(str) + '_' + df['reality'].astype(str) + '_' + df['income_type'].astype(str) + '_' + df['edu_type'].astype(str) + '_' + df['family_type'].astype(str) + '_' + df['house_type'].astype(str) + '_' + df['occyp_type'].astype(str)
```

ID 생성

각 컬럼의 값들을 더해서 고유한 사람을 파악하는 ID 컬럼 생성
한 사람이 여러 개 카드를 만들 가능성을 고려해 begin_month는 제외함

다중공선성 지수 계산

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X=train.copy()
scaler.fit(X)
vif_X_scaler = scaler.transform(X)
vif_X = pd.DataFrame(vif_X_scaler, columns=X.columns)
vif=pd.DataFrame()

from statsmodels.stats.outliers_influence import variance_inflation_factor
variance_inflation_factor(vif_X.values, 0)
```

다중공선성 컬럼 추출

라이브러리를 이용해 컬럼 별 다중공선성 지수를 계산
다중공선성 지수가 높은 컬럼을 제거하는 등의 조정을 통해 모델 정확도를 높이기 위해 노력함

수치형 변수, 범주형 변수

```
numerical_feats = train.dtypes[train.dtypes != "object"].index.tolist()
numerical_feats.remove('credit')
print("Number of Numerical features: ", len(numerical_feats))

categorical_feats = train.dtypes[train.dtypes == "object"].index.tolist()
print("Number of Categorical features: ", len(categorical_feats))
```

```
Number of Numerical features: 15
Number of Categorical features: 9
```

수치형 변수와 범주형 변수 추출

범주형 변수를 인코딩하기 위해
수치형 변수와 범주형 변수를 추출하고 정의합니다.

numerical_feats

수치형

```
['work_phone',
 'phone',
 'email',
 'family_size',
 'begin_month',
 'before_EMPLOYED',
 'income_total_befoeEMP_ratio',
 'before_EMPLOYED_m',
 'before_EMPLOYED_w',
 'Age',
 'DAYS_BIRTH_m',
```

categorical_feats

범주형

```
['gender',
 'car',
 'reality',
 'income_type',
 'edu_type',
 'family_type',
 'house_type',
 'occyp_type',
 'ID']
```

```
encoder = OrdinalEncoder(categorical_feats)
train[categorical_feats] = encoder.fit_transform(train[categorical_feats], train['credit'])
test[categorical_feats] = encoder.transform(test[categorical_feats])

train['ID'] = train['ID'].astype('int64')
test['ID'] = test['ID'].astype('int64')
```



```
kmeans_train = train.drop(['credit'], axis=1)
kmeans = KMeans(n_clusters=36, random_state=42).fit(kmeans_train)
train['cluster'] = kmeans.predict(kmeans_train)
test['cluster'] = kmeans.predict(test)
```

인코딩, 클러스터링

머신러닝을 돌리기 위해 범주형 변수를 인코딩합니다. **OrdinalEncoder**를 이용해 인코딩했습니다.

Cluster 그룹을 추가하기 위해 **kmeans**를 이용해 클러스터링합니다.

```
scaler = StandardScaler()  
train[numerical_feats] = scaler.fit_transform(train[numerical_feats])  
test[numerical_feats] = scaler.transform(test[numerical_feats])
```

```
n_est = 2000  
seed = 42  
n_fold = 15  
n_class = 3
```

```
target = 'credit'  
X = train.drop(target, axis=1)  
y = train[target]  
X_test = test
```

```
#n_est = 2000 log_loss = 0.7711250946  
#n_est = 1000 으로 시도 log_loss = 0.7774065791
```

정규화 및 하이퍼 파라미터 조정

StandardScaler를 이용해 정규화 하고

모델의 파라미터를 조정하며 예측도를 높이기
위한 작업을 합니다.

모델링 및 log loss 산출

Catboost를 이용해 모델링 합니다.

모델의 정확도를 판단하기 위한 지표로

Log loss(손실 함수)를 산출합니다.

