음악 추천 알고리즘에 진짜 탑승하기

- Spotify popularity prediction-

4조 박장호

CONTENTS

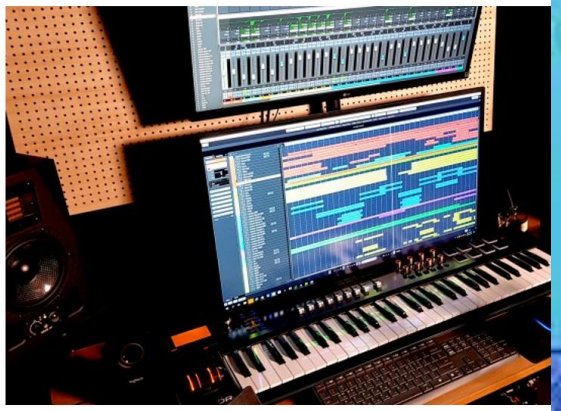
1.
Intro

1. Intro

지난 시간엔…

1. Intro

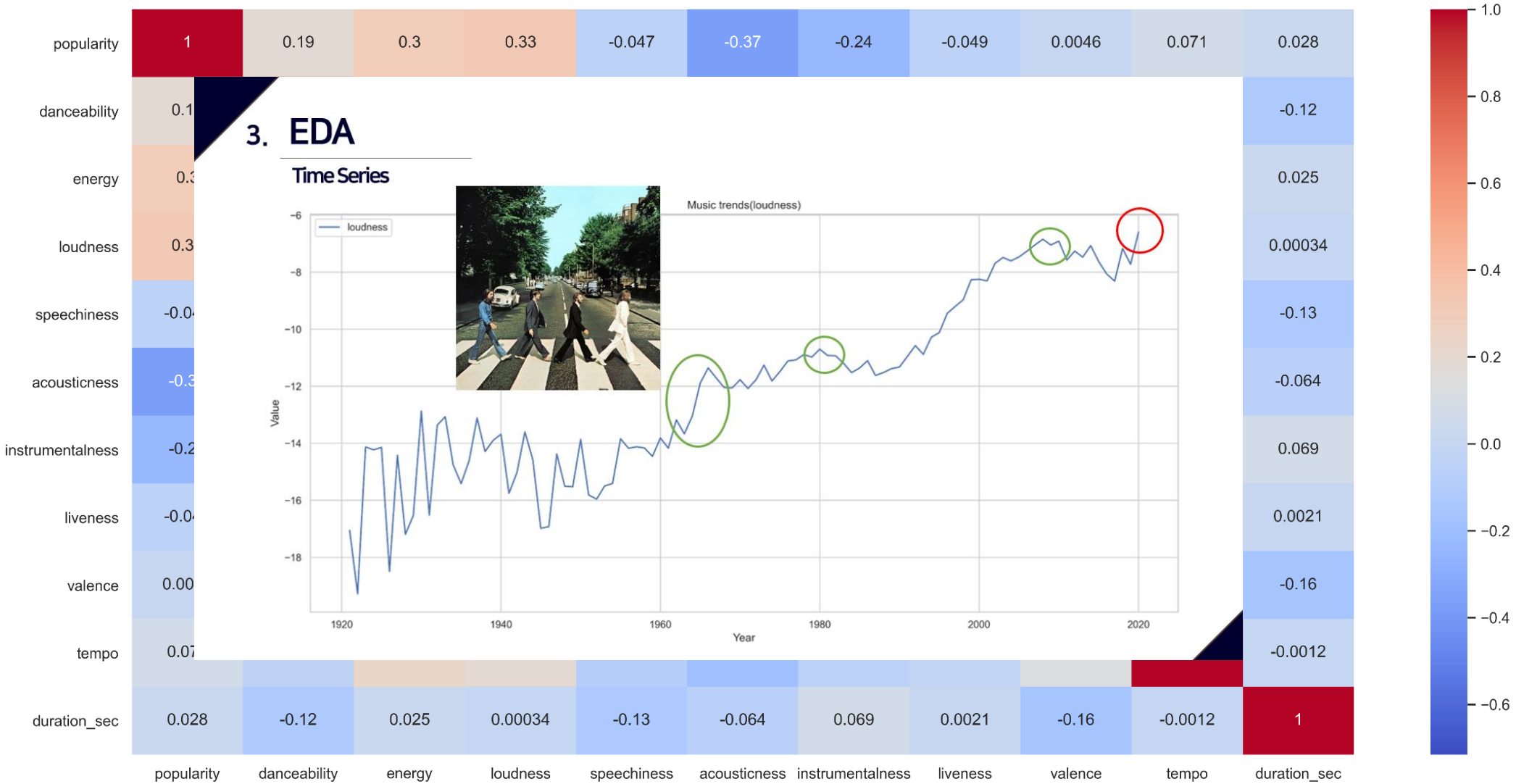
제 꿈은…



작곡!

1. Intro

지난 시간엔…



1. Intro

이 중에 있고 있었던 중요한 변인 중 하나…

- acousticness : 어쿠스틱 정도 (0.0~1.0)
- danceability : 댄스에 적합한 정도 (0.0~1.0)
- energy :격렬하고 활동적인 정도. 빠르고, 소리가 큰 경향 (0.0 ~ 1.0)
- instrumentalness : 보컬 유무 (1.0에 가까울수록 instrumental) (0.0 ~ 1.0)
- liveness : 음원에 관객 소리가 있는 정도. 0.8 이상 시 라이브 음원으로 판단 가능 (0.0 ~ 1.0)
- speechness : 목소리 정도를 감지. 토크쇼 or 오디오북은 1.0, 0.66 이상은 대부분 구어, 0.33~0.66은 음악과 구어(랩 포함), 0.33 미만은 대부분 음악이나 비 언어적 트랙 (0.0 ~ 1.0)
- valence : 긍정적인 정도 (0.0 ~ 1.0)
- tempo : 평균 beats per minute (bpm)
- duration_sec : 플레이 타임 (초)
- loudness : 트랙 전체 소리(dB) 평균화된 트랙 음량을 상대적으로 비교. (-60dB~0dB)
- mode : major(1) 혹은 minor(0)
- key : 0 – C , 1 – C#, 2 – D … (0~11)
- **Popularity** : 유명한 정도. 트랙이 플레이 된 횟수 (0~100)

