「 인기있는 음악은 어떤 특성을 가져야할까」

AIB 15th

Section 2 Project

박건후

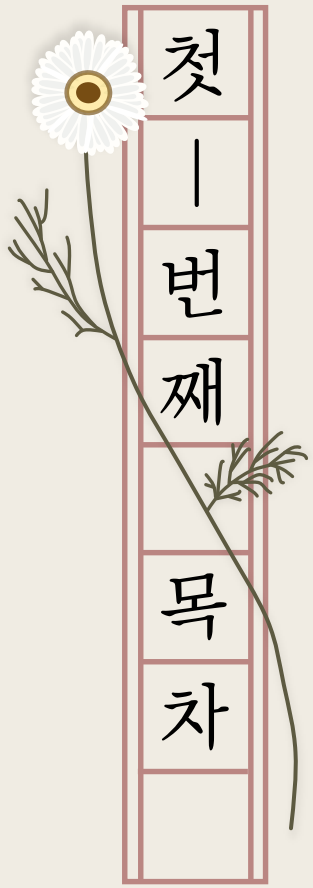
「 목 차 」

01. 첫 번째 목차 :
데이터 설명 및 문제 정의

02. 두 번째 목차 :
데이터 전처리, EDA, 시각화

03. 세 번째 목차 :
모델링 및 모델 해석

04. 네 번째 목차 :
결론



NO. 1-1

상황 가정

한국 음악 시장이 글로벌해짐에 따라 '의뢰인'은 어떤 음악이 세계적으로 성공할 수 있는지 분석을 의뢰하였습니다.

과연, 대중적으로 성공적인 음악(인기가 높은 음악)을 출시하기 위해서는 어떤 특성을 가져야할까요?

데이터 셋 특성 설명 (1)

- genre : 음악 장르
- artist_name : 아티스트 이름
- track_name : 트랙 이름
- track_id : 트랙에 대한 데이터 출처 사이트(Spotify)의 ID
- popularity : 트랙이 인기가 있는지에 대한 0 ~ 100까지의 수치이며, 값이 클수록 인기가 많은 트랙이다.
- acousticness : 트랙이 어쿠스틱한지에 대한 0.0 ~ 1.0의 측정 값이며, 값이 클수록 어쿠스틱하다.
- danceability : 템포, 리듬 안정성, 박자의 힘, 전체적인 규칙성을 포함한 음악적 요소들의 조합에 기초하여 트랙이 얼마나 춤에 적합한지를 설명. 값이 클수록 춤을 추기에 적합하다.
- duration_ms : million seconds 단위의 재생시간
- energy : 트랙의 강도와 액티비티함에 대한 0.0 ~ 1.0의 측정 값이며, 값이 클수록 빠르고 시끄럽다.
- instrumentalness – 트랙에 보컬이 없는지에 대한 예측 값이며, 값이 클수록 트랙에 보컬이 없을 가능성이 커진다.

데이터 셋 특성 설명 (2)

- instrumentalness – 트랙에 보컬이 없는지에 대한 예측 값이며, 값이 클수록 트랙에 보컬이 없을 가능성이 커진다.
- key : 트랙의 키; 음의 높낮이.
- liveness : 청중의 존재를 탐지한다. 값이 클수록 라이브 확률이 높아진다. 특히, 0.8보다 클 때 라이브일 가능성이 더 높다.
- loudness : 트랙의 전체 음량(dB). 음량 값은 전체 트랙에서 평균화되어 트랙의 상대적 음량을 비교하는 데 유용하다.
  일반적으로 값은 -60에서 0db 사이다.
- mode : 트랙의 형식(Major 또는 Minor)을 나타낸다.
- speechiness : 트랙에서 음성 단어의 존재를 감지한다.
  녹음(예: 토크쇼, 오디오북, 시)과 같은 음성 전용일수록 속성 값은 1.0에 가깝다.
  0.66 이상의 값은 전적으로 음성 단어로 구성된 트랙일 가능성이 높다.
  0.33 미만의 값은 음악 및 기타 비음성 트랙을 나타낼 가능성이 높습니다.
- tempo : 트랙의 전체 추정 속도(분당 비트 수).
  음악 용어에서 템포(tempo)는 주어진 곡의 속도 또는 속도를 말하며 평균 박자 지속 시간에서 직접 파생된다.
- time_signature : 트랙의 박자; 각 막대(또는 측정값)에 얼마나 많은 박자가 있는지 지정하는 표기법.
- valence : 트랙에 의해 전달되는 음악적 긍정성을 설명하는 0.0 ~ 1.0의 측정값.
  값이 높은 트랙은 더 긍정적으로 들리고(예: 행복, 쾌활, 행복),
  낮은 트랙은 더 부정적으로 들린다(예: 슬픔, 우울, 화).

NO. 1-3

문제 정의 및 가설 설정

문제 정의 1.	문제 정의 2.	문제 정의 3.	문제 정의 4.	문제 정의 5.
가장 인기 있는 장르는 어떤 것일까요?	재생시간은 어느 정도가 적당할까요? 재생시간이 인기도에 영향을 줄까요?	댄스 음악이 인기가 많을까요, 발라드 음악이 인기가 많을까요?	인기도에 가장 영향을 많이 주는 요소는 무엇일까요?	Genre에 따라 Danceability가 예측에 영향을 주는 정도가 다를까요?
가설 1.	가설 2.	가설 3.	가설 4.	가설 5.
가장 인기 있는 장르는 Pop일 것이다.	재생시간은 인기도에 영향을 주지 않을 것이다.	댄스 음악이 인기가 더 많을 것이다.	인기도에 가장 영향을 많이 주는 요소는 Danceability일 것이다.	Genre에 따라 Danceability가 예측에 영향을 주는 정도가 다를 것이다.

02. 데이터 전처리, EDA, 시각화

NO. 2-1

데이터 셋 EDA

01

데이터 셋 확인

	genre	artist_name	track_name	track_id	popularity	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode
0	Movie	Henri Salvador	C'est beau de faire un Show	0BRjO6ga9RKCKjfDqeFgWV	0	0.611	0.389	99373	0.910	0.000	C#	0.3460	-1.828	Major
1	Movie	Martin & les fées	Perdu d'avance (par Gad Elmaleh)	0BjC1NfoEOOusryehmNudP	1	0.246	0.590	137373	0.737	0.000	F#	0.1510	-5.559	Minor
2	Movie	Joseph Williams	Don't Let Me Be Lonely Tonight	0CoSDzoNIKCRs124s9uTVy	3	0.952	0.663	170267	0.131	0.000	C	0.1030	-13.879	Minor
3	Movie	Henri Salvador	Dis-moi Monsieur Gordon Cooper	0Gc6TVm52BwZD07Ki6tIvf	0	0.703	0.240	152427	0.326	0.000	C#	0.0985	-12.178	Major
4	Movie	Fabien Nataf	Ouverture	0IusIXpMROHdEPvSl1fTQK	4	0.950	0.331	82625	0.225	0.123	F	0.2020	-21.150	Major

데이터 정보 확인

02

```
# 데이터 정보 확인
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 232725 entries, 0 to 232724
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   genre             232725 non-null  object
 1   artist_name       232725 non-null  object
 2   track_name        232725 non-null  object
 3   track_id          232725 non-null  object
 4   popularity        232725 non-null  int64
 5   acousticness      232725 non-null  float64
 6   danceability      232725 non-null  float64
 7   duration_ms       232725 non-null  int64
 8   energy            232725 non-null  float64
 9   instrumentalness  232725 non-null  float64
 10  key               232725 non-null  object
 11  liveness          232725 non-null  float64
 12  loudness          232725 non-null  float64
 13  mode              232725 non-null  object
 14  speechiness       232725 non-null  float64
 15  tempo             232725 non-null  float64
 16  time_signature    232725 non-null  object
 17  valence           232725 non-null  float64
dtypes: float64(9), int64(2), object(7)
memory usage: 32.0+ MB
```

```
# 결측치 확인
df.isna().sum()
```

```
genre               0
artist_name         0
track_name          0
track_id            0
popularity          0
acousticness        0
danceability        0
duration_ms         0
energy              0
instrumentalness    0
key                 0
liveness            0
loudness            0
mode                0
speechiness         0
tempo               0
time_signature      0
valence             0
dtype: int64
```

03

결측치 확인

확인 결과, 없음

Target 전처리

이중 분류 문제를 풀기 위해서 target 값인 'popularity(0~100)'의 수치를 '0(인기 없음)', '1(인기 있음)'으로 치환한다.
0과 1로 나누는 척도는 새로운 데이터 셋인 '1998~2020년도의 Top 100' 데이터 셋의
popularity 값의 평균 이상의 값들을 '인기 있음'으로 분류한다.

