

## 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):
```

```

    # Index 32
    # 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} columns missing values : {df[col].isnull().sum()}')
```

```
for df in dfs:
```

```
    print(df.name)
```

```
    display_missing(df)
```



## 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 변경) , 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 관계가 가장 높음. 하지만 이것만으로 feature engineering  
하기엔 너무 부족함. 정확성을 높이기 위해 'Sex' feature도 고려하여 missing value 처리.

↓ 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 객체의 Index 값을 받음. 위의 경우인 (1, 'female')과  
같은 이름 인덱스로 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 상자를 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]
```



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(1/en(deck_names)) } # Xticks labeled every 1/16.
```

```
    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 "에 위의 과정이 동일함"



- Target value 설정화 (수치, 비율)

```
Survived_cnt = df_train['Survived'].value_counts()[1]
```

```
not_survived_cnt = df_train['Survived'].value_counts()[0]
```

$$\text{Survived\_ratio} = (\text{Survived\_cnt} / \text{df\_train.shape}[0]) * 100$$
$$\text{Not-survived ratio} = \left( \frac{\text{Not-survived}}{\text{Total}} \right) \times 100$$

plt.figure(figsize=(10,8))

```
sns.countplot(df_train['Survived'])
```

pt. x ticks (0,1), [<sup>not</sup> 'Not Survived' (Survived ratio : 2/3), 'Survived ratio' ( )]

plt. title ( )

At. 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 = {'eael\_o': 'feature1', ... }, 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)
```

 $\Delta$ -test-corr

7

```
df_test & { corr = df_train_corr['Correlation Coefficient'] > 0.1
sol.      { df_train_corr[corr]
```

```
# Heatmap
```

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

```
sns.heatmap(df_train.drop(['PassengerId'], axis=1).corr(),
```

```
ax=ax[0], annot=True, square=True,
```

```
annot_kws={'size': 14})
```

```
ticks label } sns.heatmap(df_test.drop( " " )
```

```
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],
```

```
hist=True, label='Not Survived', color=' ')
```

```
sns.histplot(df_train[surv][feature], ax=ax[0][i], " ")
```



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

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

329 { ticks  
legend  
tick\_params  
set\_title

```
plt.show()
```

### 1.5.2 Categorical Features

• 'Pclass', 'Sex' features

Why? Hasty  
homogeneity distributions 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')
```

330 { 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)`

320 {  
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



## 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=df_all['Survived'].value_counts().values,
```

```
ax=ax[0][0])
```

```
sns.countplot(x='Family-Size', hue='Survived', data=df_all, ax=ax[0][1])
```

↳ df\_test set of 'Survived' x

```
family_map = {1: 'Alone', ~ 3}
```

```
df_all['Family-Size-Grouped'] = df_all['Family-Size'].map(family_map)
```

```
plt.show()
```

### 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.



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)

# Creating 'Survival rate' & 'Survival rate\_NA' features.

```
non-unique-families = [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')['Survived', 'Family',
                                                'Family-Size'].median()
df_ticket-survival-rate = "
```

family-rates = { }

ticket-rates = { }

```
for i in range(len(df_family-survival-rate)):
    if ① df_family-survival-rate.index[i] in non-unique-families and
        ② df_family-survival-rate.iloc[i, 1] > 1:
        family-rates[i] = ② - iloc[i, 0]
```

• Ticket too