Log loss는 낮을 수록 예측도가 높은 모델입니다.

```
skfold = StratifiedKFold(n_splits=n_fold, shuffle=True, random_state=seed)
folds=[]
for train_idx, valid_idx in skfold.split(X, y):
    folds.append((train_idx, valid_idx))

cat_pred = np.zeros((X.shape[0], n_class))
cat_pred_test = np.zeros((X_test.shape[0], n_class))
cat_cols = ['income_type', 'edu_type', 'family_type', 'house_type', 'occyp_type', 'ID']
for fold in range(n_fold):
    print(f'##n----- Fold {fold} -----##n')
    train_idx, valid_idx = folds[fold]
    X_train, X_valid, y_train, y_valid = X.iloc[train_idx], X.iloc[valid_idx], y[train_idx], y[valid_idx]
    train_data = Pool(data=X_train, label=y_train, cat_features=cat_cols)
    valid_data = Pool(data=X_valid, label=y_valid, cat_features=cat_cols)

    model_cat = CatBoostClassifier()
    model_cat.fit(train_data, eval_set=valid_data, use_best_model=True, early_stopping_rounds=100, verbose=100)

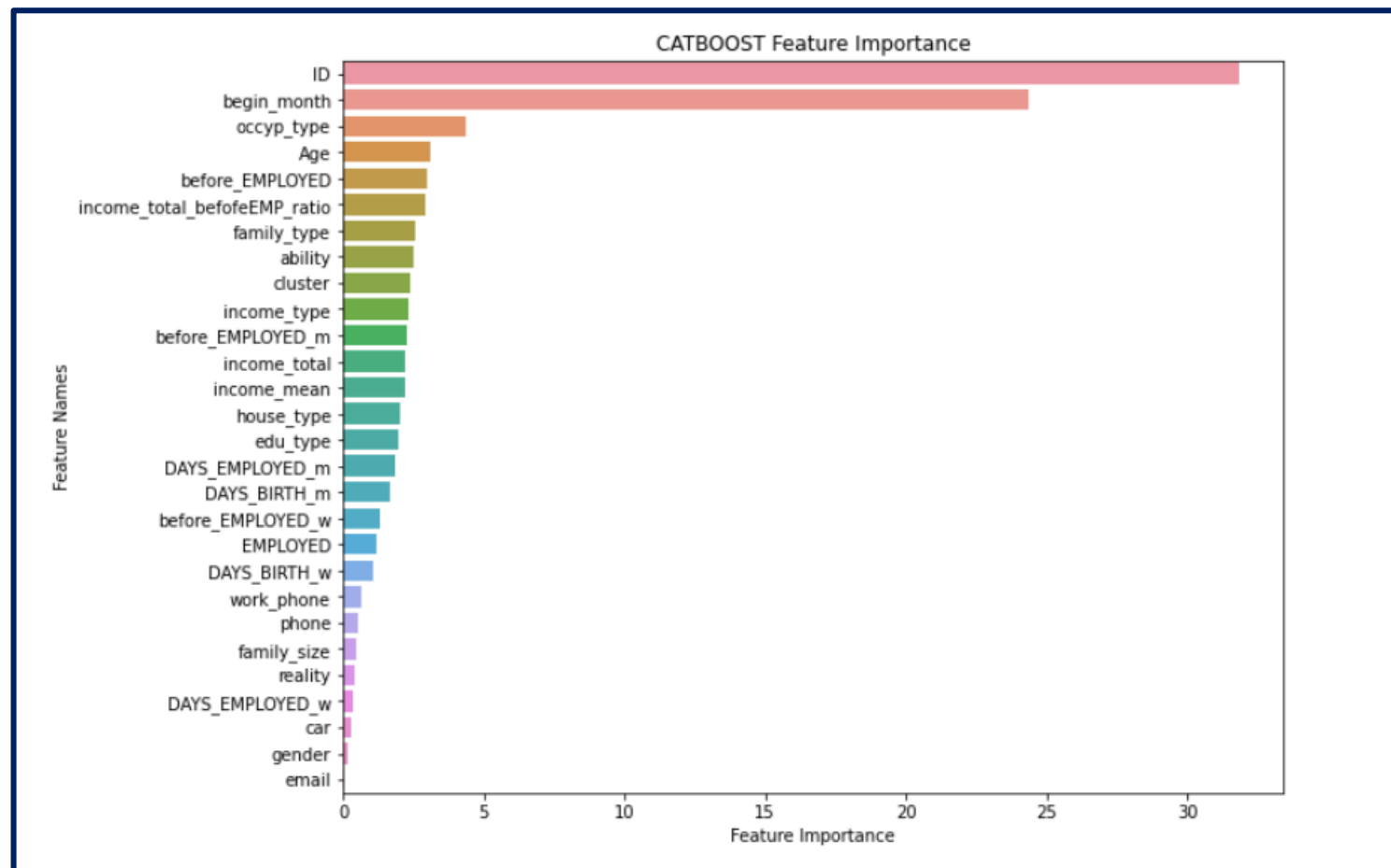
    cat_pred[valid_idx] = model_cat.predict_proba(X_valid)
    cat_pred_test += model_cat.predict_proba(X_test) / n_fold
    print(f'CV Log Loss Score: {log_loss(y_valid, cat_pred[valid_idx]):.6f}')

print(f'##tLog Loss: {log_loss(y, cat_pred):.6f}')
```

