# 추천 알고리즘 AI경진대회

## DACON

춘배사랑개

May 2023

# 목차

# 목차

# OS 및 라이브러리 버전

## Virtual Environment Information (OS)

Ubuntu: Ubuntu 22.04.2 LTS
Linux: Linux 5.15.90.1-microsoft-standard-WSL2, x86_64
CPU: AMD Ryzen 5 5600X 6-Core Processor
RAM: 48GB
GPU: NVIDIA GeForce RTX 3070 8GB

# OS 및 라이브러리 버전

## Main Versions of Python Modules

Python: 3.10.10
catboost: 1.2
cuda-python: 11.8.1
cudf: 23.4.1
cuml: 23.4.1
matplotlib: 3.7.1
pandas: 1.5.3
numpy: 1.23.5
optuna: 3.1.1
JupyterLab: 3.6.3

# 목차

# Evaluation

## Competition metric

이번 대회는 도서 평점 예측 값과 실제 도서 평점 사이의 평균 제곱근 오차로(Root Mean Squared Error) 평가됨.

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{\sum(\hat{y} - y)^2}{n}}$$

# 목차

# EDA

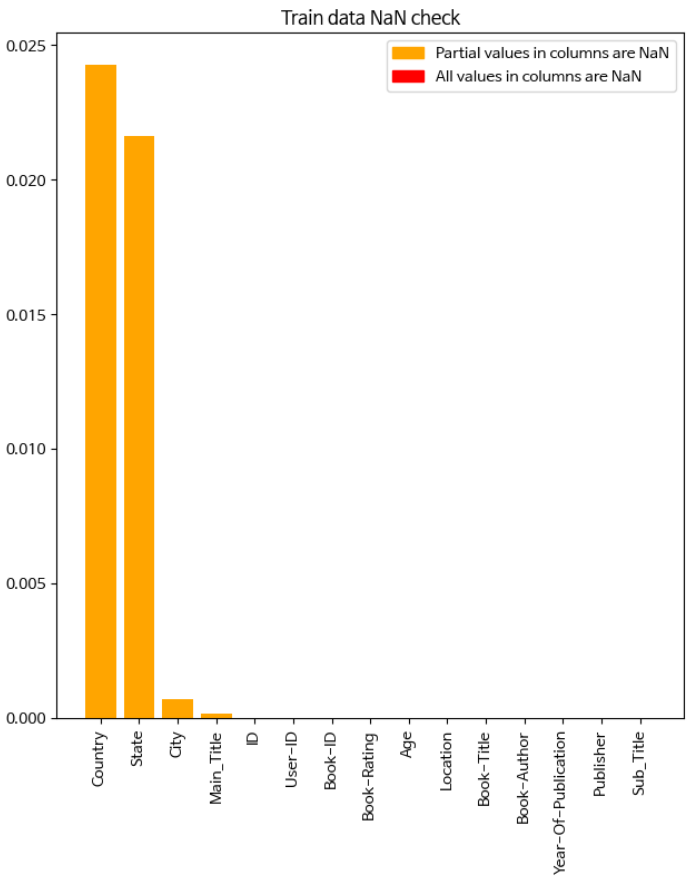
## Detaset Info

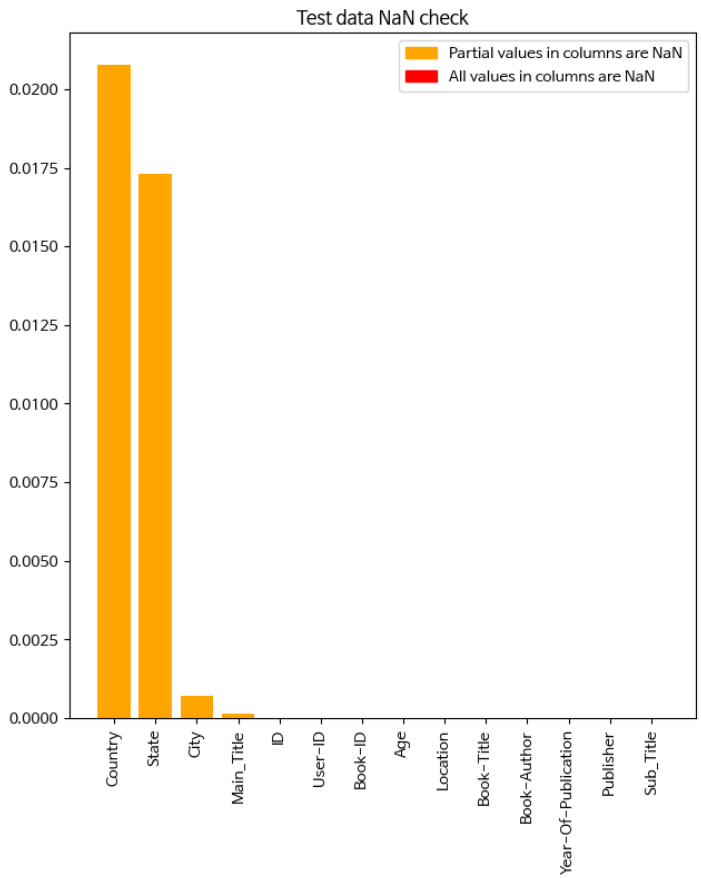
ID : 샘플 고유 ID
User-ID : 유저 고유 ID
Book-ID : 도서 고유 ID
Age : 나이
Location : 지역
Book-Title : 도서 명
Book-Author : 도서 저자
Year-Of-Publication : 도서 출판 년도 (-1일 경우 결측 혹은 알 수 없음)
Publisher : 출판사
Book-Rating : 유저가 도서에 부여한 평점 (0점 $\sim$ 10점)

# EDA


- Book-Title을 토큰화한 후 WordCloud하였을 때 그림과 같은 결과를 보였습니다. 책 제목에 많이 사용되는 키워드는 Novel, Love, Life 등이 있었으며 Mysteries와 같은 장르나 paperback(soft cover)와 같은 제본 양식 등이 자주 등장했습니다.