1. Intro

Regression

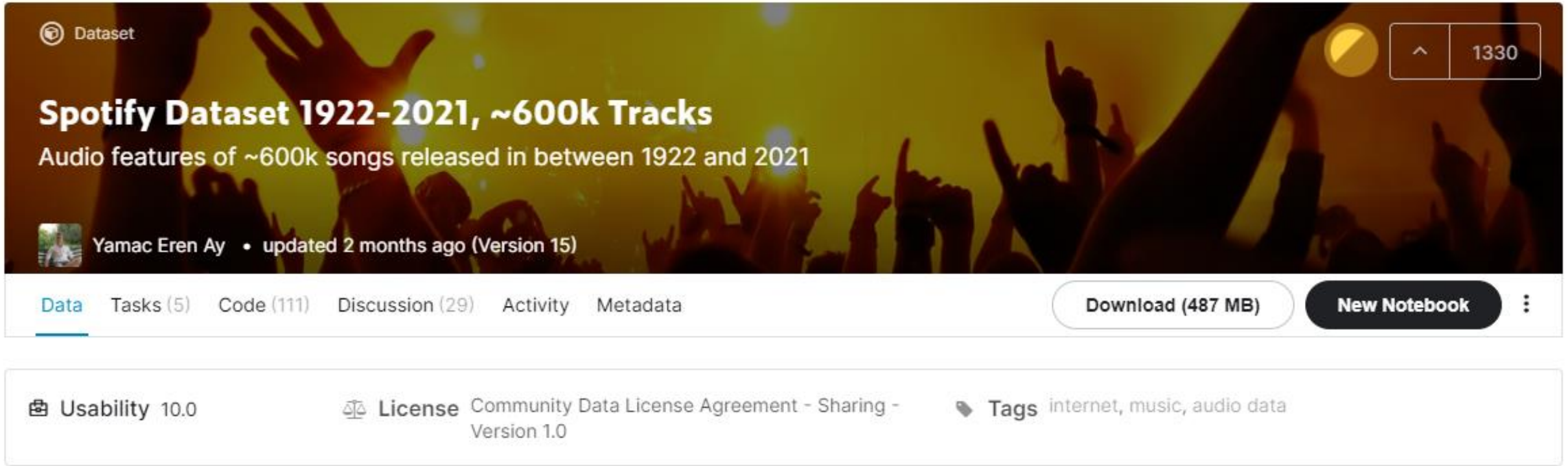
음악의 특징들을 통해
…
각 곡들의
Popularity
예측!!

- acousticness : 어쿠스틱 정도 (0.0~1.0)
- danceability : 댄스에 적합한 정도 (0.0~1.0)
- energy :격렬하고 활동적인 정도. 빠르고, 소리가 큰 경향 (0.0 ~ 1.0)
- instrumentalness : 보컬 유무 (1.0에 가까울수록 instrumental) (0.0 ~ 1.0)
- liveness : 음원에 관객 소리가 있는 정도. 0.8 이상 시 라이브 음원으로 판단 가능 (0.0 ~ 1.0)
- speechness : 목소리 정도를 감지. 토크쇼 or 오디오북은 1.0, 0.66 이상은 대부분 구어, 0.33~0.66은 음악과 구어(랩 포함), 0.33 미만은 대부분 음악이나 비 언어적 트랙 (0.0 ~ 1.0)
- valence : 긍정적인 정도 (0.0 ~ 1.0)
- tempo : 평균 beats per minute (bpm)
- duration_sec : 플레이 타임 (초)
- loudness : 트랙 전체 소리(dB) 평균화된 트랙 음량을 상대적으로 비교. (-60dB~0dB)
- mode : major(1) 혹은 minor(0)
- key : 0 – C , 1 – C#, 2 – D … (0~11)
- **Popularity** : 유명한 정도. 트랙이 플레이 된 횟수 (0~100)

2. Dataset

2. Data set

Data source

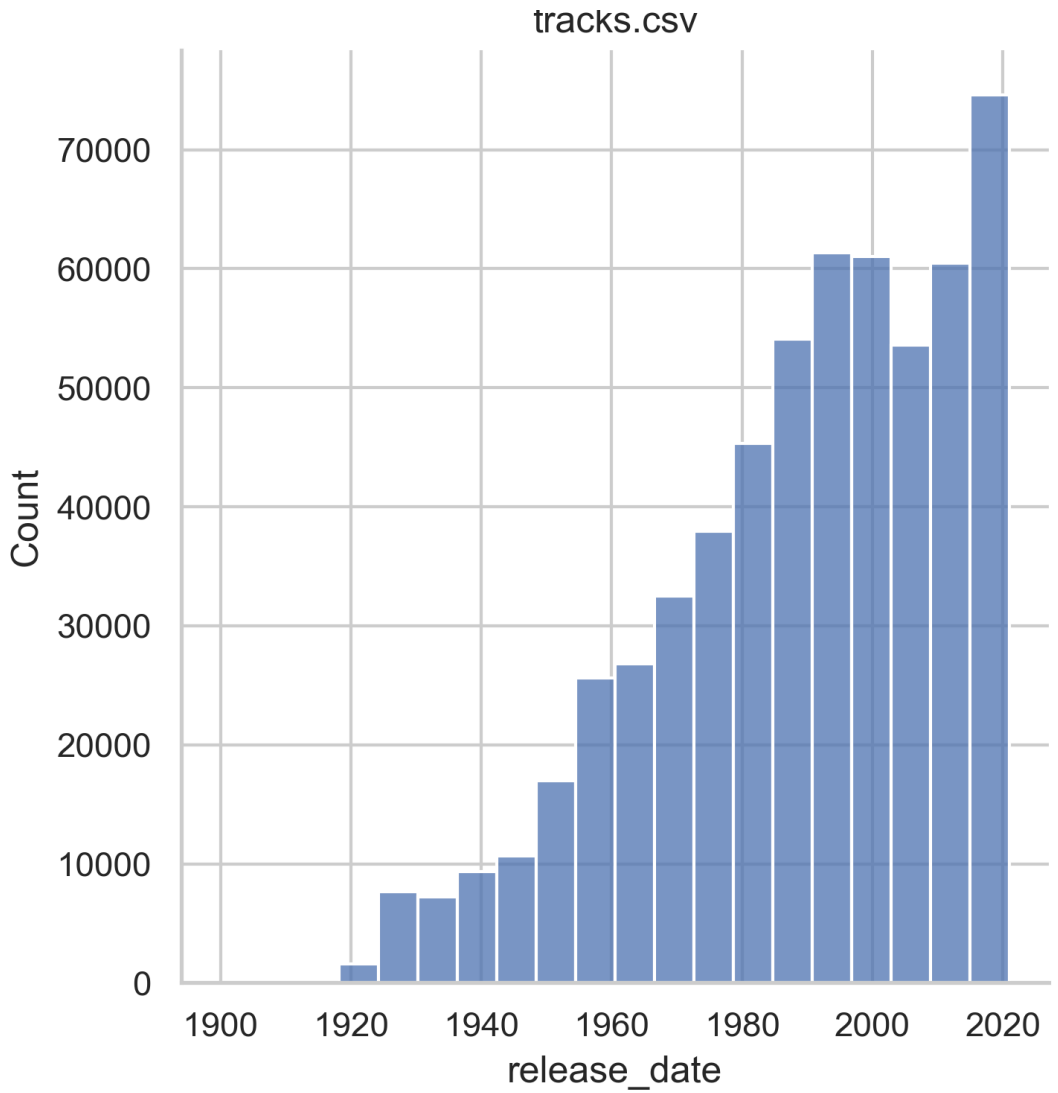


Spotify Dataset 1922-2021, ~600k Tracks (Kaggle)
-Audio features of ~600k songs released in between 1922 and 2021

