In [1]:
```python
# 데이터 분석 준비
import pandas as pd
import numpy as np
```

데이터 로딩 및 확인

In [2]:
```python
# 데이터 로딩
EPL_df = pd.read_csv('./datasets/results_00_19.csv', encoding='cp949')
EPL_test = pd.read_csv('./datasets/test20_22.csv', encoding='cp949')
```

In [3]:
```python
print(EPL_df.columns.values)
```

```
['Season' 'DateTime' 'HomeTeam' 'AwayTeam' 'FTHG' 'FTAG' 'FTR' 'HTHG'
 'HTAG' 'HTR' 'Referee' 'HS' 'AS' 'HST' 'AST' 'HC' 'AC' 'HF' 'AF' 'HY'
 'AY' 'HR' 'AR']
```

In [4]:
```python
# preview the data
EPL_df.head()
```

Out[4]:

	Season	DateTime	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	...	HST	AST	HC	AC	HF	AF	HY	AY	HR
0	Jan.00	2000-08-19T00:00:00Z	Charlton	Man City	4	0	H	2	0	H	...	14	4	6	6	13	12	1	2	0
1	Jan.00	2000-08-19T00:00:00Z	Chelsea	West Ham	4	2	H	1	0	H	...	10	5	7	7	19	14	1	2	0
2	Jan.00	2000-08-19T00:00:00Z	Coventry	Middlesbrough	1	3	A	1	1	D	...	3	9	8	4	15	21	5	3	1
3	Jan.00	2000-08-19T00:00:00Z	Derby	Southampton	2	2	D	1	2	A	...	4	6	5	8	11	13	1	1	0
4	Jan.00	2000-08-19T00:00:00Z	Leeds	Everton	2	0	H	2	0	H	...	8	6	6	4	21	20	1	3	0

5 rows × 23 columns

In [5]:
```python
EPL_df.info() # null 값 존재 x
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7600 entries, 0 to 7599
Data columns (total 23 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Season    7600 non-null   object
 1   DateTime  7600 non-null   object
 2   HomeTeam  7600 non-null   object
 3   AwayTeam  7600 non-null   object
 4   FTHG      7600 non-null   int64 
 5   FTAG      7600 non-null   int64 
 6   FTR       7600 non-null   object
 7   HTHG      7600 non-null   int64 
 8   HTAG      7600 non-null   int64 
 9   HTR       7600 non-null   object
 10  Referee   7600 non-null   object
 11  HS        7600 non-null   int64 
 12  AS        7600 non-null   int64 
 13  HST       7600 non-null   int64 
 14  AST       7600 non-null   int64 
 15  HC        7600 non-null   int64 
 16  AC        7600 non-null   int64 
 17  HF        7600 non-null   int64 
 18  AF        7600 non-null   int64 
 19  HY        7600 non-null   int64 
 20  AY        7600 non-null   int64 
 21  HR        7600 non-null   int64 
 22  AR        7600 non-null   int64 
dtypes: int64(16), object(7)
memory usage: 1.3+ MB
```

In [6]:
```python
EPL_df.describe()
```

Out[6]:

	FTHG	FTAG	HTHG	HTAG	HS	AS	HST	AST	HC	AC	
count	7600.000000	7600.000000	7600.000000	7600.000000	7600.000000	7600.000000	7600.000000	7600.000000	7600.000000	7600.000000	7
mean	1.528947	1.140132	0.682500	0.498947	13.526316	10.566842	6.255395	4.828158	6.129211	4.787632	
std	1.296113	1.131548	0.831894	0.714742	5.230781	4.529502	3.351286	2.815406	3.100482	2.730690	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	0.000000	0.000000	0.000000	10.000000	7.000000	4.000000	3.000000	4.000000	3.000000	
50%	1.000000	1.000000	0.000000	0.000000	13.000000	10.000000	6.000000	4.000000	6.000000	4.000000	
75%	2.000000	2.000000	1.000000	1.000000	17.000000	13.000000	8.000000	6.000000	8.000000	6.000000	
max	9.000000	9.000000	5.000000	5.000000	43.000000	30.000000	24.000000	20.000000	20.000000	19.000000	

Missing Value 처리

In [7]:

```
# check missing values in train dataset
EPL_df.isnull().sum()
```

Out[7]:

```
Season      0
DateTime    0
HomeTeam    0
AwayTeam    0
FTHG        0
FTAG        0
FTR         0
HTHG        0
HTAG        0
HTR         0
Referee     0
HS          0
AS          0
HST         0
AST         0
HC          0
AC          0
HF          0
AF          0
HY          0
AY          0
HR          0
AR          0
dtype: int64
```

In [8]:

```
EPL_df[EPL_df.DateTime.isnull()] # 결측이 있는 행을 찾음
```

Out[8]:

Season	DateTime	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	...	HST	AST	HC	AC	HF	AF	HY	AY	HR	AR

0 rows × 23 columns

In [9]:

```
EPL_df = EPL_df.dropna() # 레코드의 대부분이 결측이므로 레코드를 삭제
```

In [10]:

```
# 시각화 패키지
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

경기 결과와 경기 기록 비교

In [11]:

```
# 홈팀 어웨이팀으로 나뉘어있는 feature를 분석하기 좋게 하나의 새로운 feature로 표현
EPL_df['DIFF_FG'] = EPL_df['FTHG'] - EPL_df['FTAG'] # 홈팀 풀타임 골 - 어웨이팀 풀타임 골
EPL_df['DIFF_HG'] = EPL_df['HTHG'] - EPL_df['HTAG'] # 홈팀 하프타임 골 - 어웨이팀 하프타임 골
EPL_df['DIFF_SHOOT'] = EPL_df['HS'] - EPL_df['AS'] # 홈팀 슈팅 수 - 어웨이팀 슈팅 수
EPL_df['DIFF_ST'] = EPL_df['HST'] - EPL_df['AST'] # 홈팀 유효슈팅 수 - 어웨이팀 유효슈팅 수
EPL_df['DIFF_FOUL'] = EPL_df['HF'] - EPL_df['AF'] # 홈팀 파울 - 어웨이팀 파울
EPL_df['DIFF_CONER'] = EPL_df['HC'] - EPL_df['AC']# 홈팀 코너킥 - 어웨팀 코너킥
EPL_df['DIFF_YC'] = EPL_df['HY'] - EPL_df['AY'] # 홈팀 옐로카드 - 어웨이팀 옐로카드
EPL_df['DIFF_RC'] = EPL_df['HR'] - EPL_df['AR'] # 홈팀 레드카드 - 어웨이팀 레드카드
```

In [12]:

