NETFLIX D/A

DF_4

Play now

중간 발표 요약

데이터 셋: 캐글 Netflix-movies-and-tv-shows

> 1차 목표: 관객 수(imdb_votes)에 영향을 미치는 변수 회귀분석

> 2차 목표: 머신러닝을 통한 '관객수' 예측 모델링

중간 발표 이후

4가지 가설 검증

- 가설1: '제목의 길이'와 '관객 수'의 상관관계
- 가설2: '생산 국가'와 '관객 수'의 상관관계
- 가설3: '시즌 수'와 '관객 수'의 상관관계
- 가설4: '관람 등급'과 '관객 수'의 상관관계

word_count 변수 추가

country_weight 변수 추가

season 변수 예측에 사용

genres_weight 변수 추가

중간 발표 이후

최종 선택한 변수(10개)

release_year
runtime
production_countries
seasons
imdb_votes
tmdb_popularity
tmdb_score
word_count
genres_weight
country_weight

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		title	type	release_ye	runtime	productior	seasons	imdb_scor	imdb_vote	tmdb_pop	tmdb_scor	word_cour	genres_we	country_weight	
2	0	Taxi Drive	MOVIE	1976	114	92	1	8.2	808582	40.965	8.179	11	0.0365	0.016954	
3	1	Deliveranc	MOVIE	1972	109	92	1	7.7	107673	10.01	7.3	11	0.0365	0.016954	
4	2	Monty Pyt	MOVIE	1975	91	91	1	8.2	534486	15.461	7.811	31	0.055242	0.027723	
5	3	The Dirty	MOVIE	1967	150	91	1	7.7	72662	20.398	7.6	15	0.110314	0.027723	
6	4	Monty Pyt	SHOW	1969	30	91	4	8.8	73424	17.617	8.306	28	0.022292	0.027723	
7	5	Life of Bri	MOVIE	1979	94	91	1	8	395024	17.77	7.8	13	0.022292	0.027723	
8	6	Dirty Harr	MOVIE	1971	102	92	1	7.7	155051	12.817	7.5	11	0.085455	0.016954	
9	7	Bonnie an	MOVIE	1967	110	92	1	7.7	112048	15.687	7.5	16	0.053294	0.016954	
10	8	The Blue l	MOVIE	1980	104	92	1	5.8	69844	50.324	6.156	15	0.039498	0.016954	
11	9	The Guns	MOVIE	1961	158	91	1	7.5	50748	13.844	7.3	20	0.061918	0.027723	
12	10	The Profe	MOVIE	1966	117	92	1	7.3	16446	13.123	7.1	17	0.105119	0.016954	
13	11	Richard Pr	MOVIE	1979	78	92	1	8.1	5141	4.718	7.5	30	0.022292	0.016954	
14	12	White Chr	MOVIE	1954	115	92	1	7.5	42488	8.915	7.2	15	0.039498	0.016954	
15	13	Cairo Stati	MOVIE	1958	77	38	1	7.5	4471	5.546	7.3	13	0.0365	0.003786	
16	14	Hitler: A C	MOVIE	1977	150	89	1	7.5	2460	4.305	7.3	16	0.043407	0.009867	
17	15	FTA	MOVIE	1972	97	92	1	6.2	418	1.268	6.1	3	0.110314	0.016954	

TABLE OF CONTENTS

01 선형 모델

선형 회귀

In [12]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
```

[ ]
```
y = df["imdb_votes"]
x = df[['release_year', 'runtime', 'production_countries',
        'seasons', 'imdb_score','tmdb_popularity', 'tmdb_score',
        'word_count', 'genres_weight', 'country_weight']]
reg = sm.OLS(y, x).fit()
print(reg.summary())
```

선형 회귀

OLS Regression Results

Dep. Variable:	imdb_votes	R-squared (uncentered):	0.202
Model:	OLS	Adj. R-squared (uncentered):	0.201
Method:	Least Squares	F-statistic:	129.7
Date:	Tue, 11 Apr 2023	Prob (F-statistic):	2.80e-242
Time:	12:15:27	Log-Likelihood:	-65777.
No. Observations:	5129	AIC:	1.316e+05
Df Residuals:	5119	BIC:	1.316e+05
Df Model:	10		
Covariance Type:	nonrobust		

로그 변환 02

반응 변수 log 치환