Spotify Web API, Spotipy (Python module for Spotify Web Server)

2. Dataset

Data feature



track.csv

- **Rows : 586,672**
- **Columns : 20**
- **Due : 1922~2021.04**

2. Dataset

Data Modeling Environment

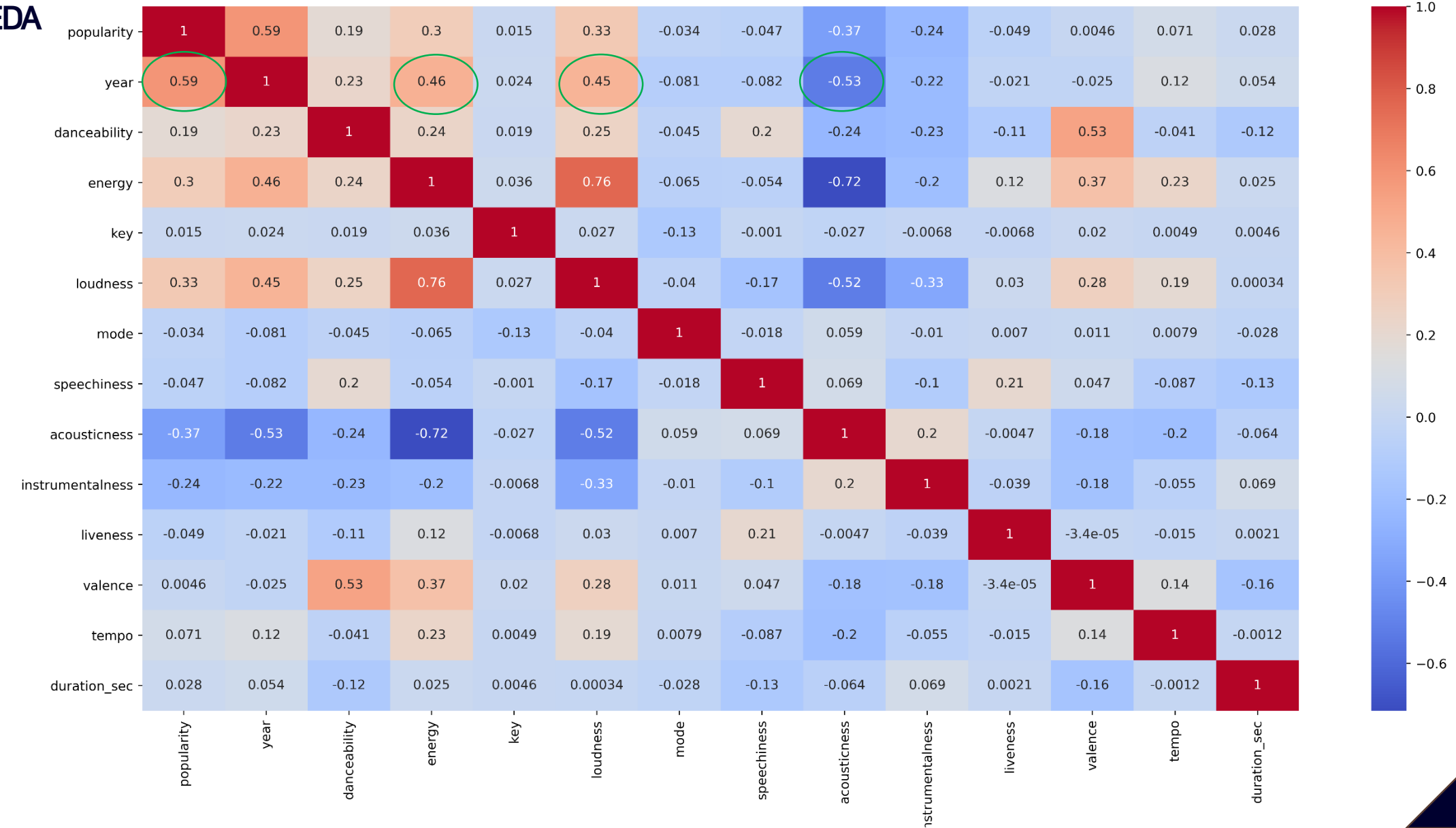
- Jupyter Notebook (Visual Studio Code)
- Python 3.8.8
- Pandas
- Numpy
- Matplotlib
- Seaborn
- Tensorflow 2.3.0
- Keras

Local GPU 활용!!

```
device_type: "GPU"
memory_limit: 4951408640
locality {
  bus_id: 1
  links {
  }
}
incarnation: 18125284504152917750
physical_device_desc: "device: 0, name: NVIDIA GeForce RTX 2060, pci bus id: 0000:1c:00.0, compute capability: 7.5",
name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
locality {
}
incarnation: 11734467077450425645
physical_device_desc: "device: XLA_GPU device"]
```

2. Dataset

EDA



3.
ML

3. ML

Data engineering

종속 변수 : Popularity

```
   1  y
 ✓ 0.7s
478627    19
132350     0
132349     0
132348     0
132347     0
          ..
211946     1
211945     0
211944     0
211955     1
444564    71
Name: popularity, Length: 586672, dtype: int64
```

