신용카드 사용자 연체 예측 AI 경진대회

3차 SEMI-PROJECT: 이지금팀

정승욱, 김민엽, 유진아, 이시현

Two-Now

이지금

바로 이 지금, 1분 1초도 흘려보내지 않고 최선을 다하는 팀

응용통계학과
ISFJ

"알이즈웰"

팀장
정승욱

경영정보학과
ENTJ

"I decide who I am"

팀원
유진아

IT 관련 전공자 4명이 모여
어우러진 팀
협업 끝판왕!!
모르는 것은 서로 물어
도와가며
프로젝트를 완성하는 데
집중

정보통신공학과
INTJ

"노려라 딥러닝 마스터"

팀원
김민엽

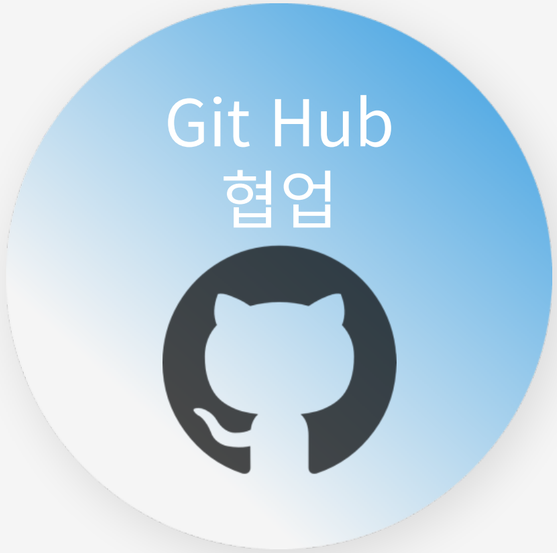
산업경영공학과
ISTJ

"Now or Never"

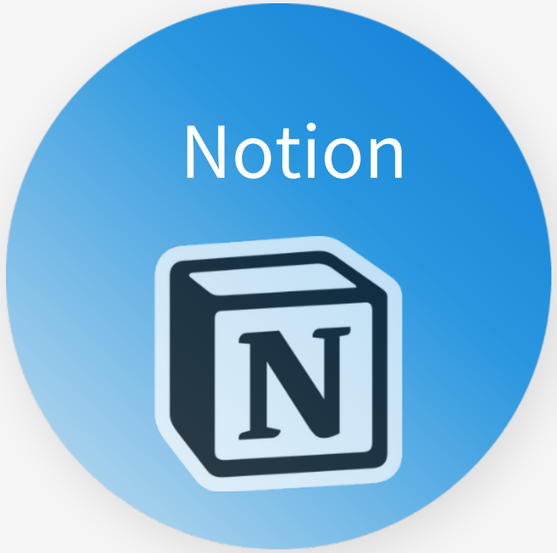
팀원
이시현

프로젝트 진행방식, R&R

-민엽: 전처리 및 코드 뼈대, DL, ML, Github merge
-승욱: EDA, ML, 발표자료
-시현: 전처리, DL, ML, 발표자료
-진아: EDA, ML, 발표자료, Github



처음 시도해봄
branch 생성 main에 merge
issue 발생시 공유, 중요한 것 노션에 저장



레퍼런스, 회의 로그, 이슈 기록

이미지 저장, 제출용 자료,
발표 슬라이드 공동 작업

Contents

신용카드 데이터 선정이유: 선정 이유 : ML, DL 모두 적용 가능한 2만개 이상의 적당한 데이터 셋

01. 데이터 소개

신용카드 사용자 개인 신상정보

데이콘 경진대회:

[월간 데이콘 14]
신용카드 사용자 연체 예측 AI 경진대회

DACON
DATA TO VALUE

배경

신용카드사는 신용카드 신청자가 제출한 개인정보와 데이터를 활용해 점수를 산정
->이 산용점수를 활용해 신청자의 향후 채무 불이행과 신용카드 대급 연체 가능성 예측

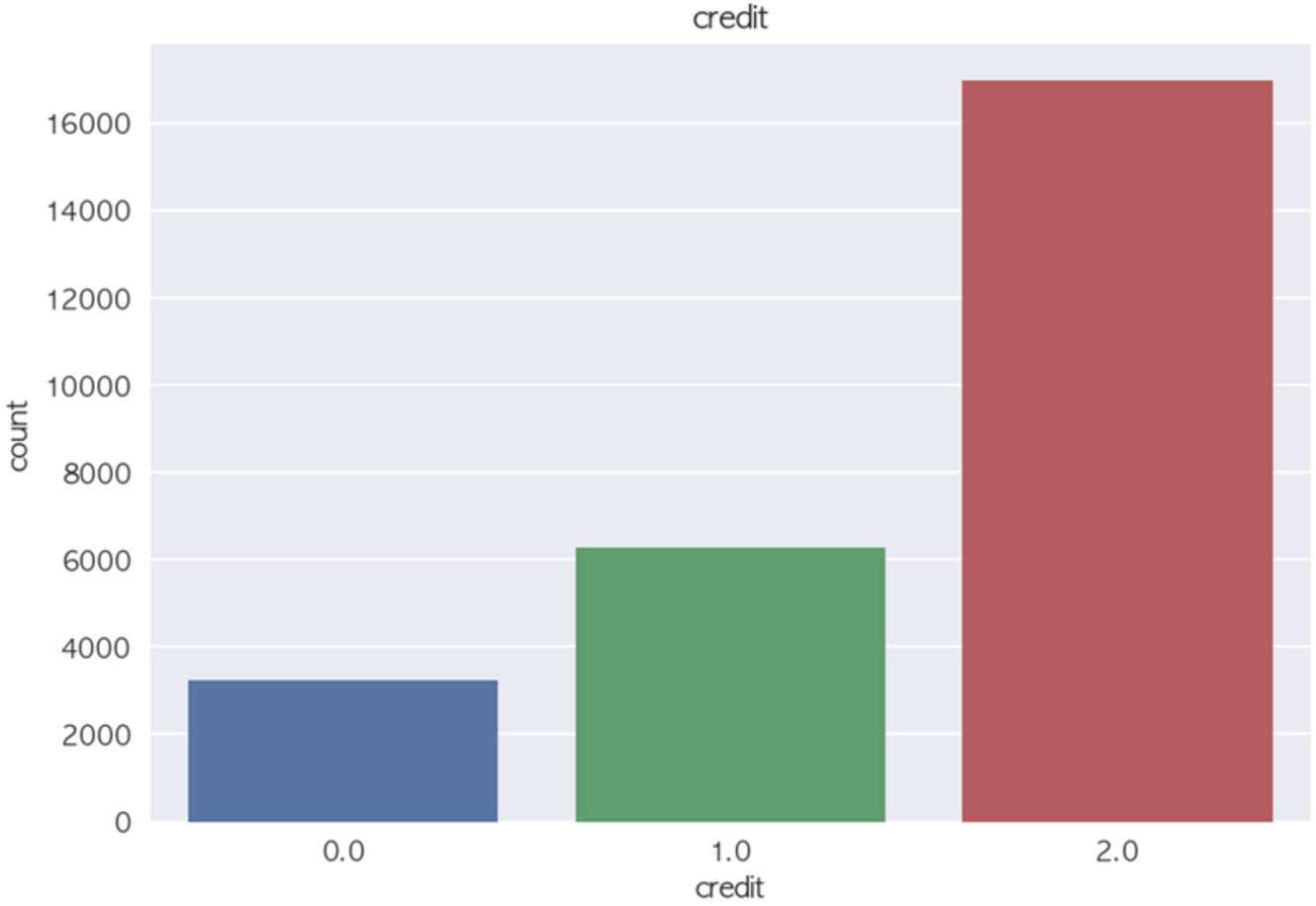
목적

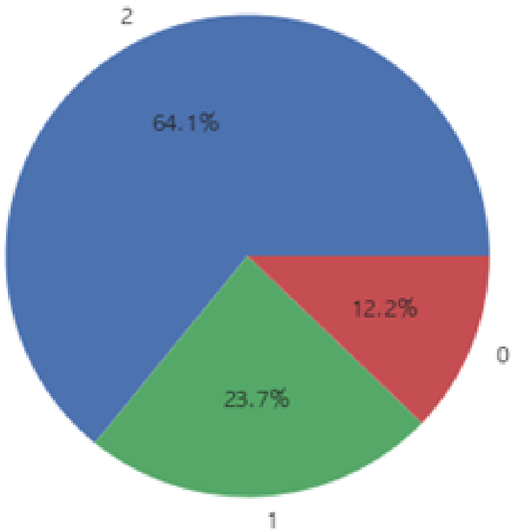
신용카드 사용자들의 개인 신상정보 데이터로 사용자의 신용카드 대금 연체 정도를 예측

```
1  train.head()
```

	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_siz
0	0	F	N	N	0	202500.0	Commercial associate	Higher education	Married	Municipal apartment	-13899	-4709	1	0	0	0	NaN	2.
1	1	F	N	Y	1	247500.0	Commercial associate	Secondary / secondary special	Civil marriage	House / apartment	-11380	-1540	1	0	0	1	Laborers	3.
2	2	M	Y	Y	0	450000.0	Working	Higher education	Married	House / apartment	-19087	-4434	1	0	1	0	Managers	2.
3	3	F	N	Y	0	202500.0	Commercial associate	Secondary / secondary special	Married	House / apartment	-15088	-2092	1	0	1	0	Sales staff	2.
4	4	F	Y	Y	0	157500.0	State servant	Higher education	Married	House / apartment	-15037	-2105	1	0	0	0	Managers	2.

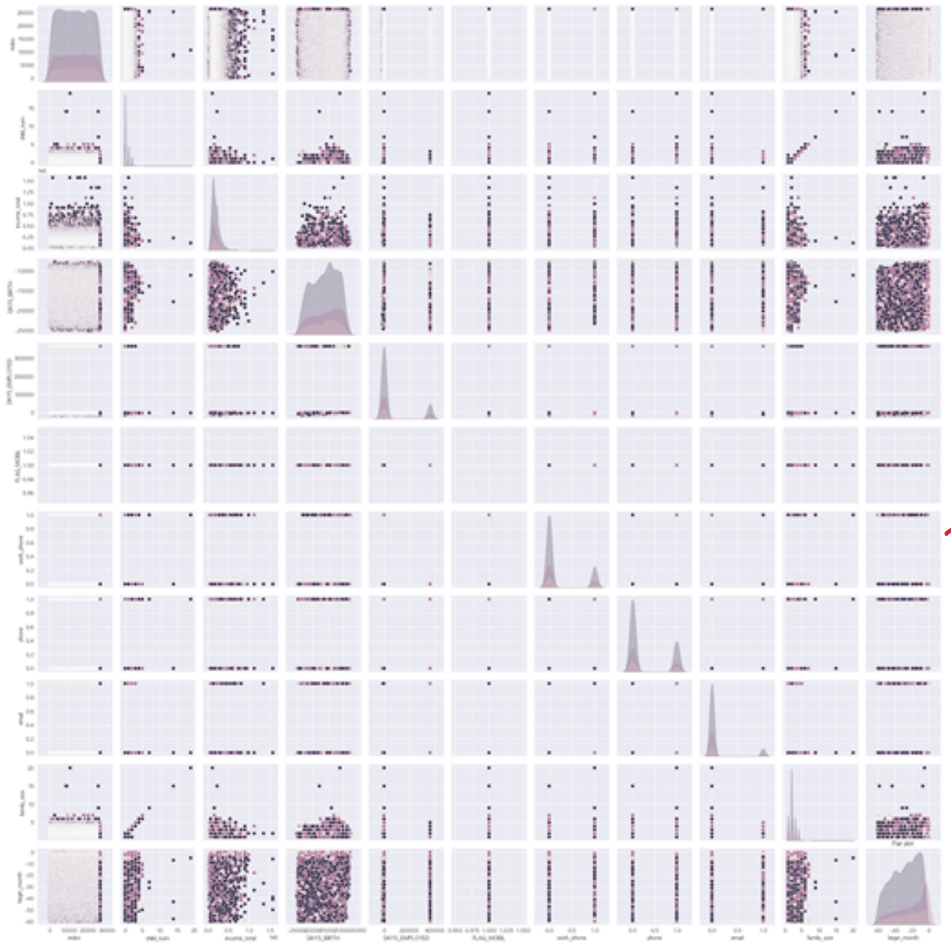
EDA target: credit 사용자의 신용카드 대금 연체 정도

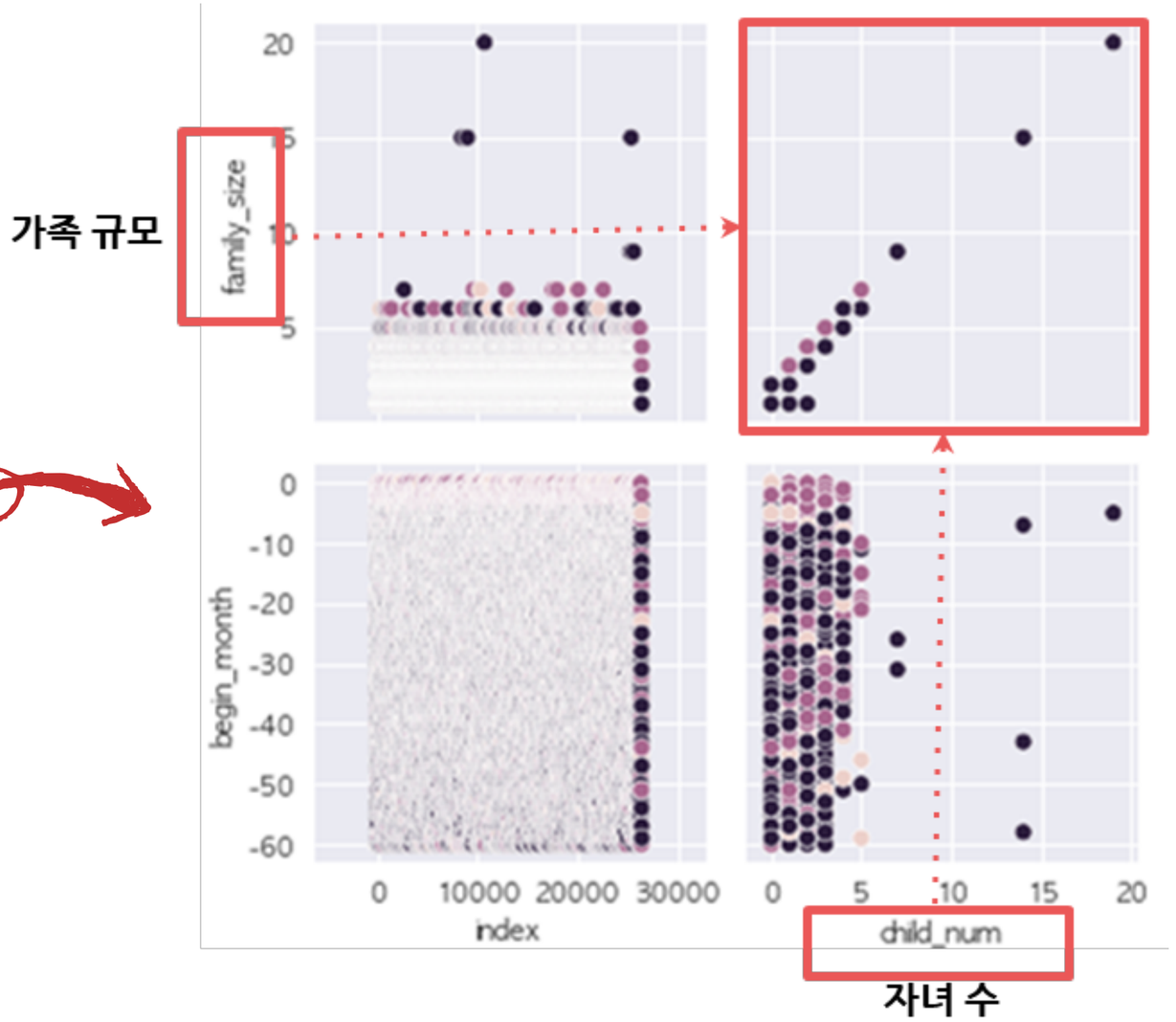




target 데이터의 불균형
train, test set 분리 시
Stratified sampling (층화추출)

