Titanic - Advanced Feature Engineering Tutorial

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

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
```

```
import String
import warnings
warnings.filterwarnings('ignore')
SEED = 42
```
} Default

0. Loading Data

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

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

```
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} colums 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)
                (이름 안줌)                          , inplace=True.
df_all_corr.rename(columns={'level_0': 'Feature_1', ... 3})
df_all_corr[df_all_corr['Feature_1'] == 'Age']
# 'Age'와 'Pclass'의 상관 관계 가장 높음. 하지만 이것만으로 feature engineering
하기엔 너무 성급함. 정확성을 높이기 위해 'Sex' feature도 고려하여 missing value 채울.
                    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'].unique().tolist():
        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 경우인 (1, 'female')과
같은 이름 안붙은 X로 받음.
```

NO.

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 정보를 확인.

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

```
def get_pclss_dist(df):

    deck_counts = {'A': {3, ... 3
    decks = df.columns.levels[0]
```

```
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 위치 지정.
    bar_width = 0.85

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

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

NO.

```
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 간의 관계 확인

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'])

plt.xticks((0,1), [f'Not Survived ({not_Survived_ratio:.2f}%)', f'Survived_ratio(  )'])
plt.title(          )
plt.show()
```

1.4 Correlations.

```
df_train_corr = df_train.drop(['PassengerId'], axis=1).corr().abs().
                unstacked().sort_values(kind='quicksort', ascending=False).
                reset_index()
df_train_corr.rename(columns={'level_0': 'feature 1', ... }, 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
```

7

NO.

df_test도 동일.

```
corr = df_train_corr['Correlation Coefficient'] > 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(  "  )
```

ticks label 제외

```
plt.show()
```

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],
                 kist=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[i][i],
             label='Training Set', color='')
sns.histplot(df_test[feature], label='Test Set', '')
```

꾸밈 { 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')
```

꾸밈 { label, tick_params, legend, title }

```
plt.show()
```

NO.

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)
```

328
- 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)
```

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 넣으면 'Survived' X

family_map = {1:'Alone', ~ 3
df_all['Family_Size_Grouped'] = df_all['Family_Size'].map(family_map)

plt.show()
```