01

새로운 데이터 셋 불러오기

```
top_100s = pd.read_csv('songs_normalize.csv')
top_100s.head()
```

	artist	song	duration_ms	explicit	year	popularity
0	Britney Spears	Oops!...I Did It Again	211160	False	2000	77
1	blink-182	All The Small Things	167066	False	1999	79
2	Faith Hill	Breathe	250546	False	1999	66
3	Bon Jovi	It's My Life	224493	False	2000	78
4	*NSYNC	Bye Bye Bye	200560	False	2000	65

```
# 평균 값을 구하고, 그 값을 기준으로 0과 1로 치환
cutline = top_100s.popularity.mean()

df[target] = df[target]>=cutline

df[target] = df[target].replace([False, True], [0, 1])
```

0과 1로 치환

02

문자형 칼럼 전처리

```
# 문자형 칼럼 카디널리티 확인
obj = df.dtypes[df.dtypes == 'object'].index
df[obj].nunique().sort_values(ascending=False)
```

```
track_id          176774
track_name        148615
artist_name        14564
genre                 27
key                   12
time_signature         5
mode                   2
dtype: int64
```

01

문자형 칼럼 카디널리티 확인

id와 name이 들어간 칼럼이
카디널리티가 높은 것을 확인.
Private한 데이터이므로 추후 삭제.

"Children's Music", 'Children’ s Music'를
동일한 문자로 통일해야함을 확인.

문자형 칼럼 정보 확인

02

```
# 문자형 칼럼 데이터 상세 확인
df.genre.unique()
```

```
array(['Movie', 'R&B', 'A Capella', 'Alternative', 'Country', 'Dance',
       'Electronic', 'Anime', 'Folk', 'Blues', 'Opera', 'Hip-Hop',
       "Children's Music", 'Children’ s Music', 'Rap', 'Indie',
       'Classical', 'Pop', 'Reggae', 'Reggaeton', 'Jazz', 'Rock', 'Ska',
       'Comedy', 'Soul', 'Soundtrack', 'World'], dtype=object)
```

```
# 'Children’ s Music'를 "Children's Music"로 교체
df['genre'] = df['genre'].replace("Children’ s Music", "Children's Music")
```

```
# 잘 바뀌었나 확인 결과, 바뀌었음을 확인.
df.genre.unique()
```

```
array(['Movie', 'R&B', 'A Capella', 'Alternative', 'Country', 'Dance',
       'Electronic', 'Anime', 'Folk', 'Blues', 'Opera', 'Hip-Hop',
       "Children's Music", 'Rap', 'Indie', 'Classical', 'Pop', 'Reggae',
       'Reggaeton', 'Jazz', 'Rock', 'Ska', 'Comedy', 'Soul', 'Soundtrack',
       'World'], dtype=object)
```

03

전처리

수치형 칼럼 전처리

duration_ms
99373
137373
170267
152427
82625

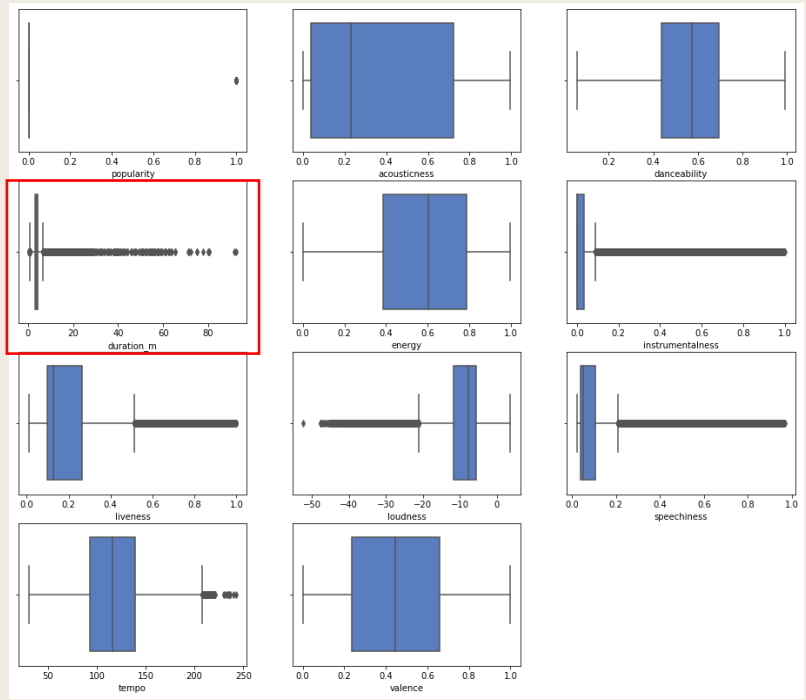
01

재생시간 단위가
micro seconds임을 확인

02

편하게 보고자
minutes 단위로 변경

```
# 'duration_ms' 칼럼 단위 변경
df['duration_ms'] = round(df['duration_ms'] / 1000 / 60, 2)
df.rename(columns={'duration_ms' : 'duration_m'}, inplace=True)
```



03

수치형 데이터
이상치 확인

'duration_m'이 이상하다는 것을 확인.

원래 0~1.0까지 분포.

'tempo' 같은 경우 200이상의 장르도 존재
ex) 하드코어 테크노, 스피드코어 등

	genre	artist_name	track_name	track_id	popularity	acousticness	danceability	duration_m
212083	Comedy	La Mesa Reñoña	Episodio 15 (Lady Orinoco, Políticas De Youtub...	76vVk4HCOLP5r7hA53SyXg	0	0.84600	0.468	92.55
162671	Reggaeton	DJ Luigi	6 : 00 Am	5p44BE1eYof6B2HSj3LTfQ	0	0.07840	0.513	91.47
211969	Comedy	La Mesa Reñoña	Episodio 14 (Machismo, Juanga, Bebé a Bordo)	3V800wAK7yLS2vckDD4WCz	0	0.85000	0.392	80.51

04

재생시간이 6분이
넘어가는 데이터 추출

40분, 80분, 90분 등등 노래 서칭 결과,
실제로 재생시간이 그러함을 확인.

중간 점검

※ 아래 사항은 train set과 test set을 나눈 후에 train set에만 적용한다.

1. 필자가 원하는 음악은 6분 이하의 음원이므로 넉넉히 7분 이상의 음원은 삭제한다.

2. 또한, 'Movie', 'Comedy'와 같은 장르는 음악이 아닌 음원이 추출된 형식이 존재한다.
   그 중에서도 'speechiness가 0.66 이상인 값은 전적으로 음성으로 만들어진 트랙을 나타내므로(데이터 특성 설명 페이지 참고)
   필자가 원하는 데이터가 아니므로 삭제해야한다.

3. 그러나 위 두 장르는 대중적인 음악과 방향이 먼, 삭제해야 할 장르에 포함되므로 한 번에 삭제한다.
   ['Anime', 'Comedy', 'Opera', 'Movie', 'A Capella', 'Classical', "Children's Music"]

즉, 1번과 3번만 시행하도록 한다.

남은 전처리

01

모델 학습에 필요 없는
특성 삭제

```
# 모델 학습에 필요없는 특성 삭제
delete = ['artist_name', 'track_name', 'track_id']
df = df.drop(delete, axis=1)
```

```
# 중복값 삭제 및 인덱스 재정렬
df = df.drop_duplicates()
df = df.reset_index(drop=True)
```