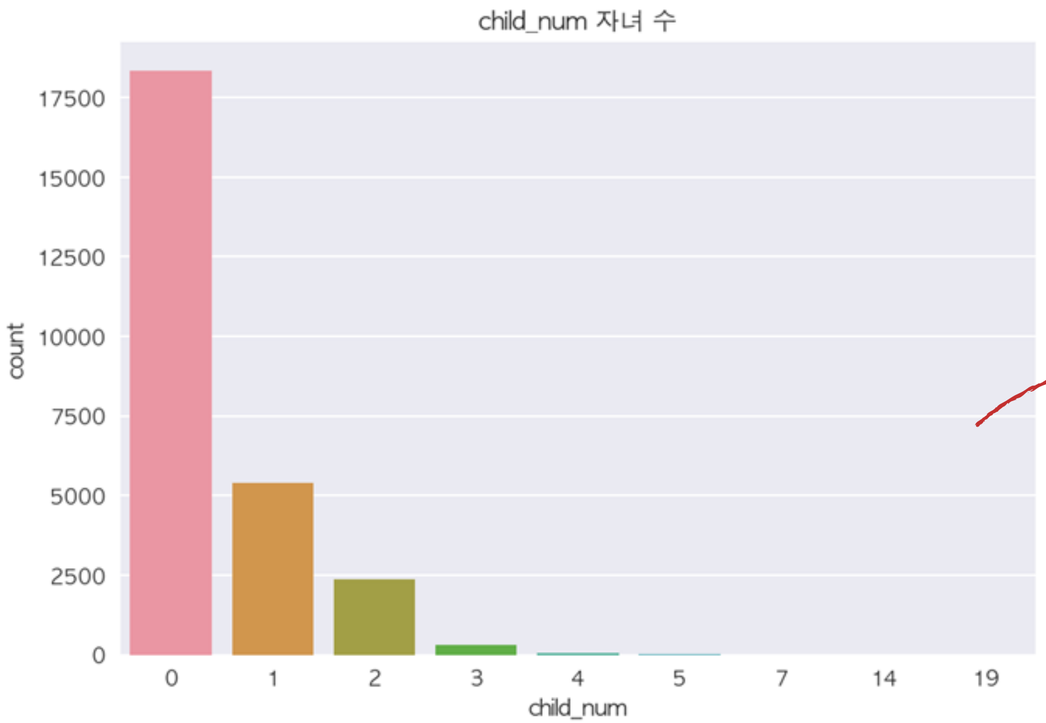
EDA의 시작,,, 다 때려넣고 Pairwise Scatter plots

