Titanic - Advanced Feature Engineering Tutorial

For EDA

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='darkgrid')
```

For Model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import StratifiedKFold
```

Default

```
import string
import warnings
warnings.filterwarnings('ignore')
SEED = 42
```

0. Loading Data

```
def concat_df(train_data, test_data):
                                                   Index 정렬              Index drop.
    return pd.concat([train_data, test_data], sort=True).reset_index(drop=True)

def divide_df(all_data)
                                  change
    return all_data.iloc[:, :890], all_data.iloc[891:, :].drop(['Survived'], axis=1)
```

```
df_train = pd.read_csv('~')
df_test =      〃
df_all = concat_df(df_train, df_test)

df_train.name = 'Training Set'
df_test.name = 'Test Set'
df_all.name = 'All set'

dfs = [df_train, df_test]
# 각 Set의 Shape, Columns 확인.
```

1. EDA.

1.1 Overview the dataset

1.2 Missing Values.

```
def display_missing(df):
    for col in df.columns.tolist():
        print(f'{col} column missing values : {df[col].isnull().sum()}')

for df in dfs:
  print(df.name)
  display_missing(df)
```

NO.

1.2.1 Age.

- How to fill NA in 'Age'? By using high correlation with other features.

```
df_all_corr = df_all.corr( ).abs( ).unstack( ).sort_values(kind = 'quicksort',
                ascending = False).reset_index( )
# df_all_corr 의 type : Series → DataFrame (by 'reset_index' method)
                     (이중 인덱스)
                                                        , inplace = True.
df_all_corr.rename(columns = {'level_0' : 'Feature_1',  ...  })
df_all_corr[df_all_corr['Feature_1'] == 'Age']
# 'Age'와 'Pclass'의 상관 관계가 가장 높음. 하지만 이것만으로 feature engineering
하기엔 너무 성급함. 정확성을 높이기 위해 'Sex' feature도 고려하여 missing value 채움.

                      groupby
age_by_pclass_sex = df_all (['Pclass', 'Sex']).median( )['Age']

for pclass in df_all['Pclass'].unique( ).tolist( ) :
    for sex in df_all['Sex']     "                 :
        print(f'{ median age of {pclass}{sex} : }', age_by_pclass_sex[pclass][sex])

df_all['Age'] = df_all.groupby(['Pclass', 'Sex'])['Age'].apply(lambda x : x.fillna(x.median( )))
# 'lambda'는 DataFrame 에서의 Index 값을 받음. 위의 경우엔 (1, 'female')과
같은 이중 인덱스를 X로 받음.
```

1.2.2 Embarked

- Need to fill two missing values. (Using outside information)

```
df_all['Embarked'] = df_all['Embarked'].fillna('S')
```

1.2.3 Fare

- Need to fill one missing value.
  - The passenger is male, third-class, and no family.

```
med_fare = df_all.groupby(['Pclass', 'Parch', 'Sibsp'])['Fare'].
                median()[3][0][0]
df_all['Fare'] = df_all['Fare'].fillna(med_fare)
```

1.2.4 Cabin

```
df_all['Deck'] = df_all['Cabin'].apply(lambda s : s[0] if pd.notnull(s)
                                        else 'M')
```

→ 'Cabin'의 첫 문자가 Deck을 의미함. Deck 별 Pclass 성격을 확인.

```
df_all_decks = df_all.groupby(['Pclass', 'Deck']).count().
                    drop(columns=[' ~ '])

def get_pclass_dist(df):

    deck_counts = {'A': { },  ...  }
    decks = df.columns.levels[0]
```

NO.

```
for deck in decks :
try:    for pclass in range(1, 4) :
            count = df[deck][pclass][0]
            deck_counts[deck][pclass] = count
        except Keyerror:
            deck_counts[deck][pclass] = 0.

df_deck = pd.DataFrame(deck_counts)
# deck_percentage = { }

for col in df_deck.columns :
    deck_percentage[col] = [ ( count / df_deck[col].sum() ) * 100
                                  for count in df_deck[col] ]

return deck_counts, deck_percentage.

def display_pclass_dist ( percentages ) :
    df_percentages = pd.DataFrame ( percentages ). transpose ( )
    deck_names = ( 'A', ... , 'T' )                    }
    bar_count = np.arange ( len ( deck_names ) )       } # Xticks label 변경 시 사용.
    bar_width = 0.85

    pclass1 = df_percentages [0]
      " 2          "         [1]
      " 3          "         [2]

    plt.figure ( figsize = (20, 10) )
```

```
  plt.bar (bar_count, pclass1, color =        ,
           width = bar_width, edgecolor =      ,
           label = 'Pclass1' )
  plt.bar ( bar_count, pclass2, bottom = pclass1,
                  "                    )
  plt.bar (bar_count, pclass3, bottom = pclass1 + pclass2,
                  "                    )

  plt.xlabel ('Deck')
  plt.ylabel ('Pclass percentage')
  plt.xticks (bar_count, deck_names)
  plt.legend (loc= 'Upper left', bbox_to_anchor = (1, 1), prop = {'size': 15})
  plt.title ('      ')
  plt.show ( )

all_deck_count, all_deck_per = get_pclass_dist (df_all_decks)
display_pclass_dist (all_deck_per)

# change 'T' deck to 'A' deck.
idx = df_all [df_all ['Deck'] == 'T'].index
df_all.loc [idx, 'Deck'] = 'A'
```

