

Titanic

NO.

Library

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

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

```
import string
```

```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
SEED = 42
```

```
def concat_df(train_data, test_data):
```

```
    return pd.concat([train_data, test_data], sort=True).reset_index(
        drop=True)
```

```
def divide_df(all_data)
```

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


1.2.1 Age.

```
df-all-corr = df-all-corr.corr().abs().unstack().sort_values(
    kind='quicksort', ascending=False).reset_index()
```

```
df-all-corr.rename(columns={'axel_0' : 'Feature 1', ... 3, inplace=True)
```

```
df-all-corr[df-all-corr['Feature 1'] == 'Age']
```

index 별

median값

찾아 올림.

```
age_by_pclass_sex = df-all.groupby(['Sex', 'pclass']).median()['Age']
```

```
df-all['Age'] = df-all.groupby(['Sex', 'pclass'])['Age'].apply(lambda x:
    x.fillna(x.median()))
```

1.2.2 Embarked.

```
df-all[df-all['Embarked'].isnull()]
```

```
df-all['Embarked'] = df-all['Embarked'].fillna('S')
```

1.2.3 Fare

['Fare']

```
med-fare = df-all.groupby(['pclass', 'parch', 'sibsp'])['Fare'].median()[3][0]
df-all['Fare'] = ( )
```

1.2.4 Cabin.

```
df-all['Deck'] = df-all['Cabin'].apply(lambda s: s[0] if pd.notnull(s)
    else 'N')
```

```
df-all-decks = df-all.groupby(['Deck', 'Pclass']).count().drop(columns=['...'])
```

```
def get_pass_dist(df):
```

```
    # Creating a dictionary for every passenger class count in every deck.
```

```
    deck_counts = {'A': {}, 'B': {}, ..., 'T': {}}
```

```
    decks = df.columns.levels[0]
```

```
    for deck in decks:
```

```
        for pclass in range(1, 4):
```

```
            try:
```

```
                count = df[deck][pclass][0]
```

```
                deck_counts[deck][pclass] = count
```

```
            except KeyError:
```

```
                deck_counts[deck][pclass] = 0
```

```
    # Percentage.
```

```
    df_decks = pd.DataFrame(deck_counts)
```

```
    deck_percentages = {}
```

```
    for col in df_decks.columns:
```

```
        deck_percentages[col] = [(count / df_decks[col].sum()) * 100
```

```
            for count in df_decks[col]]
```

```
    return deck_counts, deck_percentages
```



```
def display_pclass_dist(percentages):
```

```
    Runtime  
    df_percentages = pd.percentages, transpose)
```

```
    deck_names = ('A', ..., 'T')
```

```
    bar_count = np.arange(len(deck_names))
```

```
    bar_width = 0.85
```

```
    pclass1 = df_percentages[0]
```

```
    pclass2 = df_percentages[1]
```

```
    pclass3 = df_percentages[2]
```

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

```
    plt.bar(bar_count, pclass1, color='#659494', edgecolor='white',  
            width=bar_width, label='passenger class 1')
```

```
    plt.bar(" " , bottom=pclass1 " ")
```

```
    plt.bar(" " , bottom=pclass1 + pclass2
```

각 줄의 label 사이의 간격

```
    plt.xlabel('Deck', size=15, labelpad=20)
```

```
    plt.ylabel('Passenger Class Percentage', size=15, labelpad=20)
```

```
    plt.xticks(bar_count, deck_names) → X의 label 설정
```

```
    plt.tick_params(axis='x', labelsize=15)
```

```
    " (" " "y", " ")
```

```
    plt.legend(loc='upper left', bbox_to_anchor=(1,1), prop={'size':15})
```

```
    plt.title(' ', size=18, y=1.05)
```

```
    plt.show()
```

```
all_deck_count, all_deck_per = get_pchss_dist(df_all_decks)
display_pchss_dist(all_deck_per)
```

Passenger in the T deck is changed to A

```
idx = df_all[df_all['Deck'] == 'T'].index,
df_all.loc[idx, 'Deck'] = 'A'
```

```
df_all_decks_survived = df_all.groupby(['Deck', 'Survived']).count(),
                        drop(columns=[.....]),
                        rename(columns={ 'None': 'Count' }),
                        transpose()  ↳ 가산 column,  ↳ 빼기할 column.
```

```
def get_survived_dist(df):
```

Creating a dictionary for every survival count in every deck.

```
surv_counts = {'A': { }, ... }
```

```
decks = df.columns.levels[0]
```

```
for deck in decks:
```

```
    for survive in range(0, 2):
```

```
        surv_counts[deck][survive] = df[deck][survive].count()
```

```
df_surv = pd.DataFrame(surv_counts)
```

```
surv_percentages = { }
```

```
for col in df_surv.columns:
```

```
    surv_percentages[col] = (df_surv[col].sum() / df_surv[col].count()) * 100
```

```
    for count in df_surv[col].values:
```



```
return surv_counts, surv_percentages
```

```
def display_surv_dist(percentages):
```

```
    {
```

와 거의 동일.

```
all_surv_count, all_surv_per = get_survived_dist(df_all_decks_surv)
```

```
display_surv_dist(all_surv_per)
```

value 바뀜.

```
df_all['Deck'] = df_all['Deck'].replace(['A', 'B', 'C'], 'ABC')
```

```
}
```

```
df_all['Deck'], value_counts()
```