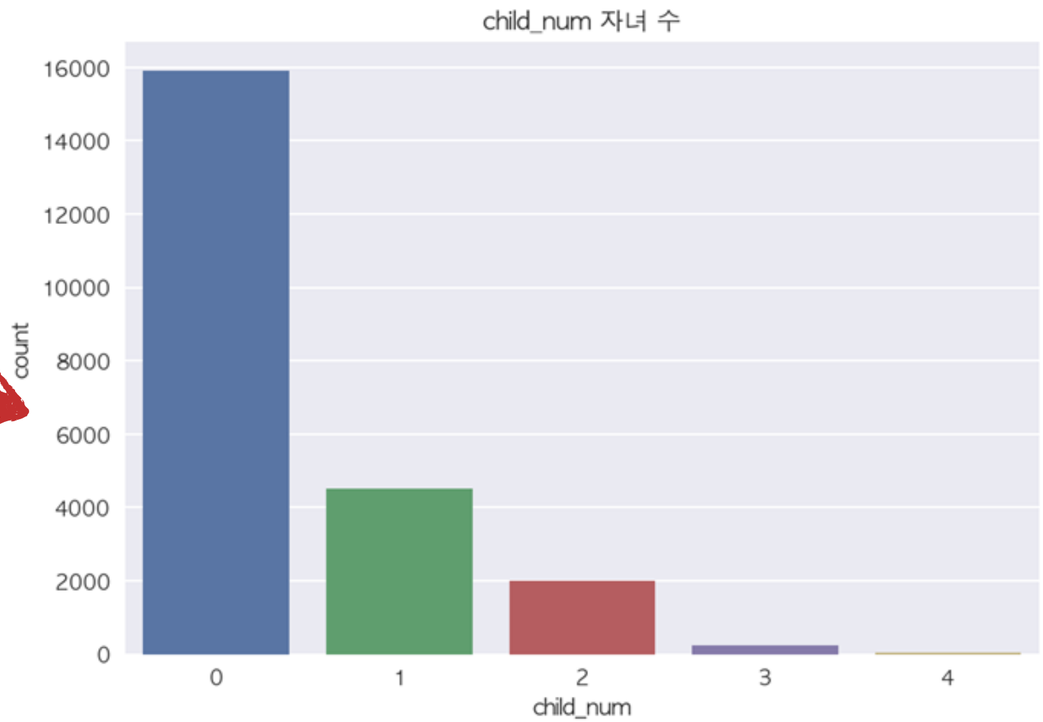




child_num, family_size 상관관계 존재
: 다중공선성(multicollinearity) 우려

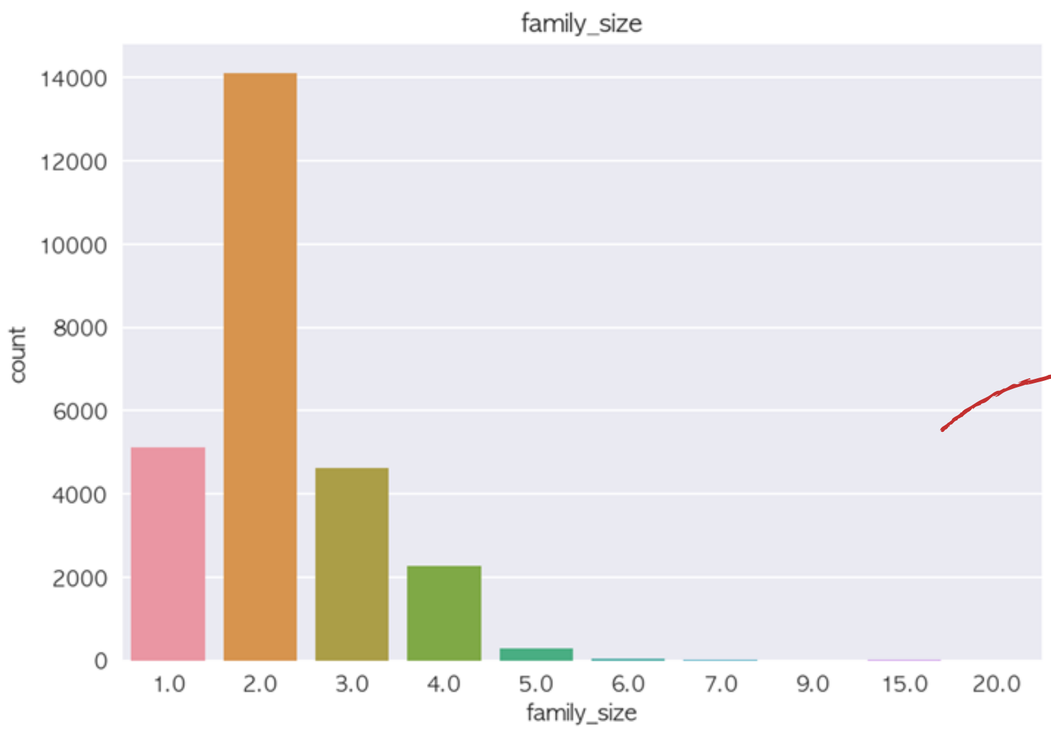
EDA child_num 자녀 수

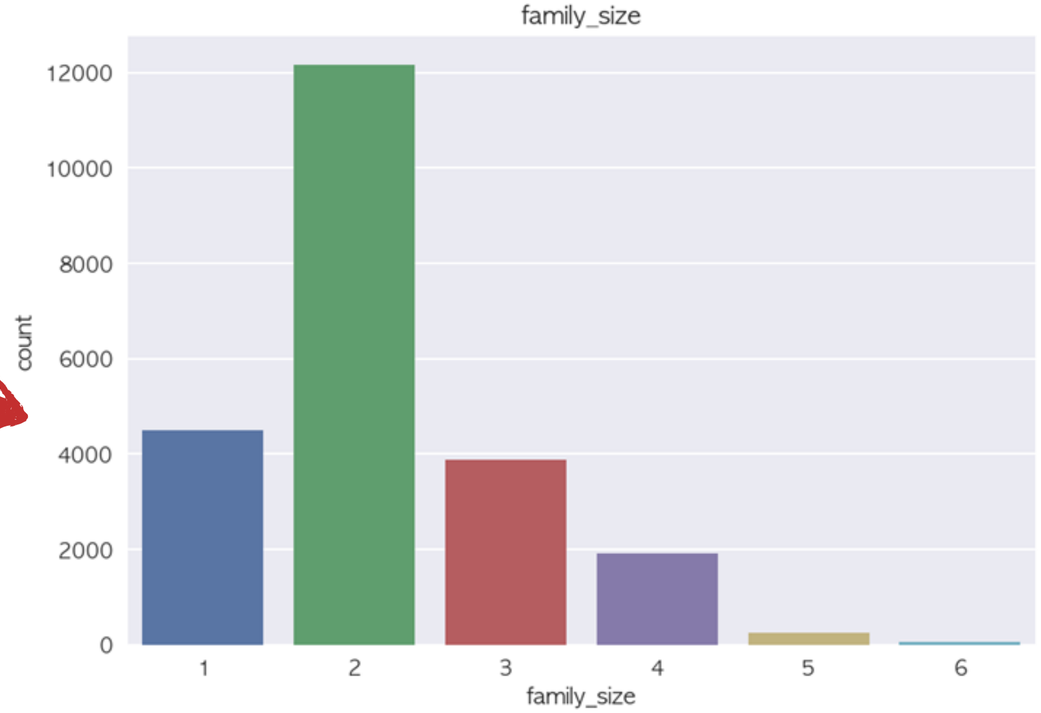




자녀 수 4 이상 한 카테고리로 묶음, 4로 처리

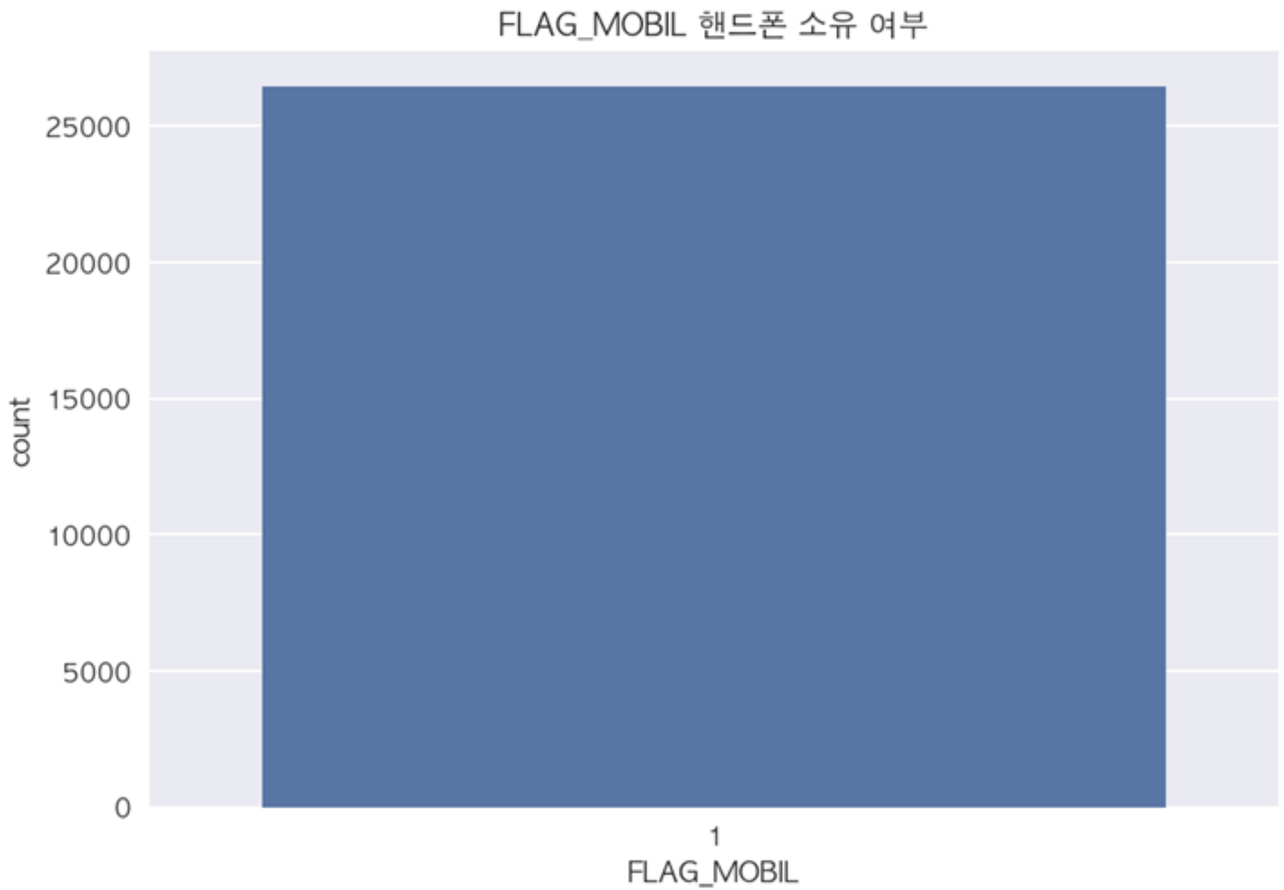
EDA family_size 가족 규모





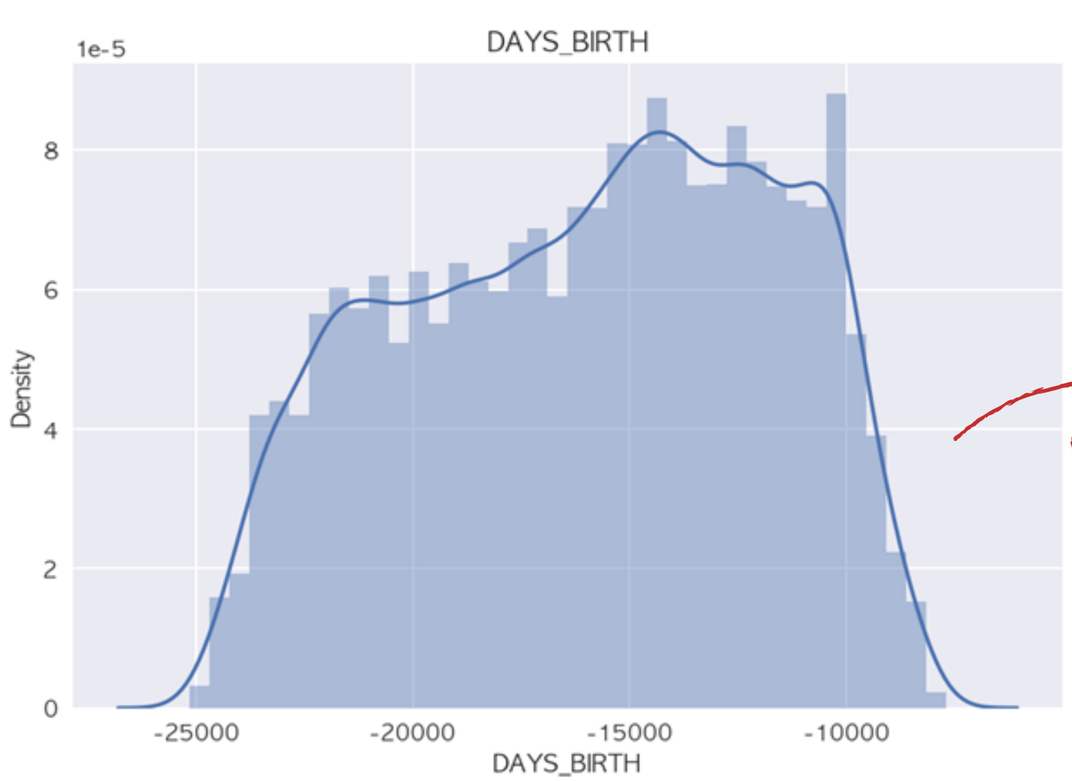
가족 수 6이상 한 카테고리로 묶음, 6으로 처리

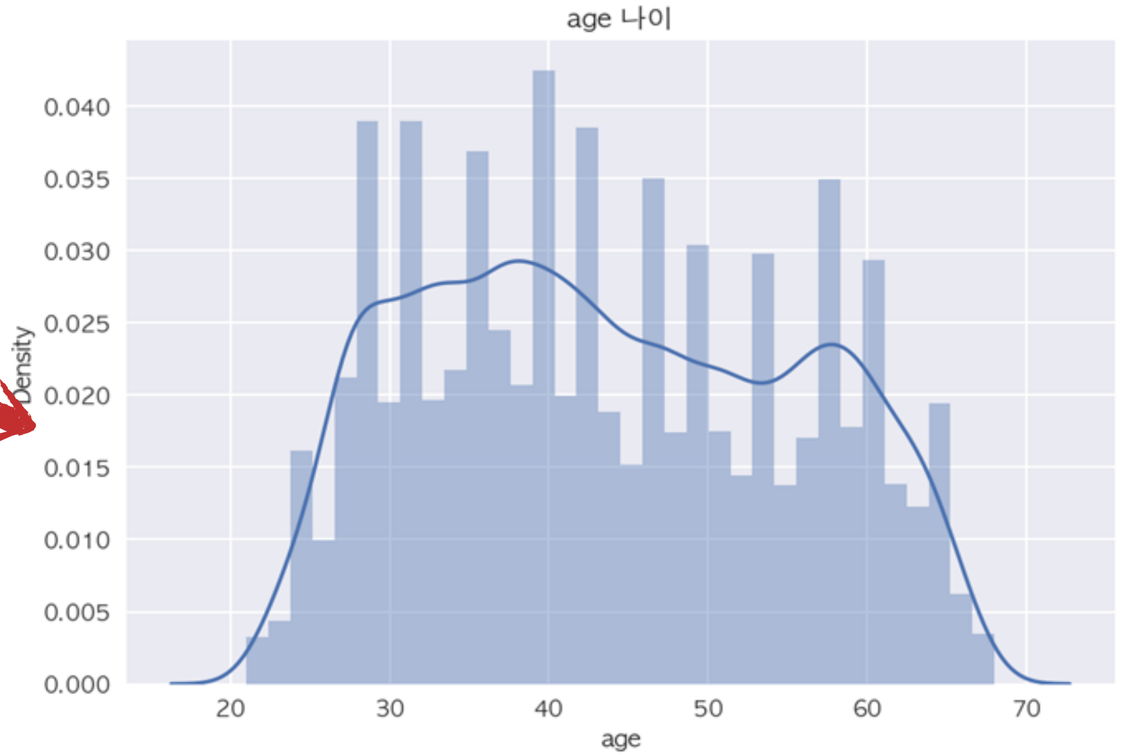
EDA FLAG_MOBIL 핸드폰 소유 여부



NA, Null 값 없이 모두 핸드폰을 소유하고 있음 -> Column 삭제

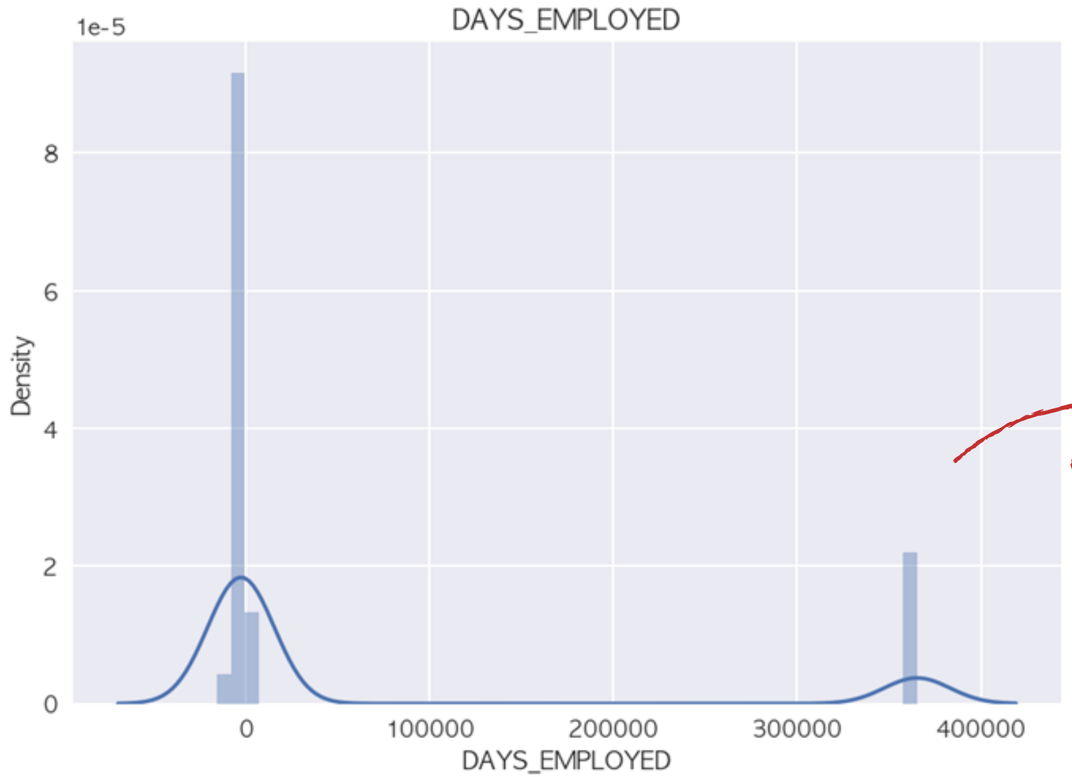
EDA age 변수 생성 from DAYS_BIRTH

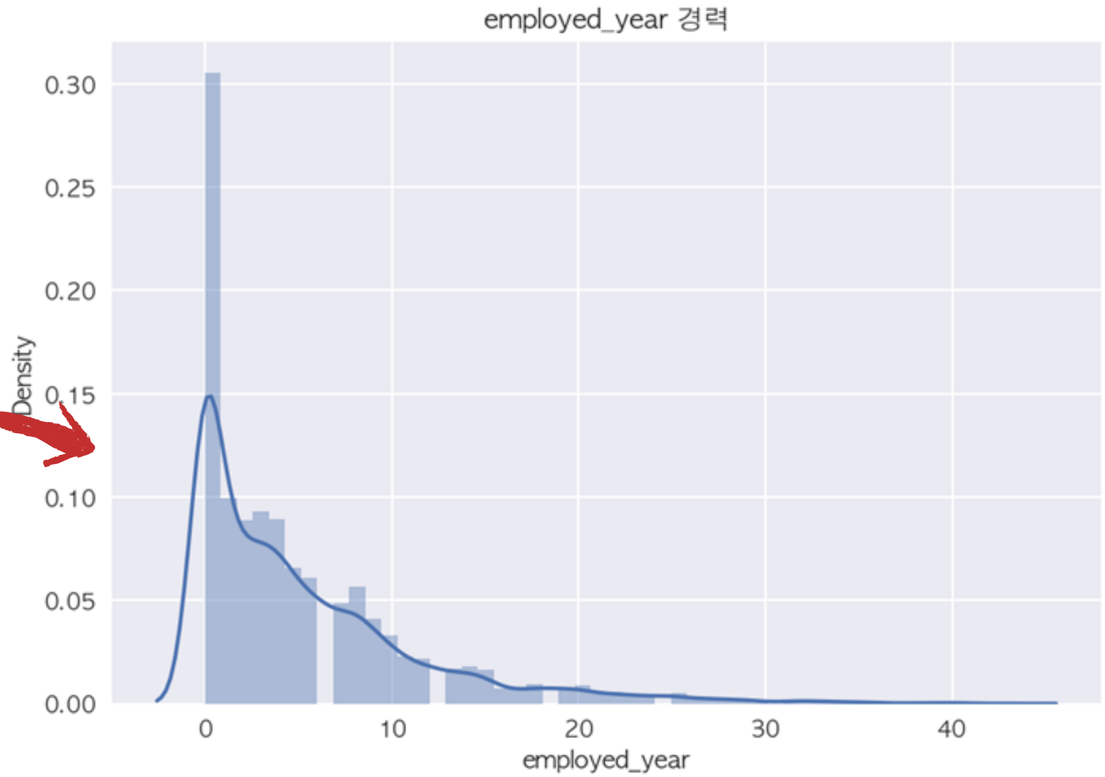




DAYS_BIRTH: 나이로 환산 (-1 곱하고 365로 나눔)
pd.cut으로 10개 구간으로 나누어 범주화한 데이터 셋도 생성 ➡ 성능 하락

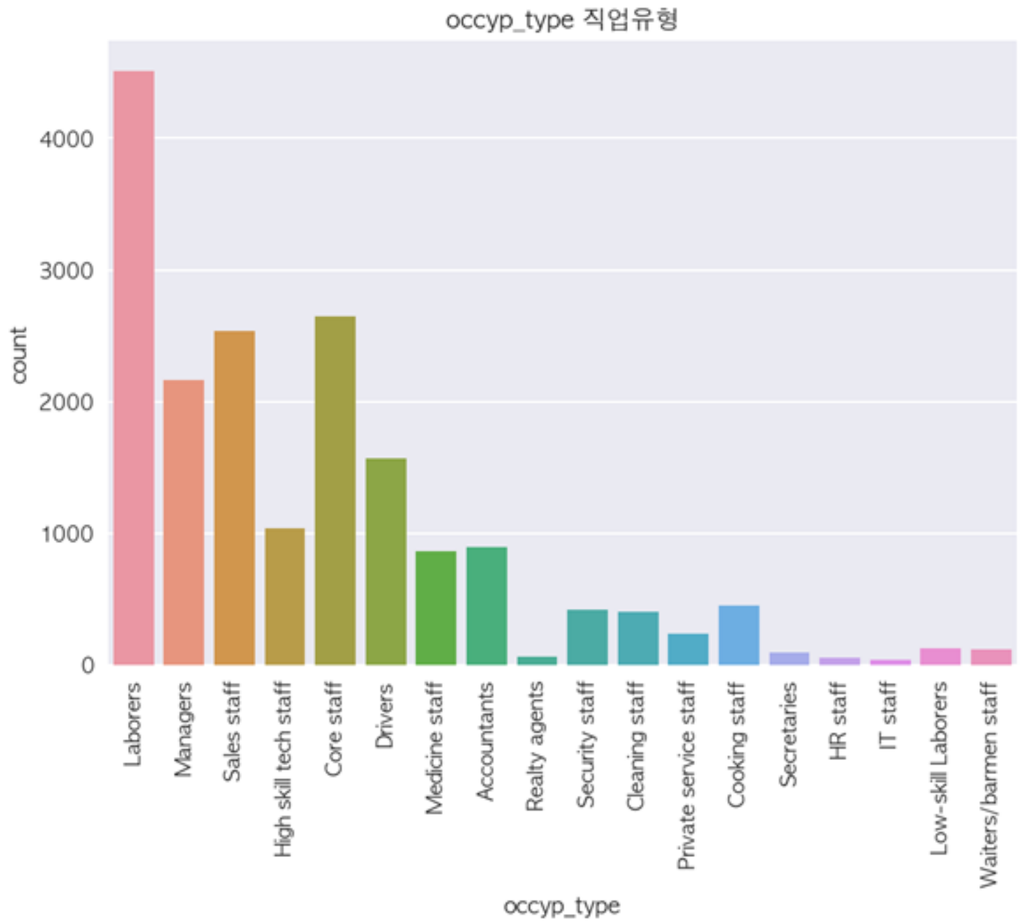
EDA employed_year 생성 from days_employed

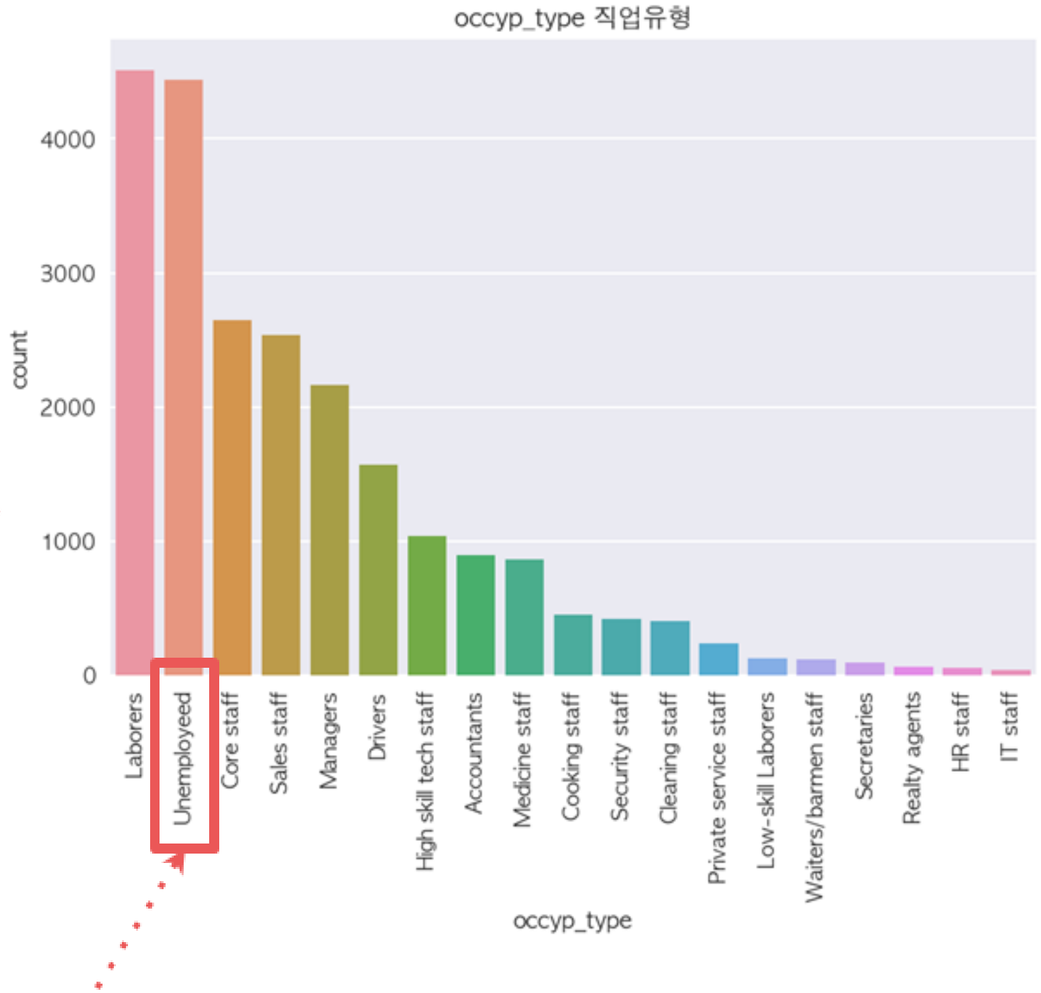




days_employed 이상치: 양수인 값이 존재 ➡ 무직으로 판단, 0으로 처리 + 고용기간(year)으로 전처리
pd.cut으로 10개 구간으로 나누어 나이를 범주화한 데이터 셋도 생성 ➡ 성능 하락