'Survived & Deck' 도 위의 과정과 동일!

1.3 Target Distribution.

- Target value 시각화 (수치, 비율)

```
Survived_cnt = df_train['Survived'].value_counts( )[1]
not_survived_cnt = df_train['Survived'].value_counts( )[0]
Survived_ratio = (Survived_cnt / df_train.shape[0]) *100
not_survived_ratio = (Not_ "                    ) *100

plt.figure (figsize = (10,8) )
sns.countplot ( df_train['Survived'] )
                                          not_
plt.xticks ( (0,1) , [f'Not Survived ({Survived_ratio:.2f3)', f'Survived_ratio(    )']
plt.title (              )
plt.show( )
```

1.4 Corelations.

```
df_train_corr = df_train.drop(['PassengerId'], axis=1 ).corr( ).abs( ).
                    unstacked( ).sort_values (Kind='quicksort', ascendig=False).
                    reset_index( )
df_train_corr.rename (colums = {'level_0':'feature 1',   ...  3, inplace=True )
df_train_corr.drop (df_train_corr.iloc[1::2].index , inplace = True )
       "            (df_train_corr[df_train_corr['Correlation Coefficient']== 1].index,
                       inplace = True )

df_test_corr
```

NO.

df-test도 동일.

```
corr = df-train-corr['Correlation Cefficient'] > 0.1
df-train-corr [corr]
```

Heatmap 시각화.

```
f, ax = plt. figure (nrows = 2, figsize = (20,20) )

sns. heatmap (df-train. drop(['PassengerId'], axis=1). corr( ),
              ax = ax[0], annot = True, square = True,
              annot_kws = {'size': 14} )
sns. heatmap (df-test. drop (        "          )

plt.show( )
```

ticks label 수정

1.5 Target Distribution in Features.

1.5.1 Continuous Features - 'Fare' & 'Age'

```
Cont_features = ['Fare', 'Age']
surv = df- train ['Survived'] == 1

f, ax = plt. subplots (nrows=2, ncols=2, figsize = (20,20) )
plt.subplots_adjust (right = 1.5 )

for i, feature in enumerate ( Cont_features ):

    sns. histplot ( df_train [~surv][feature], ax=ax[0][i],
                    hist = True, label = 'Not Survived', color = ' ' )
    sns. histplot ( df-train [surv][feature], ax = ax [0][i],    "    )
```

DESIGN K

```
    sns.histplot ( df_train[feature], hist = False, ax = ax[1][i],
                   label = 'Training Set', color = ' ' )
    sns.histplot ( df_test [feature], label = 'Test Set', ' )

조정 { ticks
       legend
       tick_params
       set_title

plt.show( )
```

1.5.2 Categorical Features.

- 'Pclass', 'Sex' features — homogeneity distributions Why? [Hasty generalization to say]

```
Cat_features = ['Embarked', 'Parch', 'Pclass', 'Sex', 'Sibsp', 'Deck']

f, ax = plt.subplots (nrows = 2, ncols = 3, figsize = (20, 20) )
plt.subplots_adjust ( right = 1.5, top = 1.25 )

for i, feature in enumerate ( Cat_features, 1 ):

    plt.subplot (2, 3, i)
    sns.countplot ( x = feature, data = df_train, hue = 'Survived' )

조정 { label
       tick_params
       legend
       title

plt.show( )
```

2. Feature Engineering.

2.1 Bining continuous features.

2.1.1 Fare.

- This feature is positively skewed, and survival rate is extremely high on the right end.

```
# Quantile
df_all ['Fare'] = pd.qcut (df_all['Fare'], 13)
```

Why 13? empirical data?

```
# Visualization.
f, ax = plt.subplots (figsize = (22,9) )
sns.countplot ( x= 'Fare', hue= 'Survived', data = df_all )

조정. { xlabel
        ylabel
        tick_params
        title
        legend

plt.show( )
```

2.1.2 Age.

- This feature shows normal distribution.

```
# Quantile
df_all ['Age'] = pd.qcut ( df_all['Age'], 10)
```

X

NO.

