# 기본 패키지 및 데이터 정보 로드

```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")


import matplotlib.pyplot as plt
import missingno # 널값 바차트로 시각화
```

```
netflix = pd.read_csv('./use_df.csv')
```

```
netflix.head(1)
```

| | Unnamed: 0 | Title | Series or Movie | COUNTRY | Release Date | GENRE | VALUE | Genre_all | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | thequeensgambit | TV | United States | 2020-10-23 | Drama | 44867 | Biography, Drama, History | Dra<br>Sl<br>B |

```
netflix.columns
```

```
Index(['Unnamed: 0', 'Title', 'Series or Movie', 'COUNTRY', 'Release Date',
       'GENRE', 'VALUE', 'Genre_all', 'Tags', 'Languages', 'Hidden Gem Score',
       'Country Availability', 'Runtime', 'Director', 'Writer', 'Actors',
       'View Rating', 'IMDb Score', 'Rotten Tomatoes Score',
       'Metacritic Score', 'Awards Received', 'Awards Nominated For',
       'Boxoffice', 'Netflix Release Date', 'Production House', 'Summary',
       'IMDb Votes'],
      dtype='object')
```

```
df_net = netflix.drop(columns = ['Unnamed: 0'],axis =1)
```

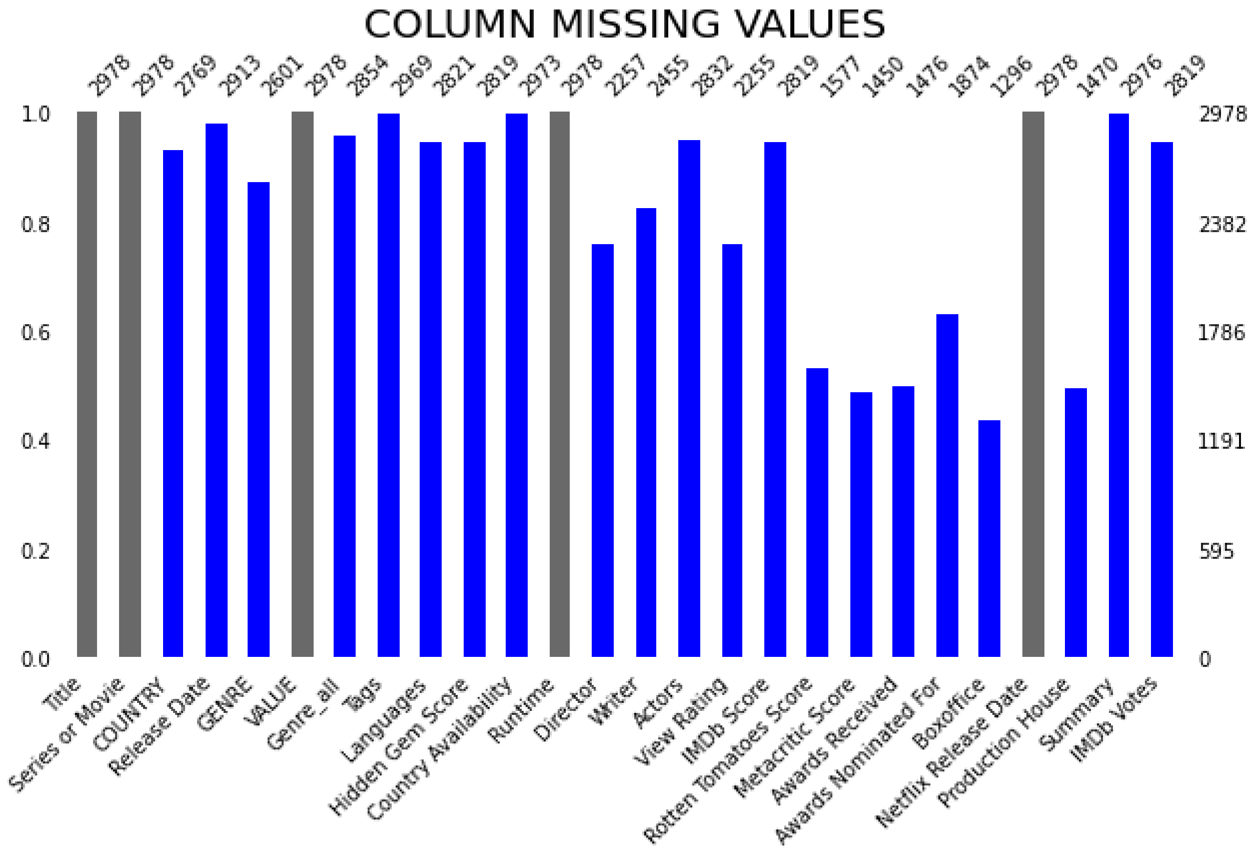
```
df_net.isnull().sum()
```

```
Title                     0
Series or Movie           0
COUNTRY                 209
Release Date             65
GENRE                   377
VALUE                     0
Genre_all               124
Tags                      9
```

```
Languages                  157
Hidden Gem Score           159
Country Availability         5
Runtime                      0
Director                   721
Writer                     523
Actors                     146
View Rating                723
IMDb Score                 159
Rotten Tomatoes Score     1401
Metacritic Score          1528
Awards Received           1502
Awards Nominated For      1104
Boxoffice                 1682
Netflix Release Date         0
Production House          1508
Summary                      2
IMDb Votes                 159
dtype: int64
```

```
color= ['dimgrey','dimgrey','blue','blue','blue','dimgrey','blue','blue','blue','blue','blue',
        'dimgrey','blue','blue','blue','blue','blue','blue','blue','blue','blue','blue','dimgrey
        'blue','blue','blue']
missingno.bar(df_net,fontsize=10,color=color,figsize=(10,5))
plt.title('COLUMN MISSING VALUES',fontsize=20)
```

```
Text(0.5, 1.0, 'COLUMN MISSING VALUES')
```