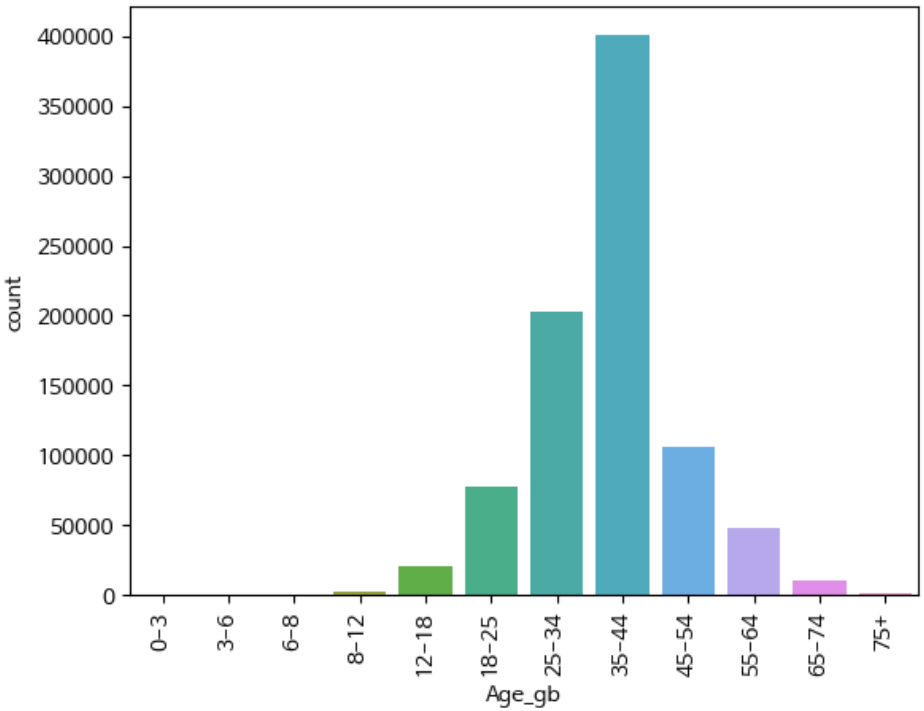
# EDA

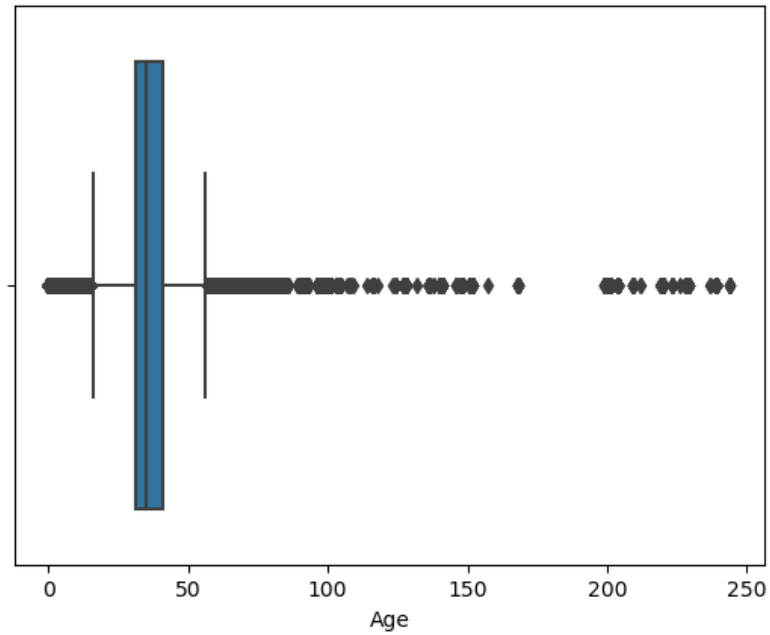



- Book-Title과 Location을 re.sub를 통해 특수문자를 제거하고, Title은 Main, Sub로 분할(' '로 구분)하고, Location은 ','를 기준으로 도시, 주, 나라로 나누었을 때 'na'나 ''의 값을 갖는 Null의 빈도는 그림과 같았음.