```
import math

df["log_votes"] = df["imdb_votes"].map(math.log10)
df[["imdb_votes", "log_votes"]]
```

	imdb_votes	log_votes
0	808582.0	5.907724
1	107673.0	5.032107
2	534486.0	5.727936
3	72662.0	4.861307
4	73424.0	4.865838
...	...	...
5124	163.0	2.212188
5125	38.0	1.579784
5126	327.0	2.514548
5127	68.0	1.832509
5128	18.0	1.255273

5129 rows × 2 columns

반응 변수 imdb_votes에
log 를 붙인 log_votes를 새롭게 만들어줌

$$log\_votes = log_{10}(imdb\_votes)$$

선형 회귀

```
from sklearn.linear_model import LinearRegression

# log(반응변수) 선형회귀
model_10 = LinearRegression()
model_10.fit(x_train, y_train)
```

▾ LinearRegression

LinearRegression()

분석

```
# 절편
model_10.intercept_
```

```
11.297895987852847
```

```
# 회귀계수
model_10.coef_
```

```
array([-8.25102131e-03, -5.27010650e-03,  6.19995555e-03,  4.35412427e+01,
        7.40735132e+00,  4.92352056e-02,  1.99857910e-01,  1.64948297e-03,
        1.89035122e-02])
```

```
# 결정계수
pred1 = model_10.predict(x_valid)
r2_score(y_valid, pred1)
```

```
0.403421917955605
```

반응변수를 log 변환 한 뒤
선형회귀를 돌려본 결과,

결정계수가 40%로
변환 전보다 설명력이 크지만
여전히 유의하지 않은 수준으로 보임

분석

```
# 밑이 2인 로그 변환
log_y = df["imdb_votes"].map(math.log2)
x_train, x_valid, y_train, y_valid = train_test_split(x, log_y, test_size=0.2)
model_2 = LinearRegression()
model_2.fit(x_train, y_train)
pred2 = model_2.predict(x_valid)
r2_score(y_valid, pred2)
```

```
0.3912336071361828
```

```
# 밑이 e(자연상수)인 로그변환
log_y = df["imdb_votes"].map(math.log)
x_train, x_valid, y_train, y_valid = train_test_split(x, log_y, test_size=0.2)
model_e = LinearRegression()
model_e.fit(x_train, y_train)
pred3 = model_e.predict(x_valid)
r2_score(y_valid, pred3)
```

```
0.3795882599240975
```

밑을 다르게 해도
설명력은 비슷하게 나옴

03 비선형 모델

“ 선형이 아닐 수도 있겠다. ”

종속변수인 “imdb_votes”를 0과 1로 바꾸어
비선형모델인 ‘로지스틱 회귀’를 사용해보기로 결정

로지스틱 분석

0과 1로 바꾸기

단순히 선형 회귀 모델로는 종속변수를 예측할 수 없다.

Array로 만들기

Numpy 라이브러리

Column 추가하기

df ['Success']

다시 분석하기

Y는
이제 0과 1 뿐

로지스틱 분석

```
[ ]  # 해결방법 2. 로지스틱 함수 (y를 흥행 / 실패 두 개로 만든다.)
     empty = []
     for y in df["imdb_votes"]:
         if y < df["imdb_votes"].median():
             empty.append(np.array([0]))
         else:
             empty.append(np.array([1]))
     # 여기에 imdb_votes가 중간값보다 낮으면 0(실패), 높으면 1(흥행)으로 나눈 값을 넣는다.
```

```
[ ]  df["Success"] = empty
     df["Success"]
     # 흥행 여부(0과 1)을 담은 새로운 칼럼 "Success"을 df에 추가
```