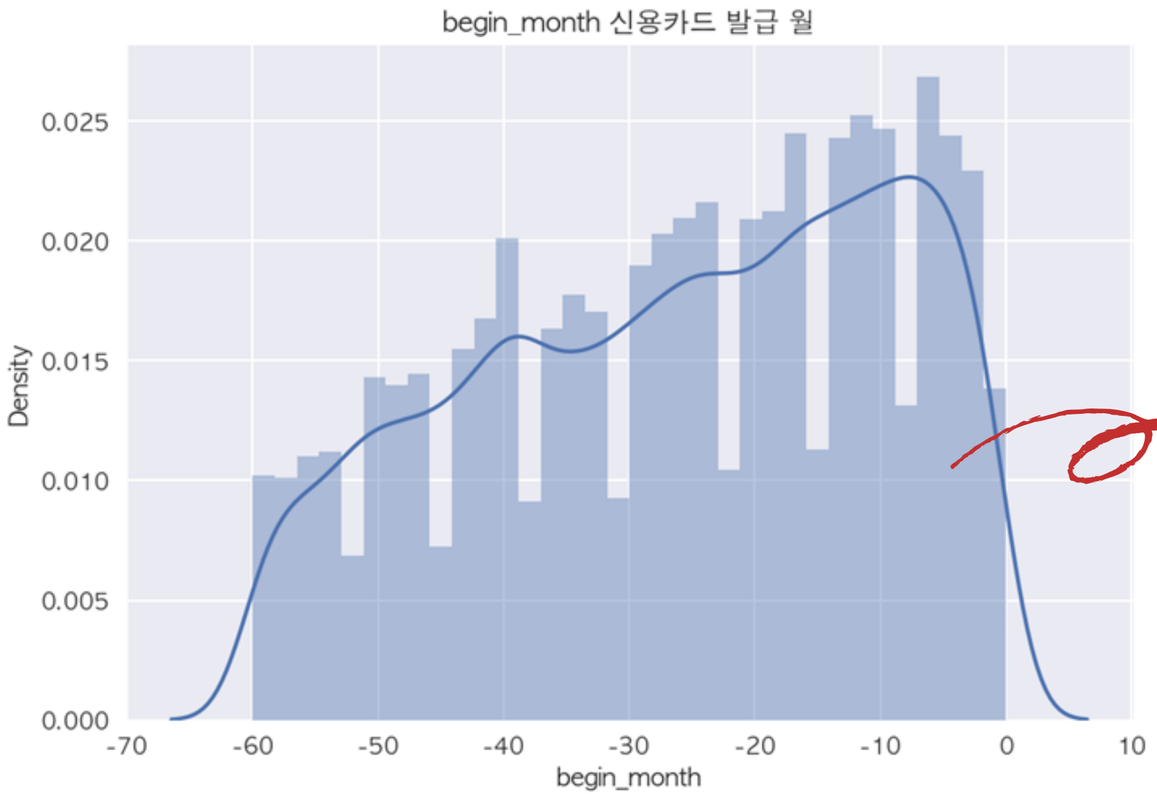
EDA occyp_type 직업유형

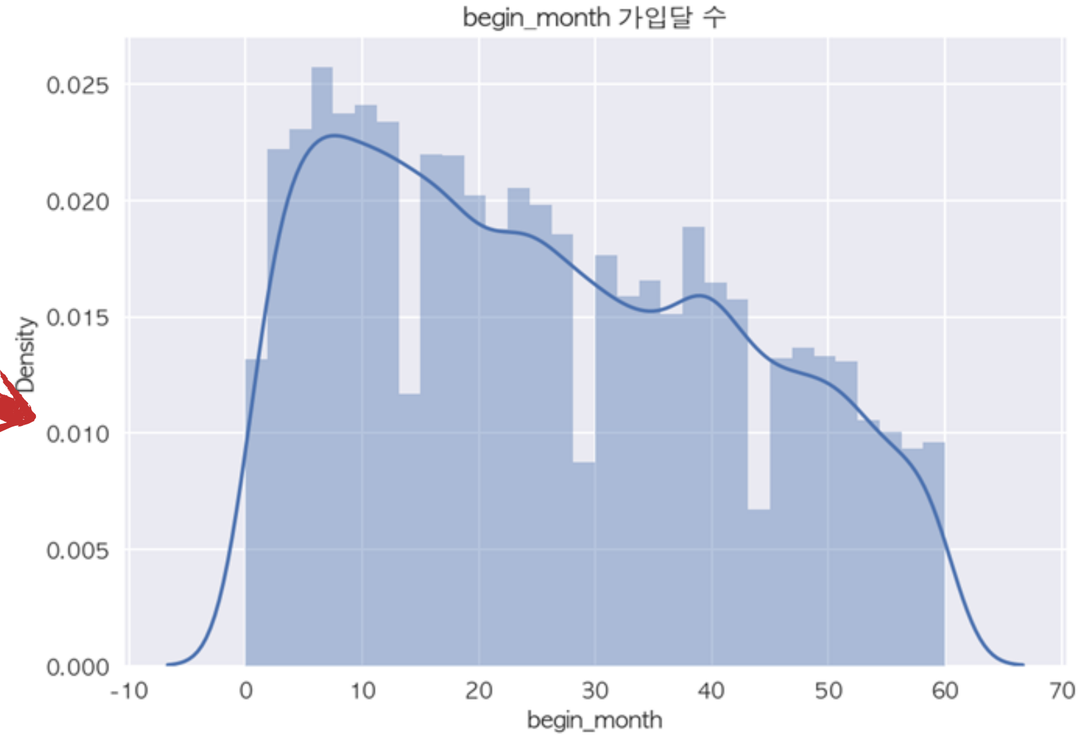




DAYS_EMPLOYED > 0 (무직) | NA values : Unemployed category 생성

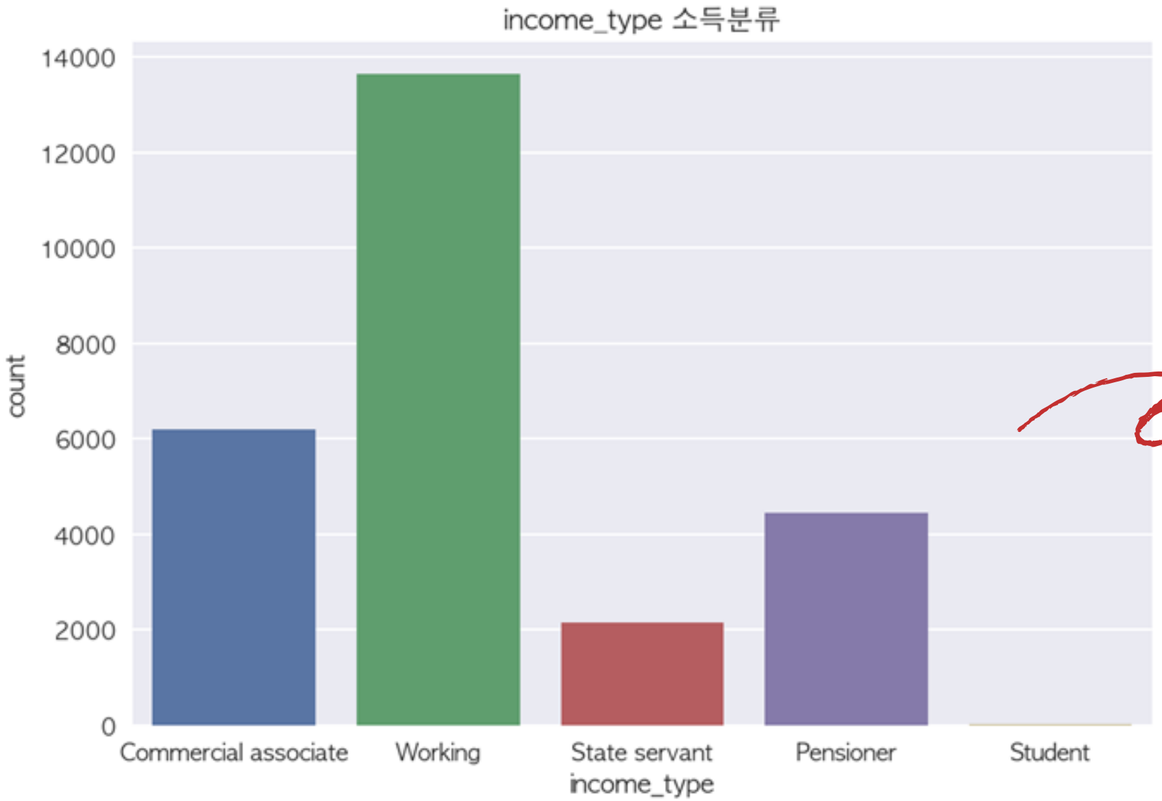
EDA begin_month 카드 가입 달수

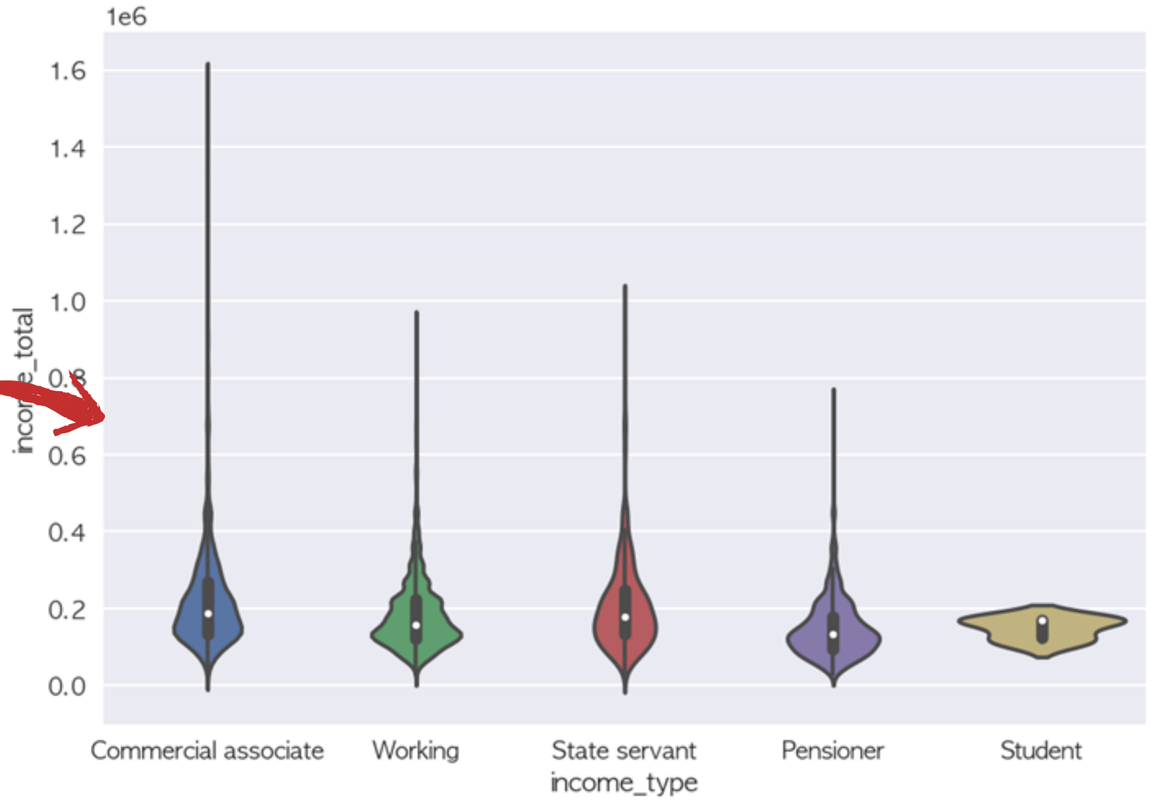




가입 달 수 < 0: -1 곱해서 양수로 환산, 그외 양수는 0으로 처리
pd.cut으로 10개 구간으로 나누어 범주화한 데이터 셋도 생성, 연속형 데이터셋 모델 성능과 비교

EDA income_type 소득





student의 수가 굉장히 적음. income_type(소득 분류)별 income_total(연간 소득) 비교
box-plot, violin-plot으로 확인 후 median, IQR 에는 큰 차이가 없음

'이지금'팀 전처리

```python
def preprocess_data(df: pd.DataFrame) -> pd.DataFrame:
    data = df.drop(['FLAG_MOBIL'], axis=1).copy()
    data['credit'] = data['credit'].astype(int)

    data = data[data['occyp_type'].notnull() | (data['DAYS_EMPLOYED'] > 0)]
    data['occyp_type'] = data['occyp_type'].fillna('Unemployeed')

    data['child_num'] = data['child_num'].apply(lambda x: 4 if x > 4 else x)
    data['family_size'] = data['family_size'].apply(lambda x: 6 if x > 6 else x)
    data['family_size'] = data['family_size'].astype(int)

    data['DAYS_BIRTH'] = data['DAYS_BIRTH'].apply(lambda x: (x*-1)/365 if x < 0 else 0)
    data['DAYS_EMPLOYED'] = data['DAYS_EMPLOYED'].apply(lambda x: (x*-1)/365 if x < 0 else 0)
    data['begin_month'] = data['begin_month'].apply(lambda x: (x*-1)/12 if x < 0 else 0)
    data.rename(columns={'DAYS_BIRTH':'age','DAYS_EMPLOYED':'employed_year',
                         'begin_month':'begin_year'}, inplace=True)

    category_dict = get_category_dict()
    for column, cat_dict in category_dict.items():
        data[column].replace(cat_dict, inplace=True)

    return data
```

'이지금'팀 전처리

```python
def get_category_dict() -> dict:
    category_dict = dict()

    category_dict['gender'] = {'M':0,'F':1}
    category_dict['car'] = {'N':0,'Y':1}
    category_dict['reality'] = {'N':0,'Y':1}
    category_dict['income_type'] = {'Working': 0, 'Commercial associate': 1, 'Pensioner': 2, 'State servant': 3, 'Student': 4}
    category_dict['edu_type'] = {'Secondary / secondary special': 0, 'Higher education': 1, 'Incomplete higher': 2,
                                 'Lower secondary': 3, 'Academic degree': 4}
    category_dict['family_type'] = {'Married': 0, 'Single / not married': 1, 'Civil marriage': 2, 'Separated': 3, 'Widow': 4}
    category_dict['house_type'] = {'House / apartment': 0, 'With parents': 1, 'Municipal apartment': 2, 'Rented apartment': 3,
                                   'Office apartment': 4, 'Co-op apartment': 5}
    category_dict['occyp_type'] = {'Unemployeed': 0, 'Laborers': 1, 'Core staff': 2, 'Sales staff': 3, 'Managers': 4, 'Drivers': 5,
                                   'High skill tech staff': 6, 'Accountants': 7, 'Medicine staff': 8, 'Cooking staff': 9,
                                   'Security staff': 10, 'Cleaning staff': 11, 'Private service staff': 12, 'Low-skill Laborers': 13,
                                   'Waiters/barmen staff': 14, 'Secretaries': 15, 'Realty agents': 16, 'HR staff': 17, 'IT staff': 18}

    return category_dict
```

index	gender	car	reality	child_num	income_total	income_type	edu_type	family_type	house_type	age	employed_year	work_phone	phone	email	occyp_type	family_size	begin_year	credit
1	1	0	1	1	247500	1	0	2	0	31.1780822	4.219178082	0	0	1	1	3	0.41666667	1
2	0	1	1	0	450000	0	1	0	0	52.2931507	12.14794521	0	1	0	4	2	1.83333333	2
3	1	0	1	0	202500	1	0	0	0	41.3369863	5.731506849	0	1	0	3	2	3.08333333	0
4	1	1	1	0	157500	3	1	0	0	41.1972603	5.767123288	0	0	0	4	2	2.16666667	2
5	1	0	1	2	270000	0	0	0	0	36.7479452	13.68767123	0	0	1	6	4	1.5	1

'소회의실'팀 전처리

```python
def preprocess_data(df: pd.DataFrame, name='train') -> pd.DataFrame:
    data = df.fillna('NaN').copy()
    data = data.drop(['FLAG_MOBIL'], axis=1)
    client_input = data.columns.tolist()

    data['DAYS_EMPLOYED'] = data['DAYS_EMPLOYED'].apply(lambda x: 0 if x > 0 else x)
    data['occyp_type'] = ['Unemployed' if emp == 0 else occ
                          for emp, occ in zip(data['DAYS_EMPLOYED'],data['occyp_type'])]

    data['family_size'] = data['family_size'].apply(lambda x: 6 if x > 6 else x)
    data[['work_phone','phone','email']] = data[['work_phone','phone','email']].astype('object')

    date_columns = ['DAYS_BIRTH', 'DAYS_EMPLOYED', 'begin_month']
    data[date_columns] = data[date_columns].apply(lambda x: abs(x))

    data = set_extra_features(data, client_input)
    data.drop(['child_num','DAYS_BIRTH','DAYS_EMPLOYED'], axis=1, inplace=True)
    data = set_ordinal_encoding(data, name)
    data = set_clustering(data, name)

    columns = data.drop(['index','credit'], axis=1).columns.tolist()
    data = data.reindex(columns=['index']+sorted(columns)+['credit'])
    data.set_index('index').to_csv(f'credit_data/{name}.csv')

    return data
```

- ✅ 결측치가 있는 행을 버리지 않고 사용
- ✅ 기존 열들로부터 파생한 데이터를 생성하고 파생변수와 상관관계가 있는 기존 데이터를 제거
- ✅ dictionary를 사용한 수작업 대신 OrdinalEncoder 활용
- ✅ KMeans로 군집화한 결과를 새로운 데이터로 활용

'소회의실'팀 전처리

```python
def set_extra_features(df: pd.DataFrame, client_input: list) -> pd.DataFrame:
    data = df.copy()

    data['age'] = data['DAYS_BIRTH'] // 365
    data['month_birth'] = np.floor(data['DAYS_BIRTH']/30) - ((np.floor(data['DAYS_BIRTH']/30)/12).astype(int)*12)
    data['week_birth'] = np.floor(data['DAYS_BIRTH']/7) - ((np.floor(data['DAYS_BIRTH']/7)/4).astype(int)*4)

    data['career'] = data['DAYS_EMPLOYED'] // 365
    data['days_unemployed'] = data['DAYS_BIRTH'] - data['DAYS_EMPLOYED']
    data['month_unemployed'] = np.floor(data['days_unemployed']/30) - ((np.floor(data['days_unemployed']/30)/12).astype(int)*12)
    data['week_unemployed'] = np.floor(data['days_unemployed']/7) - ((np.floor(data['days_unemployed']/7)/4).astype(int)*4)

    data['days_income'] = data['income_total'] / (data['DAYS_BIRTH']+data['DAYS_EMPLOYED'])
    data['income_per'] = data['income_total'] / data['family_size']

    # id 열은 새로운 데이터에 적용하기 적절하지 않기 때문에 제거, 향후 중복 여부를 판단하는 열로 변경해서 시도
    # data['id'] = str()
    # for column in client_input:
    #     data['id'] += data[column].astype(str) + '_'

    return data
```

DAYS_BIRTH와 DAYS_EMPLOYED를 각각 연, 월, 주 단위로 구분, 원본 데이터 제거

DAYS_EMPLOYED와 family_size에 대한 income_total 계산 결과를 새로운 데이터로 활용

'소회의실'팀 전처리

```python
def set_ordinal_encoding(df: pd.DataFrame, name: str) -> pd.DataFrame:
    data = df.copy()

    num_features = data.dtypes[data.dtypes != 'object'].index.tolist()
    cat_features = data.dtypes[data.dtypes == 'object'].index.tolist()
    if name == 'train':
        ordianl_encoder = OrdinalEncoder(cat_features)
        data[cat_features] = ordianl_encoder.fit_transform(data[cat_features],data['credit'])
        joblib.dump(ordianl_encoder, 'credit_data/train_ord_pipe.pkl')
    else:
        ordianl_encoder = joblib.load('credit_data/train_ord_pipe.pkl')
        data[cat_features] = ordianl_encoder.transform(data[cat_features])
    # data['id'] = data['id'].astype('int64')

    if name == 'train':
        make_pipeline(data, num_features, cat_features)

    return data
```

`OrdinalEncoder`를 활용해 범주형 데이터를 한번에 숫자로 변경

train 데이터 전처리 시에 한해서 `OneHotEncoder` 등이 포함된 `Pipeline` 추가 생성

'소회의실'팀 전처리

```python
def set_clustering(df: pd.DataFrame, name: str) -> pd.DataFrame:
    data = df.copy()

    train = data
    if name == 'train':
        train = data.drop(['credit'], axis=1)
        kmeans = KMeans(n_clusters=36, random_state=42).fit(train)
        joblib.dump(kmeans, 'models/model_train_kmeans.pkl')

    kmeans = joblib.load('models/model_train_kmeans.pkl')
    data['cluster'] = kmeans.predict(train)

    return data
```

KMeans를 통해 군집화된 결과를 새로운 데이터로 활용

Pipeline 설정

```python
def make_pipeline(df: pd.DataFrame, num_features: list, cat_features: list):
    data = df.drop(['index','credit'], axis=1).copy()
    num_features = [feature for feature in num_features if feature not in ['index','credit']]
    cat_features = [feature for feature in cat_features if feature not in ['index','credit']]

    numerical_transformer = StandardScaler()
    categorical_transformer = OneHotEncoder(categories='auto', handle_unknown='ignore')

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numerical_transformer, num_features),
            ('cat', categorical_transformer, cat_features)])

    pipe = Pipeline(steps=[('preprocessor', preprocessor)])
    pipe.fit(data)
    joblib.dump(pipe, 'credit_data/train_pipe.pkl')
```