중복값 삭제

02

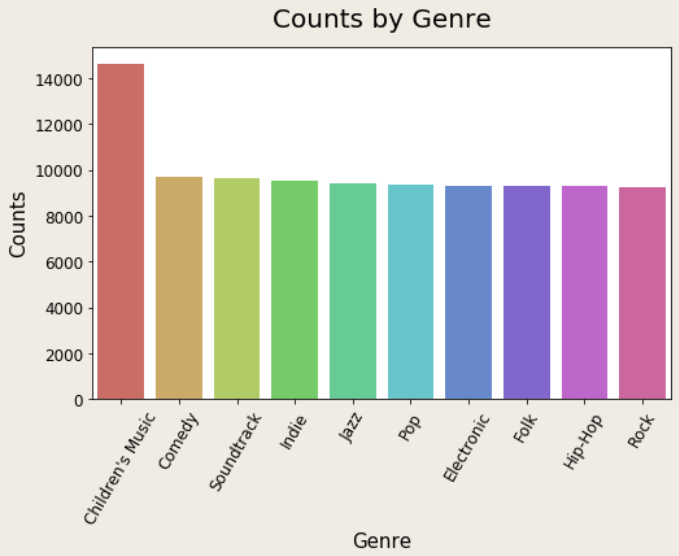
	genre	popularity	acousticness	danceability	duration_m	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence
0	Movie	0	0.61100	0.389	1.66	0.910	0.000000	C#	0.3460	-1.828	Major	0.0525	166.969	4/4	0.814
1	Movie	0	0.24600	0.590	2.29	0.737	0.000000	F#	0.1510	-5.559	Minor	0.0868	174.003	4/4	0.816
2	Movie	0	0.95200	0.663	2.84	0.131	0.000000	C	0.1030	-13.879	Minor	0.0362	99.488	5/4	0.368
3	Movie	0	0.70300	0.240	2.54	0.326	0.000000	C#	0.0985	-12.178	Major	0.0395	171.758	4/4	0.227
4	Movie	0	0.95000	0.331	1.38	0.225	0.123000	F	0.2020	-21.150	Major	0.0456	140.576	4/4	0.390
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
232199	Soul	0	0.00384	0.687	5.44	0.714	0.544000	D	0.0845	-10.626	Major	0.0316	115.542	4/4	0.962
232200	Soul	0	0.03290	0.785	4.71	0.683	0.000880	E	0.2370	-6.944	Minor	0.0337	113.830	4/4	0.969
232201	Soul	0	0.90100	0.517	2.78	0.419	0.000000	D	0.0945	-8.282	Major	0.1480	84.135	4/4	0.813
232202	Soul	0	0.26200	0.745	3.71	0.704	0.000000	A	0.3330	-7.137	Major	0.1460	100.031	4/4	0.489
232203	Soul	0	0.09730	0.758	5.38	0.470	0.000049	G#	0.0836	-6.708	Minor	0.0287	113.897	4/4	0.479

232204 rows × 15 columns

03

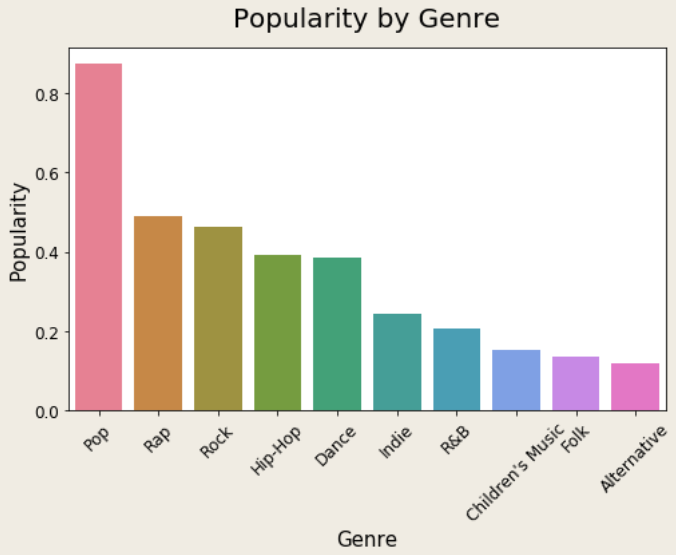
전처리한
데이터 셋 확인

문제 정의 1. 가장 인기 있는 장르는 어떤 것일까요?



01

출시된 장르의 수를 확인



02

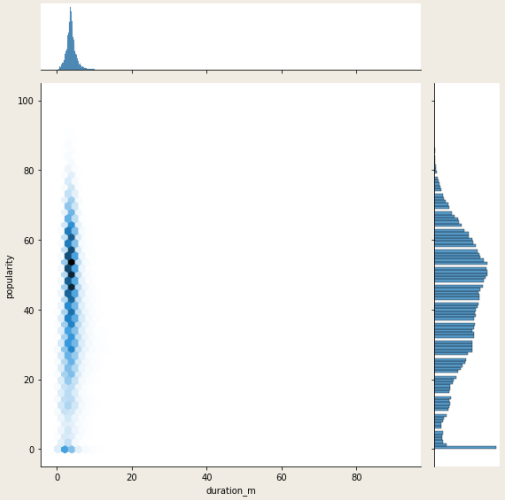
인기도가 높은 Top 10 장르를 확인

※ 가설 1. 가장 인기 있는 장르는 Pop일 것이다. == True

또한, 출시되는 갯수가 많다고 인기를 차지하는 비중이 높지 않다는 것을 알 수 있다.

문제 정의 2. 재생시간은 어느 정도가 적당할까요?
재생시간이 인기도에 영향을 줄까요?

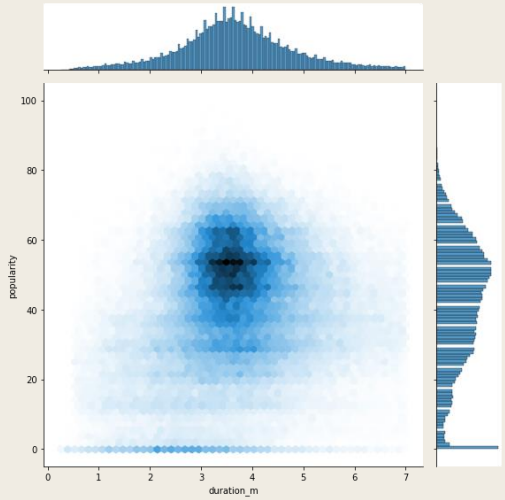
NO. 2-2 EDA 및 시각화



01

재생시간 분포 확인

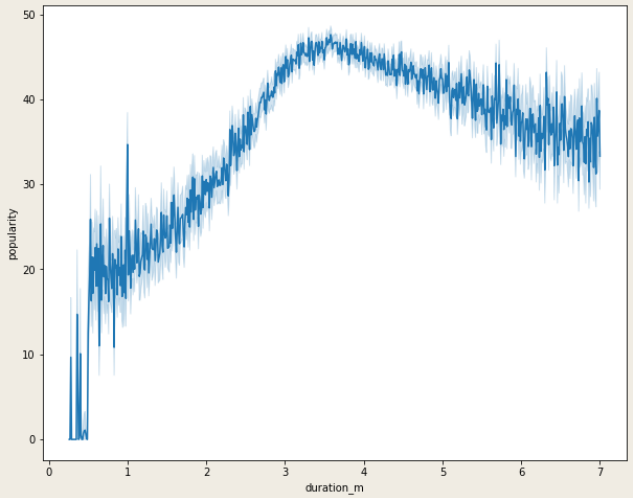
데이터가 몰려있음을 볼 수 있다.



02

재생시간이 7분을 초과하는
데이터를 삭제 후 확인

3~4분 사이의 음원에서
인기도가 높음을 알 수 있으므로
이 시간대가 가장 적당하다.



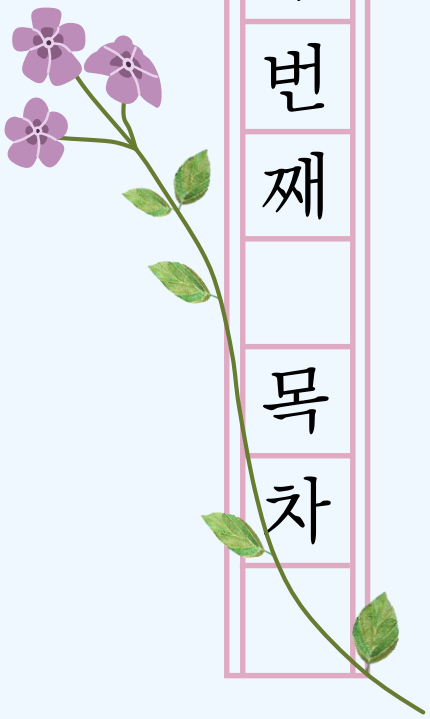
03

더 확실히 보기 위해
선 그래프로 확인

특정 구간의 재생시간이 넘어가면
재생시간이 길어질수록 인기도는 하락

※ 가설 2. 재생시간은 인기도에 영향을 주지 않을 것이다. == False

세 | 번 째 목 차



Train set 전처리

※ 중간 점검 때 체크해 둔 사항을 실행한다.

```
# train, validation, test으로 데이터를 분할
train_val, test = train_test_split(df, test_size=0.2, random_state=42)
train, val = train_test_split(train_val, test_size=0.2, random_state=42)

# 각 데이터 세트의 shape 확인
print("train set : ",train.shape)
print("val set : ",val.shape)
print("test set : ",test.shape)
```

```
train set :  (148610, 15)
val set :  (37153, 15)
test set :  (46441, 15)
```

01

각 데이터 셋으로
분할 후 shape 확인

재생시간이 7분 초과인
트랙 삭제

02

```
# 재생시간이 7분 초과인 트랙 삭제
long_dur = train.query("duration_m>7").index
train = train.drop(long_dur)
train.shape
```