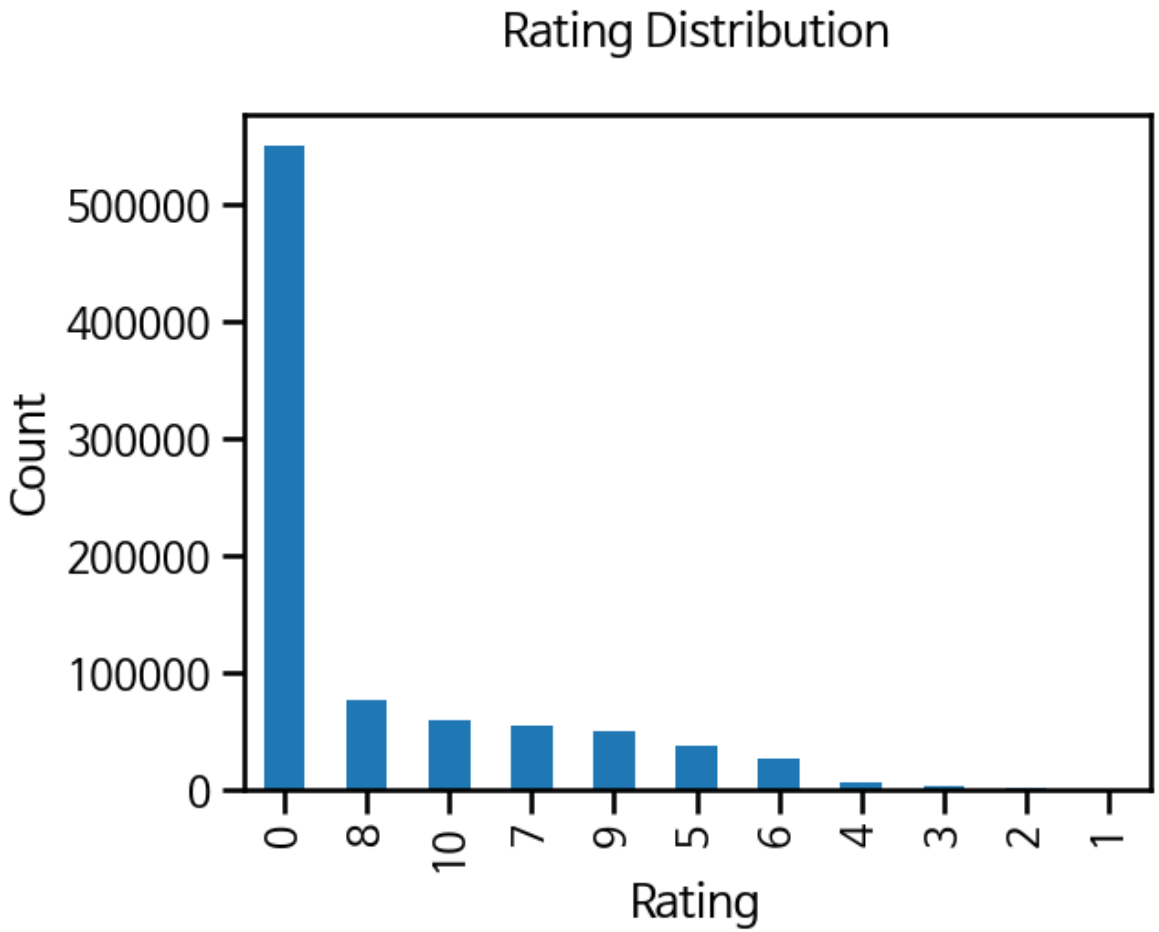
# EDA




- 나이대 범주화는 미국 노동통계국에서 발표한 'Spending and employment related to books and other reading materials'를 참고하였습니다. 35-44 사이의 나이대에 리뷰가 제일 많았습니다. 또한 boxplot으로 표현하였을 때, 0세 근방과 100세 이상의 이상치를 확인할 수 있었습니다.

# EDA




- Train 데이터에서 점수 분포는 그림과 같습니다. 약 87만 개의 데이터 중에 55만개 데이터가 0의 평가입니다. 즉, 유저가 해당 도서에 관심이 없고 관련이 없는 경우가 가장 많았습니다.

```
df['Book-Title'] = [re.sub(r'[^0-9a-zA-Z:,]',  ' ',str(i)) for i in df['Book-Title']]
df['Main_Title'] = [i.split('  ')[0] for i in df['Book-Title']]
df['Sub_Title'] = [''.join(i.split('  ')[1:]) for i in df['Book-Title']]
df['Sub_Title'] = np.where(df['Sub_Title'] == '', 'No_SUB', df['Sub_Title'])

df['Location'] = [re.sub(r'[^0-9a-zA-Z:,]',  ' ',str(i)) for i in df['Location']]

df['City'] = [(i.split(',')[0]).lstrip().lower() for i in df['Location']]
df['State'] = [(i.split(',')[1]).lstrip().lower() for i in df['Location']]
df['Country'] = [(i.split(',')[2]).lstrip().lower() for i in df['Location']]
```

- Book-title은 특수문자 제거 후 Main과 Sub Title로 나누었습니다. 이유는 특정 출판사명이나 책 시리즈 등의 부제가 붙기 때문에 나누어 피쳐로 두는 것이 좋다고 생각했습니다. Sub Title이 없는 경우엔 'NO_SUB'로 바꾸었습니다. Title을 Split한 점이 모델 예측에 좋았습니다.