2.2 Frequency Encoding

- 'Family_size' feature which is derived feature from Parch, Sibsp is also able to predict the survival rate.

◦ Family_size

1	Alone
2,3,4	Small
5,6	Medium
7~	Large.

```
df_all['Family_Size'] = df_all['Parch'] + df_all['Sibsp'] + 1


f, ax = plt.subplots(nrows=2, ncols=2, figsize=(20,20))
plt.subplots_adjust(right=1.5)


sns.barplot(x=df_all['Family_Size'].value_counts.index,
            y=      〃                          .values,
            ax=ax[0][0])
sns.countplot(x='Family_Size', hue='Survived', data=df_all, ax=ax[0][1])
                                   ↳ df_test Setalz 'Survived' x

family_map = {1 : 'Alone',  ~   }
df_all['Family_Size_Grouped'] = df_all['Family_Size'].map(family_map)


      x


plt.show()
```

NO.

2.3 Title & Is married

- 'Title' is created by extracting the prefix before 'Name'
- 'Is married' is binary feature based on 'Mrs' title.

```
df_all ['Title'] = df_all ['Name'].str.split(', ', expand=True)[1]
                    .str.split('.', expand=True)[0]
df_all ['Is_Married'] = 0
df_all ['Is_Married'].loc [df_all ['Title'] == 'Mrs'] = 1

# Visualization (Before & After Grouping the title)
f, ax = plt.subplots (nrows=2, figsize=(20,20))
sns.barplot (x=df_all ['Title'].value_counts.index,
        y=        "               .values, ax=ax[0])

df_all ['Title'] = df_all ['Title'].replace( [  ~  ], 'Miss/Mrs/Ms' )
              "              ( [      ],                 )
   "

plt.show( )
```

2.4 Target Encoding.

- Creating 'Family' feature to group passengers in the same family by using 'Name' feature.

DESIGN K

```
def extract_surname ( data ) :

    families = [ ]

    for i in range ( len(data) ) :
        name = data.iloc[i]

        if '(' in name :
            name_no_brace = name.split('(')[0]
        else :
            name_no_brace = name

        family = name_no_brace.split(', ')[0]

        for c in string.punctuation :
            family = family.replace( c, '').strip( )

        families.append( family )

    return families

df_all['Family'] = extract_surname( df_all['Name'] )
df_train, df_test = divide_df( df_all )
```

NO.

```
# Creating 'Survival rate' & 'Survival rate_NA' features.

non_unique_familes = [x for x in df_train['Family'].unique() \
                                  if x in df_test['Family'].unique() ]

non_unique_tickets = [              〃         ['Ticket']  〃
                                  〃      ['Ticket']   〃      ]

median? { df_family_survival_rate = df_train.groupby('Family')['Surived', 'Family',
                                                      'Family_Size'].median()
        { df_ticket_survival_rate =         〃

family_rates = { }
ticket_rates = { }

{ for i in range( len(df_family_survival_rate) ):
{     if ①df_family_survival_rate.index[i] in non_unique_familes and
{         ②df_family_survival_rate.iloc[i, 1] >1 :
{         family_rates[ ①] = ② .iloc[i,0]

→ Ticket too
mean_survival_rate = np.mean (df_train ['Survived'])

train_family_survival_rate = [ ]
      "                    _NA = [ ]
test_       "               = [ ]
      "                    _NA = [ ]

for i in range ( len (df_train) ) :

    if df_train ['Famiy'][i] in family_rates :
        train_family_survival_rate. append (family_rates [df_train ['Family'] [i] ] )
        train_family_survival_rate_NA. append ( 1 )
    else :
        train_family_survival_rate. append ( mean_survival_rate )
                 "             _NA. append ( 0 )

For test set, too

    7

df_train ['Family_survival_rate'] = train_family_survival_rate
      "                       _NA'] =      "                   _NA

For test set, too
    → 7

    → For 'Ticket' feature, too

        7
```

```
for df in [df_train, df_test]:
    df['Survival_Rate'] = (df['Ticket_Survival_Rate'] + df['Family_Survival_Rate'])/2
    df[   "        _NA'] =        ;
```