Numeric 열과 Categorical 열을 구분해서 각각의 Encoder 적용

처음 생성 시 train 데이터에 fitting한 상태로 저장해놓고 다른 함수에서 활용

'이지금'팀 전처리 vs '소회의실'팀 전처리

```
1  x_train, x_test, y_train, y_test = credit_data.load_data()
2  print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(15906, 67) (6818, 67) (15906, 1) (6818, 1)
```

CheckPoint 예측 결과는 모든 test 데이터에 대해서 측정해야함
train 데이터와 동일한 방법으로 test 데이터의 행을 제거하면 안되는 것!
(두 팀 간 정확한 비교가 어려움 인지함)

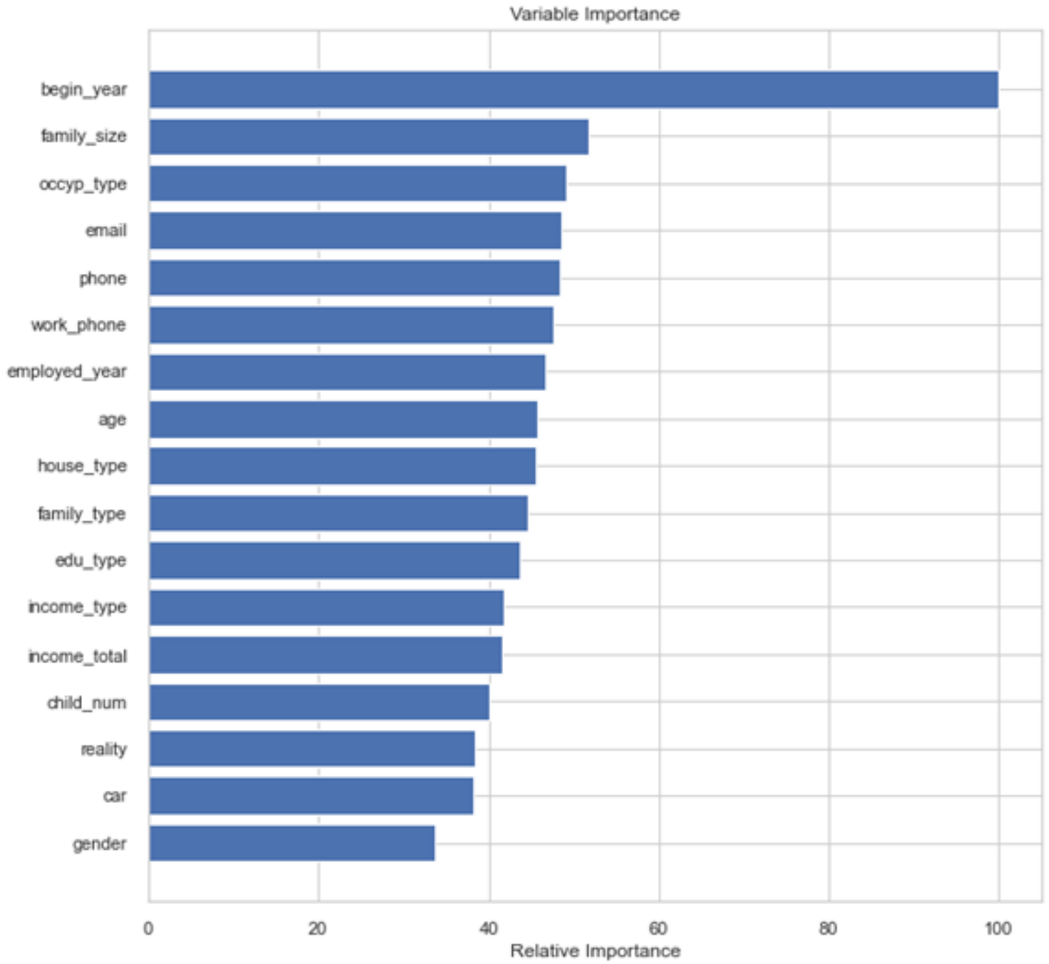
```
1  x_train, x_test, y_train, y_test = credit_data.load_train_data()
2  print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

✓ 0.2s

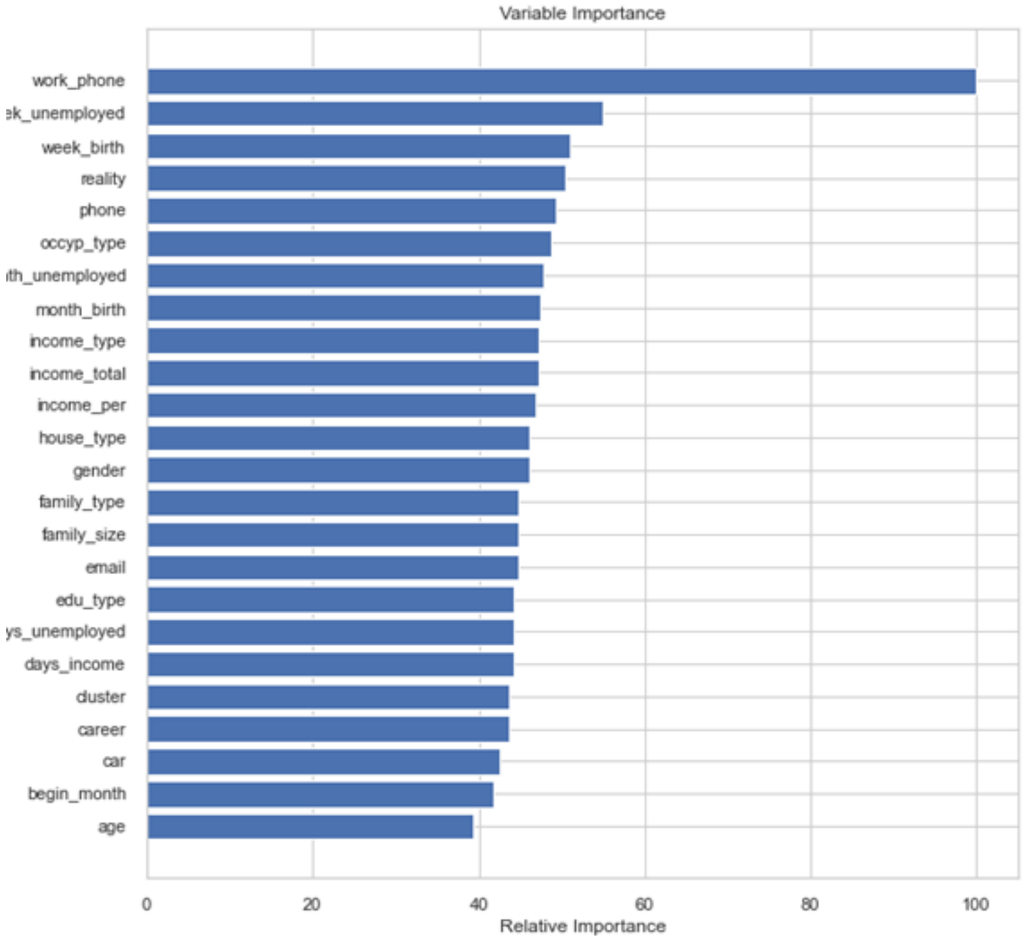
```
(18519, 65) (7938, 65) (18519, 1) (7938, 1)
```

Feature Importances (xgboost)

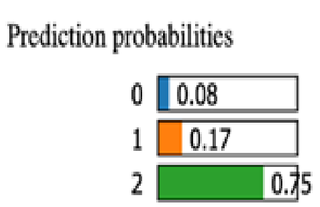
이지금팀

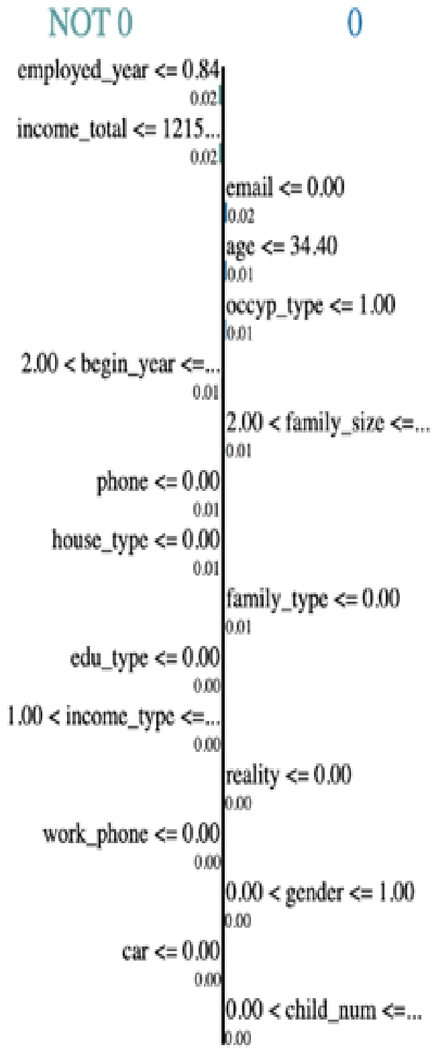


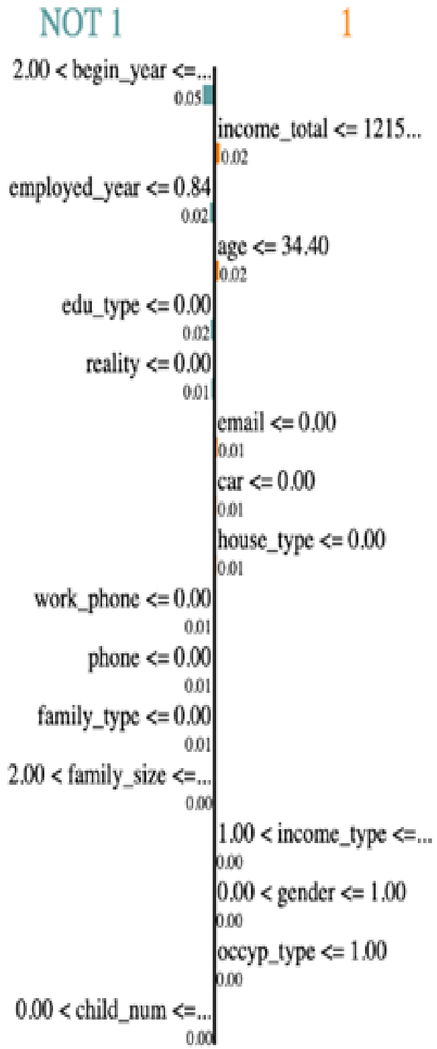
소회의실팀

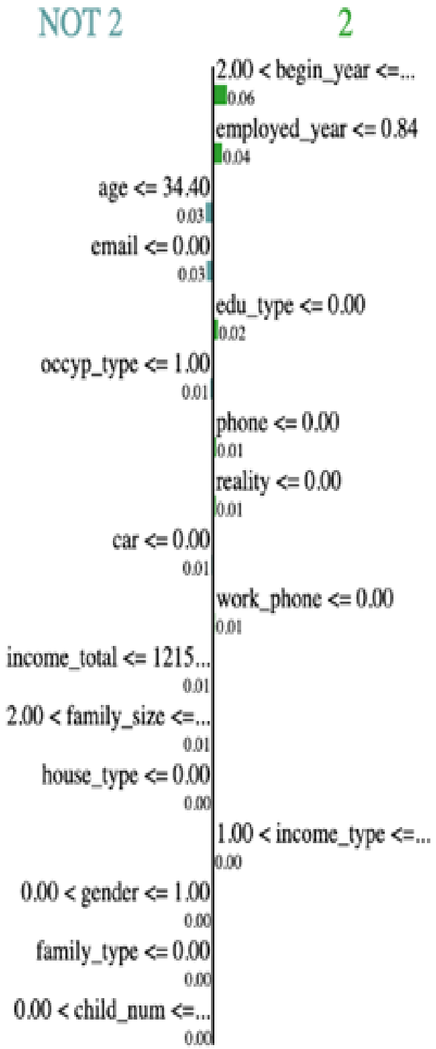


이지금 팀 test 데이터 첫 번째 행 LIME



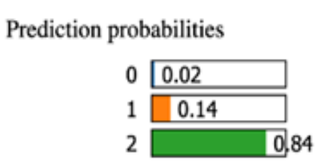


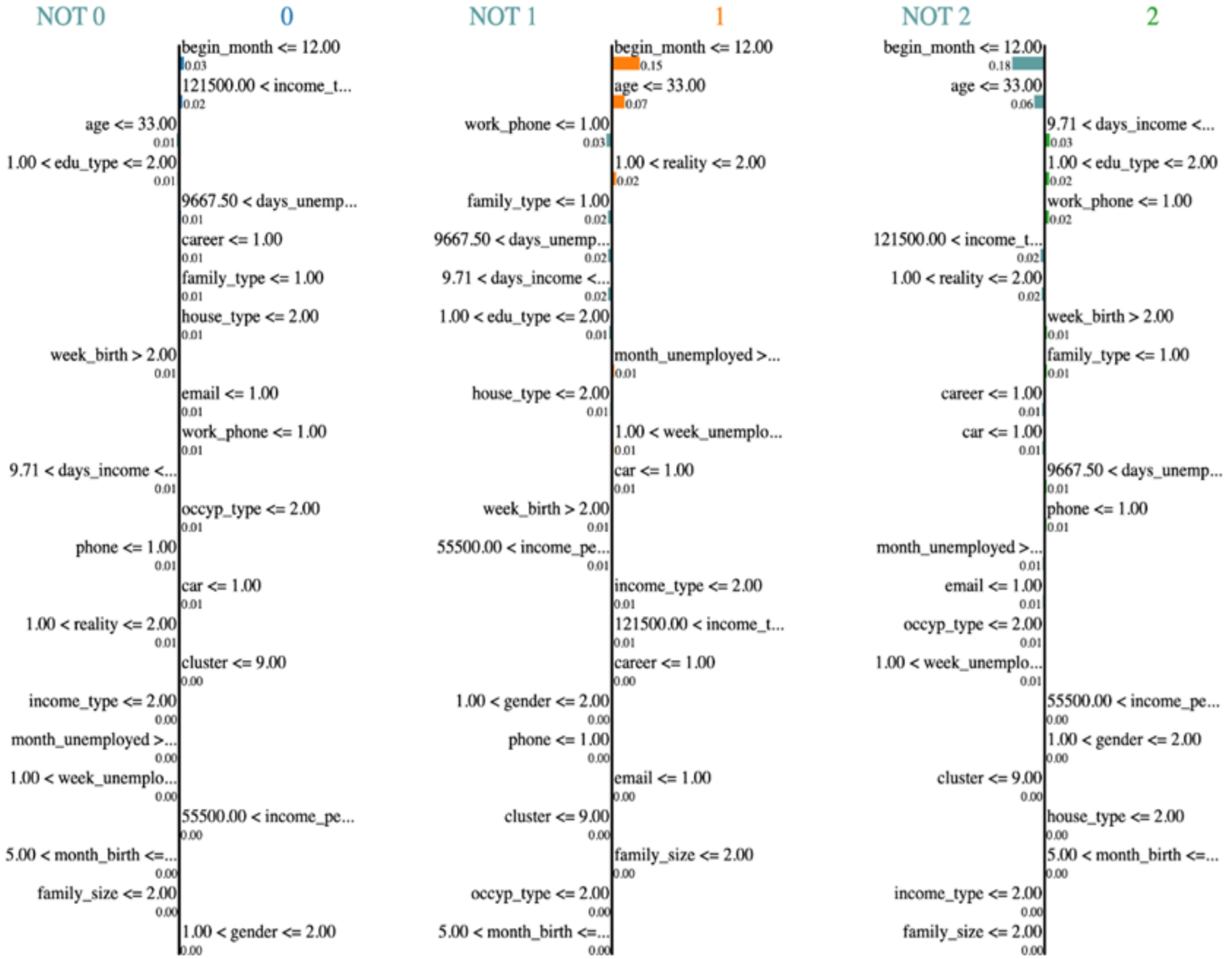




Feature	Value
employed_year	0.00
income_total	90000.00
email	0.00
age	27.02
occyp_type	0.00
begin_year	2.42
family_size	3.00
phone	0.00
house_type	0.00
family_type	0.00
edu_type	0.00
income_type	2.00
reality	0.00

소회의실 팀 test 데이터 첫 번째 행 LIME

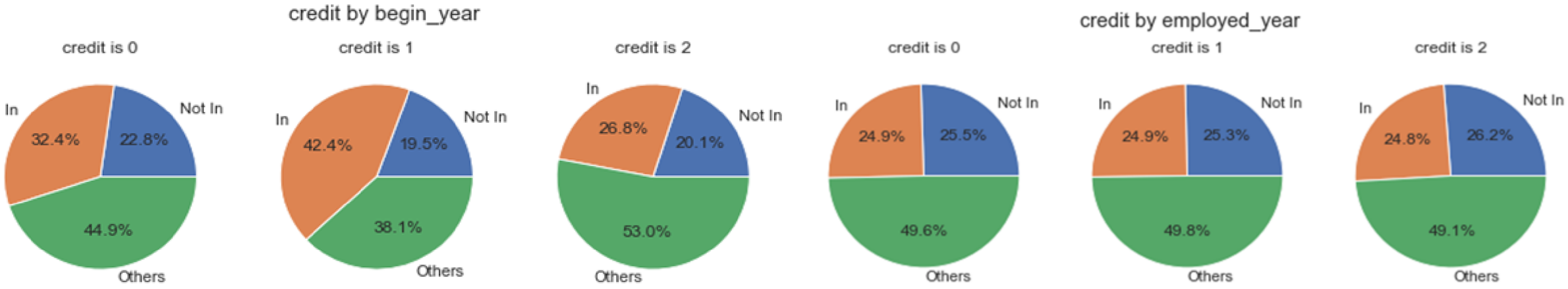




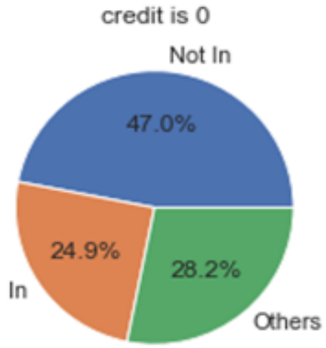
Feature	Value
begin_month	6.00
income_total	153000.00
age	33.00
edu_type	2.00
days_unemployed	11864.00
career	0.00
family_type	1.00
house_type	2.00
week_birth	3.00
email	1.00
work_phone	1.00
days_income	12.26
occyp_type	2.00

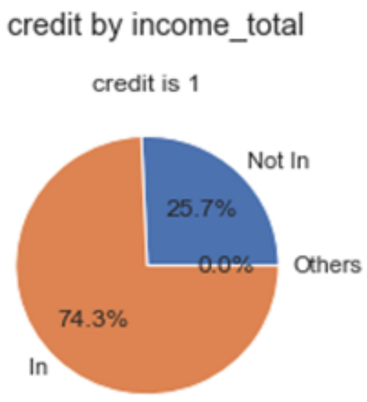
이지금 팀 LIME 기준 EDA (1)

LIME이 특정 credit 범주에 속하거나 속하지 않는다고 판단한 범위가 얼마나 정확한지 시각화
각각의 범주에서 In의 비중이 클수록, Not In의 비중이 작을수록 정확한 판단을 내렸다고 생각할 수 있음
Others는 판단이 불명확한 범위



이지금 팀 LIME 기준 EDA (2)



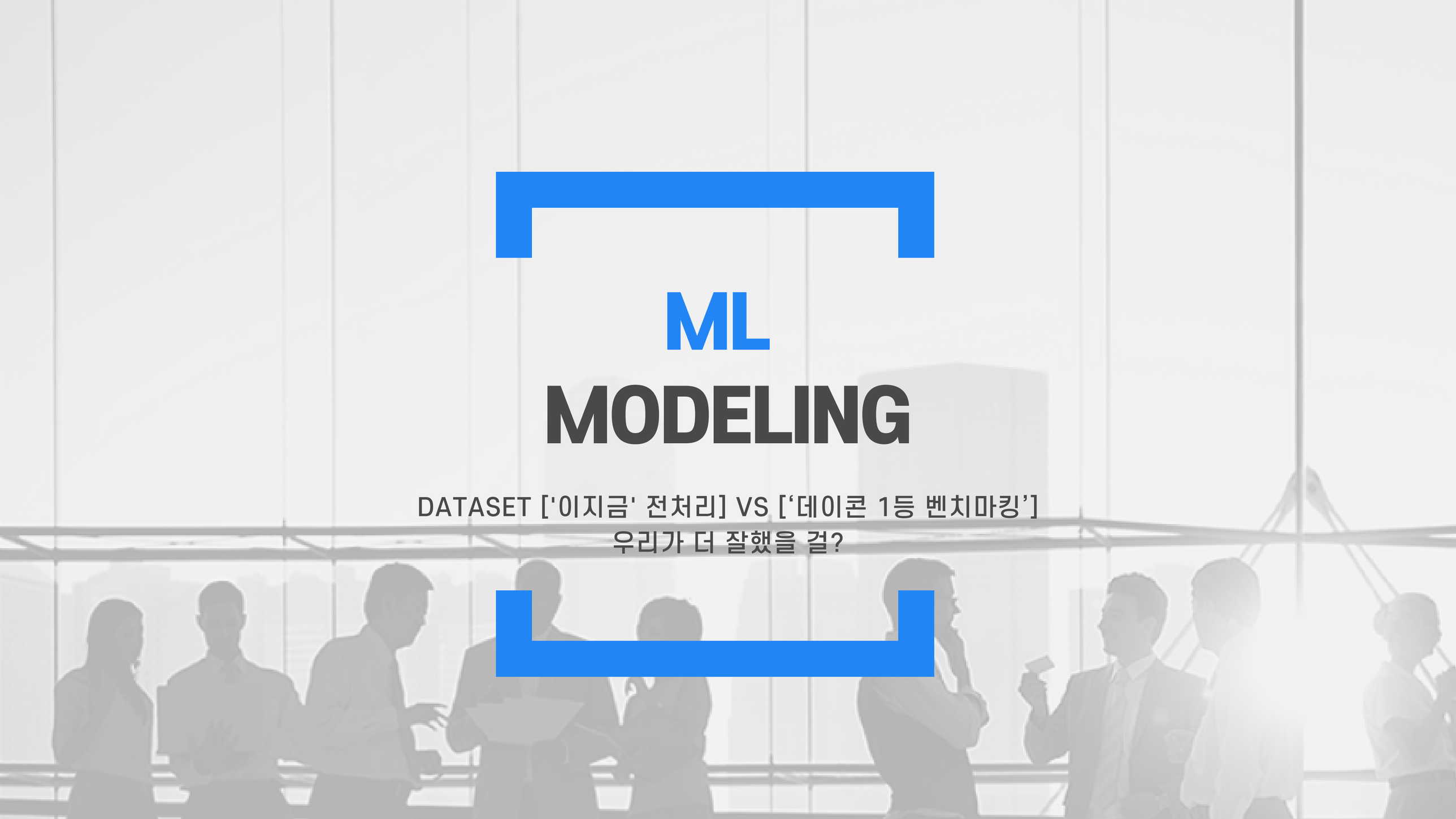


ML

MODELING

DATASET ['이지금' 전처리] VS [‘데이콘 1등 벤치마킹’]

우리가 더 잘했을 걸?



INTRO

MLMODEL 시도

AutoML

Pycaret

Base

개별 모델 비교

(XGB, LGBM, NGBoost etc.)

Ensemble

Stacking

Pycaret - 이지금팀 전처리 기반 결과-1 (모델별 비교)

1) logloss 값 기준으로 정렬 후 상위 5개의 모델 선택!