# EDA

## Preprocessing

```
# 최빈값을 사용하기 위해 새로운 데이터 프레임 생성(pd.Series.mode를 이용하면 같은 count수의 값을 list로 묶어서 정확하지 않음)
new_state = train_lb.groupby(['City'])['State'].value_counts().to_frame().rename(columns = {'State' : 'count'}).reset_index()
new_state = new_state[(~new_state['City'].isna())&(~new_state['State'].isna())&(new_state['count']!=1)]
new_state = new_state.sort_values(by=['City', 'count'], ascending=[True, False]).drop_duplicates(subset='City', keep='first')
new_state = new_state.rename(columns = {'State' : 'N_State'})
new_state = new_state.drop(columns = ['count'])

new_country = train_lb.groupby(['State'])['Country'].value_counts().to_frame().rename(columns = {'Country' : 'count'}).reset_index()
new_country = new_country[(~new_country['State'].isna())&(~new_country['Country'].isna())&(new_country['count']!=1)]
new_country = new_country.sort_values(by=['State', 'count'], ascending=[True, False]).drop_duplicates(subset='State', keep='first')
new_country = new_country.rename(columns = {'Country' : 'N_Country'})
new_country = new_country.drop(columns = ['count'])

df = pd.merge(df, new_country, on = 'State', how = 'left')
df = pd.merge(df, new_state, on = 'City', how = 'left')

df['Country'] = np.where((df['Country'] == '')|(df['Country'].astype(str) == 'nan'), df['N_Country'], df['Country'])
df['State'] = np.where((df['State'] == '')|(df['State'].astype(str) == 'nan'), df['N_State'], df['State'])

# 채워지지 않은 값은 Unknown 처리
df[['Country', 'State', 'City']] = df[['Country', 'State', 'City']].fillna(value= 'Unknown')
df = df.drop(columns = ['N_Country', 'N_State'])
```

- Location을 도시, 주, 나라로 나눈 뒤 Null은 최빈값으로 대체하였습니다. 처음에는 pandas.Series.mode를 사용하였지만, 해당 agg를 groupby에 적용하면 공통 최빈값은 하나의 list로 묶여져서 value_counts 후 sort_values에서 중복 제거 후 첫번째 행을 가져오는 방법을 선택했습니다.

```
labels = ['0-3','3-6','6-8','8-12','12-18','18-25','25-34','35-44','45-54','55-64','65-74','75+']
bins = [0, 3, 6, 8, 12, 18, 25, 34, 44, 54, 64, 74, 250]

df.loc[(df['Age'] > 90) | (df['Age'] < 3), 'Age'] = np.nan

# 평균값으로 대체
df['Age'] = df['Age'].fillna(df['Age'].mean())
df['Age'] = df['Age'].astype(np.int32)


df['Age_gb'] = pd.cut(df.Age, bins, labels = labels,include_lowest = True)
```

- Age 변수를 나눌 기준을 세우고, 이상치에 해당되는 90세 이상, 3살 미만의 데이터는 Null로 처리한 뒤 평균값으로 대처하였습니다. 기존에 3살 미만은 3살로, 100살 이상은 100살로 고정하는 것보단 좋은 성능을 보였습니다.

# 목차

# Model Selection

모델을 선택하기 앞서, 파라미터를 지정하지 않고, random_state만 고정시킨 뒤, Random forest, Linear Regression, LGB, XGB, Catboost를 돌려서 기본적인 Valid 점수를 확인하고 Catboost를 메인으로 하였습니다. User-ID가 핵심이라 생각해서, 과적합을 방지하기 위해 K-fold 대신 StratifiedKFold를 통해 교차검증하였습니다.