## 인코딩

viewpoint 예측에 사용될 변수들을 일단은 타입, 생산 국가, 상영 국가와 장르, 태그, 출시 일, 런타임, 연령, imdb 투표수와 점수, 로튼토마토의 숨겨진 명작 점수들을 가지고 먼저 예측초기모델을 만들어보자

## null값 처리

```
df = df_net[['Series or Movie', 'COUNTRY', 'Country Availability', 'Hidden Gem Score',
             'Release Date', 'GENRE','Tags',
             'View Rating', 'Runtime','VALUE','IMDb Score', 'IMDb Votes']]
```

```
df.isnull().sum()
```

```
Series or Movie           0
COUNTRY                 209
Country Availability      5
Hidden Gem Score        159
Release Date             65
GENRE                   377
Tags                      9
View Rating             723
Runtime                   0
VALUE                     0
IMDb Score              159
IMDb Votes              159
dtype: int64
```

```
df['Release Date'] = df['Release Date'].fillna(0)  # 날짜 데이터는 null값을 0으로 채움
```

```
df["Release Date"] = pd.to_datetime(df['Release Date'])
df['month'] = df['Release Date'].dt.month
df = df.drop(columns = ['Release Date'],axis =1)
```

```
df['Series or Movie'] = df['Series or Movie'].replace({'Movie': 1, 'TV': 2})  # type은 1,2로 인
```

```
df['Series or Movie'].value_counts() # 영화 데이터가 4배정도 많은 걸로 나타남
```

```
1    2293
2     685
Name: Series or Movie, dtype: int64
```

```
df['COUNTRY'] = df['COUNTRY'].fillna('NA')  # 국가의 널값은 NA로 대체
```

```
df['GENRE'] = df['GENRE'].fillna('NA') # 장르의 널값도 NA로 대체
```

```
rows = len(df['Tags'])  # 태그는 태그 수를 사용
df['Tag_count'] = 0
for i in range(rows):
    df.Tag_count[i] = len(str(df.Tags.iloc[i]).split(','))
df = df.drop(['Tags'], axis = 1)
```

```
rows = len(df['Country Availability'])  # 상영국가는 상영국가 수를 사용
df['C_count'] = 0
for i in range(rows):
    df.C_count[i] = len(str(df['Country Availability'].iloc[i]).split(','))
df = df.drop(['Country Availability'], axis = 1)
```

```
for i in range(rows):
    if df['View Rating'][i] in ['PG', 'TV-PG', 'TV-G', 'TV-Y7', 'TV-Y', 'G', 'Unrated', 'TV-Y7-F
        df['View Rating'][i] = 1  # 어린이 컨텐츠 = 1
    elif df['View Rating'][i] in ['PG-13', 'TV-14', 'R', 'GP', 'Passed', 'X']:
        df['View Rating'][i] = 2  # 청소년 컨텐츠 = 2
    elif df['View Rating'][i] in ['TV-MA', 'Not Rated','NC-17']:
        df['View Rating'][i] = 3  # 성인컨텐츠 = 3
    else:
        df['View Rating'][i] = 0 # null값을 0으로 처리