```
0       [1]
1       [1]
2       [1]
3       [1]
4       [1]
       ...
5126    [0]
```

로지스틱 분석

```
[ ]  # 로지스틱을 돌리기 위한 코드
     from sklearn.model_selection import train_test_split

     train_features, test_features, train_labels, test_labels = train_test_split(x,  y)

     from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()

     train_features = scaler.fit_transform(train_features)
     test_features = scaler.transform(test_features)

     from sklearn.linear_model import LogisticRegression

     model = LogisticRegression()
     model.fit(train_features, train_labels)
```

```
▾ LogisticRegression
LogisticRegression()
```

예측 모델의 정확도

Train-set: 76.3%

```
print(model.score(train_features, train_labels))
# 모델의 정확도: 대략 76%
```

```
0.7628705148205929
```

Test-set: 76.0%

```
print(model.score(test_features, test_labels))
# 테스트 정확도: 76% 정도
```

```
0.7607170693686672
```

Movie #1

Inception (2010)

- IMDB_VOTES: 2,294,231

- 러닝타임: 148분
- 국가: 미국
- 시즌: 1개
- 평점: 8.8 / 8.4
- 제목길이: 9자
- 장르: 액션

Movie #2

Squid Game (2021)

- IMDB_VOTES: 426,967

- 러닝타임: 55분
- 국가: 한국
- 시즌: 2개
- 평점: 8.0 / 7.8
- 제목길이: 10자
- 장르: 액션

Movie #3

Lokillo (2021)

- IMDB_VOTES: 68

- 러닝타임: 90분
- 국가: 호주
- 시즌: 1개
- 평점: 3.8 / 6.3
- 제목길이: 7자
- 장르: 코미디

로지스틱 분석

```
[ ]  # 예측력 테스트하기
     #'release_year', 'runtime', 'production_countries','seasons', 'imdb_score','tmdb_popularity', 'tmdb_score','word_count', 'genres_weight', 'country_weight' 집어넣기

     Squid_game = np.array([2021, 55, 63, 2.0, 8.0, 361.925, 7.8210000000000015, 10, 0.0619181141740407, 0.007858757670362794])
     # 오징어게임은 흥행(1)한다고 나와야 한다.
     Inception = np.array([2010, 148, 92, 1.0, 8.8, 108.284, 8.4, 9, 0.0619181141740407, 0.016953682702195032])
     # 인셉션은 흥행(1)한다고 나와야 한다.
     Lokillo = np.array([2021, 90, 22, 1.0, 3.8, 26.005, 6.3, 7, 0.022292220482381, 0.0008129749314168408])
     # 로킬로는 실패(0)한다고 나와야 한다.
     sample_movies = np.array([Squid_game, Inception, Lokillo])
```

Interpretation

	Inception	Squid Game	Lokillo
Success (1)	99.1%	97.6%	2.5%
Fail (0)	0.01%	2.3%	97.5%

로지스틱 분석

```
[ ]  print(model.predict(sample_movies))

     print(model.predict_proba(sample_movies))
     # [1 1 0]은 첫번째, 두번째 영화는 흥행, 세번째 영화는 실패할 것임을 예측해준다.
     # 각 영화의 흥행할 확률도 말해준다. 오징어 게임 97%, 인셉션 99%, 로킬로 2.4%
```