```
list_dif = ["FTR", "DIFF_FG", "DIFF_HG", "DIFF_SHOOT","DIFF_ST","DIFF_FOUL","DIFF_CONER","DIFF_YC","DIFF_RC"]
EPL_df[list_dif].groupby(['FTR'], as_index=False).mean().sort_values(by='FTR', ascending=False)
```

Out[12]:

	FTR	DIFF_FG	DIFF_HG	DIFF_SHOOT	DIFF_ST	DIFF_FOUL	DIFF_CONER	DIFF_YC	DIFF_RC
2	H	1.884387	0.836498	5.034287	3.215642	-0.746387	1.491924	-0.516860	-0.085577
1	D	0.000000	0.006792	2.684431	1.080982	-0.635841	1.549634	-0.377743	-0.019331
0	A	-1.713027	-0.727863	-0.191006	-1.191470	-0.214650	0.910987	-0.075568	0.060732

데이터 전처리: 속성 조정

In [13]:
```python
# train / test split
from sklearn.model_selection import train_test_split
```

In [14]:
```python
# 필요한 feature만 뽑아 새로운 df에 저장
# 풀타임 결과(H, D, R), 슈팅수, 유효슈팅수, 파울, 코너킥, 옐로카드, 레드카드
EPL_df_ext = EPL_df[['FTR', 'HS', 'AS', 'HST', 'AST', 'HF', 'AF',
                     'HC', 'AC', 'HY', 'AY', 'HR', 'AR']]

EPL_test_ext = EPL_test[['FTR', 'HS', 'AS', 'HST', 'AST', 'HF', 'AF',
                     'HC', 'AC', 'HY', 'AY', 'HR', 'AR']]
print(EPL_df_ext)
```

```
     FTR  HS  AS  HST  AST  HF  AF  HC  AC  HY  AY  HR  AR
0      H  17   8   14    4  13  12   6   6   1   2   0   0
1      H  17  12   10    5  19  14   7   7   1   2   0   0
2      A   6  16    3    9  15  21   8   4   5   3   1   0
3      D   6  13    4    6  11  13   5   8   1   1   0   0
4      H  17  12    8    6  21  20   6   4   1   3   0   0
...   ..  ..  ..  ...  ...  ..  ..  ..  ..  ..  ..  ..  ..
7595   A  14   7    3    3  12  11   3   3   1   4   1   0
7596   H  31   5   10    4   7   4   9   0   1   1   0   0
7597   A   3  14    2    6  11   5   2   4   1   0   0   0
7598   H  13   5    4    3   9  16   9   1   0   1   0   0
7599   D  10  13    1    4  16  13   0   7   2   1   0   0

[7600 rows x 13 columns]
```

In [15]:
```python
# FTR 무승부 지우고 binary 형태로
# 무승부 경기를 제외하고 홈팀이 이긴 것을 RESULT의 1로
EPL_df_ext = EPL_df_ext[EPL_df_ext.FTR != 'D']
EPL_df_ext['RESULT'] = np.where(EPL_df_ext['FTR']=='H', 1, 0)
EPL_df_ext.drop('FTR', axis=1, inplace=True)
print(EPL_df_ext)

EPL_test_ext = EPL_test_ext[EPL_test_ext.FTR != 'D']
EPL_test_ext['RESULT'] = np.where(EPL_test_ext['FTR']=='H', 1, 0)
EPL_test_ext.drop('FTR', axis=1, inplace=True)
print(EPL_test_ext)
```

```
      HS  AS  HST  AST  HF  AF  HC  AC  HY  AY  HR  AR  RESULT
0     17   8   14    4  13  12   6   6   1   2   0   0       1
1     17  12   10    5  19  14   7   7   1   2   0   0       1
2      6  16    3    9  15  21   8   4   5   3   1   0       0
4     17  12    8    6  21  20   6   4   1   3   0   0       1
6     16   3   10    2   8   8   6   1   1   1   0   0       1
...   ..  ..  ...  ...  ..  ..  ..  ..  ..  ..  ..  ..     ...
7594  13  13    5    7  11   9   2   5   1   0   0   0       0
7595  14   7    3    3  12  11   3   3   1   4   1   0       0
7596  31   5   10    4   7   4   9   0   1   1   0   0       1
7597   3  14    2    6  11   5   2   4   1   0   0   0       0
7598  13   5    4    3   9  16   9   1   0   1   0   0       1

[5686 rows x 13 columns]
     HS  AS  HST  AST  HF  AF  HC  AC  HY  AY  HR  AR  RESULT
0     5  13    2    6  12  12   2   3   2   2   0   0       0
1     5   9    3    5  14  11   7   3   2   1   0   0       1
2    22   6    6    3   9   6   9   0   1   0   0   0       1
3    15  15    3    2  13   7   8   7   2   2   0   0       0
4     7  13    1    7  12   9   2   5   1   1   0   0       0
..   ..  ..  ...  ...  ..  ..  ..  ..  ..  ..  ..  ..     ...
683   8  10    1    5   8  12   3   8   2   0   0   0       0
684   9  11    8    5  12  14   9   3   2   3   0   0       0
685  15   5    7    1   2   6   4   6   0   1   0   0       1
686  12  11    3    3  11  12   3   4   1   1   0   0       1
687  17  18    6    4  12  10   6   7   1   1   0   0       1

[534 rows x 13 columns]
```

In [16]:
```python
# 데이터 내에서 학습 집합과 테스트 집합을 나눔(8:2)
# train, test = train_test_split(EPL_df_ext, test_size=0.2, random_state=12)
train = EPL_df_ext
test = EPL_test_ext

X_train = train.drop("RESULT", axis=1) # 열 지움
Y_train = train["RESULT"]
X_test = test.drop("RESULT", axis=1)
Y_test = test["RESULT"]
X_train.shape, Y_train.shape, X_test.shape, Y_test.shape
print(X_train)
```

```
      HS  AS  HST  AST  HF  AF  HC  AC  HY  AY  HR  AR
0     17   8   14    4  13  12   6   6   1   2   0   0
1     17  12   10    5  19  14   7   7   1   2   0   0
2      6  16    3    9  15  21   8   4   5   3   1   0
4     17  12    8    6  21  20   6   4   1   3   0   0
6     16   3   10    2   8   8   6   1   1   1   0   0
...   ..  ..  ...  ...  ..  ..  ..  ..  ..  ..  ..  ..
7594  13  13    5    7  11   9   2   5   1   0   0   0
7595  14   7    3    3  12  11   3   3   1   4   1   0
7596  31   5   10    4   7   4   9   0   1   1   0   0
7597   3  14    2    6  11   5   2   4   1   0   0   0
7598  13   5    4    3   9  16   9   1   0   1   0   0