3. ML

Data engineering

독립 변수 : features of songs

```
1  X
✓ 0.1s
```

	year	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_sec
478627	1900	0.659	0.791	2	-4.895	1	0.0295	0.1390	0.000002	0.1610	0.956	141.999	234
132350	1922	0.567	0.663	2	-5.334	1	0.0318	0.9920	0.878000	0.2680	0.853	103.394	233
132349	1922	0.483	0.060	1	-9.499	1	0.0420	0.9820	0.000089	0.0498	0.381	136.044	196
132348	1922	0.578	0.462	8	-7.217	1	0.0398	0.9950	0.903000	0.0767	0.513	89.876	195
132347	1922	0.565	0.334	10	-6.802	1	0.0309	0.9780	0.032900	0.2560	0.550	97.167	190
...	...	...	...	...	...	...	...	...	...	...	...	...	...
211946	2021	0.777	0.714	11	-4.296	1	0.0532	0.1600	0.000000	0.1150	0.590	90.987	237
211945	2021	0.804	0.786	10	-3.837	0	0.0735	0.1440	0.000000	0.0928	0.575	91.992	309
211944	2021	0.807	0.606	3	-8.871	0	0.0872	0.0946	0.000000	0.1190	0.304	92.988	228
211955	2021	0.855	0.710	1	-5.321	1	0.0939	0.0426	0.000000	0.3370	0.591	89.977	247
444564	2021	0.896	0.459	1	-8.937	1	0.0515	0.0737	0.000084	0.0981	0.484	125.939	169

586672 rows × 13 columns

정규화 필요!!

3. ML

Data engineering

MinMaxScaler()

```python
# 정규화
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
X_scale = min_max_scaler.fit_transform(X)
X_scale
```

✓ 1.1s

```
array([[0.        , 0.66498486, 0.791     , ..., 0.956     , 0.57633908,
        0.04111784],
       [0.18181818, 0.57214934, 0.663     , ..., 0.853     , 0.41965087,
        0.04093984],
       [0.18181818, 0.48738648, 0.06      , ..., 0.381     , 0.5521692 ,
        0.03435386],
       ...,
       [1.        , 0.81432896, 0.606     , ..., 0.304     , 0.37741547,
        0.04004984],
       [1.        , 0.86276488, 0.71      , ..., 0.591     , 0.36519456,
        0.04343183],
       [1.        , 0.90413724, 0.459     , ..., 0.484     , 0.51115549,
        0.02954788]])
```

3. ML

Modeling

1. Linear Regression

```
1  1. Linear Regression
```

```
1  from sklearn.linear_model import LinearRegression
2
3  # 단순회귀분석 모형 객체 생성 후 학습시키기
4  lr = LinearRegression().fit(X_train, y_train)
5
6  y_predict = lr.predict(X_test)
7  y_predict
```

✓ 0.1s

```
array([33.59207929, 13.98322289, 39.78063207, ..., 39.59211601,
       26.6677326 , 20.59572694])
```

```
1  from sklearn.metrics import mean_squared_error
2  from sklearn.metrics import mean_absolute_error
3  print(mean_squared_error(y_test, y_predict))
4  print(mean_absolute_error(y_test, y_predict))
```

✓ 0.2s

```
212.3150693832738
11.215909848827625
```

3. ML

Modeling

2. Lasso Regression

2. Lasso Regression

```
# sklearn 라이브러리에서 선형회귀분석 모듈 가져오기
from sklearn.linear_model import Lasso

# 다항 회귀분석 모형 객체 생성 후 학습 시키기
lasso001 = Lasso(alpha=0.01).fit(X_train,y_train)

y_predict = lasso001.predict(X_test)
y_predict
```

✓ 0.2s

```
array([33.47298472, 13.78331047, 39.62343135, ..., 39.35493381,
       26.53256778, 21.08804533])
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test, y_predict))
print(mean_absolute_error(y_test, y_predict))
```

✓ 0.1s

```
212.406688236076
11.217383408911711
```

3. ML

Modeling

3. Ridge Regression

3. Ridge Regression

```
# sklearn 라이브러리에서 선형회귀분석 모듈 가져오기
from sklearn.linear_model import Ridge

# 다항 회귀분석 모형 객체 생성 후 학습 시키기
ridge50 = Ridge(alpha=5.0).fit(X_train,y_train)
y_predict = ridge50.predict(X_test)
```

✓ 0.7s

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test, y_predict))
print(mean_absolute_error(y_test, y_predict))
```

✓ 0.2s

```
212.31386666336195
11.216558863481639
```

3. ML

Modeling

4. Decision Tree

4. Decision Tree

```python
from sklearn import tree
tree_model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

✓ 0.3s

```python
# train 데이터를 가지고 모델 학습
tree_model.fit(X_train, y_train)

# test 데이터를 가지고 모델 예측
y_predict = tree_model.predict(X_test)
```

✓ 2.1s

```python
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test, y_predict))
print(mean_absolute_error(y_test, y_predict))
```

✓ 0.9s

```
451.4306257883433
14.356058453881205
```

3. ML

Modeling

5. Random Forest

5. Random Forest

```
from sklearn.ensemble import RandomForestRegressor

random = RandomForestRegressor(n_estimators=5,
                               random_state=0).fit(X_train,y_train)
```

✓ 10.9s

```
random.fit(X_train, y_train)
y_predict = tree_model.predict(X_test)
```

✓ 11.6s

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test, y_predict))
print(mean_absolute_error(y_test, y_predict))
```

✓ 0.8s

```
451.4306257883433
14.356058453881205
```

3. ML

Modeling

6. Gradient Boosting

6. Gradient Boosting Regressor

```
1  from sklearn.ensemble import GradientBoostingRegressor
2
3  reg = GradientBoostingRegressor(random_state=777)
4  reg.fit(X_train, y_train)
5  y_predict = reg.predict(X_test)
```

✓ 55.5s

```
1  from sklearn.metrics import mean_squared_error
2  from sklearn.metrics import mean_absolute_error
3  print(mean_squared_error(y_test, y_predict))
4  print(mean_absolute_error(y_test, y_predict))
```

✓ 0.5s

```
185.04981548565968
10.344073883774701
```

3. ML

Modeling

7. Neural Network

7. Linear (nn)

```
# train / validation 데이터를 7:3 비율로 분리
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
                                                  test_size=0.3,
                                                  random_state=777)
```

✓ 0.6s

```
print(X_train.shape, X_val.shape)
```

✓ 0.7s

```
(201228, 13) (86241, 13)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(13,)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))  # 하나의 값을 출력 -> popularity
```

✓ 0.7s