```
[1 1 0]
[[0.02328919 0.97671081]
 [0.00862129 0.99137871]
 [0.97537036 0.02462964]]
```

Insight

비선형성

단순히 선형 회귀 모델로는
종속변수를 예측할 수 없다.

로지스틱

로지스틱은 imdb_votes
예측값을 알 수 없다.

다른 모델

다른 예측 모델이
필요하다.

예측 모델 04

Modeling

Random Forest Regressor

- 분류/회귀 모두 사용 가능한 앙상블 모델
- 다수의 의사 결정 트리를 만드는 방식

XGBoost Regressor

- 약한 분류기를 세트로 묶어 정확도를 예측
- 이전 모델의 에러를 분석하여 회귀 정확도 향상

선형 모델 로그 변화 비선형 모델 예측 모델

	release_year	runtime	production_countries	seasons	imdb_score	tmdb_popularity	tmdb_score	word_count	genres_weight	country_weight
0	1976	114	92	1.0	8.2	40.965	8.179	11	0.036500	0.016954
1	1972	109	92	1.0	7.7	10.010	7.300	11	0.036500	0.016954
2	1975	91	91	1.0	8.2	15.461	7.811	31	0.055242	0.027723
3	1967	150	91	1.0	7.7	20.398	7.600	15	0.110314	0.027723
4	19									0.027723
...										...
5126	20									0.002049
5127	20									0.001147
5128	20									0.003786
5129	20									0.000813
5130	20									0.001147

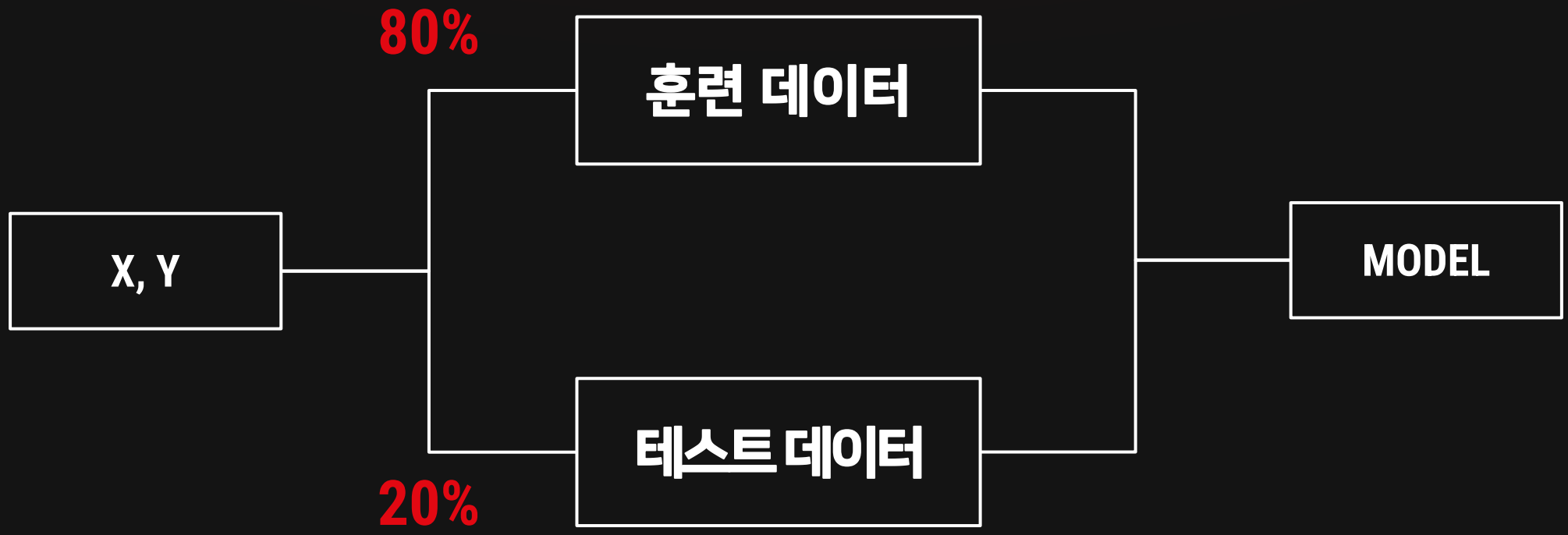
```
[ ]  from sklearn.preprocessing import StandardScaler # 표준 정규화(평균 0, 표준편차 1)
     scaler = StandardScaler() # 스케일러 불러오기
     x = scaler.fit_transform(x) # x 데이터 스케일
     x
```