[5686 rows x 12 columns]
```

기계학습

In [17]:

```
# 기계 학습
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB  # 이산형이 아닌 연속형에 대한
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.cluster import KMeans
```

Random Forest

In [18]:

```
from sklearn.ensemble import RandomForestClassifier

# n_estimators: 결정 트리 개수(클수록 수행 시간 증가)
rf = RandomForestClassifier(n_estimators = 100, oob_score = True, random_state=123456)

rf.fit(X_train, Y_train)

from sklearn.metrics import accuracy_score

pred_rf = rf.predict(X_test)

# oob_score = out of bag score 예측이 얼마나 정확한지에 대한 추정치
# print(f'Out of bag score estimate: {rf.oob_score_:.9}')
print(f'accuracy: {accuracy_score(Y_test,pred_rf): .4f}')
```

```
accuracy:  0.7697
```

Xgboost

In [19]:

```
from xgboost import XGBClassifier
import xgboost as xgb
from xgboost import plot_importance
from matplotlib import pyplot

xgb_model  = XGBClassifier(n_estimators = 100, random_state=123456, eval_metric='logloss', use_label_encoder=Fal
xgb_model.fit(X_train,Y_train)
pred_xgb = xgb_model.predict(X_test)

print(f'accuracy: {accuracy_score(Y_test, pred_xgb): .4f}')
```

```
accuracy:  0.7715
```

Support Vector Machines

In [20]:

```
# Support Vector Machines
# SVM 모델 학습
svc = SVC()
svc.fit(X_train, Y_train)
Y_pred_svc = svc.predict(X_test)
acc_svc = svc.score(X_test, Y_test)



print(f'accuracy: {acc_svc: .4f}')
```

```
accuracy:  0.7903
```

Logistic Regression

In [21]:

```
# 수정
```

```
# Logistic Regression training
logreg = LogisticRegression(solver='liblinear')
logreg.fit(X_train, Y_train)
# Logistic Regression prediction
Y_pred_logreg = logreg.predict(X_test)

acc_log = logreg.score(X_test, Y_test)

print(f'accuracy: {acc_log: .4f}')
```

```
accuracy:  0.7903
```

보완필요

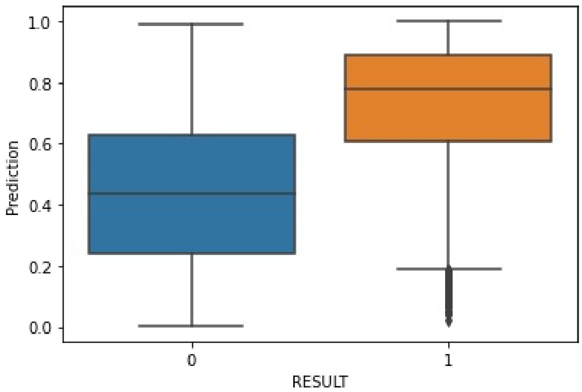
In [22]:

```
#####보완필요#####
import statsmodels.api as sm

model_fb = sm.Logit.from_formula("RESULT ~ HR+AS+HST+AST+HF+AF+HC+AC+HY+AY+HR+AR",EPL_df_ext )
result_fb = model_fb.fit()
print(result_fb.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.509022
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                 RESULT   No. Observations:                 5686
Model:                          Logit   Df Residuals:                     5674
Method:                           MLE   Df Model:                           11
Date:                Mon, 25 Apr 2022   Pseudo R-squ.:                  0.2331
Time:                        22:36:57   Log-Likelihood:                -2894.3
converged:                       True   LL-Null:                       -3774.1
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.3346      0.199      1.678      0.093      -0.056       0.725
HR            -1.0371      0.133     -7.804      0.000      -1.298      -0.777
AS             0.0171      0.011      1.543      0.123      -0.005       0.039
HST            0.3035      0.013     23.876      0.000       0.279       0.328
AST           -0.3711      0.018    -21.087      0.000      -0.406      -0.337
HF             0.0233      0.009      2.513      0.012       0.005       0.041
AF            -0.0088      0.009     -0.990      0.322      -0.026       0.009
HC            -0.0996      0.012     -8.408      0.000      -0.123      -0.076
AC             0.1260      0.014      9.291      0.000       0.099       0.153
HY            -0.2059      0.029     -7.015      0.000      -0.263      -0.148
AY             0.0858      0.028      3.038      0.002       0.030       0.141
AR             0.6668      0.125      5.332      0.000       0.422       0.912
==============================================================================
```

In [23]:

```
import seaborn as sns
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

EPL_df_ext["Prediction"] = result_fb.predict(EPL_df_ext)
sns.boxplot(x="RESULT", y="Prediction", data=EPL_df_ext)
plt.show()
```



In [24]:

```
model_fb = sm.Logit.from_formula("RESULT ~ HST + HF+ AC + AY + AR",EPL_df_ext )
result_fb = model_fb.fit()
print(result_fb.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.597698
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                 RESULT   No. Observations:                 5686
Model:                          Logit   Df Residuals:                     5680
Method:                           MLE   Df Model:                            5
Date:                Mon, 25 Apr 2022   Pseudo R-squ.:                 0.09951
Time:                        22:36:57   Log-Likelihood:                -3398.5
converged:                       True   LL-Null:                       -3774.1
Covariance Type:            nonrobust   LLR p-value:                4.361e-160
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -1.0399      0.131     -7.924      0.000      -1.297      -0.783
HST            0.2449      0.011     23.192      0.000       0.224       0.266
HF            -0.0161      0.008     -2.094      0.036      -0.031      -0.001
AC             0.0174      0.011      1.614      0.106      -0.004       0.038
AY             0.0600      0.024      2.534      0.011       0.014       0.106
AR             0.6467      0.112      5.749      0.000       0.426       0.867
==============================================================================
```

k-Nearest Neighbor

In [25]:

```
knn  = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred_knn = knn.predict(X_test)

acc_knn = knn.score(X_test, Y_test)

print(f'accuracy: {acc_knn: .4f}')
```

```
accuracy:  0.6966
```

Naive Bayes classifiers

In [26]:

```
# Gaussian Naive Bayes
gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred_NB = gaussian.predict(X_test)

acc_gaussian = gaussian.score(X_test, Y_test)

print(f'accuracy: {acc_gaussian: .4f}')
```

```
accuracy:  0.7154
```

Decision tree

In [27]:

```
# Decision Tree
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred_DT = decision_tree.predict(X_test)

acc_decision_tree = decision_tree.score(X_test, Y_test)
print(f'accuracy: {acc_decision_tree: .4f}')
```

```
accuracy:  0.6966
```

Artificial Neural Network

In [28]:

```
# 수정
ANN = MLPClassifier(solver='lbfgs', alpha=1, hidden_layer_sizes=(30, 10), random_state=1, max_iter=5000)
ANN.fit(X_train, Y_train)
Y_pred_ANN = ANN.predict(X_test)

acc_ANN = ANN.score(X_test, Y_test)

print(f'accuracy: {acc_ANN: .4f}')
```

```
accuracy:  0.7715
```

Keras : ANN 모델

In [29]:

```
from keras import layers, models, datasets
from keras.utils import np_utils
```

```
Using TensorFlow backend.
```

In [30]:

```
def plot(hist):
```

```
    fig, loss_ax = plt.subplots()
    acc_ax = loss_ax.twinx()

    loss_ax.plot(hist.history['loss'], 'y', label='train loss')
    loss_ax.plot(hist.history['val_loss'], 'r', label='val loss')

    acc_ax.plot(hist.history['accuracy'], 'b', label='train acc')
    acc_ax.plot(hist.history['val_accuracy'], 'g', label='val acc')

    loss_ax.set_xlabel('epoch')
    loss_ax.set_ylabel('loss')
    acc_ax.set_ylabel('accuray')

    loss_ax.legend(loc='upper left')
    acc_ax.legend(loc='lower left')