<History of LB scores>

| Model | Describe | Public Score |
|---|---|---|
| Catboost | No_cat_features 5-folds | 3.54836 |
| Catboost | Add_cat_features 5-folds | 3.27553 |
| Catboost | Add_cat_features + bertopic 5-folds+optuna | 3.27097 |
| Catboost | Add_cat_features 10-folds | 3.26106 |
| Catboost | stacking(XGB, RF) | 3.40884 |
| Catboost | Add_cat_features 20-folds + preprocessing | 3.26013 |
| Catboost | Add_cat_features 20-folds + preprocessing (mean age) | 3.26007 |
| Catboost | Add_cat_features 20-folds + preprocessing + title_split | 3.25811 |

## Optuna hyperparameter tuning

```
cbrm_param = {
    'l2_leaf_reg':trial.suggest_float("l2_leaf_reg",1e-5, 1e-1),
    'max_bin': trial.suggest_int("max_bin", 200, 400),
    'learning_rate' : trial.suggest_float('learning_rate', 0.001, 0.01),
    'max_depth': trial.suggest_categorical('max_depth', [5,7,9,11,13,15]),
    'min_data_in_leaf': trial.suggest_int('min_data_in_leaf', 1, 300),
    'iterations':trial.suggest_int("iterations", 1000, 10000),
    #'random_strength' :trial.suggest_int('random_strength', 0, 10),
    'subsample': trial.suggest_float('subsample', 0.4, 1.0),
    #'min_child_samples': trial.suggest_int("min_child_samples", 5, 100),
    'eval_metric' : 'RMSE',
    'task_type': 'GPU',
    'random_state': 113,
    'use_best_model': True,
    'bootstrap_type' : 'Poisson',
    'early_stopping_rounds' : 100
}
```

```
from catboost import CatBoostRegressor, Pool
model = CatBoostRegressor(random_seed = 113,
                          l2_leaf_reg = 0.003426034644149707,
                          max_bin = 358,
                          subsample = 0.9974697184313627,
                          learning_rate = 0.009464402227606937,
                          max_depth = 11,
                          min_data_in_leaf = 139,
                          eval_metric = 'RMSE',
                          iterations = 8694,
                          task_type='GPU',
                          bootstrap_type = 'Poisson',
                          early_stopping_rounds = 100,
                          verbose=500
                          )
```

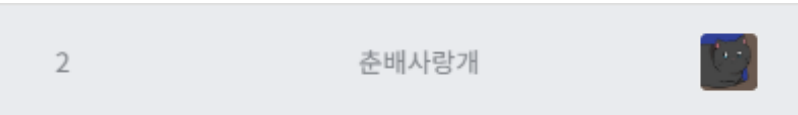
- 대회 metric이 RMSE이고, 점수 복원을 위한 Random state, GPU, 등을 기본적으로 고정하고 중요해보이는 파라미터들만 튜닝하였습니다.
- learning_rate: 가중치 업데이트 기준
- min_data_in_leaf: leaf의 최소 훈련 샘플 수
- subsample: 각 트리를 훈련하는 데 사용될 features의 비율 (randomly selected)

# 목차

# Conclusion

## Overall Summary

[Final LB Score]