```
model.compile(optimizer='adam', loss='mse', metrics=['mae','mse'])
```

✓ 0.2s

7. Neural Network

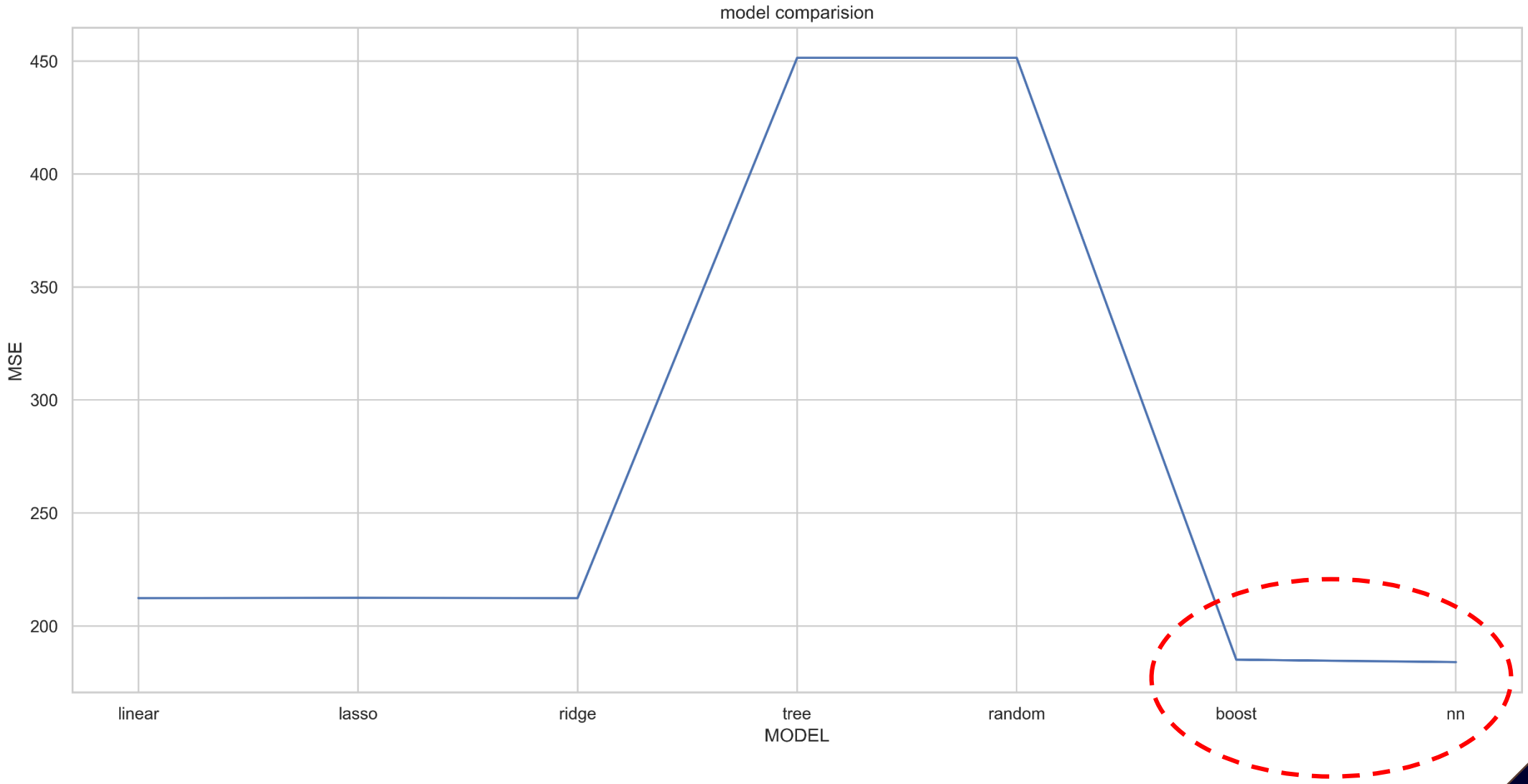
```python
# 모델 학습하기
history = model.fit(X_train,y_train,
                    batch_size=128,
                    epochs = 500,
                    validation_data = (X_val, y_val))
```

✓ 1776.1s