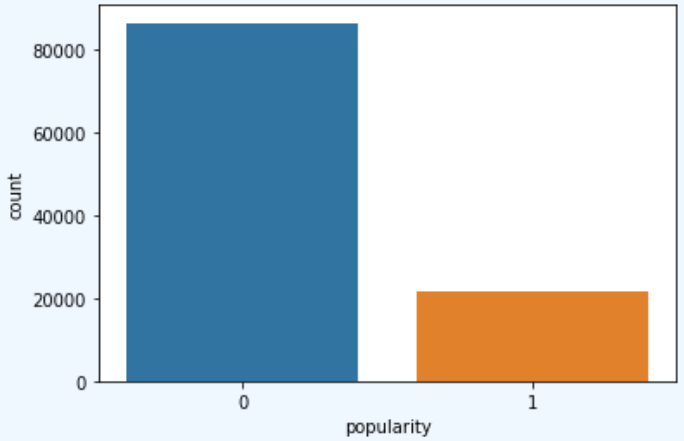
```
(142720, 15)
```

```
# 장르 삭제
delete = ['Anime', 'Comedy', 'Opera', 'Movie', 'A Capella', 'Classical', "Children's Music"]
train = train[~train.genre.isin(delete)]
train.shape
```

```
(107990, 15)
```

03

불필요한 장르 삭제



04

Target 값 비율 확인

확인 결과, 불균형하다는 것을
알 수 있다.

Target 분리 및 기준 모델 설정

```
features = df.drop(target, axis=1).columns

# X와 y를 나누는 함수 만들기
def x_y_split(df) :
    X = df[features]
    y = df[target]
    return X, y

# 각 데이터 세트를 X와 y로 분할
X_train, y_train = x_y_split(train)
X_val, y_val = x_y_split(val)
X_test, y_test = x_y_split(test)
```

```
# 데이터가 잘 나눠졌는지 shape을 확인
print(f'X_train: {X_train.shape}, y_train: {y_train.shape}')
print(f'X_train: {X_val.shape}, y_train: {y_val.shape}')
print(f'X_test: {X_test.shape}, y_test: {y_test.shape}')
```

```
X_train: (107990, 14), y_train: (107990,)
X_train: (37153, 14), y_train: (37153,)
X_test: (46441, 14), y_test: (46441,)
```

01

각 데이터 셋을
Feature와 Target으로 분리

```
def score(y, y_pred) :
    accuracy = accuracy_score(y, y_pred).round(3)
    precision = precision_score(y, y_pred).round(3)
    recall = recall_score(y, y_pred).round(3)
    f1 = f1_score(y, y_pred).round(3)
    return accuracy, precision, recall, f1
```

```
def auc(y, y_pred_proba) :
    auc = roc_auc_score(y, y_pred_proba).round(3)

    return auc
```

각 평가 지표 점수를
계산하는 함수 작성

02

```
# 기준모델 설정
base = y_train.mode()[0]
baseline = len(y_train) * [base]

baseline_accuracy, baseline_precision, baseline_recall, baseline_f1 = score(y_train, baseline)
baseline_auc = auc(y_train, baseline)
```

```
print("BaseLine Accuracy :", baseline_accuracy)
print("BaseLine Precision Score :", baseline_precision)
print("BaseLine Recall Score :", baseline_recall)
print("BaseLine F1 Score :", baseline_f1)
print("BaseLine AUC :", baseline_auc)
```

```
BaseLine Accuracy : 0.801
BaseLine Precision Score : 0.0
BaseLine Recall Score : 0.0
BaseLine F1 Score : 0.0
BaseLine AUC : 0.5
```

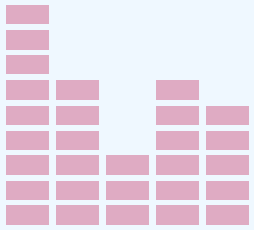
03

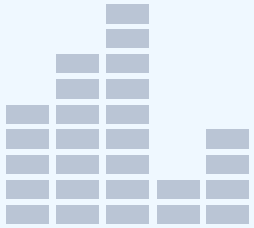
기준 모델 설정 후
점수 계산

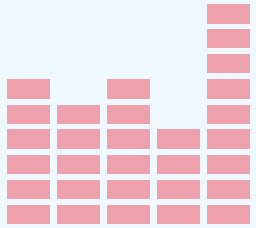
Random Forest
Classfier

XGB
Classfier

LGBM
Classfier







Random Forest 모델

```
# 하이퍼파라미터 서치에 사용할 값 범위 설정
params = {
    "randomforestclassifier__n_estimators": hp.quniform("n_estimators", 100, 2500, 100),
    # "randomforestclassifier__min_samples_split": hp.choice("min_samples_split", [2, 3, 5, 7, 9]),
    "randomforestclassifier__max_depth": hp.choice("max_depth", [1, 2, 3, 4, 5, 10 ,20, 30]),
    # "randomforestclassifier__max_leaf_nodes": hp.quniform("max_leaf_nodes", 1, 5, 1),
    # "randomforestclassifier__oob_score": hp.choice("oob_score", [True, False]),
    # "randomforestclassifier__max_features": hp.choice("max_features", ['auto', 'sqrt', 'log2']),
    # "randomforestclassifier__criterion": hp.choice("criterion", ['gini', 'entropy']),
}
```

```
# 베이지안 서치 시행
def get_pipe(params):
    params["randomforestclassifier__max_depth"] = int(
        params["randomforestclassifier__max_depth"]
    )
    params["randomforestclassifier__n_estimators"] = int(
        params["randomforestclassifier__n_estimators"]
    )
    pipe = make_pipeline(
        OrdinalEncoder(),
        RandomForestClassifier(
            class_weight='balanced',
            random_state=42,
            n_jobs=-1,
        ),
    )
    pipe = pipe.set_params(**params)
    return pipe


def fit_and_eval(params):
    pipe = get_pipe(params)
    score = cross_val_score(pipe, X_train, y_train, cv=3, scoring="roc_auc")
    avg_cv_score = np.mean(score)
    return {"loss": -avg_cv_score, "status": STATUS_OK}


rf_trials = (
    Trials()
)

best_params = fmin(
    fn=fit_and_eval, trials=rf_trials, space=params, algo=tpe.suggest, max_evals=30
)
```

```
100%|██████████████████████████████████████████████████| 30/30 [32:10<00:00, 64.34s/trial, best loss: -0.8933716129788603]
```

```
print("Random Forest 최적의 하이퍼파라미터 :", rf_trials.best_trial["misc"]["vals"])
print("Random Forest 최적의 AUC :", -rf_trials.best_trial["result"]["loss"])
```

```
Random Forest 최적의 하이퍼파라미터 : {'max_depth': [7], 'n_estimators': [1600.0]}
Random Forest 최적의 AUC : 0.8933716129788603
```

01

베이지안 서치를 통한
최적의 파라미터 조정

모델 학습 후
각 검증 스코어 확인

02

```
# 랜덤포레스트분류 모델
rf_pipe = make_pipeline(
            OrdinalEncoder(),
            RandomForestClassifier(
                class_weight='balanced',
                random_state=42,
                n_jobs=-1,
                n_estimators=1600,
                max_depth=7,
        ),
)

rf_pipe.fit(X_train, y_train)

y_rf_val_pred = rf_pipe.predict(X_val)
y_rf_val_pred_proba = rf_pipe.predict_proba(X_val)[:,1]

rf_val_accuracy, rf_val_precision, rf_val_recall, rf_val_f1 = score(y_val, y_rf_val_pred)
rf_val_auc = auc(y_val, y_rf_val_pred_proba)
```

```
print("Random Forest Validation Accuracy :", rf_val_accuracy)
print("Random Forest Validation Precision Score :", rf_val_precision)
print("Random Forest Validation Recall Score :", rf_val_recall)
print("Random Forest Validation F1 Score :", rf_val_f1)
print("Random Forest Validation AUC :", rf_val_auc)
```

```
Random Forest Validation Accuracy : 0.686
Random Forest Validation Precision Score : 0.304
Random Forest Validation Recall Score : 0.784
Random Forest Validation F1 Score : 0.438
Random Forest Validation AUC : 0.792
```