```

```
df['View Rating'] = df['View Rating'].astype(int)
```

```
df['Runtime'] = df['Runtime'].replace({'< 30 minutes' : 0,
                                       '30-60 mins' : 1,
                                       '1-2 hour' : 2,
                                       '> 2 hrs' : 3 })
```

```
df['Runtime'] = df['Runtime'].astype(int)
```

```
df['Runtime'].value_counts()
```

```
2    1716
0     740
3     506
1      16
Name: Runtime, dtype: int64
```

```
df.isnull().sum()
```

```
Series or Movie       0
COUNTRY               0
Hidden Gem Score    159
GENRE                 0
View Rating           0
Runtime               0
VALUE                 0
IMDb Score          159
IMDb Votes          159
month                 0
Tag_count             0
C_count               0
dtype: int64
```

```
df = df.dropna(how='any',axis=0)  # hidden gem score, imdb score, imdb vote의 null값이 같으므로
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2819 entries, 0 to 2977
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Series or Movie   2819 non-null   int64  
 1   COUNTRY           2819 non-null   object 
 2   Hidden Gem Score  2819 non-null   float64
 3   GENRE             2819 non-null   object 
 4   View Rating       2819 non-null   int64  
 5   Runtime           2819 non-null   int64  
 6   VALUE             2819 non-null   int64  
 7   IMDb Score        2819 non-null   float64
 8   IMDb Votes        2819 non-null   float64
 9   month             2819 non-null   int64  
 10  Tag_count         2819 non-null   int64  
 11  C_count           2819 non-null   int64  
dtypes: float64(3), int64(7), object(2)
memory usage: 286.3+ KB
```

## 장르와 제작국가는 원핫인코딩으로 인코딩

```
pd_df = pd.get_dummies(df[['COUNTRY','GENRE']])
df_result = pd.concat([df, pd_df], axis=1)
```

```
df_result = df_result.drop(['COUNTRY','GENRE'], axis = 1)
```

```
df_taget = df_result['VALUE']
df_result = df_result.drop(['VALUE'], axis = 1)
```

```
df_result.head(1)
```

|   | Series or Movie | Hidden Gem Score | View Rating | Runtime | IMDb Score | IMDb Votes | month | Tag_count | C_count | COUN |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2 | 4.3 | 2 | 0 | 7.3 | 104495.0 | 10 | 4 | 36 |  |

1 rows × 99 columns

```
df_taget.head(1)
```

```
0    44867
Name: VALUE, dtype: int64
```

# 딥러닝을 통해 모든 변수들을 넣고 예측

```
from sklearn.preprocessing import StandardScaler
```

```
X = df_result.iloc[:].values
y = df_taget.iloc[:].values
```

```
y = y.reshape(-1,1)
```

```
y
```

```
array([[44867],
       [42149],
       [27138],
       ...,
       [    1],
       [    1],
       [    1]])
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X)
y_train = sc.fit_transform(y)
```

```