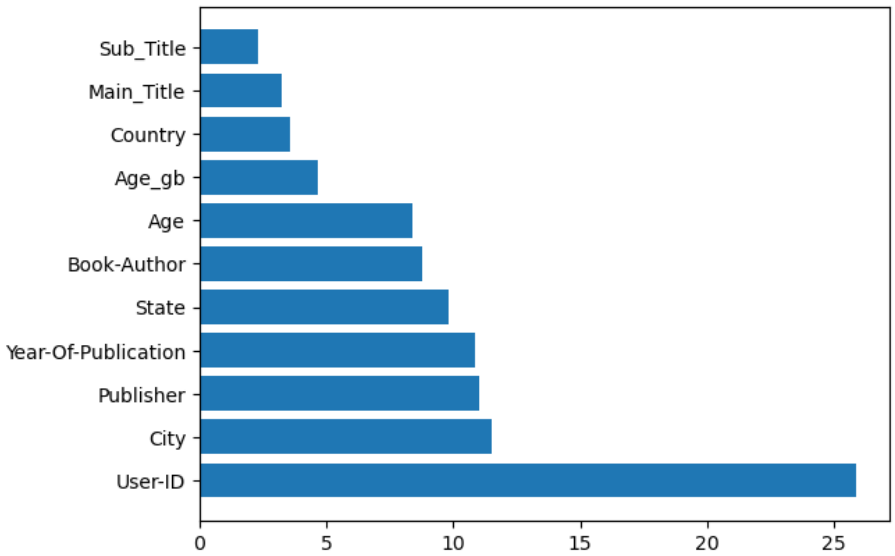


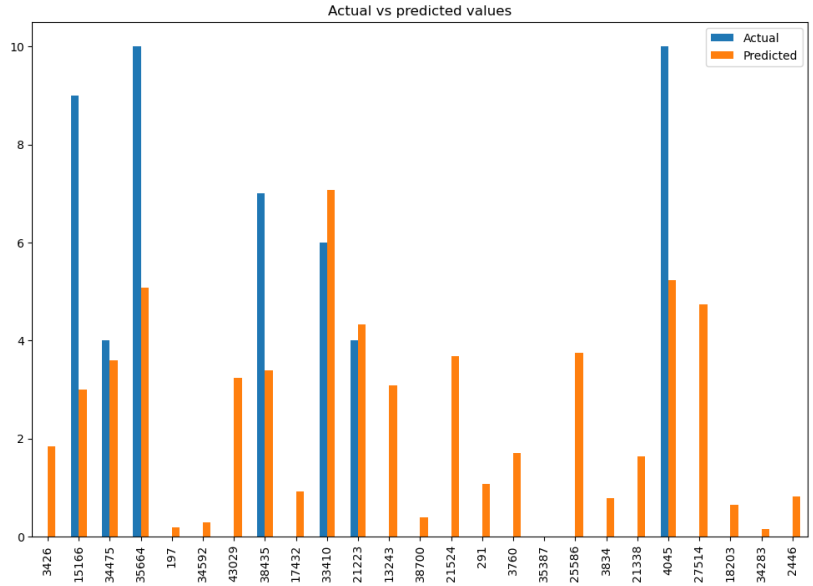

| LB | Score |
|---|---|
| Public | 3.25811 |
| Private | 3.27139 |

- Catboost Regressor를 사용하여 전처리 + 나이 범주화 + 하이퍼파라미터 튜닝 + Ordinal Encoding을 통해 최종 Public 점수는 3.25811, Private 점수는 3.27139를 기록하였습니다.
- 다양한 변수(Title 기반 토픽 모델링, 언어구분, 출판연도 구분, counting 등)를 적용하였지만 기존 변수를 전처리하는 것이 제일 좋은 성능을 보였습니다.

# Conclusion

## Overall Summary




- Catboost의 기능 중 feature_importances_를 이용하여 트리에서 피쳐가 클래스를 나누는데 얼마나 영향을 미쳤는지 확인할 수 있었습니다. Title은 영향이 상대적으로 적었으며 User-ID가 가장 높고 City, Publisher, Year-Of-Publication이 다음으로 크게 작용했습니다.
- Valid set을 얼마나 잘 예측하는지 확인하기 위해서 fold과정에 시각화하는 함수를 넣어 두었습니다. 낮은 점수대는 근사하게 예측하는 반면, 높은 점수대는 잘 예측 못하는 것을 확인할 수 있었습니다.

# Conclusion

## Limitation of study

- Catboost + 20-folds를 돌렸을 때, 약 4시간 정도 시간이 소모되었는데 fold를 하지않고, 더 많은 것(feature_importances_ 관련 feature engineering)을 테스트해봐야 했습니다..
- Catboost가 아닌 모델들을 Optuna하여 앙상블 모델을 구현했어야 하는데, 시간 관계상 Catboost에만 집중하게 됐습니다.
- 트리 모델이 아닌 FFM(Field-aware Factorization Machines)도 사용해봤지만 Valid가 3.32까지 나와서 적용하지 못했습니다. 다른 딥러닝 모델도 사용해봐야 했습니다.
- 점수 분포가 0점에 몰려있어 고 득점대의 예측이 힘들었음. 해당 문제를 해결했다면 RMSE을 조금이라도 줄일 수 있지 않았을까 생각이 들었습니다.

# Thank You!