```
#svm, ridge는 predict_proba 미지원으로 제외
best5 = compare_models(fold = 5, sort = 'logloss', n_select = 5, exclude=['svm','ridge'])
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss	TT (Sec)
rf	Random Forest Classifier	0.7043	0.7354	0.4973	0.6774	0.6682	0.3318	0.3578	0.7765	2.556
lightgbm	Light Gradient Boosting Machine	0.6943	0.6849	0.4237	0.6773	0.6181	0.2326	0.3110	0.7792	0.758
xgboost	Extreme Gradient Boosting	0.6943	0.6912	0.4405	0.6652	0.6309	0.2557	0.3130	0.7808	9.940
gbc	Gradient Boosting Classifier	0.6912	0.6531	0.4109	0.6568	0.6066	0.2119	0.3025	0.7966	8.542
lda	Linear Discriminant Analysis	0.6419	0.6125	0.3387	0.5274	0.5100	0.0174	0.0594	0.8632	0.204
dummy	Dummy Classifier	0.6400	0.5000	0.3333	0.4096	0.4995	0.0000	0.0000	0.8832	0.050
nb	Naive Bayes	0.6400	0.5688	0.3333	0.4096	0.4995	0.0000	0.0000	0.8836	0.068
lr	Logistic Regression	0.6400	0.5141	0.3333	0.4096	0.4995	0.0000	0.0000	0.9008	1.284
ada	Ada Boost Classifier	0.6910	0.6277	0.4062	0.6299	0.6021	0.2032	0.3045	1.0802	0.828
et	Extra Trees Classifier	0.6780	0.7134	0.5013	0.6507	0.6535	0.3039	0.3152	1.0906	3.006
knn	K Neighbors Classifier	0.5323	0.5095	0.3365	0.4845	0.5039	0.0055	0.0056	4.9191	0.556
qda	Quadratic Discriminant Analysis	0.5972	0.5018	0.3364	0.4851	0.5060	0.0048	0.0068	13.9123	0.120
dt	Decision Tree Classifier	0.5812	0.6125	0.4542	0.5904	0.5855	0.2077	0.2079	14.4660	0.252

Pycaret - 이지금팀 전처리 기반 결과-2 (모델별 비교)

2) logloss, accuracy, auc 모두 고려해 모델 재선택!

```
[20] #lda 와 nb는 losloss가 좋지만 accuracy, auc가 상대적으로 좋지 않아 커스텀모델 생성
     custom_model = compare_models(fold = 5, sort = 'logloss', n_select = 6, include=['lightgbm','gbc','rf','ada', 'lda', 'nb'])
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss	TT (Sec)
rf	Random Forest Classifier	0.7043	0.7354	0.4973	0.6774	0.6682	0.3318	0.3578	0.7765	2.664
lightgbm	Light Gradient Boosting Machine	0.6943	0.6849	0.4237	0.6773	0.6181	0.2326	0.3110	0.7792	1.866
gbc	Gradient Boosting Classifier	0.6912	0.6531	0.4109	0.6568	0.6066	0.2119	0.3025	0.7966	8.598
lda	Linear Discriminant Analysis	0.6419	0.6125	0.3387	0.5274	0.5100	0.0174	0.0594	0.8632	0.214
nb	Naive Bayes	0.6400	0.5688	0.3333	0.4096	0.4995	0.0000	0.0000	0.8836	0.070
ada	Ada Boost Classifier	0.6910	0.6277	0.4062	0.6299	0.6021	0.2032	0.3045	1.0802	0.826

Pycaret - 이지금팀 전처리 기반 결과-3 (Logloss반영)

3) 선택된 모델로 Blended 모델 생성! 정확도 0.6888 , logloss 0.8045

```
blended_custom = blend_models(estimator_list = custom_model, fold = 5, optimize = 'logloss')
pred_holdout_custom = predict_model(blended_custom)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
0	0.6936	0.7209	0.4103	0.6845	0.6066	0.2129	0.3136	0.7993
1	0.6913	0.7263	0.4105	0.6656	0.6062	0.2097	0.3036	0.7987
2	0.6944	0.7159	0.4125	0.7071	0.6088	0.2156	0.3158	0.8018
3	0.6970	0.7122	0.4144	0.6918	0.6113	0.2219	0.3255	0.7984
4	0.6885	0.7183	0.4042	0.7378	0.5987	0.1946	0.2962	0.8040
Mean	0.6929	0.7187	0.4104	0.6973	0.6063	0.2110	0.3109	0.8004
Std	0.0029	0.0047	0.0035	0.0242	0.0042	0.0091	0.0101	0.0021

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
0	Voting Classifier	0.6888	0.722	0.4095	0.6768	0.6001	0.2073	0.3065	0.8045

Pycaret - 소회의실팀 전처리 기반 결과-1 (모델별 비교)

1) logloss 값 기준으로 정렬 후 상위 5개의 모델 선택!

```
[53] #svm, ridge는 predict_proba 미지원으로 제외
     best5 = compare_models(fold = 5, sort = 'logloss', n_select = 5, exclude=['svm','ridge'])
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.6968	0.6965	0.4263	0.6864	0.6206	0.2378	0.3188	0.7719	0.656
xgboost	Extreme Gradient Boosting	0.6957	0.7056	0.4393	0.6670	0.6302	0.2552	0.3160	0.7737	9.628
rf	Random Forest Classifier	0.7067	0.7461	0.5135	0.6798	0.6770	0.3523	0.3712	0.7848	2.376
gbc	Gradient Boosting Classifier	0.6929	0.6564	0.4136	0.6560	0.6092	0.2179	0.3064	0.7957	8.336
lda	Linear Discriminant Analysis	0.6435	0.6101	0.3400	0.5563	0.5122	0.0199	0.0707	0.8640	0.198
dummy	Dummy Classifier	0.6405	0.5000	0.3333	0.4103	0.5002	0.0000	0.0000	0.8840	0.046
nb	Naive Bayes	0.6405	0.5672	0.3333	0.4103	0.5002	0.0000	0.0000	0.8847	0.060
lr	Logistic Regression	0.6405	0.5182	0.3333	0.4103	0.5002	0.0000	0.0000	0.8928	0.890
ada	Ada Boost Classifier	0.6918	0.6283	0.4079	0.6515	0.6035	0.2062	0.3057	1.0806	0.780
et	Extra Trees Classifier	0.6778	0.7194	0.5089	0.6518	0.6565	0.3129	0.3217	1.1438	2.280
knn	K Neighbors Classifier	0.5341	0.5055	0.3382	0.4864	0.5058	0.0091	0.0093	4.9439	0.480
qda	Quadratic Discriminant Analysis	0.5986	0.5034	0.3367	0.4886	0.5082	0.0079	0.0110	13.8638	0.116
dt	Decision Tree Classifier	0.5801	0.6094	0.4613	0.5900	0.5846	0.2074	0.2076	14.5040	0.248

Pycaret - 소회의실팀 전처리 기반 결과-2 (모델 블렌딩)

2) 선택된 모델로 Blended 모델 생성! 정확도 0.6861 , logloss 0.8029

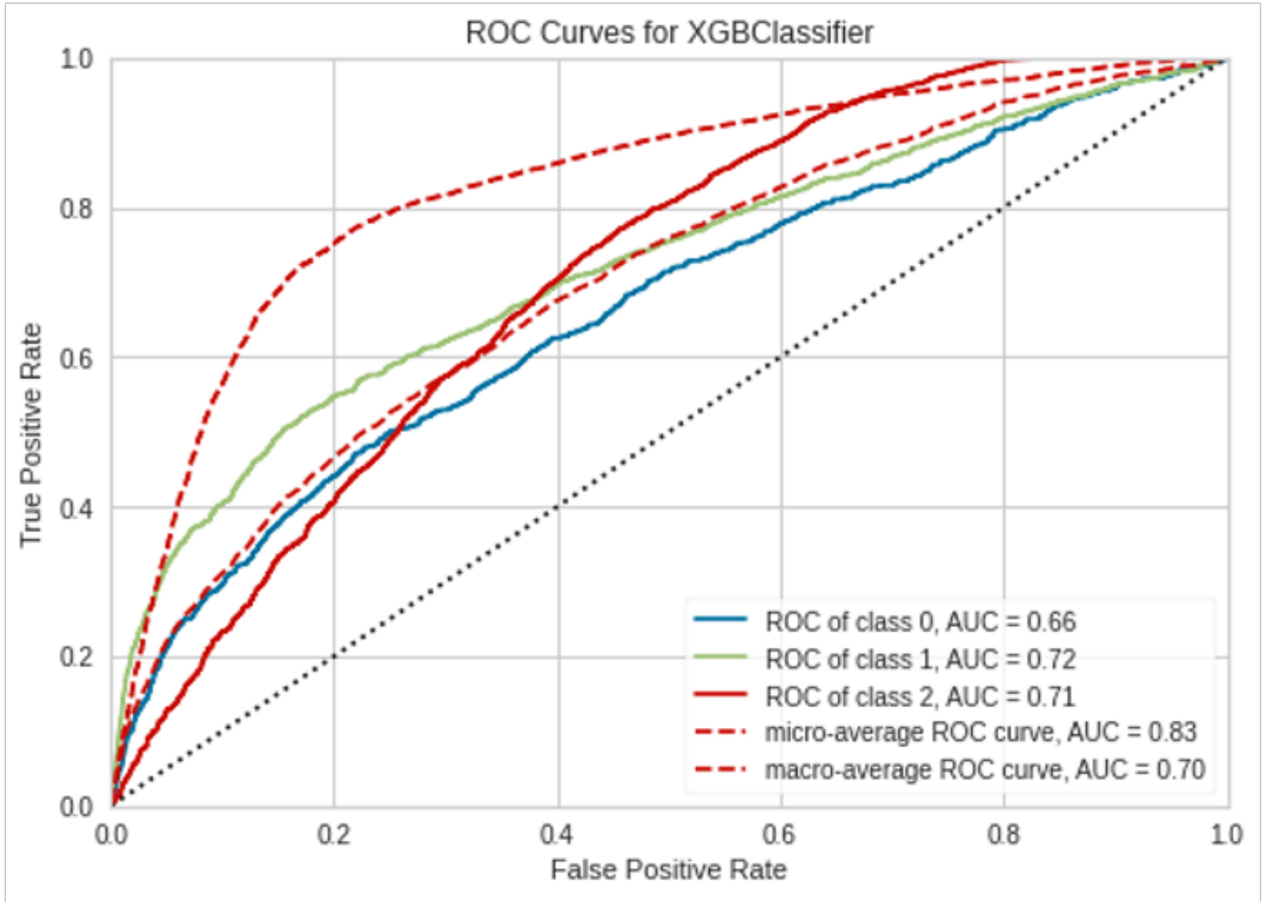
```
[56] blended_custom = blend_models(estimator_list = custom_model, fold = 5, optimize = 'logloss')
     pred_holdout_custom = predict_model(blended_custom)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
0	0.6976	0.7240	0.4170	0.6659	0.6137	0.2273	0.3238	0.7944
1	0.6922	0.7257	0.4090	0.6630	0.6047	0.2077	0.3079	0.7997
2	0.6969	0.7446	0.4157	0.7053	0.6118	0.2211	0.3246	0.7910
3	0.6917	0.7303	0.4075	0.7395	0.6030	0.2061	0.3055	0.7980
4	0.6904	0.7291	0.4042	0.6775	0.5998	0.1993	0.3014	0.7987
Mean	0.6938	0.7307	0.4107	0.6902	0.6066	0.2123	0.3126	0.7964
Std	0.0029	0.0073	0.0049	0.0288	0.0053	0.0103	0.0097	0.0032