```

```
1.0
```

```
from keras import models
from keras import layers
```

```
from keras.layers.core import Dropout
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(150, activation='relu',
                          input_shape=(X_train.shape[1],)))
    model.add(layers.Dense(280, activation='relu'))
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(99, activation='relu'))
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(24, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

```
X_train.shape
```

```
(2819, 99)
```

```
import numpy as np

k = 4
all_mae_histories = []

num_val_samples = len(X_train) // k
num_epochs = 300
all_scores = []
for i in range(k):
    print('처리중인 폴드 #', i)
    val_data = X_train[i * num_val_samples: (i + 1) * num_val_samples]  # 검증 데이터 준비: k번
    val_targets = y_train[i * num_val_samples: (i + 1) * num_val_samples]

    partial_train_data = np.concatenate(  # 훈련 데이터 준비: 다른 분할 전체
        [X_train[:i * num_val_samples],
         X_train[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [y_train[:i * num_val_samples],
         y_train[(i + 1) * num_val_samples:]],
        axis=0)

model = build_model()  # 케라스 모델 구성(컴파일 포함)
history = model.fit(partial_train_data, partial_train_targets,  # 모델 훈련(verbose=0이므로 훈련
                    validation_data=(val_data, val_targets),
                    epochs=num_epochs, batch_size=1, verbose=True)
mae_history = history.history['val_mae']
all_mae_histories.append(mae_history)

val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)  # 검증 세트로 모델 평가
all_scores.append(val_mae)
```

```
Epoch 272/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.8857 - mae: 0.3238 - va
Epoch 273/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.6399 - mae: 0.2937 - va
Epoch 274/300
2115/2115 [==============================] - 7s 3ms/step - loss: 1.3532 - mae: 0.3559 - va
Epoch 275/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.6957 - mae: 0.3204 - va
Epoch 276/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.6180 - mae: 0.3007 - va
Epoch 277/300
2115/2115 [==============================] - 7s 3ms/step - loss: 1.2328 - mae: 0.3276 - va
Epoch 278/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.8611 - mae: 0.3388 - va
Epoch 279/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.8876 - mae: 0.3368 - va
Epoch 280/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.8779 - mae: 0.3361 - va
Epoch 281/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.9594 - mae: 0.3322 - va
Epoch 282/300
2115/2115 [==============================] - 7s 3ms/step - loss: 1.2147 - mae: 0.3516 - va
Epoch 283/300
```

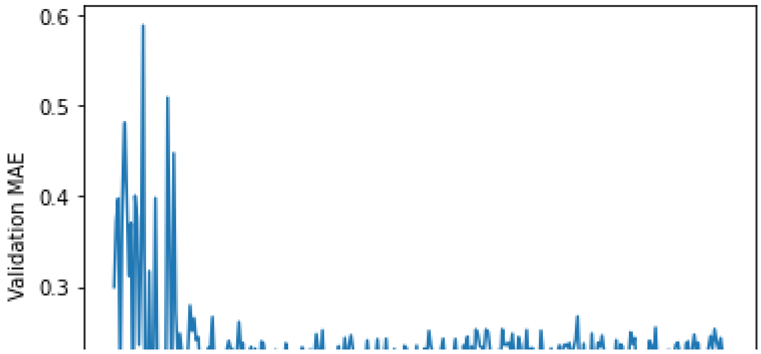
```
2115/2115 [==============================] - 7s 3ms/step - loss: 0.9742 - mae: 0.3441 - va
Epoch 284/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.7248 - mae: 0.2978 - va
Epoch 285/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.6148 - mae: 0.2972 - va
Epoch 286/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.9927 - mae: 0.3530 - va
Epoch 287/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.9435 - mae: 0.3458 - va
Epoch 288/300
2115/2115 [==============================] - 7s 3ms/step - loss: 1.1736 - mae: 0.3512 - va
Epoch 289/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.8567 - mae: 0.3225 - va
Epoch 290/300
2115/2115 [==============================] - 7s 3ms/step - loss: 1.1947 - mae: 0.3527 - va
Epoch 291/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.9071 - mae: 0.3460 - va
Epoch 292/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.7951 - mae: 0.3181 - va
Epoch 293/300
2115/2115 [==============================] - 7s 3ms/step - loss: 1.3460 - mae: 0.3475 - va
Epoch 294/300
2115/2115 [==============================] - 7s 3ms/step - loss: 1.0178 - mae: 0.3460 - va
Epoch 295/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.6361 - mae: 0.3008 - va
Epoch 296/300
2115/2115 [==============================] - 7s 3ms/step - loss: 1.1702 - mae: 0.3416 - va
Epoch 297/300
2115/2115 [==============================] - 7s 3ms/step - loss: 1.1392 - mae: 0.3387 - va
Epoch 298/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.7437 - mae: 0.3269 - va
Epoch 299/300
2115/2115 [==============================] - 7s 3ms/step - loss: 1.2100 - mae: 0.3618 - va
Epoch 300/300
2115/2115 [==============================] - 7s 3ms/step - loss: 0.8869 - mae: 0.3297 - va
```