    plt.show()
```

In [31]:

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation

model = Sequential()
model.add(Dense(64, activation='relu', input_dim=12))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy'])

history = model.fit(X_train, Y_train, epochs=200,
        batch_size=32, validation_split=0.2)
performance_test = model.evaluate(X_test, Y_test, batch_size=32)
classes = model.predict(X_test, batch_size=128)
performance_test
```

```
2022-04-25 22:37:11.265254: I tensorflow/core/platform/cpu_feature_guard.cc:145] This TensorFlow binary is opti
mized with Intel(R) MKL-DNN to use the following CPU instructions in performance critical operations:  SSE4.1 S
SE4.2 AVX AVX2 FMA
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate compiler flags.
2022-04-25 22:37:11.265758: I tensorflow/core/common_runtime/process_util.cc:115] Creating new thread pool with
default inter op setting: 12. Tune using inter_op_parallelism_threads for best performance.
Train on 4548 samples, validate on 1138 samples
Epoch 1/200
4548/4548 [==============================] - 1s 261us/step - loss: 0.7993 - accuracy: 0.6240 - val_loss: 0.5115
- val_accuracy: 0.7671
Epoch 2/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.6143 - accuracy: 0.6840 - val_loss: 0.5104
- val_accuracy: 0.7671
Epoch 3/200
4548/4548 [==============================] - 0s 103us/step - loss: 0.5717 - accuracy: 0.7040 - val_loss: 0.5030
- val_accuracy: 0.7680
Epoch 4/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5659 - accuracy: 0.7128 - val_loss: 0.4911
- val_accuracy: 0.7654
Epoch 5/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5480 - accuracy: 0.7265 - val_loss: 0.4779
- val_accuracy: 0.7698
Epoch 6/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5461 - accuracy: 0.7269 - val_loss: 0.4886
- val_accuracy: 0.7768
Epoch 7/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5455 - accuracy: 0.7227 - val_loss: 0.4848
- val_accuracy: 0.7786
Epoch 8/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5398 - accuracy: 0.7271 - val_loss: 0.4711
- val_accuracy: 0.7830
Epoch 9/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.5319 - accuracy: 0.7355 - val_loss: 0.4745
- val_accuracy: 0.7786
Epoch 10/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5285 - accuracy: 0.7379 - val_loss: 0.4792
- val_accuracy: 0.7865
Epoch 11/200
4548/4548 [==============================] - 0s 105us/step - loss: 0.5312 - accuracy: 0.7381 - val_loss: 0.4986
- val_accuracy: 0.7786
Epoch 12/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5253 - accuracy: 0.7397 - val_loss: 0.4767
- val_accuracy: 0.7786
Epoch 13/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5276 - accuracy: 0.7357 - val_loss: 0.4815
- val_accuracy: 0.7935
Epoch 14/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5265 - accuracy: 0.7392 - val_loss: 0.4736
- val_accuracy: 0.7847
Epoch 15/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.5213 - accuracy: 0.7416 - val_loss: 0.4704
- val_accuracy: 0.7882
```

```
Epoch 16/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5200 - accuracy: 0.7482 - val_loss: 0.4790
- val_accuracy: 0.7777
Epoch 17/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5206 - accuracy: 0.7458 - val_loss: 0.4758
- val_accuracy: 0.7856
Epoch 18/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5175 - accuracy: 0.7392 - val_loss: 0.4656
- val_accuracy: 0.7856
Epoch 19/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5164 - accuracy: 0.7456 - val_loss: 0.4659
- val_accuracy: 0.7900
Epoch 20/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5196 - accuracy: 0.7463 - val_loss: 0.4843
- val_accuracy: 0.7900
Epoch 21/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.5153 - accuracy: 0.7454 - val_loss: 0.4692
- val_accuracy: 0.7865
Epoch 22/200
4548/4548 [==============================] - 0s 104us/step - loss: 0.5158 - accuracy: 0.7436 - val_loss: 0.4776
- val_accuracy: 0.7768
Epoch 23/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.5124 - accuracy: 0.7498 - val_loss: 0.4523
- val_accuracy: 0.7873
Epoch 24/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.5159 - accuracy: 0.7383 - val_loss: 0.4693
- val_accuracy: 0.7909
Epoch 25/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.5123 - accuracy: 0.7489 - val_loss: 0.4590
- val_accuracy: 0.7882
Epoch 26/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.5125 - accuracy: 0.7423 - val_loss: 0.4666
- val_accuracy: 0.7891
Epoch 27/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.5079 - accuracy: 0.7544 - val_loss: 0.4652
- val_accuracy: 0.7856
Epoch 28/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.5114 - accuracy: 0.7454 - val_loss: 0.4615
- val_accuracy: 0.7891
Epoch 29/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.5115 - accuracy: 0.7469 - val_loss: 0.4698
- val_accuracy: 0.7856
Epoch 30/200
4548/4548 [==============================] - 1s 115us/step - loss: 0.5122 - accuracy: 0.7397 - val_loss: 0.4615
- val_accuracy: 0.7847
Epoch 31/200
4548/4548 [==============================] - 1s 115us/step - loss: 0.5092 - accuracy: 0.7456 - val_loss: 0.4641
- val_accuracy: 0.7873
Epoch 32/200
4548/4548 [==============================] - 1s 118us/step - loss: 0.5086 - accuracy: 0.7524 - val_loss: 0.4626
- val_accuracy: 0.7891
Epoch 33/200
4548/4548 [==============================] - 1s 121us/step - loss: 0.5060 - accuracy: 0.7478 - val_loss: 0.4627
- val_accuracy: 0.7926
Epoch 34/200
4548/4548 [==============================] - 1s 114us/step - loss: 0.5023 - accuracy: 0.7542 - val_loss: 0.4670
- val_accuracy: 0.7891
Epoch 35/200
4548/4548 [==============================] - 1s 114us/step - loss: 0.5085 - accuracy: 0.7458 - val_loss: 0.4635
- val_accuracy: 0.7847
Epoch 36/200
4548/4548 [==============================] - 1s 114us/step - loss: 0.5045 - accuracy: 0.7520 - val_loss: 0.4684
- val_accuracy: 0.7891
Epoch 37/200
4548/4548 [==============================] - 1s 113us/step - loss: 0.5015 - accuracy: 0.7526 - val_loss: 0.4650
- val_accuracy: 0.7900
Epoch 38/200
4548/4548 [==============================] - 1s 114us/step - loss: 0.4981 - accuracy: 0.7509 - val_loss: 0.4665
- val_accuracy: 0.7847
Epoch 39/200
4548/4548 [==============================] - 1s 116us/step - loss: 0.5009 - accuracy: 0.7537 - val_loss: 0.4687
- val_accuracy: 0.7803
Epoch 40/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4992 - accuracy: 0.7577 - val_loss: 0.4645
- val_accuracy: 0.7891
Epoch 41/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.5029 - accuracy: 0.7524 - val_loss: 0.4633
- val_accuracy: 0.7847
Epoch 42/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.5032 - accuracy: 0.7504 - val_loss: 0.4715
- val_accuracy: 0.7821
Epoch 43/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4989 - accuracy: 0.7509 - val_loss: 0.4699
- val_accuracy: 0.7882
Epoch 44/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.5017 - accuracy: 0.7489 - val_loss: 0.4552
- val_accuracy: 0.7900
Epoch 45/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.5006 - accuracy: 0.7500 - val_loss: 0.4732
```

```
- val_accuracy: 0.7821
Epoch 46/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4995 - accuracy: 0.7487 - val_loss: 0.4748
- val_accuracy: 0.7803
Epoch 47/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.5006 - accuracy: 0.7465 - val_loss: 0.4749
- val_accuracy: 0.7821
Epoch 48/200
4548/4548 [==============================] - 1s 119us/step - loss: 0.4987 - accuracy: 0.7535 - val_loss: 0.4628
- val_accuracy: 0.7882
Epoch 49/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4997 - accuracy: 0.7542 - val_loss: 0.4688
- val_accuracy: 0.7830
Epoch 50/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4960 - accuracy: 0.7531 - val_loss: 0.4647
- val_accuracy: 0.7891
Epoch 51/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4958 - accuracy: 0.7515 - val_loss: 0.4570
- val_accuracy: 0.7891
Epoch 52/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4967 - accuracy: 0.7568 - val_loss: 0.4634
- val_accuracy: 0.7794
Epoch 53/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.5006 - accuracy: 0.7526 - val_loss: 0.4728
- val_accuracy: 0.7803
Epoch 54/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4923 - accuracy: 0.7614 - val_loss: 0.4642
- val_accuracy: 0.7917
Epoch 55/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4895 - accuracy: 0.7551 - val_loss: 0.4734
- val_accuracy: 0.7812
Epoch 56/200
4548/4548 [==============================] - 0s 110us/step - loss: 0.4952 - accuracy: 0.7564 - val_loss: 0.4597
- val_accuracy: 0.7891
Epoch 57/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4933 - accuracy: 0.7542 - val_loss: 0.4654
- val_accuracy: 0.7856
Epoch 58/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4930 - accuracy: 0.7579 - val_loss: 0.4580
- val_accuracy: 0.7803
Epoch 59/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4876 - accuracy: 0.7584 - val_loss: 0.4570
- val_accuracy: 0.7856
Epoch 60/200
4548/4548 [==============================] - 0s 110us/step - loss: 0.4894 - accuracy: 0.7522 - val_loss: 0.4712
- val_accuracy: 0.7777
Epoch 61/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4941 - accuracy: 0.7573 - val_loss: 0.4777
- val_accuracy: 0.7891
Epoch 62/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4938 - accuracy: 0.7555 - val_loss: 0.4705