2.5 Feature Transformation.

2.5.1 Label encoding Non-Numerical features to Numeric

- Object type : 'Embarked', 'Sex', 'Deck', 'Title',
  'Family_Size_Grouped'
- Category type : 'Fare', 'Age'

⇒ LabelEncoder

```
non_numeric_features = [                ]
```

'df_all'로 transforming 하는 편이 리소스 관리 측면에서 효율적일듯?
(1) fit_transform

```
for df in dfs:
    for feature in non_numeric_features:
        df[feature] = LabelEncoder().fit_transform(df[feature])
```

2.5.2 One-Hot Encoding the categorical features

```
cat_features = ['Pclass', 'Deck', 'Sex', 'Family_Size_Grouped',
                'Embarked', 'Title']
encoded_features = [ ]
```

Series → Array

```
for df in dfs :
    for feature in cat_features :
        encoded_feat = OneHotEncoder( ). fit_transform (
                       df[feature]. values.reshape (-1,1) ). toarray( )
        n = df[feature ]. nunique( )
        cols = [f'{feature}_{i}' for i in range( 1, n+1) ]
        encoded df = pd. DataFrame ( encoded feat, columns = cols )
        encoded_df. index = df. index
        encoded_features. append ( encoded_df )

df_train  = pd. concat ([df_train, *encoded features [ :6] ], axis =1 )
df_test   =                  "             [6: ]
```

2.6 Conclusion.

3. Model.

```
                                                          drop.
           ①
X_train = StandardScaler( ). fit_transform ( df_train. drop ( columns = cols) )
y_train = df_train ['Survived']. values.
X_test  = ① "
```

3.1 Random Forest.

- Compare Single model with leaderboard model, and try to experiment hyperparameter tuning

NO.

```
Single_best_model = RandomForestClassifier ( criterion = 'gini',
                                              n_estimator = 1100, # 생성 트리 갯수
                                              max_depth = 5,
                                              min_samples_split = 4,
                                              min_samples_leaf = 5,
                                              max_features = 'auto',
                                              oob_score = True,
                                              random_state = SEED,
                                              n_jobs = -1
                 # Printing 'wordy' informations ~ verbose = 1 )


N = 5      # KFold num.
oob = 0
probs = pd.DataFrame ( np.zeros( (len(X_test), N*2) ),
                               columns = [ 'Fold{}_Prob_{}'.format (i, j)
                                           for i in range(1, N+1)
                                           for j in range(2) ])
importances = pd.DataFrame ( np.zeros( (X_train.shape[1], N) ),
                                   columns = ['Fold_{}'.format (i) for i in range(1,N+1
                                   index = df_all.columns )

fprs, tprs, scores = [], [], []

skf = StratifiedKFold ( n_split = N, random_state = N, shuffle = True )
for fold, (trn_idx, val_idx) in enumerate( skf.split( X_train, y_train), 1):

    print(f'Fold {fold}')

    # Fitting the model.
    leaderboard_model.fit ( X_train [trn_idx], y_train [val_idx] )

    # Computing Train AUC score.
    trn_fpr, trn_tpr, trn_thresholds = roc_curve (y_train [trn_idx],
                                                  leaderboard_model.predict_proba
                                                  (X_train [trn_idx])[:, 1] )

    trn_auc_score = auc (trn_fpr, trn_tpr)
    # Computing validation AUC score.
    val_fpr, val_tpr, val_thresholds = roc_curve (y_train [val_idx],
                                                  leaderboard_model.predict_proba
                                                  (X_train [val_idx])[:, 1] )
    val_auc_score = auc (val_fpr, val_tpr)

    scores.append ( (trn_auc_score, val_auc_score) )
    fprs.append ( val_fpr)
    tprs.append ( val_tpr )

    # X_test prob.                      .format(fold)
    probs.loc [:, 'Fold_{}_prob_0' ] = leaderboard_model.predict_proba(
                                                  X_test) [:, 0] )
         "                           1

    importances.iloc [:, fold-1] = leaderboard_model.feature_importances_
```

```
        oob += leaderboard_model.oob_score_ / N
        print(f'Fold {fold} OOB score: {leaderboard_model.oob_score_}')

print(f'Ave OOB score: {oob}')
```

3.2 Feature Importances

```
importances['Mean_importances'] = importances.mean(axis=1)
importances.sort_values(by='Mean_importances', ascending=False,
                        inplace=True)

sns.barplot(x='Mean_importances', y=importances.index, data=importances)
plt.show()
```