```
all_scores
```

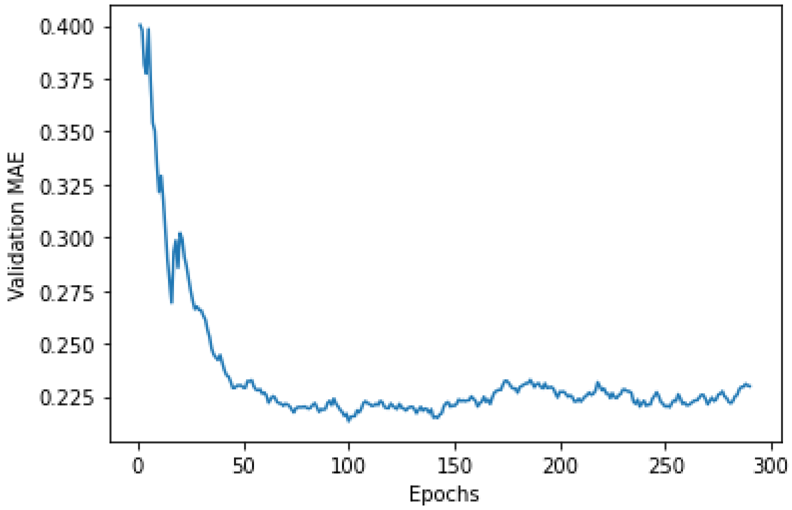
```
[0.22708107531070711]
```

```
average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

```
import matplotlib.pyplot as plt

plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



```
def smooth_curve(points, factor=0.9):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point * (1 - factor))
        else:
            smoothed_points.append(point)
    return smoothed_points

smooth_mae_history = smooth_curve(average_mae_history[10:])

plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



**에포크 300으로 딥러닝을 진행했을 때 오차가 0.227, 227정도로 viewpoint가 1~40000인것을 감안하면 오차가 좀 큰걸로 나옴**

- all_scores = [0.2270810753107071]

```
num_epochs = 150
all_mae_histories = []
for i in range(k):
    print('처리중인 폴드 #', i)
    val_data = X_train[i * num_val_samples: (i + 1) * num_val_samples]  #검증 데이터 준비: k번째
    partial_train_data = np.concatenate(  # 훈련 데이터 준비: 다른 분할 전체
        [X_train[:i * num_val_samples],
        [X_train[:i * num_val_samples],
         X_train[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [y_train[:i * num_val_samples],
         y_train[(i + 1) * num_val_samples:]],
        axis=0)

model = build_model()  # 케라스 모델 구성(컴파일 포함)
history = model.fit(partial_train_data, partial_train_targets,  # 모델 훈련(verbose=0이므로 훈련
                    validation_data=(val_data, val_targets),
                    epochs=num_epochs, batch_size=1, verbose=True)
mae_history = history.history['val_mae']
all_mae_histories.append(mae_history)
```