- val_accuracy: 0.7926
Epoch 63/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4889 - accuracy: 0.7551 - val_loss: 0.4627
- val_accuracy: 0.7821
Epoch 64/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4874 - accuracy: 0.7614 - val_loss: 0.4722
- val_accuracy: 0.7794
Epoch 65/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4883 - accuracy: 0.7579 - val_loss: 0.4622
- val_accuracy: 0.7900
Epoch 66/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4875 - accuracy: 0.7579 - val_loss: 0.4688
- val_accuracy: 0.7865
Epoch 67/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4882 - accuracy: 0.7621 - val_loss: 0.4673
- val_accuracy: 0.7856
Epoch 68/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4823 - accuracy: 0.7623 - val_loss: 0.4780
- val_accuracy: 0.7847
Epoch 69/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4869 - accuracy: 0.7586 - val_loss: 0.4670
- val_accuracy: 0.7856
Epoch 70/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4804 - accuracy: 0.7630 - val_loss: 0.4723
- val_accuracy: 0.7873
Epoch 71/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4880 - accuracy: 0.7577 - val_loss: 0.4636
- val_accuracy: 0.7909
Epoch 72/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4821 - accuracy: 0.7632 - val_loss: 0.4734
- val_accuracy: 0.7821
Epoch 73/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4813 - accuracy: 0.7544 - val_loss: 0.4615
- val_accuracy: 0.7873
Epoch 74/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4855 - accuracy: 0.7658 - val_loss: 0.4722
- val_accuracy: 0.7926
Epoch 75/200
```

```
4548/4548 [==============================] - 0s 106us/step - loss: 0.4826 - accuracy: 0.7647 - val_loss: 0.4850
- val_accuracy: 0.7900
Epoch 76/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4834 - accuracy: 0.7630 - val_loss: 0.4695
- val_accuracy: 0.7882
Epoch 77/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4782 - accuracy: 0.7639 - val_loss: 0.4773
- val_accuracy: 0.7838
Epoch 78/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4804 - accuracy: 0.7643 - val_loss: 0.4808
- val_accuracy: 0.7742
Epoch 79/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4782 - accuracy: 0.7645 - val_loss: 0.4658
- val_accuracy: 0.7856
Epoch 80/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4807 - accuracy: 0.7680 - val_loss: 0.4788
- val_accuracy: 0.7847
Epoch 81/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4794 - accuracy: 0.7645 - val_loss: 0.4677
- val_accuracy: 0.7873
Epoch 82/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4788 - accuracy: 0.7603 - val_loss: 0.4930
- val_accuracy: 0.7768
Epoch 83/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4783 - accuracy: 0.7630 - val_loss: 0.4866
- val_accuracy: 0.7821
Epoch 84/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4748 - accuracy: 0.7628 - val_loss: 0.4745
- val_accuracy: 0.7935
Epoch 85/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4767 - accuracy: 0.7612 - val_loss: 0.4761
- val_accuracy: 0.7856
Epoch 86/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4771 - accuracy: 0.7698 - val_loss: 0.4768
- val_accuracy: 0.7786
Epoch 87/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4725 - accuracy: 0.7687 - val_loss: 0.4738
- val_accuracy: 0.7821
Epoch 88/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4739 - accuracy: 0.7628 - val_loss: 0.4621
- val_accuracy: 0.7856
Epoch 89/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4767 - accuracy: 0.7650 - val_loss: 0.4786
- val_accuracy: 0.7891
Epoch 90/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4705 - accuracy: 0.7707 - val_loss: 0.4825
- val_accuracy: 0.7873
Epoch 91/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4746 - accuracy: 0.7700 - val_loss: 0.4704
- val_accuracy: 0.7847
Epoch 92/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4693 - accuracy: 0.7698 - val_loss: 0.4856
- val_accuracy: 0.7891
Epoch 93/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4727 - accuracy: 0.7685 - val_loss: 0.4949
- val_accuracy: 0.7847
Epoch 94/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4725 - accuracy: 0.7632 - val_loss: 0.4865
- val_accuracy: 0.7821
Epoch 95/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4755 - accuracy: 0.7704 - val_loss: 0.4849
- val_accuracy: 0.7803
Epoch 96/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4720 - accuracy: 0.7685 - val_loss: 0.4762
- val_accuracy: 0.7882
Epoch 97/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4644 - accuracy: 0.7700 - val_loss: 0.4849
- val_accuracy: 0.7794
Epoch 98/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4734 - accuracy: 0.7674 - val_loss: 0.4764
- val_accuracy: 0.7821
Epoch 99/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4637 - accuracy: 0.7726 - val_loss: 0.4767
- val_accuracy: 0.7882
Epoch 100/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4666 - accuracy: 0.7806 - val_loss: 0.4845
- val_accuracy: 0.7847
Epoch 101/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4664 - accuracy: 0.7733 - val_loss: 0.4972
- val_accuracy: 0.7750
Epoch 102/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4707 - accuracy: 0.7619 - val_loss: 0.4934
- val_accuracy: 0.7654
Epoch 103/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4636 - accuracy: 0.7779 - val_loss: 0.4872
- val_accuracy: 0.7803
Epoch 104/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4678 - accuracy: 0.7740 - val_loss: 0.4923
- val_accuracy: 0.7794
```

```
Epoch 105/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4710 - accuracy: 0.7654 - val_loss: 0.4843
- val_accuracy: 0.7900
Epoch 106/200
4548/4548 [==============================] - 1s 110us/step - loss: 0.4722 - accuracy: 0.7678 - val_loss: 0.4966
- val_accuracy: 0.7856
Epoch 107/200
4548/4548 [==============================] - 1s 114us/step - loss: 0.4636 - accuracy: 0.7654 - val_loss: 0.4956
- val_accuracy: 0.7830
Epoch 108/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4639 - accuracy: 0.7751 - val_loss: 0.5002
- val_accuracy: 0.7821
Epoch 109/200
4548/4548 [==============================] - 1s 116us/step - loss: 0.4668 - accuracy: 0.7726 - val_loss: 0.4967
- val_accuracy: 0.7838
Epoch 110/200
4548/4548 [==============================] - 1s 114us/step - loss: 0.4661 - accuracy: 0.7715 - val_loss: 0.4931
- val_accuracy: 0.7733
Epoch 111/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4688 - accuracy: 0.7726 - val_loss: 0.4797
- val_accuracy: 0.7900
Epoch 112/200
4548/4548 [==============================] - 1s 116us/step - loss: 0.4608 - accuracy: 0.7720 - val_loss: 0.4914
- val_accuracy: 0.7847
Epoch 113/200
4548/4548 [==============================] - 1s 117us/step - loss: 0.4665 - accuracy: 0.7718 - val_loss: 0.5020
- val_accuracy: 0.7786
Epoch 114/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4665 - accuracy: 0.7757 - val_loss: 0.4902
- val_accuracy: 0.7777
Epoch 115/200
4548/4548 [==============================] - 1s 113us/step - loss: 0.4628 - accuracy: 0.7687 - val_loss: 0.4903
- val_accuracy: 0.7882
Epoch 116/200
4548/4548 [==============================] - 1s 113us/step - loss: 0.4623 - accuracy: 0.7715 - val_loss: 0.4950
- val_accuracy: 0.7856
Epoch 117/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4656 - accuracy: 0.7634 - val_loss: 0.4926
- val_accuracy: 0.7935
Epoch 118/200
4548/4548 [==============================] - 1s 110us/step - loss: 0.4641 - accuracy: 0.7661 - val_loss: 0.4912
- val_accuracy: 0.7891
Epoch 119/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4639 - accuracy: 0.7746 - val_loss: 0.4935
- val_accuracy: 0.7794
Epoch 120/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4663 - accuracy: 0.7773 - val_loss: 0.4928
- val_accuracy: 0.7856
Epoch 121/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4542 - accuracy: 0.7768 - val_loss: 0.4892
- val_accuracy: 0.7900
Epoch 122/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4595 - accuracy: 0.7773 - val_loss: 0.4951
- val_accuracy: 0.7821
Epoch 123/200
4548/4548 [==============================] - 1s 113us/step - loss: 0.4539 - accuracy: 0.7808 - val_loss: 0.5159
- val_accuracy: 0.7830
Epoch 124/200
4548/4548 [==============================] - 1s 114us/step - loss: 0.4589 - accuracy: 0.7740 - val_loss: 0.4963
- val_accuracy: 0.7873
Epoch 125/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4567 - accuracy: 0.7792 - val_loss: 0.4933
- val_accuracy: 0.7882
Epoch 126/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4538 - accuracy: 0.7812 - val_loss: 0.5058
- val_accuracy: 0.7865
Epoch 127/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4606 - accuracy: 0.7726 - val_loss: 0.5076
- val_accuracy: 0.7821
Epoch 128/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4586 - accuracy: 0.7682 - val_loss: 0.5105
- val_accuracy: 0.7750
Epoch 129/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4589 - accuracy: 0.7735 - val_loss: 0.5032
- val_accuracy: 0.7777
Epoch 130/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4532 - accuracy: 0.7808 - val_loss: 0.5062
- val_accuracy: 0.7812
Epoch 131/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4544 - accuracy: 0.7715 - val_loss: 0.5014
- val_accuracy: 0.7838
Epoch 132/200
4548/4548 [==============================] - 1s 113us/step - loss: 0.4546 - accuracy: 0.7757 - val_loss: 0.4962