XGB 모델

```
# 하이퍼파라미터 서치에 사용할 값 범위 설정
params = {
    "xgbclassifier__n_estimators": hp.quniform("n_estimators", 100, 2500, 100),
    "xgbclassifier__learning_rate": hp.quniform("learning_rate", 0.01, 0.06, 0.01),
    "xgbclassifier__colsample_bytree": hp.quniform("colsample_bytree", 0.5, 1.0, 0.1),
    "xgbclassifier__min_child_weight": hp.quniform("min_child_weight", 1, 21, 1),
    "xgbclassifier__max_depth": hp.choice("max_depth", [1, 2, 3, 4, 5, 10 ,20, 30]),
    "xgbclassifier__reg_lambda": hp.choice("reg_lambda", [0, 1, 10, 20, 100, 150, 200]),
    "xgbclassifier__reg_alpha": hp.quniform("reg_alpha", 0, 0.2, 0.1),
    "xgbclassifier__scale_pos_weight": hp.uniform("scale_pos_weight", 2.5, 3.5),
        }
```

```
# 베이지안 서치 시행
def get_pipe(params):
    params["xgbclassifier__max_depth"] = int(
        params["xgbclassifier__max_depth"]
    )
    params["xgbclassifier__n_estimators"] = int(
        params["xgbclassifier__n_estimators"]
    )
    pipe = make_pipeline(
        OrdinalEncoder(),
        XGBClassifier(
            objective="binary:logistic",
            eval_metric="error",
            random_state=42,
            n_jobs=-1,
        ),
    )
    pipe = pipe.set_params(**params)
    return pipe


def fit_and_eval(params):
    pipe = get_pipe(params)
    score = cross_val_score(pipe, X_train, y_train, cv=3, scoring="roc_auc")
    avg_cv_score = np.mean(score)
    return {"loss": -avg_cv_score, "status": STATUS_OK}


xgb_trials = (
    Trials()
)

best_params = fmin(
    fn=fit_and_eval, trials=xgb_trials, space=params, algo=tpe.suggest, max_evals=30
)
```

```
100%|██████████████████████████████████████████████| 30/30 [1:36:15<00:00, 192.52s/trial, best loss: -0.9116167417540769]
```

```
print("XGB 최적의 하이퍼파라미터 :", xgb_trials.best_trial["misc"]["vals"])
print("XGB 최적의 AUC :", -xgb_trials.best_trial["result"]["loss"])
```

```
XGB 최적의 하이퍼파라미터 : {'colsample_bytree': [0.7000000000000001], 'learning_rate': [0.04], 'max_depth': [7], 'min_child_weight': [11.0], 'n_estima
tors': [2500.0], 'reg_alpha': [0.0], 'reg_lambda': [6], 'scale_pos_weight': [2.70458527192024]}
XGB 최적의 AUC : 0.9116167417540769
```

01

베이지안 서치를 통한
최적의 파라미터 조정

모델 학습 후
각 검증 스코어 확인

02

```
# XGB분류 모델
xgb_pipe = make_pipeline(
            OrdinalEncoder(),
            XGBClassifier(
                objective="binary:logistic",
                eval_metric="error",
                random_state=42,
                n_jobs=-1,
                colsample_bytree=0.7000000000000001,
                learning_rate=0.04,
                max_depth=7,
                min_child_weight=11,
                n_estimators=2500,
                reg_alpha=0,
                reg_lambda=6,
                scale_pos_weight=2.70458527192024,
            ))

xgb_pipe.fit(X_train, y_train)

y_xgb_val_pred = xgb_pipe.predict(X_val)
y_xgb_val_pred_proba = xgb_pipe.predict_proba(X_val)[:,1]

xgb_val_accuracy, xgb_val_precision, xgb_val_recall, xgb_val_f1 = score(y_val, y_xgb_val_pred)
xgb_val_auc = auc(y_val, y_xgb_val_pred_proba)
```

```
print("XGB Validation Accuracy :", xgb_val_accuracy)
print("XGB Validation Precision Score :", xgb_val_precision)
print("XGB Validation Recall Score :", xgb_val_recall)
print("XGB Validation F1 Score :", xgb_val_f1)
print("XGB Validation AUC :", xgb_val_auc)
```

```
XGB Validation Accuracy : 0.779
XGB Validation Precision Score : 0.391
XGB Validation Recall Score : 0.745
XGB Validation F1 Score : 0.512
XGB Validation AUC : 0.855
```

LGBM 모델

```
# 하이퍼파라미터 서치에 사용할 값 범위 설정
params = {
    "lgbmclassifier__n_estimators": hp.quniform("n_estimators", 100, 2500, 100),
    "lgbmclassifier__learning_rate": hp.quniform("learning_rate", 0.01, 0.06, 0.01),
    "lgbmclassifier__subsample": hp.quniform("subsample", 0.3, 1.0, 0.1),
    "lgbmclassifier__colsample_bytree": hp.quniform("colsample_bytree", 0.5, 1.0, 0.1),
    "lgbmclassifier__scale_pos_weight": hp.uniform("scale_pos_weight", 2.5, 3.5),
        }
```

```
# 베이지안 서치 시행
def get_pipe(params):
    params["lgbmclassifier__n_estimators"] = int(
        params["lgbmclassifier__n_estimators"]
    )
    pipe = make_pipeline(
        OrdinalEncoder(),
        LGBMClassifier(
            max_depth=-1,
            objective='binary',
            boosting_type='dart',
            random_state=42,
            n_jobs=-1)
            )
    pipe = pipe.set_params(**params)
    return pipe


def fit_and_eval(params):
    pipe = get_pipe(params)
    score = cross_val_score(pipe, X_train, y_train, cv=3, scoring="roc_auc")
    avg_cv_score = np.mean(score)
    return {"loss": -avg_cv_score, "status": STATUS_OK}


lgbm_trials = (
    Trials()
)

best_params = fmin(
    fn=fit_and_eval, trials=lgbm_trials, space=params, algo=tpe.suggest, max_evals=30
)
```

```
100%|██████████████████████████████████████████████████| 30/30 [40:52<00:00, 81.74s/trial, best loss: -0.8835739538897883]
```

```
print("LGBM 최적의 하이퍼파라미터 :", lgbm_trials.best_trial["misc"]["vals"])
print("LGBM 최적의 AUC :", -lgbm_trials.best_trial["result"]["loss"])
```

```
LGBM 최적의 하이퍼파라미터 : {'colsample_bytree': [0.9], 'learning_rate': [0.06], 'n_estimators': [2500.0], 'scale_pos_weight': [3.186529635416566], 's
ubsample': [0.7000000000000001]}
LGBM 최적의 AUC : 0.8835739538897883
```

01

베이지안 서치를 통한
최적의 파라미터 조정

모델 학습 후
각 검증 스코어 확인

02

```
# LGBM분류 모델
lgbm_pipe = make_pipeline(
            OrdinalEncoder(),
            LGBMClassifier(
                max_depth=-1,
                objective='binary',
                boosting_type='dart',
                random_state=42,
                n_jobs=-1,
                n_estimators=2500,
                learning_rate=0.05,
                subsample=0.7000000000000001,
                colsample_bytree=0.9,
                scale_pos_weight=3.186529635416566,
            ))

lgbm_pipe.fit(X_train, y_train)

y_lgbm_val_pred = lgbm_pipe.predict(X_val)
y_lgbm_val_pred_proba = lgbm_pipe.predict_proba(X_val)[:,1]

lgbm_val_accuracy, lgbm_val_precision, lgbm_val_recall, lgbm_val_f1 = score(y_val, y_lgbm_val_pred)
lgbm_val_auc = auc(y_val, y_lgbm_val_pred_proba)
```

```
print("LGBM Validation Accuracy :", lgbm_val_accuracy)
print("LGBM Validation Precision Score :", lgbm_val_precision)
print("LGBM Validation Recall Score :", lgbm_val_recall)
print("LGBM Validation F1 Score :", lgbm_val_f1)
print("LGBM Validation AUC :", lgbm_val_auc)
```

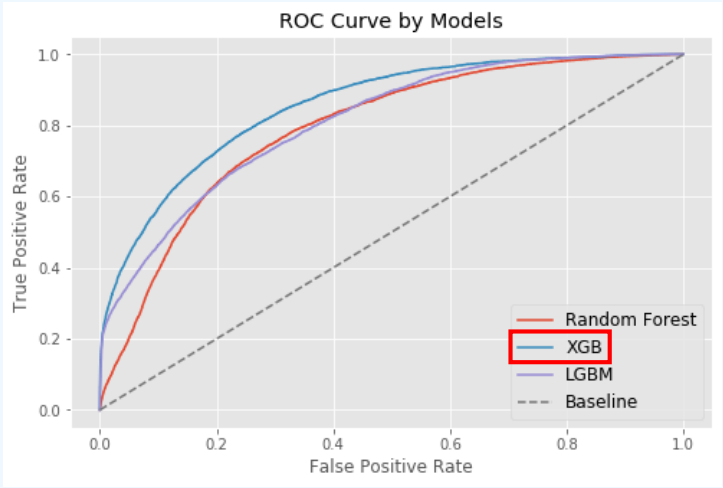
```
LGBM Validation Accuracy : 0.675
LGBM Validation Precision Score : 0.294
LGBM Validation Recall Score : 0.776
LGBM Validation F1 Score : 0.427
LGBM Validation AUC : 0.81
```

모델별 성능 비교

	BaseLine	RandomForest	XGB	LGBM
Accuracy	0.801	0.686	0.779	0.675
Precision	0.000	0.304	0.391	0.294
Recall	0.000	0.784	0.745	0.776
F1	0.000	0.438	0.512	0.427
AUC	0.500	0.792	0.855	0.810

01

각 모델별
검증 점수 비교



02

ROC Curve를 통한 비교

※ 모든 모델의 점수와 그래프를 비교한 결과,
XGB 모델이 가장 뛰어나므로 최종 모델로 XGB 모델을 고른다.