```
Epoch 1/500
1573/1573 [==============================] - 4s 2ms/step - loss: 236.7004
Epoch 2/500
1573/1573 [==============================] - 4s 2ms/step - loss: 204.6170
Epoch 3/500
1573/1573 [==============================] - 4s 2ms/step - loss: 201.9599
```

3. ML

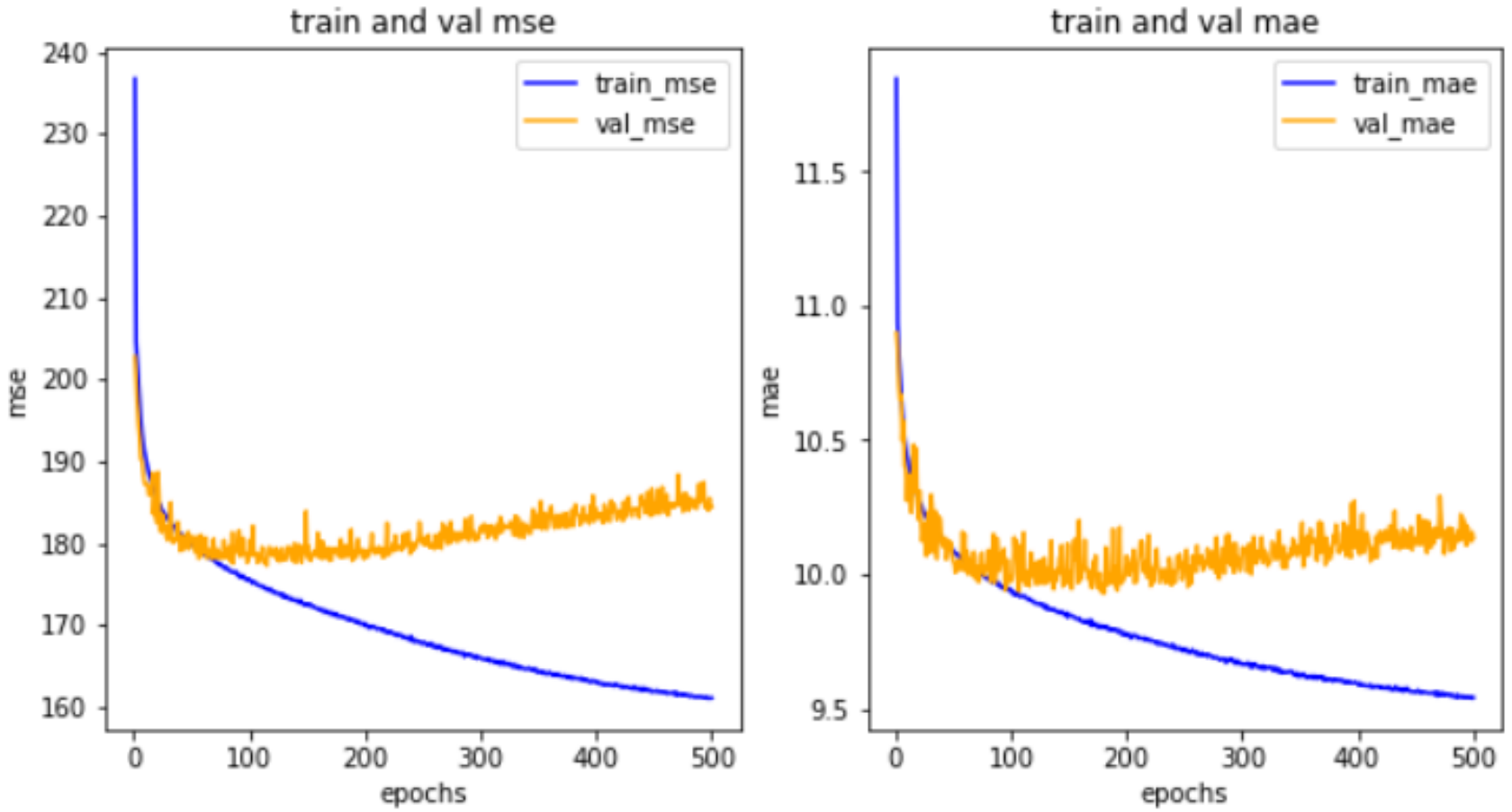
Evaluation



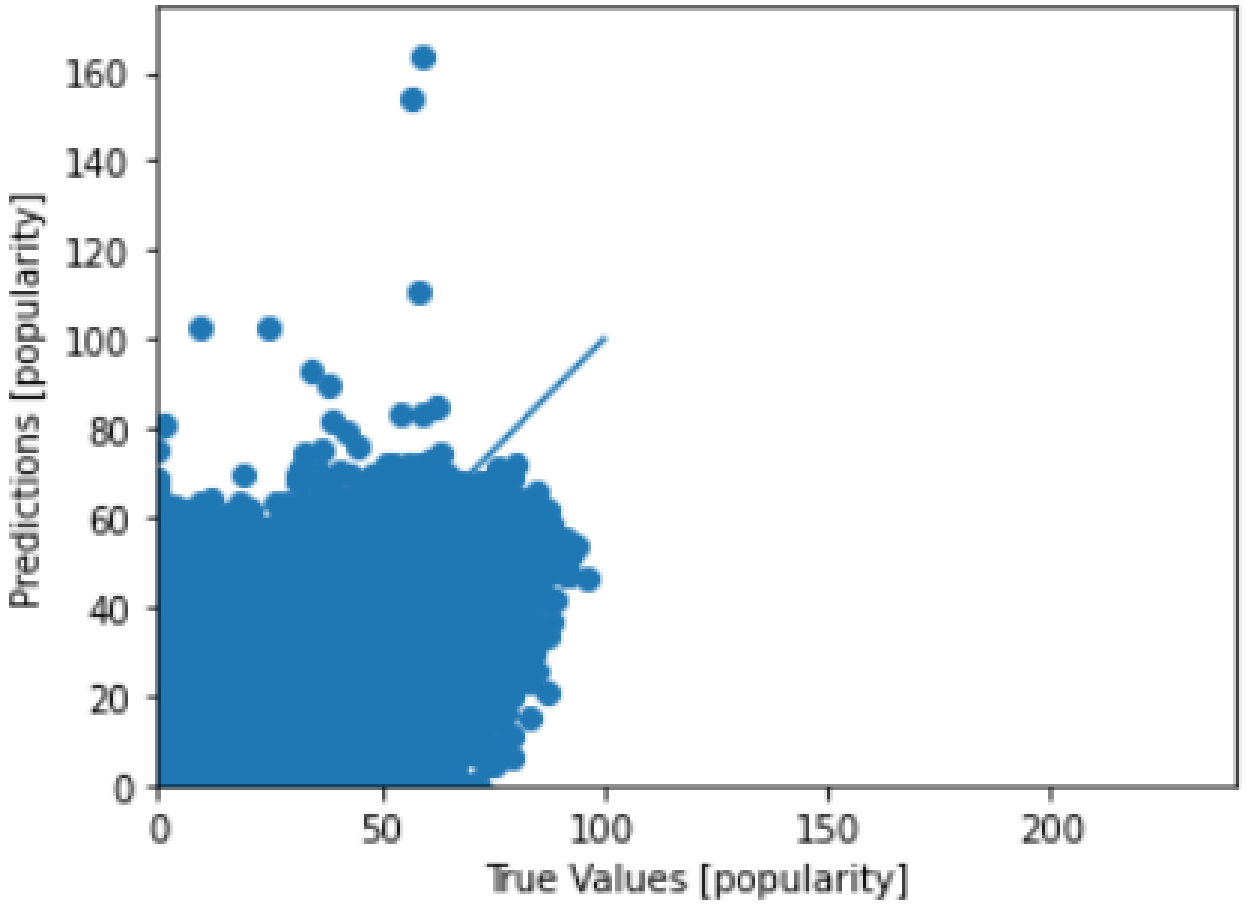
3. ML

Evaluation

7. Neural Network



7. Neural Network



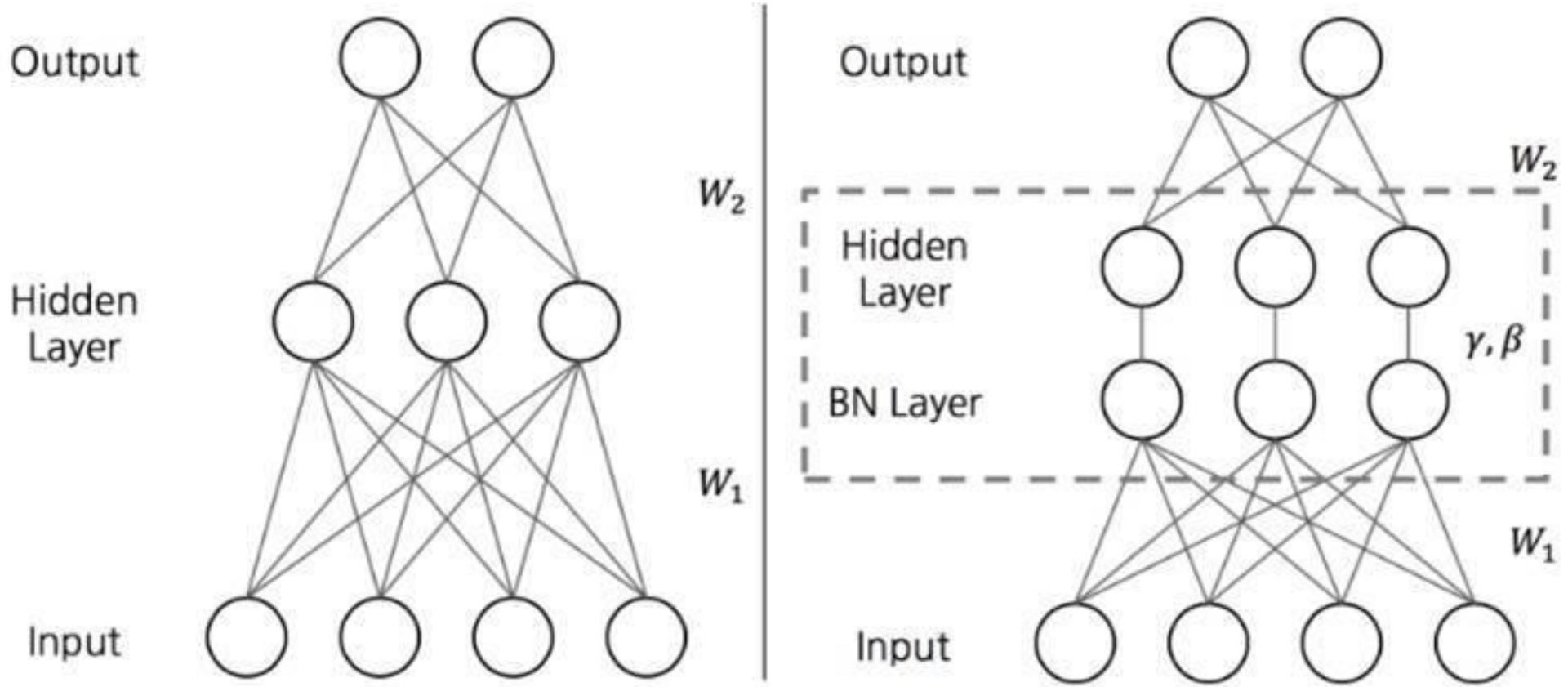
MSE : 183.96

3. ML

Evaluation

Avoid Overffiting…

Batch normalization



3. ML

Evaluation

7. Neural Network (Batch normalization)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization

model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(13,)))
model.add(BatchNormalization())
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(1))  # 하나의 값을 출력 -> popularity
```