- val_accuracy: 0.7830
Epoch 133/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4579 - accuracy: 0.7759 - val_loss: 0.5037
- val_accuracy: 0.7812
Epoch 134/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4602 - accuracy: 0.7742 - val_loss: 0.5049
```

```
- val_accuracy: 0.7900
Epoch 135/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4509 - accuracy: 0.7817 - val_loss: 0.5058
- val_accuracy: 0.7821
Epoch 136/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4546 - accuracy: 0.7744 - val_loss: 0.5043
- val_accuracy: 0.7847
Epoch 137/200
4548/4548 [==============================] - 1s 110us/step - loss: 0.4516 - accuracy: 0.7777 - val_loss: 0.5084
- val_accuracy: 0.7873
Epoch 138/200
4548/4548 [==============================] - 1s 110us/step - loss: 0.4554 - accuracy: 0.7740 - val_loss: 0.4963
- val_accuracy: 0.7794
Epoch 139/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4527 - accuracy: 0.7753 - val_loss: 0.5129
- val_accuracy: 0.7821
Epoch 140/200
4548/4548 [==============================] - 1s 110us/step - loss: 0.4497 - accuracy: 0.7885 - val_loss: 0.5208
- val_accuracy: 0.7873
Epoch 141/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4495 - accuracy: 0.7755 - val_loss: 0.5211
- val_accuracy: 0.7900
Epoch 142/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4559 - accuracy: 0.7744 - val_loss: 0.5079
- val_accuracy: 0.7909
Epoch 143/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4546 - accuracy: 0.7801 - val_loss: 0.5080
- val_accuracy: 0.7838
Epoch 144/200
4548/4548 [==============================] - 0s 110us/step - loss: 0.4514 - accuracy: 0.7795 - val_loss: 0.5043
- val_accuracy: 0.7794
Epoch 145/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4479 - accuracy: 0.7821 - val_loss: 0.5179
- val_accuracy: 0.7794
Epoch 146/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4522 - accuracy: 0.7762 - val_loss: 0.5099
- val_accuracy: 0.7838
Epoch 147/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4497 - accuracy: 0.7832 - val_loss: 0.5082
- val_accuracy: 0.7821
Epoch 148/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4481 - accuracy: 0.7781 - val_loss: 0.5233
- val_accuracy: 0.7698
Epoch 149/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4548 - accuracy: 0.7790 - val_loss: 0.4969
- val_accuracy: 0.7900
Epoch 150/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4544 - accuracy: 0.7764 - val_loss: 0.5137
- val_accuracy: 0.7821
Epoch 151/200
4548/4548 [==============================] - 1s 120us/step - loss: 0.4541 - accuracy: 0.7755 - val_loss: 0.5269
- val_accuracy: 0.7873
Epoch 152/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4466 - accuracy: 0.7790 - val_loss: 0.5140
- val_accuracy: 0.7768
Epoch 153/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4486 - accuracy: 0.7817 - val_loss: 0.5118
- val_accuracy: 0.7794
Epoch 154/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4487 - accuracy: 0.7792 - val_loss: 0.5147
- val_accuracy: 0.7768
Epoch 155/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4467 - accuracy: 0.7850 - val_loss: 0.5078
- val_accuracy: 0.7882
Epoch 156/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4488 - accuracy: 0.7810 - val_loss: 0.5185
- val_accuracy: 0.7777
Epoch 157/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4479 - accuracy: 0.7880 - val_loss: 0.5118
- val_accuracy: 0.7698
Epoch 158/200
4548/4548 [==============================] - 1s 113us/step - loss: 0.4485 - accuracy: 0.7852 - val_loss: 0.5303
- val_accuracy: 0.7777
Epoch 159/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4459 - accuracy: 0.7819 - val_loss: 0.5300
- val_accuracy: 0.7759
Epoch 160/200
4548/4548 [==============================] - 1s 113us/step - loss: 0.4452 - accuracy: 0.7786 - val_loss: 0.5231
- val_accuracy: 0.7786
Epoch 161/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4509 - accuracy: 0.7799 - val_loss: 0.5211
- val_accuracy: 0.7742
Epoch 162/200
4548/4548 [==============================] - 1s 112us/step - loss: 0.4485 - accuracy: 0.7880 - val_loss: 0.5277
- val_accuracy: 0.7812
Epoch 163/200
4548/4548 [==============================] - 1s 111us/step - loss: 0.4468 - accuracy: 0.7869 - val_loss: 0.5363
- val_accuracy: 0.7794
Epoch 164/200
```

```
4548/4548 [==============================] - 0s 109us/step - loss: 0.4397 - accuracy: 0.7887 - val_loss: 0.5336
- val_accuracy: 0.7786
Epoch 165/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4369 - accuracy: 0.7889 - val_loss: 0.5422
- val_accuracy: 0.7838
Epoch 166/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4464 - accuracy: 0.7812 - val_loss: 0.5226
- val_accuracy: 0.7759
Epoch 167/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4413 - accuracy: 0.7834 - val_loss: 0.5268
- val_accuracy: 0.7777
Epoch 168/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4519 - accuracy: 0.7812 - val_loss: 0.5256
- val_accuracy: 0.7786
Epoch 169/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4497 - accuracy: 0.7775 - val_loss: 0.5170
- val_accuracy: 0.7750
Epoch 170/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4414 - accuracy: 0.7823 - val_loss: 0.5227
- val_accuracy: 0.7873
Epoch 171/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4402 - accuracy: 0.7834 - val_loss: 0.5319
- val_accuracy: 0.7715
Epoch 172/200
4548/4548 [==============================] - 1s 116us/step - loss: 0.4367 - accuracy: 0.7869 - val_loss: 0.5321
- val_accuracy: 0.7750
Epoch 173/200
4548/4548 [==============================] - 1s 113us/step - loss: 0.4389 - accuracy: 0.7836 - val_loss: 0.5288
- val_accuracy: 0.7803
Epoch 174/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4414 - accuracy: 0.7902 - val_loss: 0.5349
- val_accuracy: 0.7821
Epoch 175/200
4548/4548 [==============================] - 1s 113us/step - loss: 0.4474 - accuracy: 0.7902 - val_loss: 0.5339
- val_accuracy: 0.7838
Epoch 176/200
4548/4548 [==============================] - 0s 110us/step - loss: 0.4401 - accuracy: 0.7874 - val_loss: 0.5301
- val_accuracy: 0.7803
Epoch 177/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4459 - accuracy: 0.7858 - val_loss: 0.5259
- val_accuracy: 0.7750
Epoch 178/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4457 - accuracy: 0.7854 - val_loss: 0.5318
- val_accuracy: 0.7838
Epoch 179/200
4548/4548 [==============================] - 0s 109us/step - loss: 0.4406 - accuracy: 0.7931 - val_loss: 0.5076
- val_accuracy: 0.7794
Epoch 180/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4478 - accuracy: 0.7795 - val_loss: 0.5129
- val_accuracy: 0.7759
Epoch 181/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4441 - accuracy: 0.7861 - val_loss: 0.5168
- val_accuracy: 0.7865
Epoch 182/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4410 - accuracy: 0.7861 - val_loss: 0.5196
- val_accuracy: 0.7742
Epoch 183/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4392 - accuracy: 0.7858 - val_loss: 0.5100
- val_accuracy: 0.7838
Epoch 184/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4420 - accuracy: 0.7834 - val_loss: 0.5312
- val_accuracy: 0.7794
Epoch 185/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4354 - accuracy: 0.7971 - val_loss: 0.5329
- val_accuracy: 0.7733
Epoch 186/200
4548/4548 [==============================] - 0s 110us/step - loss: 0.4374 - accuracy: 0.7872 - val_loss: 0.5437
- val_accuracy: 0.7742
Epoch 187/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4363 - accuracy: 0.7856 - val_loss: 0.5491
- val_accuracy: 0.7953
Epoch 188/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4356 - accuracy: 0.7951 - val_loss: 0.5290
- val_accuracy: 0.7689
Epoch 189/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4346 - accuracy: 0.7863 - val_loss: 0.5613
- val_accuracy: 0.7742
Epoch 190/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4469 - accuracy: 0.7828 - val_loss: 0.5134
- val_accuracy: 0.7759
Epoch 191/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4343 - accuracy: 0.7847 - val_loss: 0.5409
- val_accuracy: 0.7777
Epoch 192/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4401 - accuracy: 0.7812 - val_loss: 0.5341
- val_accuracy: 0.7786
Epoch 193/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4367 - accuracy: 0.7874 - val_loss: 0.5397
- val_accuracy: 0.7750
```

```
Epoch 194/200
4548/4548 [==============================] - 0s 107us/step - loss: 0.4450 - accuracy: 0.7790 - val_loss: 0.5438
- val_accuracy: 0.7671
Epoch 195/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4345 - accuracy: 0.7907 - val_loss: 0.5481
- val_accuracy: 0.7768
Epoch 196/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4289 - accuracy: 0.7898 - val_loss: 0.5230
- val_accuracy: 0.7812
Epoch 197/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4347 - accuracy: 0.7920 - val_loss: 0.5536
- val_accuracy: 0.7847
Epoch 198/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4413 - accuracy: 0.7889 - val_loss: 0.5355
- val_accuracy: 0.7856
Epoch 199/200
4548/4548 [==============================] - 0s 106us/step - loss: 0.4413 - accuracy: 0.7907 - val_loss: 0.5393
- val_accuracy: 0.7724
Epoch 200/200
4548/4548 [==============================] - 0s 108us/step - loss: 0.4410 - accuracy: 0.7865 - val_loss: 0.5543
- val_accuracy: 0.7750
534/534 [==============================] - 0s 31us/step
```