최종 모델의 일반화 성능 평가

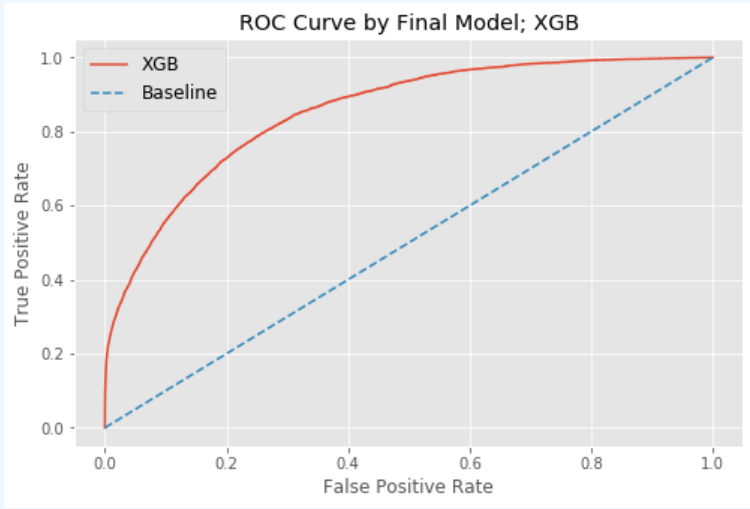
```
# XGB분류 모델 사용
y_xgb_test_pred = xgb_pipe.predict(X_test)
y_xgb_test_pred_proba = xgb_pipe.predict_proba(X_test)[:,1]

xgb_test_accuracy, xgb_test_precision, xgb_test_recall, xgb_test_f1 = score(y_test, y_xgb_test_pred)
xgb_test_auc = auc(y_test, y_xgb_test_pred_proba)
```

```
print("XGB Test Accuracy :", xgb_test_accuracy)
print("XGB Test Precision Score :", xgb_test_precision)
print("XGB Test Recall Score :", xgb_test_recall)
print("XGB Test F1 Score :", xgb_test_f1)
print("XGB Test AUC :", xgb_test_auc)
```

```
XGB Test Accuracy : 0.778
XGB Test Precision Score : 0.395
XGB Test Recall Score : 0.749
XGB Test F1 Score : 0.517
XGB Test AUC : 0.853
```

01

최종 모델인 XGB 모델로
Test Set 예측



02

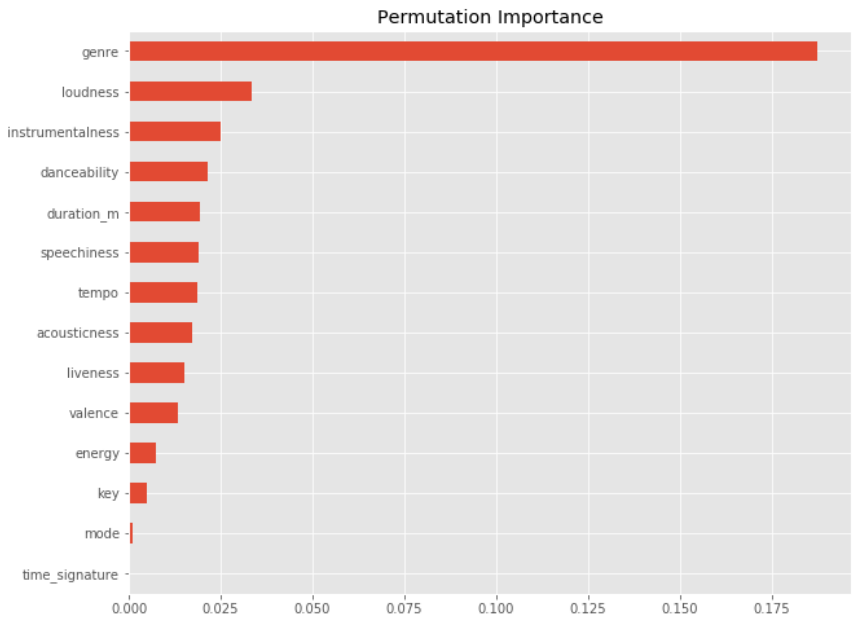
ROC Curve 확인

```
XGB Validation AUC : 0.855
```

※ Test set의 AUC 점수가 Validation set의 AUC 점수랑 비슷하게 나왔다.
나름 일반화가 잘 된 모델이라고 할 수 있다.

문제 정의 4. 인기도에 가장 영향을 많이 주는 특성은 무엇일까요?

NO. 3-2 모델 해석



Weight	Feature
0.1873 ± 0.0050	genre
0.0334 ± 0.0023	loudness
0.0249 ± 0.0016	instrumentalness
0.0213 ± 0.0010	danceability
0.0193 ± 0.0012	duration_m
0.0191 ± 0.0013	speechiness
0.0186 ± 0.0010	tempo
0.0174 ± 0.0015	acousticness
0.0152 ± 0.0008	liveness
0.0132 ± 0.0011	valence
0.0074 ± 0.0015	energy
0.0050 ± 0.0005	key
0.0011 ± 0.0002	mode
0.0005 ± 0.0003	time_signature

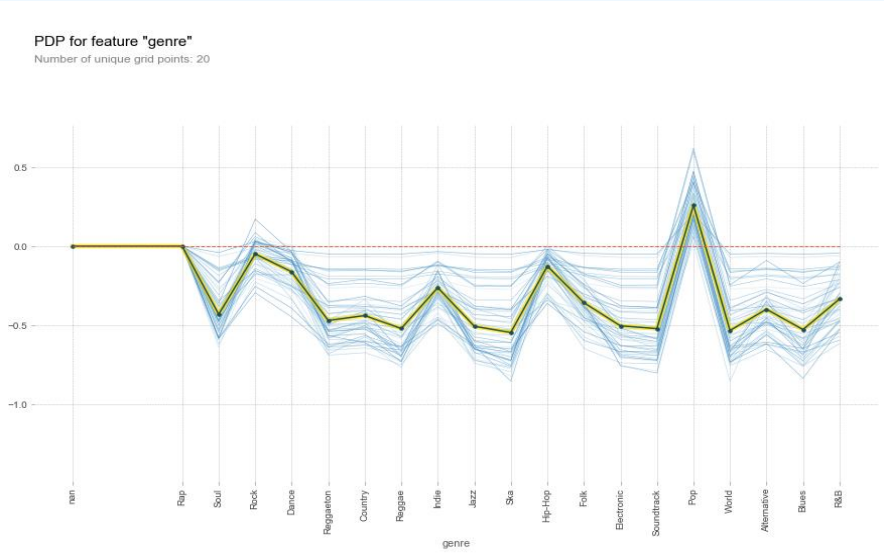
01

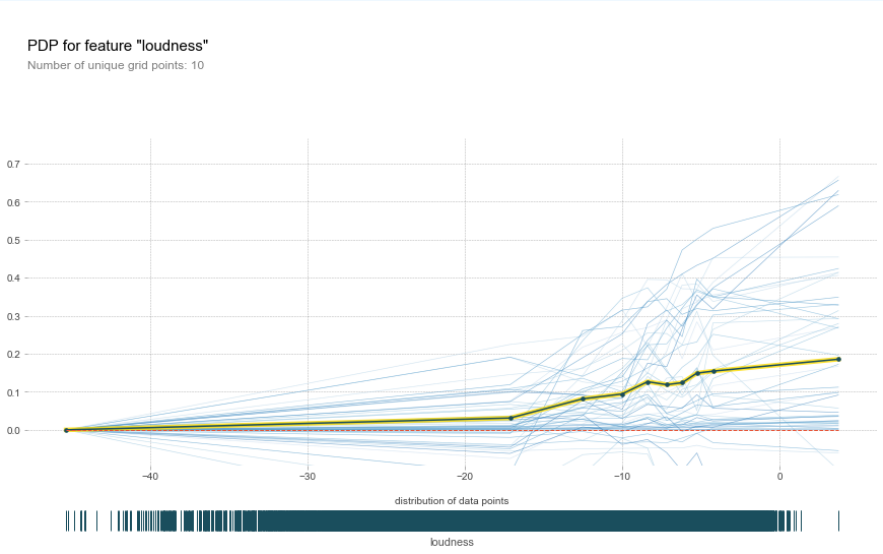
Permutation Importance와
그 수치를 이용한 특성 중요도 파악

※ 가설 4. 인기도에 가장 영향을 많이 주는 특성은 Danceability일 것이다. == False

영향을 가장 많이 주는 특성은 Genre이다.