Feature Importance

Catboost 모델링 Feature Importance 시각화 결과입니다.

ID의 중요도가 가장 높게 나타났습니다.



Part 4 데이콘 순위

1

조정 없이 Catboost만 돌렸을 때

0.66769

2

하이퍼 파라미터 조정 후

0.66769



0.6678545748
0.6589218264

3

파생 변수 조정 후

0.6678545748
0.6589218264



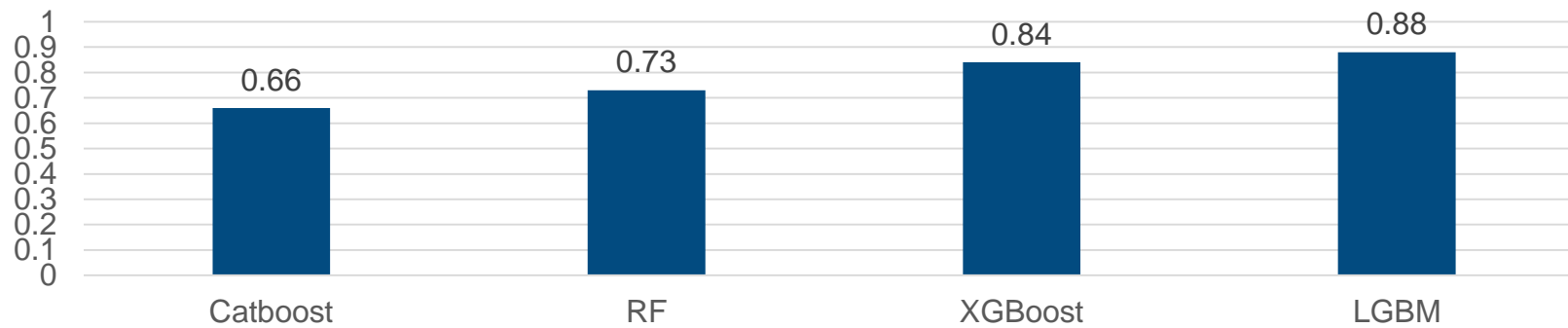
0.6591885657

최종 순위 결과

● WINNER ● 1% ● 4% ● 10%			
#	팀	팀 멤버	점수
17	Platium		0.66769
Public 17위			
#	팀	팀 멤버	최종점수
1	소회의실		0.6581
2	dswook		0.65862
3	js4756		0.65913
4	초보산님		0.66003
5	Team SsulleBal		0.66016
Private 4위			
Here!			

Final Ranking

Catboost > RF > XGBoost > LGBM





감사합니다.