```
array([[-5.72576107,  0.91022157,  0.81354681, ..., -0.63456145,
        -0.12917051,  0.56430079],
       [-6.29425277,  0.78071942,  0.81354681, ..., -0.63456145,
        -0.12917051,  0.56430079],
       [-5.86788399,  0.3145117 ,  0.76850345, ...,  1.28130015,
         0.49313879,  1.99703759],
       ...,
       [ 0.66977056,  0.36631256, -1.61879455, ..., -0.92194069,
        -0.02961715, -1.18748048],
       [ 0.66977056,  0.28861127, -2.33948828, ..., -1.01773377,
        -0.60094062, -1.58307585],
       [ 0.66977056, -1.86112433, -2.38453164, ...,  1.56867939,
        -0.9590506 , -1.53862693]])
```

예측

여
향상

Modeling

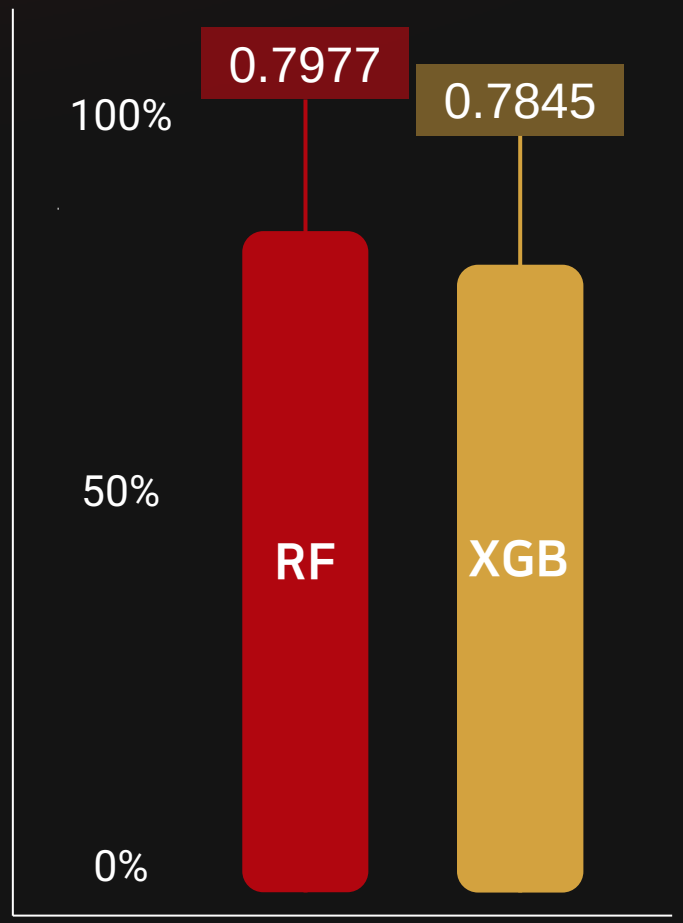


Modeling

```
[ ]  from sklearn.ensemble import RandomForestRegressor # RandomForest(RF)
     model2 = RandomForestRegressor()
     model2.fit(x_train, y_train)
     pred2 = model2.predict(x_test)
```

```
[ ]  import xgboost as xgb # XGBoost(XGB)
     model3 = xgb.XGBRegressor()
     model3.fit(x_train, y_train)
     pred3 = model3.predict(x_test)
```

```
[ ]  from sklearn.metrics import mean_squared_error, r2_score
     import numpy as np