Dropping the Cabin feature

```
df_all.drop(['Cabin'], axis=1, inplace=True)
```

```
df_train, df_test = divide_df(df_all)
```

```
dfs = [df_train, df_test]
```

```
for df in dfs:
```

```
    display_missing(df)
```

1.3 Survival Distribution

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

```
not_survived = " " [0]
```

```
survived_per = survived / df_train.shape[0] * 100
```

```
not " not " " "
```

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

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

1.4 Correlations

```
fig, ax = plt.subplots(nrow=2, figsize=(20,20))
```

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

```
ax=ax[0], annot=True, annot_kws={'size': 14},
```

```
square=True, cmap='coolwarm')
```

```
" df_test "
```


1.5 Survival Distribution in Features.

```
cont_features = ['Age', 'Fare']
```

```
surv = df_train['Survived'] == 1
```

```
fig, axs = plt.subplots(ncols=2, nrows=2, figsize=(20, 20))
```

```
plt.subplots_adjust(right=1.5)
```

```
for i, feature in enumerate(cont_features):
```

```
    # Distribution of survival in feature
```

```
    sns.distplot(df_train[~surv][feature], label='Not Survived', hist=True,  
                 color=' ', ax=axs[0][i])
```

```
    sns.distplot(df_train[surv][feature], label='Survived',  
                 " ")
```

```
axs[0][i].set_xlabel(" ") } X2 label None. 1/4  
[1] " ( )
```

```
for j in range(2):
```

```
    axs[i][j].tick_params(axis='x', labelsize=20)
```

```
    " y'
```

1.5.2 Categorical Features

```
cat_features = [ ~ ]
```

```
fig, axs = plt.subplots(ncols=2, nrows=3, figsize=(20,20))
```

```
plt.subplots_adjust(right=1.5, top=1.25)
```

```
for i, feature in enumerate(cat_features, start=1):
```

```
    plt.subplot(2, 3, i)  # Categorical label
```

```
    sns.countplot(x=feature, hue='Survived', data=df_train)
```

1.6 Conclusion

```
df_all = concat_df(df_train, df_test)
```

```
df_all.head()
```


2. Feature Engineering

2.1 Binning the continuous features

2.1.1 Fare

타이타닉의 객실의 크기
→ 13개의 구간으로 나눔.

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

```
fig, axes = plt.subplots(figsize=(22, 9))
```

```
sns.catplot(x='Fare', hue='Survived', data=df_all)
```

} Option

2.1.2 Age

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

```
fig, axes = plt.subplots(figsize=(22, 9))
```

```
sns.catplot(x='Age', hue='Survived', data=df_all)
```

}

2.3 Title & Is married

```
df_all['Title'] = df_all['Name'].str.split(';', expand=True)[1].\
    str.split('.', expand=True)[0]
```

Why? {

```
df_all['Is_Married'] = 0
df_all['Is_Married'] += 1 / (df_all['Title'] == 'Mrs')
```

Subplot

```
sns.barplot(x=df_all['Title'], value_counts().index,
            y=      , values=      ,
            data=df_all, ax=axs[0])
```

2.4 Survival rate

```
def extract_surname(data):
```

```
    families = [ ]
```

```
    for i in range(len(data)):
```

```
        name = data.iloc[i]
```

```
        if 'C' in name:
```

```
            name_no_bracket = name.split('(')[0]
```

```
        else:
```

```
            name_no_bracket = name
```

family = name.no_bracket.split(',')[0]

title = " ".join(strip).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_all.ix[:890]

df_test = " " [891:]

dfs = [df_train, df_test]

Creating a list of families and tickets

that are occurring in both training & test sets.

non-unique-families = [x for x in df_train['Family'].unique()

if x in df_test['Family'].unique()]

non-unique-tickets = [" " ['Ticket'] " [,]]

df_family_survival_rate = df_train.groupby('Family')['Survived', 'Family', 'Family_size']
 .median()

df_ticket = " " = " " ('Ticket') * 'Ticket'

family_rates = { }

ticket_rates = { }

for i in range(len(df_family_survival_rates)):

if df_family_survival_rates.index[i] in non_unique_families

and df_family_survival_rates.iloc[i,1] > 1:

family_rates[df_family_survival_rates.index[i]] = df_family_survival_rates.iloc[i,1]

Ticket → for

}

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

~ 전체 인원 중 몇 퍼센트의 인원이 살아남았는지?

train_survival_rate = []

family

= _NA = []

test_survival_rate = []

family

test = _NA = []

```
for i in range(len(df_train)):
```

```
    if df_train['Family'][i] in family_rates:
```

```
        train_family_survival_rate.append(family_rates[i])
```

```
    train_family_survival_rate.append(1)
```

```
else:
```

```
    train_family_survival_rate.append(mean_survival_rate)
```

```
    train_family_survival_rate.append(0)
```

```
df_test
```

```
7
```

```
df_train['Family_survival_rate'] = train_family_survival_rate
```

```
df_train['Family_survival_rate'] = train_family_survival_rate
```

```
df_test
```

```
//
```

2.5 Feature Transformation

← 2.5.1 Label Encoding

```
non_numeric_features = ['Embarked', 'Sex', 'Deck', ...]
```

```
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 = [~]

encoded_features = []

for df in dts:

for feature in Cat-features:

encoded_feat = OnehotEncoder().fit_transform(df[feature], values.

reshape(-1,1)).toarray()

n = df[feature].nunique()

cols = ['{ }_ { }'.format(feature, n) for n 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, ^{Unpacking the []} *encoded_features[:6]], axis=1)

df_test