PDP





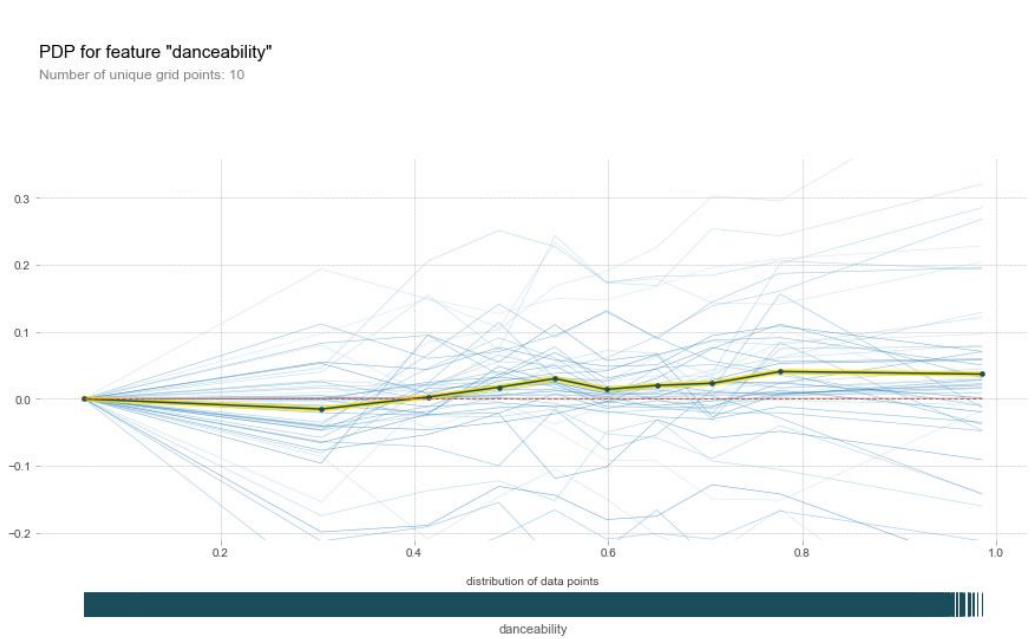
01

Permutation Importance를 통한 특성 중요도 상위 2개의 특성의 PDP 확인

Pop, Rap, Rock인 장르일수록,
데시벨이 높을 수록,
인기도 상승에 영향을 준다.

문제 정의 3. 댄스 음악이 인기가 많을까요, 발라드 음악이 인기가 많을까요?

NO. 3-2 모델 해석



01

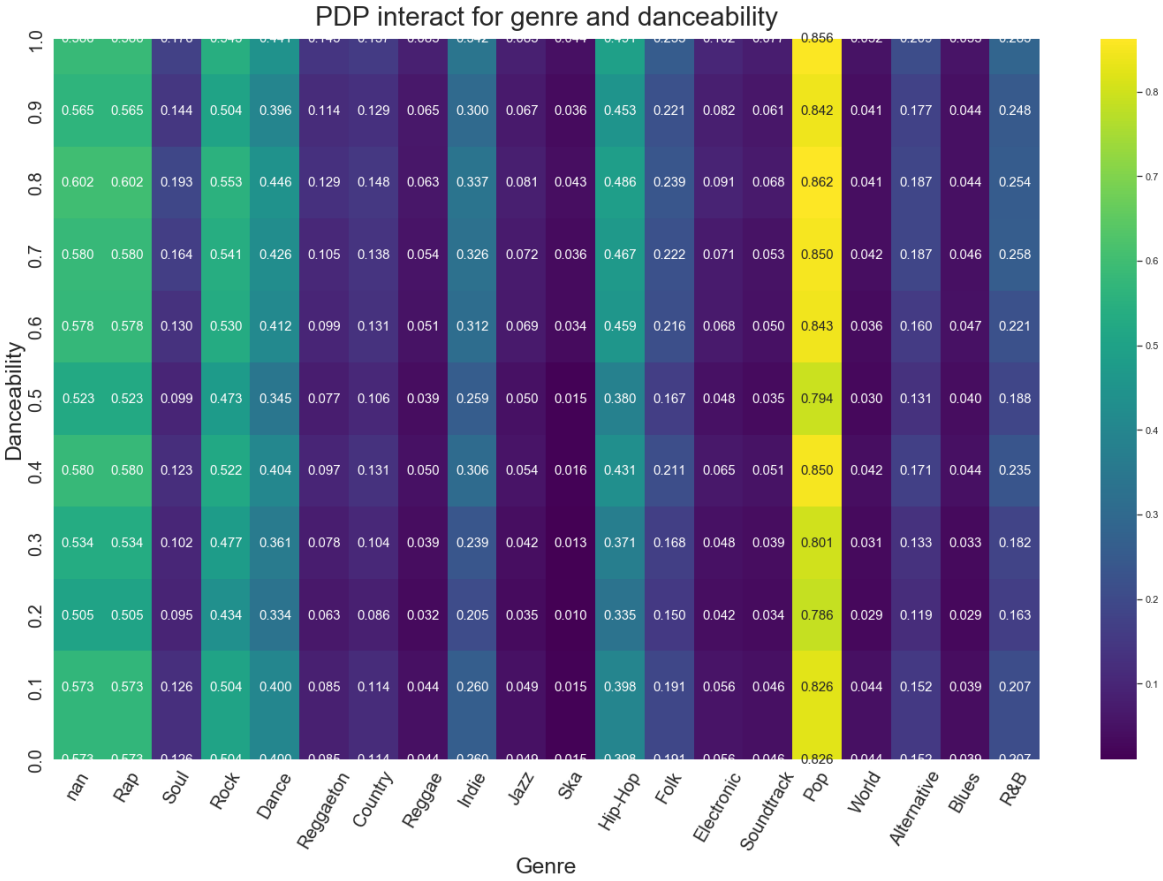
PDP를 통해 Danceability에 의한 인기도 파악

※ 가설 3. 댄스 음악이 인기가 더 많을 것이다. == True

수치가 낮을 때보다 높을 때 인기도가 상승하는 것을 볼 수 있다.

문제 정의 5. Genre에 따라 Danceability가 예측에 영향을 주는 정도가 다를까요?

NO. 3-2 모델 해석



01

2특성 PDP를 통해
Genre에 따른 Danceability의
영향력 파악

※ 가설 5. Genre에 따라 Danceability가 예측에 영향을 주는 정도가 다를 것이다. == True

04. 결론

※ 대중적으로 성공적인 음악(인기도가 높은 음악)을 출시하기 위해서는 어떤 특성을 가져야할까요?

각 특성의 PDP를 그려보았고, 그를 통해 분석하였다.
자세한 그래프는 PPT 맨 뒤 부록 참고.

- 중요 특성

  - genre : Pop, Rock, Rap 장르가 가장 좋다.
  - loudness : 데시벨이 높아야한다. 약 -17db 이상의 지표를 가져야한다.
  - instrumentalness : 값이 상승할수록 인기도는 떨어진다. 특정 지표 없이 그저 음악보다는 보컬이 있어야한다.
  - danceability : 발라드보다는 춤을 출 수 있어야하는 신나는 음악이어야 하고, 약 0.4~0.55의 지표와 0.7 이상의 지표를 가져야한다.