     print('model2', np.sqrt(mean_squared_error(y_test, pred2))) # RMSE
     print('model2', r2_score(y_test, pred2)) # R^2
     print('model3', np.sqrt(mean_squared_error(y_test, pred3)))
     print('model3', r2_score(y_test, pred3))
```



Hyper parameter tunning

max_features

max_depth

각 모델별 하이퍼파라미터 후보

n_estimators

learning_rate

Grid Search CV

best_score_

하이퍼 파라미터 조합으로
가장 좋은 모델의 점수
그리고 그 때의 하이퍼 파라미터

best_estimator_

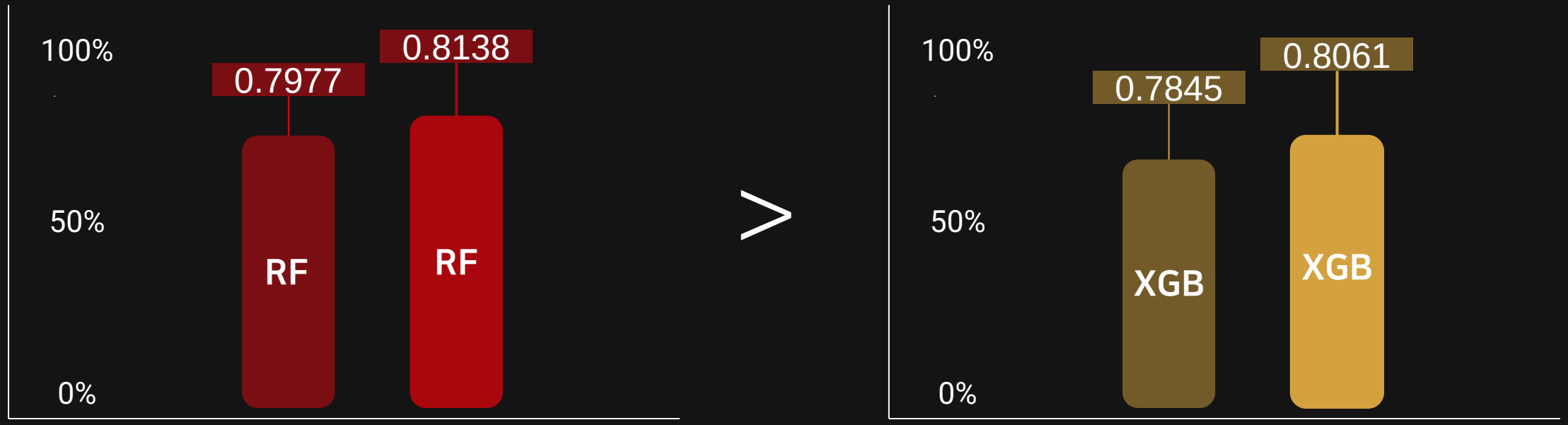
```
[ ]  from sklearn.model_selection import GridSearchCV
     import xgboost as xgb

     rf = RandomForestRegressor()
     xgb = xgb.XGBRegressor()

     parameters2 =

     parameters3 =

     model5 = Grid                                                    squared_error')
     model6 = Grid                                                    ean_squared_error')

     model5.fit(x_
     model6.fit(x_train, y_train)

     print('rf: ', model5.best_estimator_)
     print('rf: ', model5.best_score_)
     print('xgb: ', model6.best_estimator_)
     print('xgb: ', model6.best_score_)
```

```
rf:  RandomForestRegressor(bootstrap=False, max_features=3, n_estimators=10)
rf:  -2871705264.6827607
xgb:  XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.7, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.07, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=6, max_leaves=None,
             min_child_weight=4, missing=nan, monotone_constraints=None,
             n_estimators=500, n_jobs=None, nthread=4, num_parallel_tree=None,
             objective='reg:linear', ...)
xgb:  -51394.04081624999
```

Hyper parameter tunning

```
[ ]  pred5 = model5.predict(x_test) # RF
     print('model5', np.sqrt(mean_squared_error(y_test, pred5)))
     print('model5', r2_score(y_test, pred5))
```

```
[ ]  pred6 = model6.predict(x_test) # XGB
     print('model6', np.sqrt(mean_squared_error(y_test, pred6)))
     print('model6', r2_score(y_test, pred6))
```



결론

기존 목표 : 70%의 설명력을 가지는 예측 모델 생성

↓

회귀모델(LR, RF, XGB)과 분류모델(Logistic)을 통해 예측 해본 결과,
RandomForest 회귀 모델이 약 81%의 설명력을 보임

↓

Disney+와 같은 다른 OTT 플랫폼의 데이터를 가지고
모델에 적용시켜볼 수 있을 것으로 예상