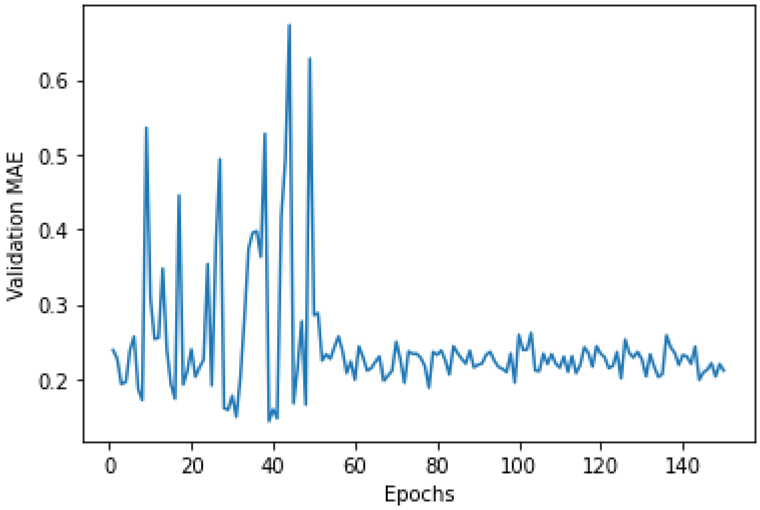
```
Epoch 122/150
2115/2115 [==============================] - 7s 3ms/step - loss: 0.9277 - mae: 0.3688 - va
Epoch 123/150
2115/2115 [==============================] - 7s 3ms/step - loss: 0.9781 - mae: 0.3760 - va
Epoch 124/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.7292 - mae: 0.3314 - va
Epoch 125/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.9285 - mae: 0.3616 - va
Epoch 126/150
2115/2115 [==============================] - 6s 3ms/step - loss: 1.6072 - mae: 0.3402 - va
Epoch 127/150
2115/2115 [==============================] - 7s 3ms/step - loss: 1.4358 - mae: 0.3939 - va
Epoch 128/150
2115/2115 [==============================] - 7s 3ms/step - loss: 1.0121 - mae: 0.3505 - va
Epoch 129/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.8880 - mae: 0.3620 - va
Epoch 130/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.6747 - mae: 0.3347 - va
Epoch 131/150
2115/2115 [==============================] - 7s 3ms/step - loss: 1.0936 - mae: 0.3781 - va
Epoch 132/150
2115/2115 [==============================] - 7s 3ms/step - loss: 0.9382 - mae: 0.3562 - va
Epoch 133/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.8323 - mae: 0.3542 - va
Epoch 134/150
2115/2115 [==============================] - 6s 3ms/step - loss: 1.0838 - mae: 0.3635 - va
Epoch 135/150
2115/2115 [==============================] - 6s 3ms/step - loss: 1.0026 - mae: 0.3694 - va
Epoch 136/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.6345 - mae: 0.3146 - va
Epoch 137/150
2115/2115 [==============================] - 7s 3ms/step - loss: 0.6693 - mae: 0.3327 - va
Epoch 138/150
2115/2115 [==============================] - 7s 3ms/step - loss: 0.9233 - mae: 0.3238 - va
Epoch 139/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.9082 - mae: 0.3482 - va
Epoch 140/150
2115/2115 [==============================] - 7s 3ms/step - loss: 0.6979 - mae: 0.3211 - va
Epoch 141/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.7273 - mae: 0.3190 - va
Epoch 142/150
2115/2115 [==============================] - 7s 3ms/step - loss: 0.8994 - mae: 0.3578 - va
Epoch 143/150

2115/2115 [==============================] - 6s 3ms/step - loss: 0.7782 - mae: 0.3352 - va
Epoch 144/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.8608 - mae: 0.3532 - va
```

```
Epoch 145/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.9644 - mae: 0.3453 - va
Epoch 146/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.8177 - mae: 0.3376 - va
Epoch 147/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.9545 - mae: 0.3398 - va
Epoch 148/150
2115/2115 [==============================] - 6s 3ms/step - loss: 0.9691 - mae: 0.3646 - va
Epoch 149/150
2115/2115 [==============================] - 7s 3ms/step - loss: 0.9609 - mae: 0.3440 - va
Epoch 150/150
2115/2115 [==============================] - 6s 3ms/step - loss: 1.0555 - mae: 0.3604 - va
```

```
mae_history = history.history['val_mae']
all_mae_histories.append(mae_history)
```