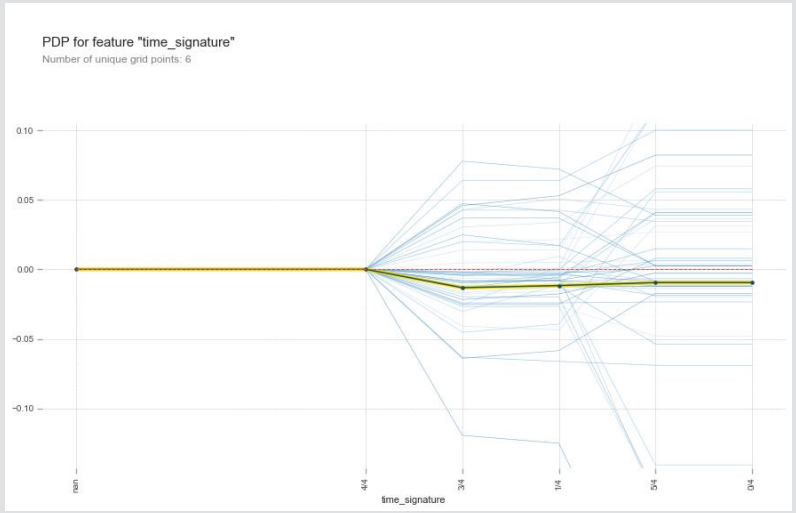
- 참고할만한 특성

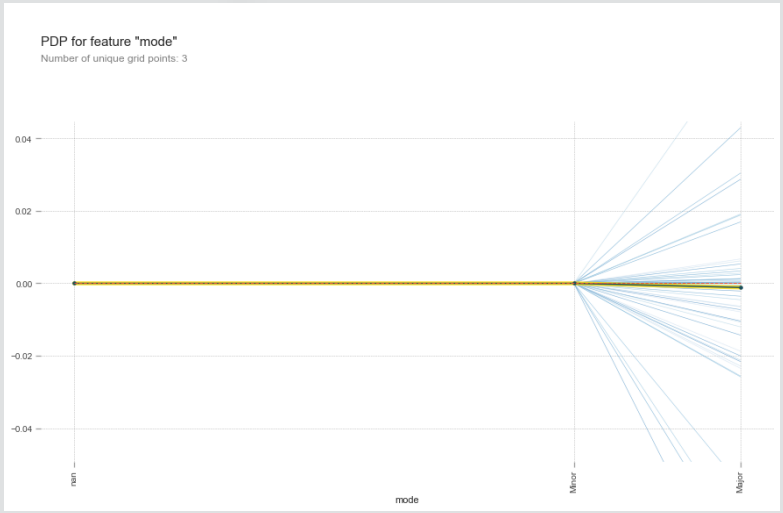
  - duration_m : 짧은 시간보다는 긴 시간이 인기도가 높고, 그 중 3~5분 사이의 재생시간이 좋다. 그 이후의 시간은 영향력이 없다.
  - speechiness : 음성만있는 음원보다 음악과 어울어진 음원이 좋다. 약 0.05~0.15까지의 지표가 가장 좋고 그 이후로는 하락한다.
  - tempo : 너무 빠른 템포보다는 느린 템포가 더 좋다. 그 중 90~140 정도의 템포가 가장 좋다.
  - acousticness : 어쿠스틱 하지 않을 때가 좋고, 약 0.03의 지표가 가장 좋다.
    그 이후로는 영향력이 거의 없다가 0.8 이후로 하락한다. 즉, 너무 어쿠스틱하면 인기도가 하락한다.
  - liveness : 라이브 음원보다는 AR 음원이 좋고, 지표가 약 0.1일 때가 가장 좋다. 그 이후로는 인기도가 계속 하락한다.
  - valence : 영향력이 거의 없고, 그나마 밝은 음악일 때, 인기도가 상승한다.
  - energy : 약 0.4~0.5의 지표가 가장 좋다.
  - key : 영향력이 거의 없고, 그나마 F#일 때, 인기도가 높다.
  - mode : 영향력이 없다. 그래프로도 판단이 불가하다.
  - time_signature : 영향력이 없다. 그나마 4/4 박자.

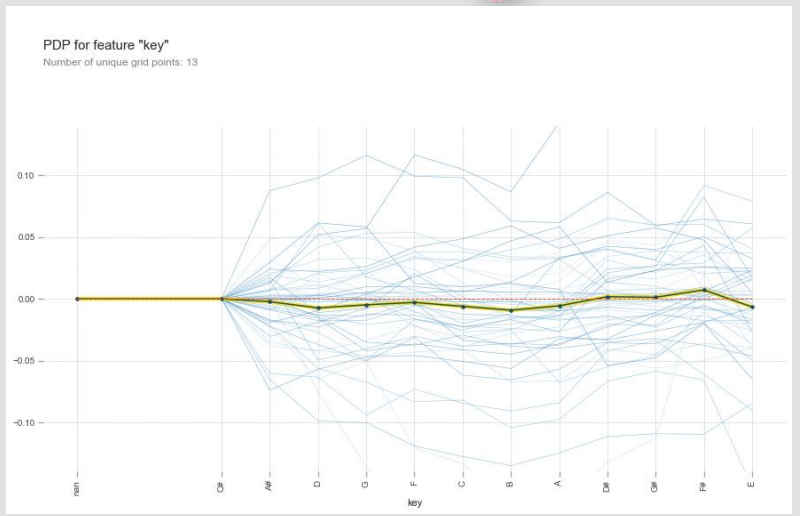
감사합니다.

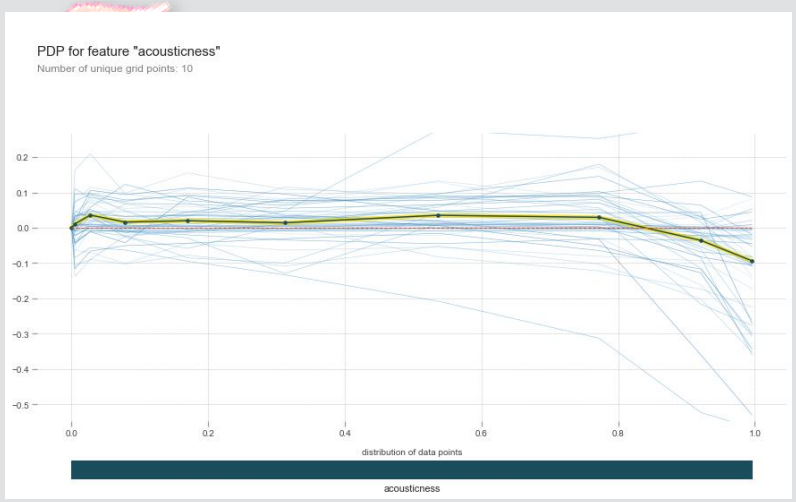
부록 (1)

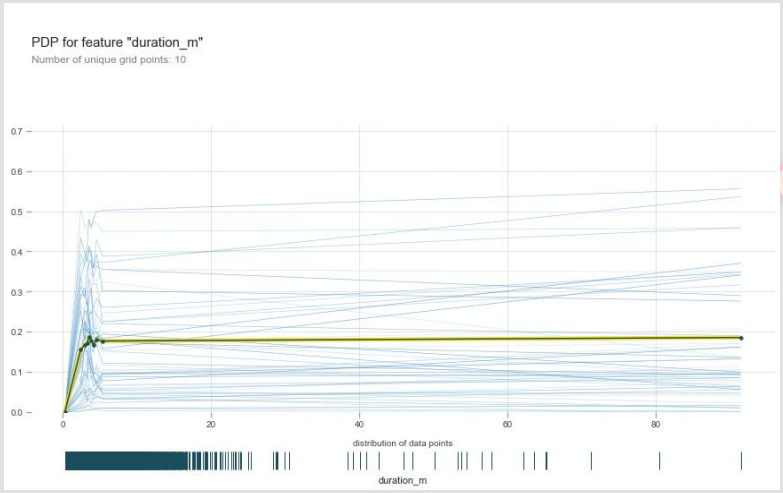
※각 특성에 대한 PDP

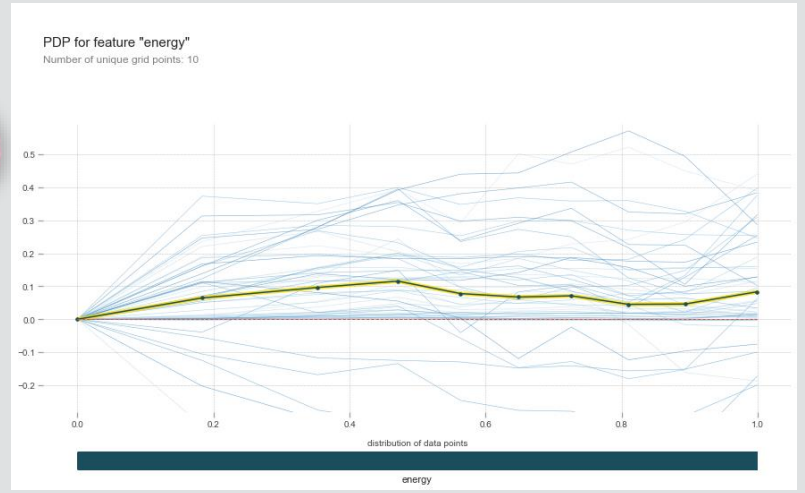












부록 (2)

※각 특성에 대한 PDP