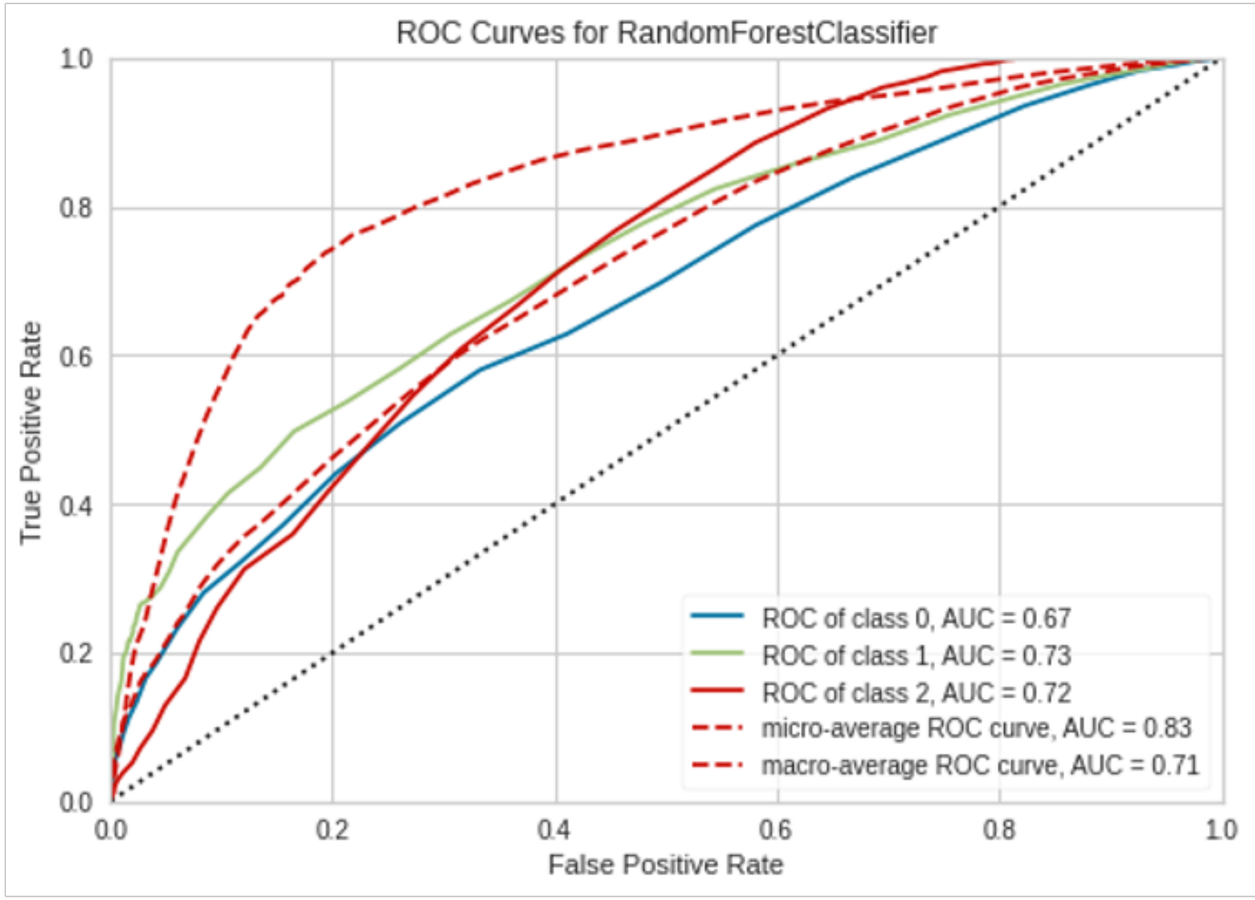
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
0	Voting Classifier	0.6861	0.7377	0.4112	0.7459	0.5982	0.2106	0.3152	0.8029

AUC 곡선 비교

이지금팀 정확도 0.6888 , logloss 0.8045



소희의실팀 정확도 0.6861 , logloss 0.8029



개별 모델 비교 - 이지금팀 vs 소회의실팀 (dacon)

model / Accuracy	이지금 팀	소회의실 팀
XGBoost	0.707	0.716
LightGBM	0.706	0.716
NGBoost	0.642	0.692
NaiveBayes	0.634	0.642
RandomForest	0.692	0.695
Decision Tree	0.691	0.693
SVM	0.681	0.681
KNN	0.647	0.652
Logistic Reg.	0.642	0.643

졌다..

그럼에도 '크게' 밀리진 않는다

Model Stacking (Table! with code)

	정확도	이지금 팀	소회의실 팀
5-fold	train	0.703	0.7065
	test	0.71	0.7189
10-fold	train	0.71	0.7146
	test	**0.7119**	**0.7199**

이지금팀

5-fold
SVM, RandomForest, NaiveBayes, XGBoost

10 - fold
SVM, RandomForest, NaiveBayes, XGBoost, KNN, LightGBM

소회의실 팀

5-fold
SVM, RandomForest, XGBoost, DecisionTree

10-fold
KNN, RandomForest, XGBoost, NaiveBayes, LightGBM

ML Model 성능 비교 총 정리

이지금팀		소회의실팀
0.6668	Auto ML - Pycaret	0.6861
9패	개별 모델 비교	9승
정확도 0.7119	Model Stacking	정확도 0.7199

DL
MODELING

DATASET ['이지금' 전처리] VS [‘데이콘 1등 벤치마킹’]
우리가 더 잘했을 걸?

[DL Modelling]

‘이지금’ 팀 최적화 가즈아!

INTRO

DLMODEL 시도

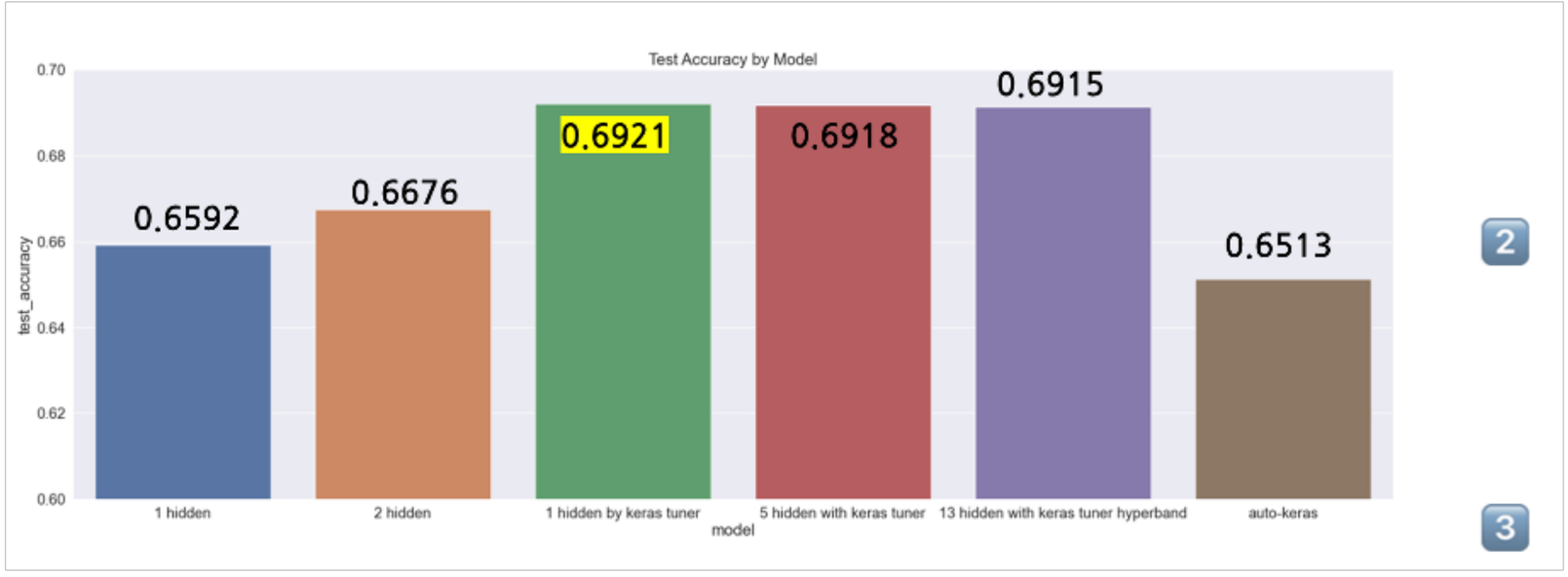
NN & DNN

Keras tuner

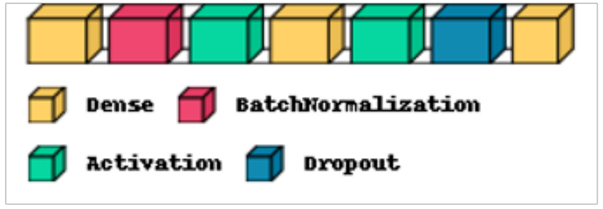
Auto Keras

Tabnet

NN, DNN, Auto keras 성능 비교



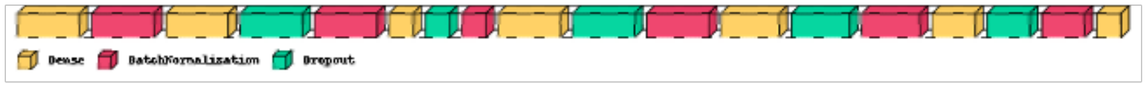
1



2

3

4



6

5

Keras Tuner


```python
def build_hyper_model(hp):
    model = models.Sequential()

    hp_units = hp.Int('units_input', min_value=32, max_value=512, step=32)
    hp_activations = hp.Choice('activation_input', values=['relu', 'elu'])
    model.add(layers.Dense(input_dim=67, units=hp_units, activation=hp_activations, kernel_initializer=he_uniform()))
    model.add(layers.BatchNormalization())

    for layer_num in range(hp.Int('num_layers', min_value=1, max_value=5)):
        hp_units = hp.Int('units_' + str(layer_num), min_value=32, max_value=512, step=32)
        hp_activations = hp.Choice('activation_' + str(layer_num), values=['relu', 'elu'])
        model.add(layers.Dense(hp_units, activation=hp_activations, kernel_initializer=he_uniform()))
        hp_dropouts = hp.Float('dropout_' + str(layer_num), 0.0, 0.5, step=0.1)
        model.add(layers.Dropout(hp_dropouts))
        model.add(layers.BatchNormalization())
    model.add(layers.Dense(units=3, activation='softmax'))

    hp_learning_rate = hp.Choice('learning_rate', values = [1e-2, 1e-3, 1e-4])
    model.compile(optimizer=Adam(hp_learning_rate),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Keras Tuner


```
tuner = kt.BayesianOptimization(build_hyper_model,
                                objective = 'val_accuracy',
                                max_trials = 100,
                                directory = 'keras_tuner',
                                project_name = 'bayes_depth5')
```

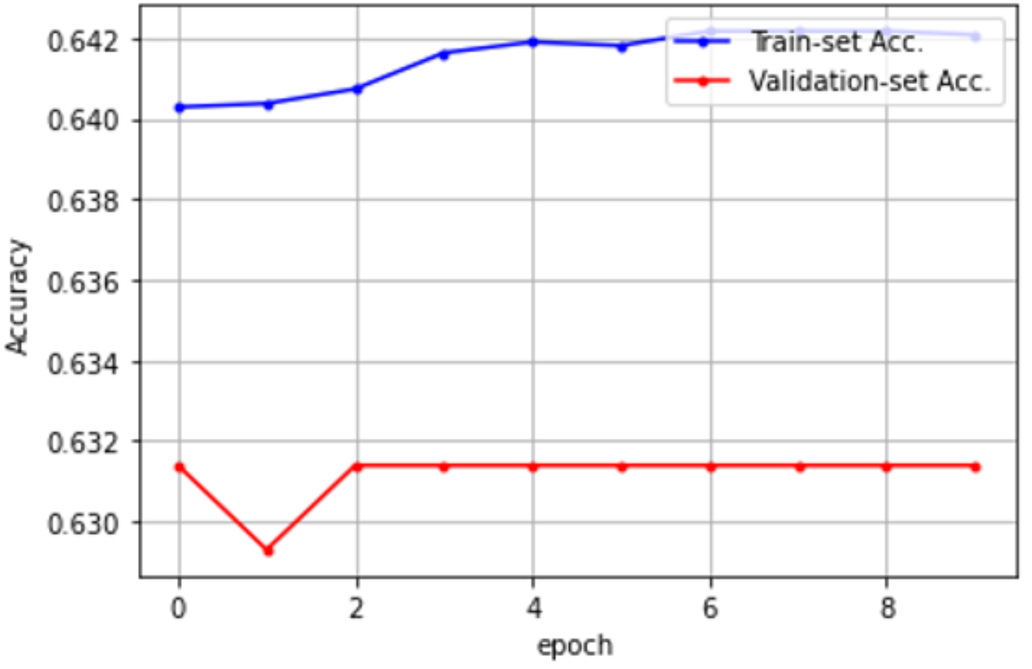
```
1  tuner.search(train_data, train_label, batch_size=100, epochs=10, validation_data=(test_data, test_label))
```

```
Trial 100 Complete [00h 01m 14s]
val_accuracy: 0.6887650489807129

Best val_accuracy So Far: 0.6921384930610657
Total elapsed time: 00h 22m 01s
```

AutoKeras

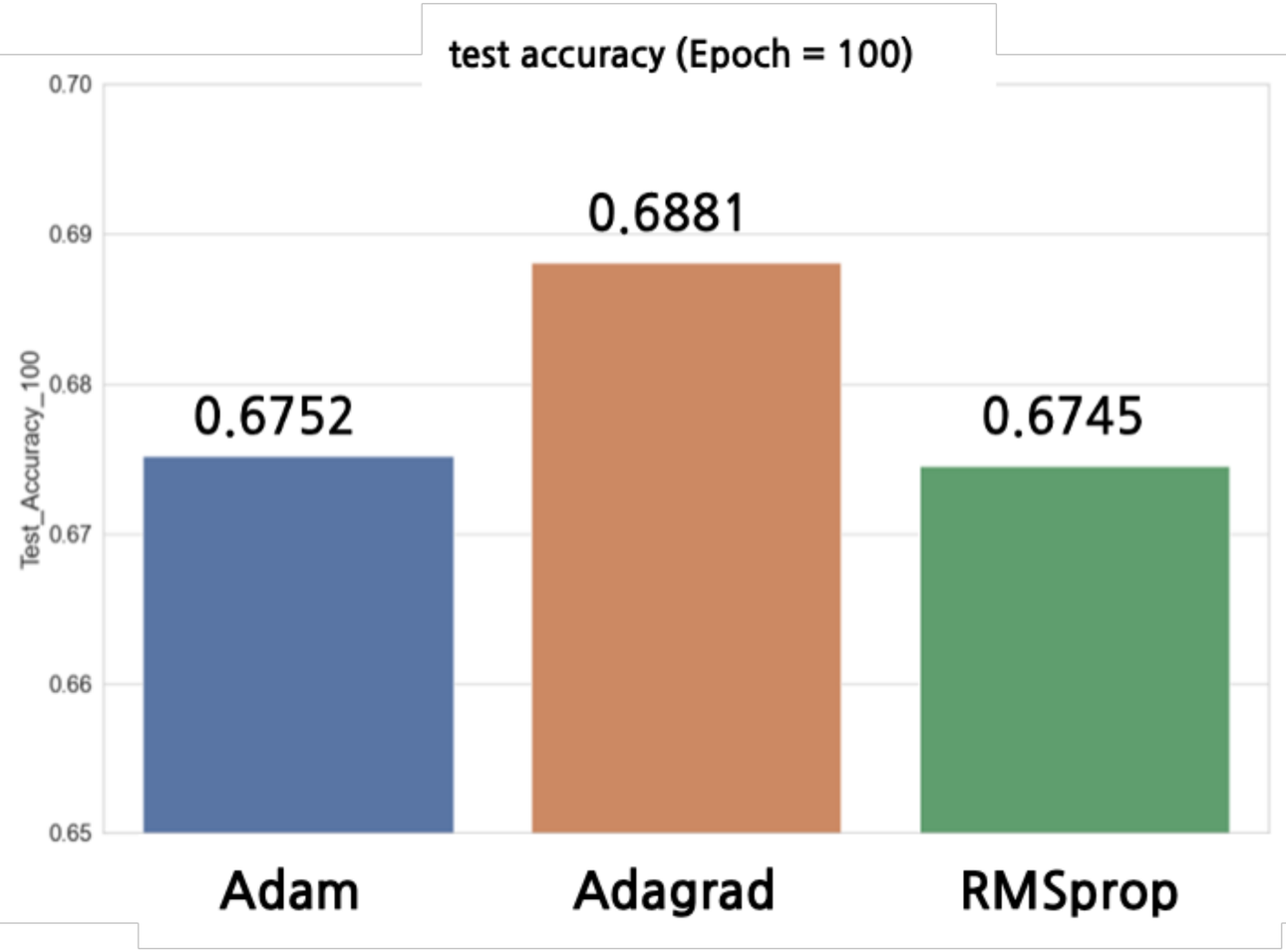