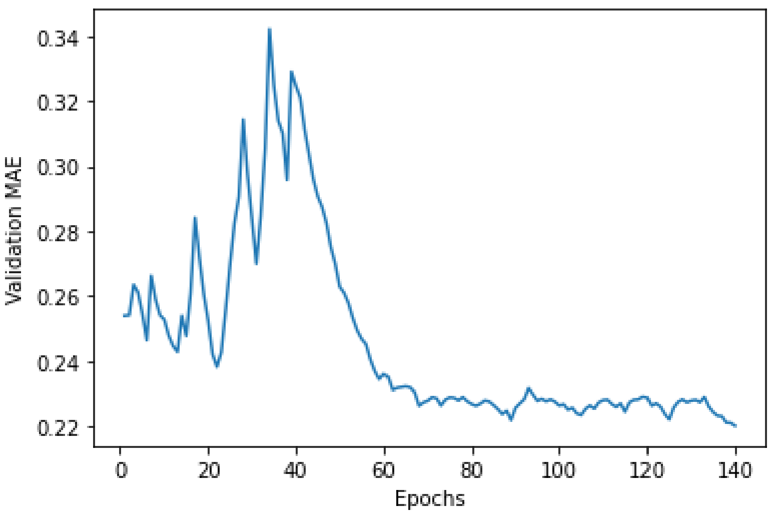
```
average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

```
import matplotlib.pyplot as plt

plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



```
def smooth_curve(points, factor=0.9):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point * (1 - factor))
        else:
            smoothed_points.append(point)
    return smoothed_points

smooth_mae_history = smooth_curve(average_mae_history[10:])
```

```
plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



에포크 90정도 지점부터 오차가 줄어들지 않는것으로 보인다. 이 지점이 과대적합 지점이라고 생각하고 새로운 모델을 만들어서 테스트 해보았다.

```
model = build_model()  # 새롭게 컴파일된 모델을 얻습니다.
model.fit(X_train, y_train,  # 전체 데이터로 훈련시킵니다.
          epochs=90, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(X_train, y_train)
```

```
89/89 [==============================] - 0s 2ms/step - loss: 0.2012 - mae: 0.1685
```

```
# batch_size = 1
# test_mse_score, test_mae_score = model.evaluate(X_train, y_train)
# 89/89 [==============================] - 0s 2ms/step - loss: 1.0165 - mae: 0.3206
# test_mae_score = 0.32060846686363322
```

```
89/89 [==============================] - 0s 2ms/step - loss: 1.0165 - mae: 0.3206
```

```
test_mae_score
```

```
0.1685454249382019
```

## 최종 결과

- batch_size = 16 , epochs = 90
- 89/89 [==============================] - 0s 2ms/step - loss: 0.2012 - mae: 0.1685
- test_mae_score = 0.1685454249382019

- 최종 결과 epochs 90지점에서 오차가 0.169정도로 나왔고, 최종 오차는 169정도 나는걸로 보인다.
- 데이터셋이 부족하여 오차를 더 줄이기는 어려워 보인다.
- 추후에 데이터셋을 더 추가하고, 딥러닝 코드를 손을 보면 더 좋은 예측모델이 나올것으로 보인다.

```
# pd_df = pd.get_dummies(df_net['Tags'])
# df_result = pd.concat([df_net, pd_df], axis=1)
```

✓ 0초 오후 3:53에 완료됨 ● ×