Out[31]:

```
[0.5510589545139213, 0.7640449404716492]
```

In [32]:

```
classes2_list = classes.tolist()
model_keras_ANN = []
for i in classes2_list:
    model_keras_ANN.append(round(i[0]))
```

모델 검증

In [33]:

```
#report
from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_pred_ANN, digits=4))
```

```
              precision    recall  f1-score   support

           0     0.8053    0.7000    0.7490       260
           1     0.7468    0.8394    0.7904       274

    accuracy                         0.7715       534
   macro avg     0.7760    0.7697    0.7697       534
weighted avg     0.7753    0.7715    0.7702       534
```

앙상블모델 만들기

In [34]:

```
model_knn = knn.predict(X_test)
model_decision_tree = decision_tree.predict(X_test)
model_gaussian = gaussian.predict(X_test)
model_logreg = logreg.predict(X_test)
model_svc = svc.predict(X_test)
model_ANN = ANN.predict(X_test)
# model_keras_ANN
```

In [35]:

```
score = []
# for i in range(0,962):
for i in range(0,534):
    list = []
    #list.append(model_knn[i])
    #list.append(model_decision_tree[i])
    #list.append(model_gaussian[i])
    #list.append(model_svc[i])
    list.append(model_logreg[i])
    list.append(model_ANN[i])
    list.append(model_keras_ANN[i])
    if list.count(1) > list.count(0):
        score.append(1)
    else:
        score.append(0)
print(Y_test)
```

```
0      0
1      1
2      1
3      0
4      0
      ..
683    0
684    0
685    1
686    1
687    1
Name: RESULT, Length: 534, dtype: int64
```

In [36]:

```
ensemble = pd.DataFrame({
        "result": Y_test,
        "prediction_ensemble": score
    })
ensemble['RESULT'] = np.where(ensemble["result"]==ensemble["prediction_ensemble"], 1, 0)
```

In [37]:

```
#앙상블모델 정확도
acc_Ens = round(((sum(ensemble['RESULT'])/534))*100, 2)
acc_Ens
```

Out[37]:

```
77.72
```

In [38]:

```
print(classification_report(Y_test,score))
```

```
              precision    recall  f1-score   support

           0       0.81      0.70      0.75       260
           1       0.75      0.85      0.80       274

    accuracy                           0.78       534
   macro avg       0.78      0.78      0.78       534
weighted avg       0.78      0.78      0.78       534
```