```
1  clf = ak.StructuredDataClassifier()
2  clf.fit(train_data, train_label)
3
4  accuracy = clf.evaluate(test_data, test_label)
5  result = clf.predict(test_data)
6
7  print(accuracy)
8  print(result)
```

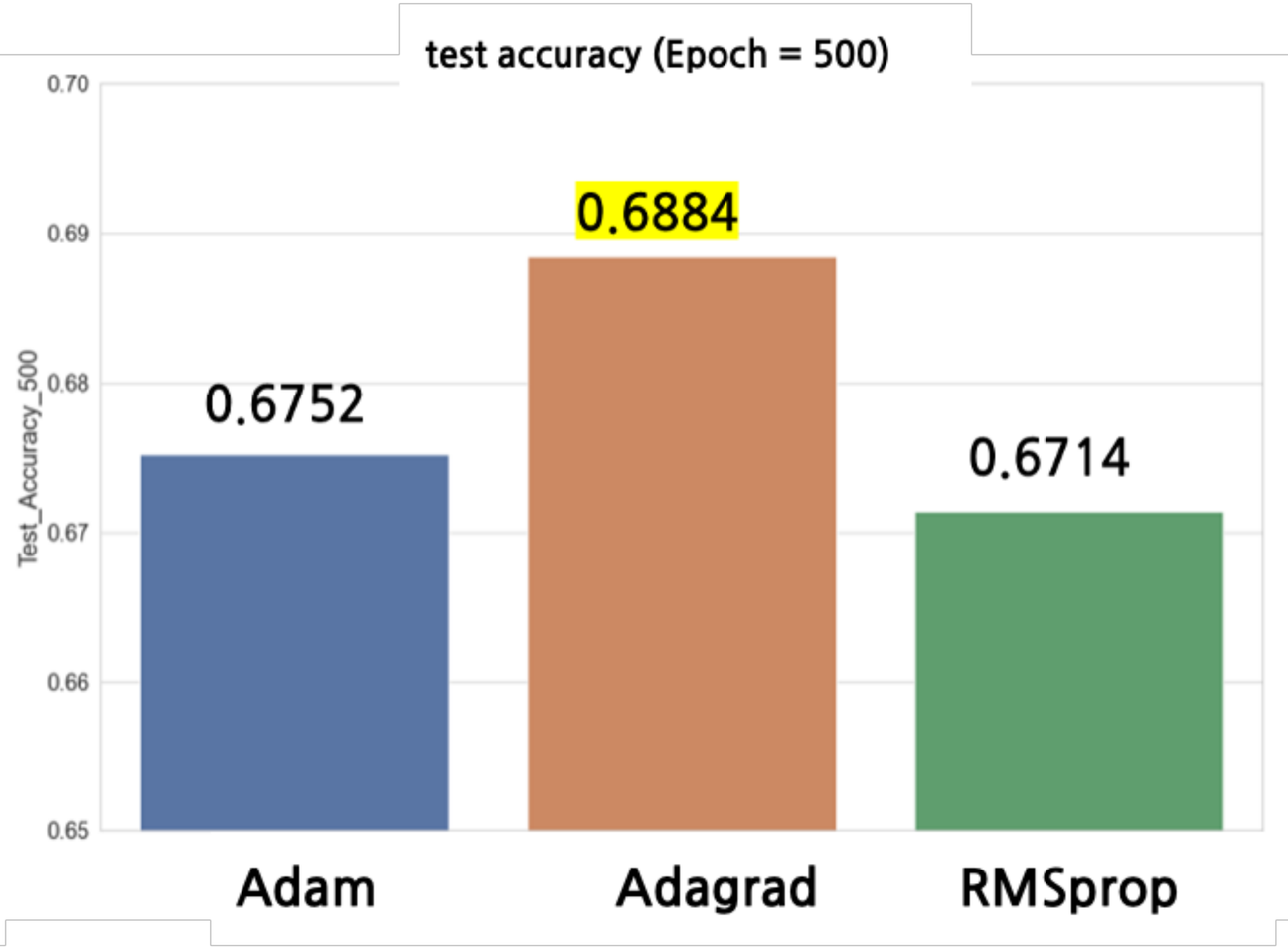
```
Trial 65 Complete [00h 02m 50s]
val_accuracy: 0.6347992420196533

Best val_accuracy So Far: 0.6513702869415283
Total elapsed time: 03h 46m 35s
```



Tabnet

Table (정형 데이터)를 위한 효과적인 딥러닝(인공신경망) 모델
Optimizer 와 epoch 을 다양하게 적용!





DL 성능 비교 총 정리!!

Model	Accuracy
🥇 Keras Tuner	0.6921
🥈 TabNet	0.6884
🥉 Auto Keras	0.6513

ML VS DL
성능비교

이지금 팀

ML Model 성능 비교 총 정리

ML 최고성능		DL 최고성능
Model Stacking		Keras Tuner
0.7119	정확도	0.6921

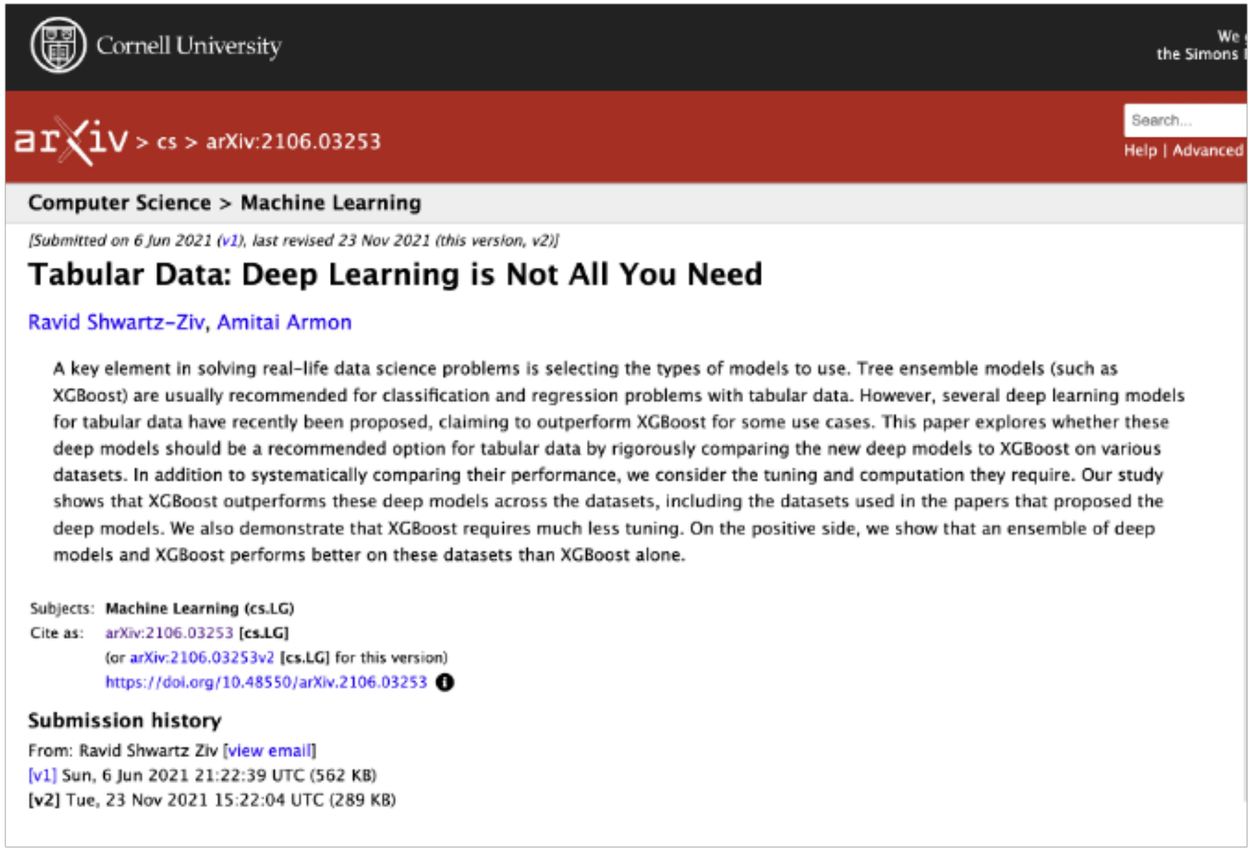
왜 머신러닝이 더 잘 나올까??

[Tabular Data: Deep Learning is Not All You Need]

이 논문에 따르면 tabular 데이터에서
XGBoost 알고리즘은 딥러닝 모델들보다

우수한 성능을 낸다.
- even less tuned!

https://arxiv.org/abs/2106.03253



Cornell University

arXiv > cs > arXiv:2106.03253

Search... Help | Advanced

Computer Science > Machine Learning

[Submitted on 6 Jun 2021 (v1), last revised 23 Nov 2021 (this version, v2)]

Tabular Data: Deep Learning is Not All You Need

Ravid Shwartz-Ziv, Amitai Armon

A key element in solving real-life data science problems is selecting the types of models to use. Tree ensemble models (such as XGBoost) are usually recommended for classification and regression problems with tabular data. However, several deep learning models for tabular data have recently been proposed, claiming to outperform XGBoost for some use cases. This paper explores whether these deep models should be a recommended option for tabular data by rigorously comparing the new deep models to XGBoost on various datasets. In addition to systematically comparing their performance, we consider the tuning and computation they require. Our study shows that XGBoost outperforms these deep models across the datasets, including the datasets used in the papers that proposed the deep models. We also demonstrate that XGBoost requires much less tuning. On the positive side, we show that an ensemble of deep models and XGBoost performs better on these datasets than XGBoost alone.

Subjects: Machine Learning (cs.LG)
Cite as: arXiv:2106.03253 [cs.LG]
(or arXiv:2106.03253v2 [cs.LG] for this version)
https://doi.org/10.48550/arXiv.2106.03253

Submission history

From: Ravid Shwartz Ziv [view email]
[v1] Sun, 6 Jun 2021 21:22:39 UTC (562 KB)
[v2] Tue, 23 Nov 2021 15:22:04 UTC (289 KB)

Extra. 더 시도해보고 싶었던 것들

- 모든 열을 조합해 id 카테고리를 생성 :

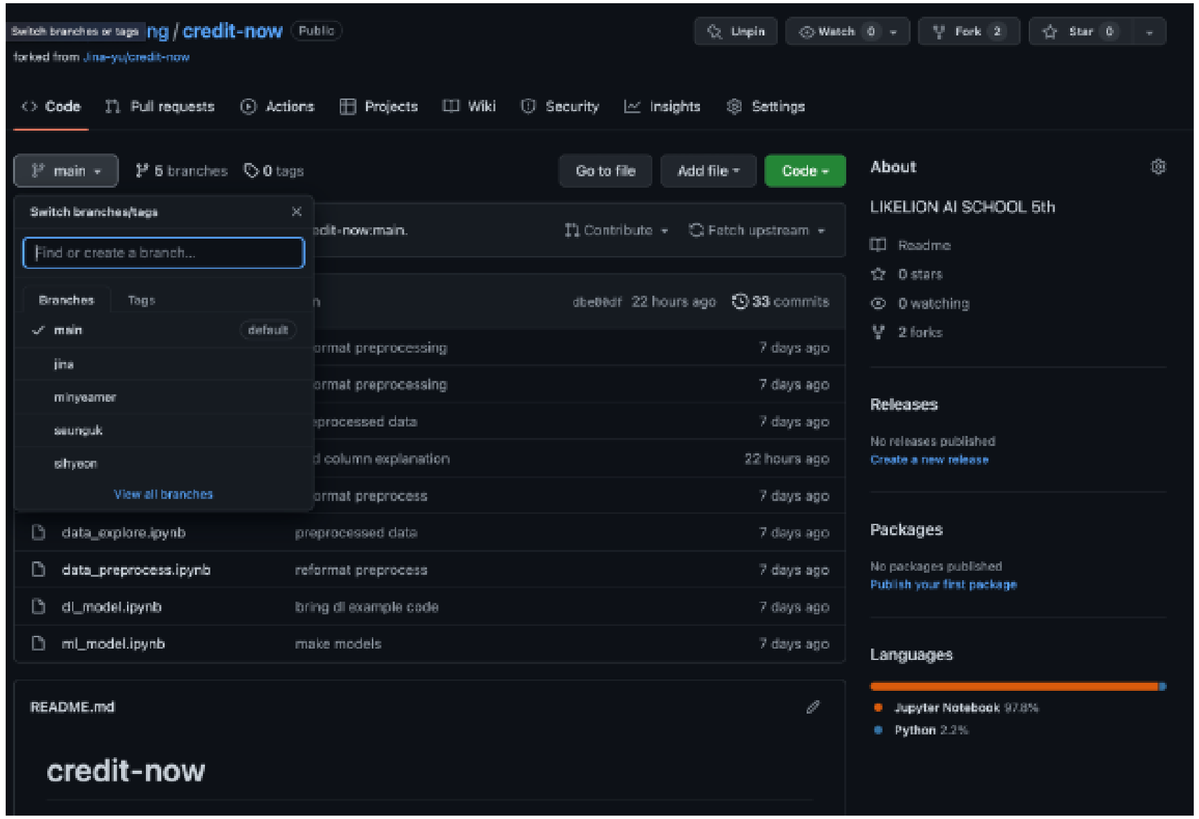
  고유값을 생성해 신용카드 중복 사용 유저를 걸러내기 위함.

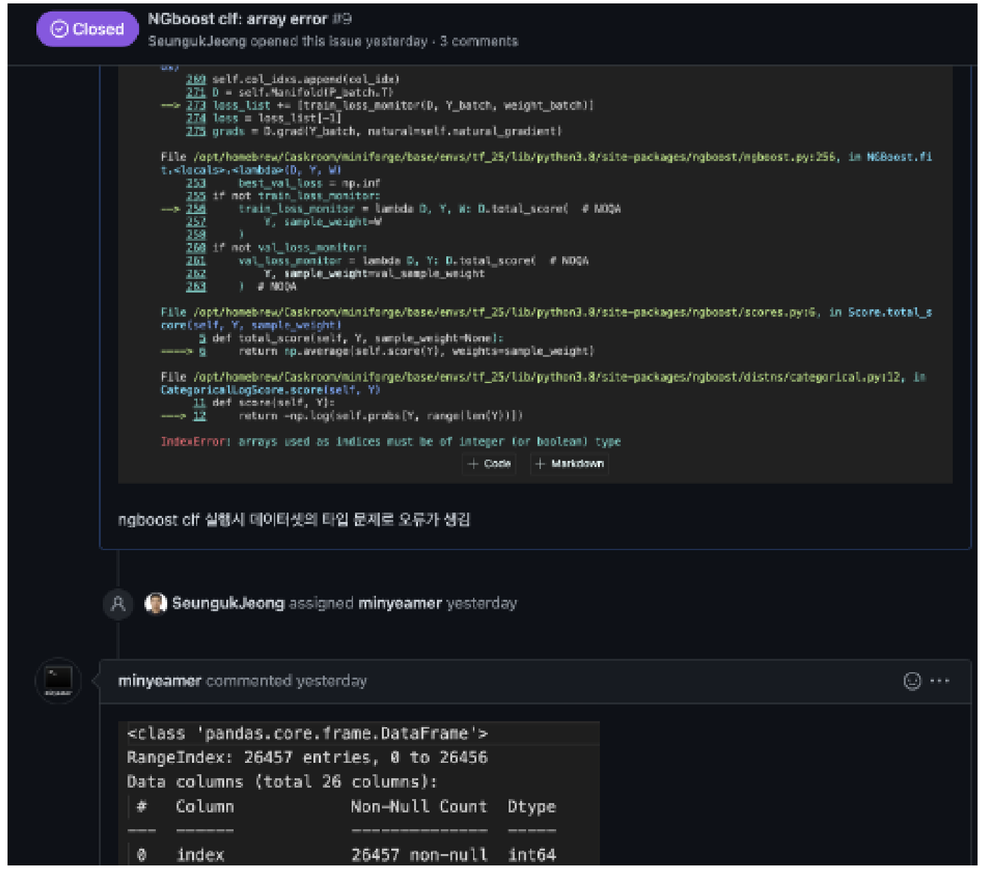
- Numerical 로 id의 속성을 지정해 성능을 높였으나

  그 이유를 뒷받침 하지 못하며 성능 향상의 이유를 찾을 수 없음

```
#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)
```

Extra. Github 사용 소감: 협업과 이슈 정리의 편리함





각자 브랜치를 생성하고

민엽님이 만드신 모듈을 import하는 방식

정해진 규칙 안에서 각자의 자유도를 높일 수 있었다.

이슈 정리도 브랜치 내 코드에 남기거나 캡처하며 적극적으로 소통

Q&A

궁금한 사항에 대해 편하게 질문주세요

THANK YOU

감사합니다.

AI SCHOOL 5기
정승욱, 김민엽, 유진아, 이시현