✓ ✓ 0.1s

```
model.compile(optimizer='adam', loss='mse', metrics=['mae','mse'])
```

✓ ✓ ✓ ✓ 0.1s

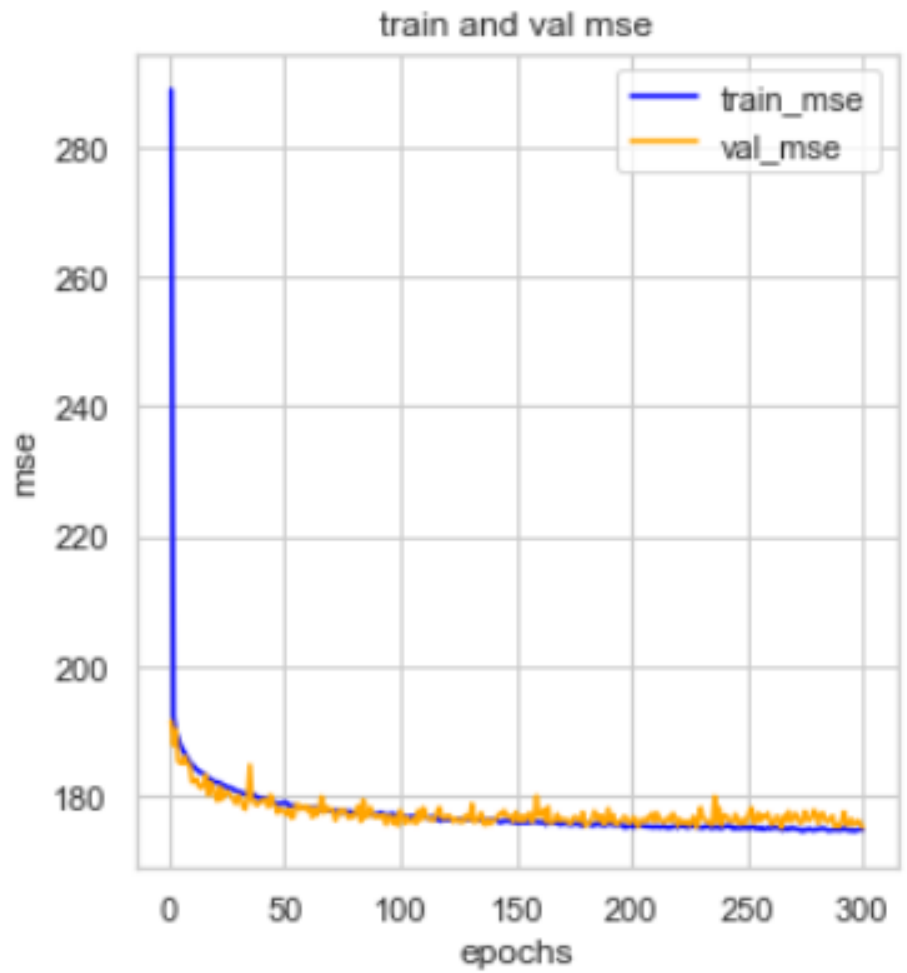
```
# 모델 학습하기
history222 = model.fit(x_train,y_train,
                       batch_size=128,
                       epochs = 300,
                       validation_data = (x_val, y_val))
```

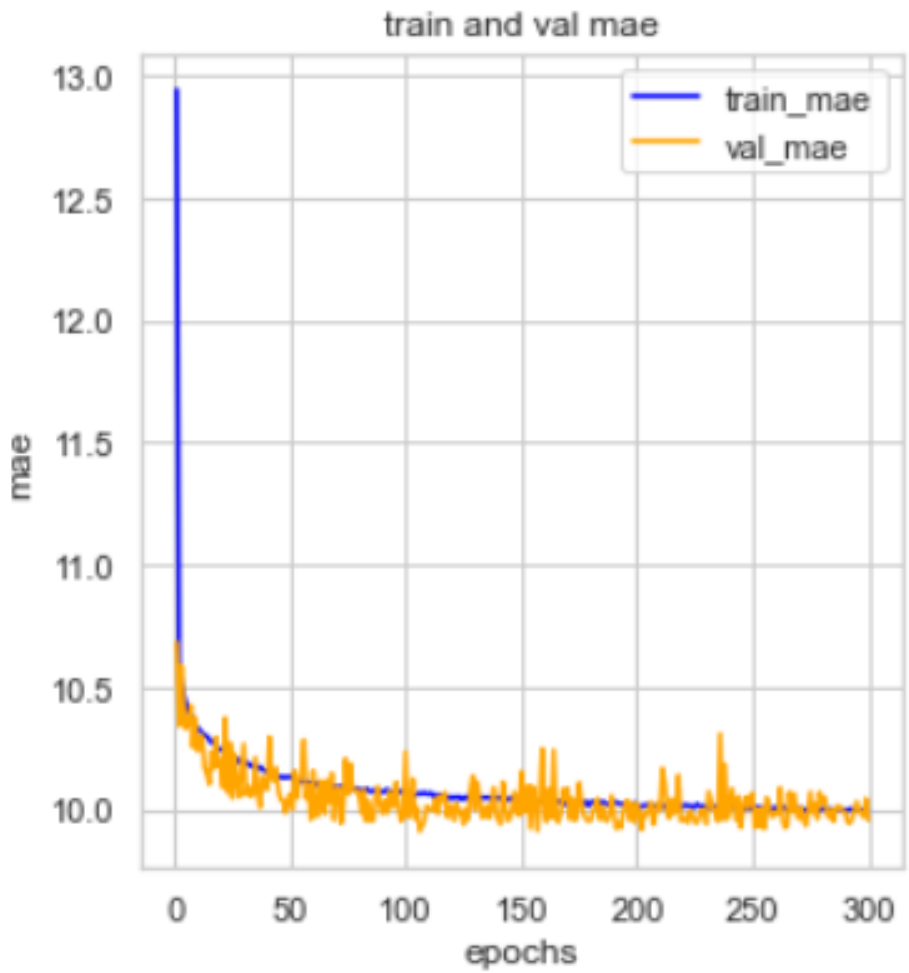
✓ 2109.7s

3. ML

Evaluation

7. Neural Network (Batch normalization)

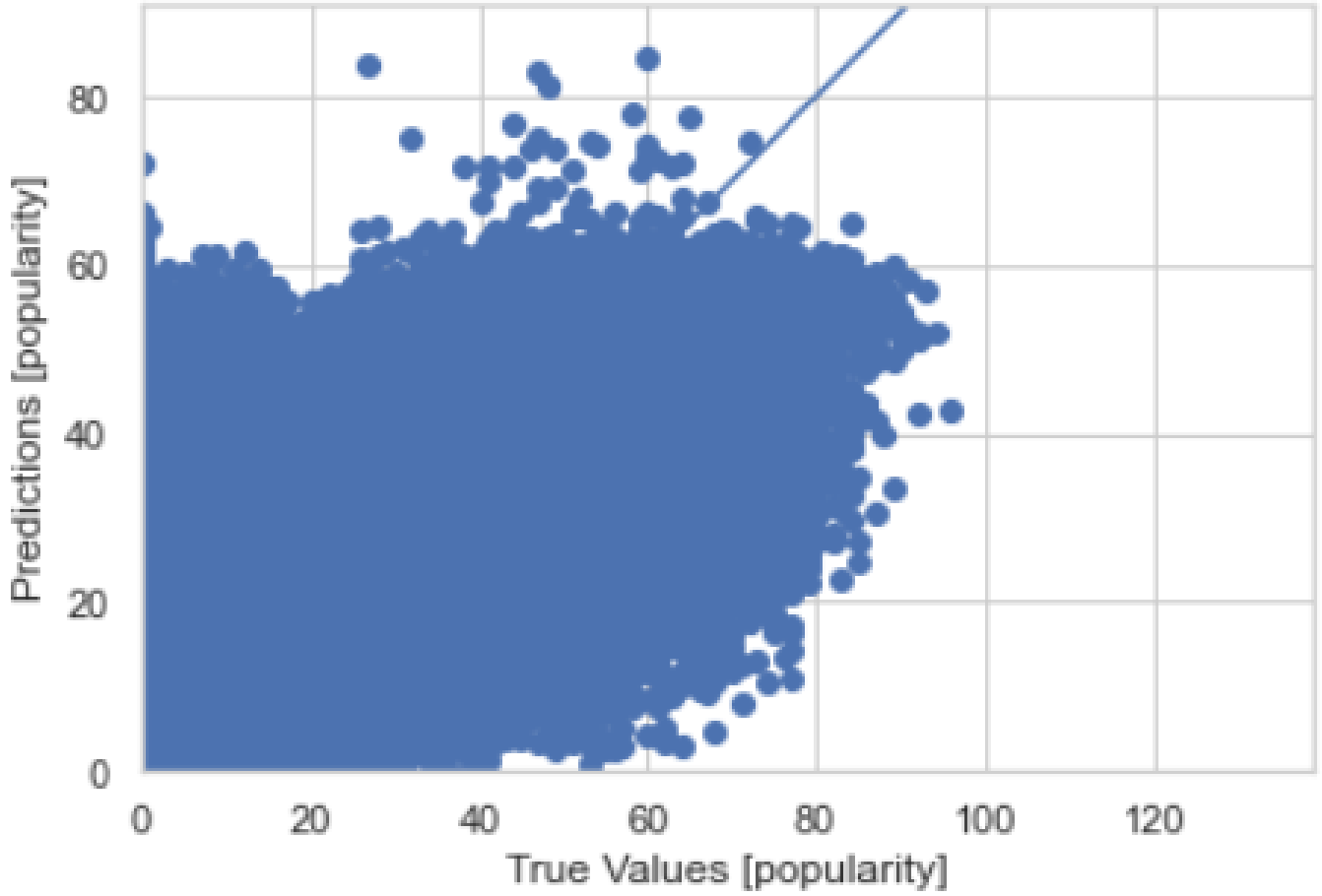




3. ML

Evaluation

7. Neural Network (Batch normalization)



MSE : 175.61

3. Outro

Conclusion & Feedback

1. API로 주어진 feature들 이외의 요소들 또한 스포티파이 추천 알고리즘의 key point로 보인다.

   (장르 분석, 텍스트 분석, 협업 필터링… etc)

2. 신경망의 학습 설계 방법은 무궁무진!

3. 신경망에서도 피할 수 없는 Overffiting… 여러 대처법 필요!

4. Batch_size를 줄이고 여유있게 학습했다면 더 좋은 결과가 나오지 않았을까.

5. GPU만세!!

4. Outro

Reference

- **Dataset : https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks**
- https://muzukphysics.tistory.com/entry/DL-5-딥러닝-Overfitting-방지-방법-오버피팅-과적합 [물리학과 직장인]

Thank you