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

NO.

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

df_train.name = 'Train Set'
df_test.name = 'Test Set'
df_all.name = 'All Set'

print('Number of training Sets : {}'.format(df_train.shape[0]))
print('  "      Test    "  :  ',  "
print('Training Sets X Shape : {}'.format(  *  x))
```

```
1. Exploratory Data Analysis.
  1.1 overview
print(df_train.info())
df_train.sample(3)

  1.2 missing values.
  def display_missing(df):
      for col in df.columns.tolist():
          print('{col, df[col].isnull().sum()} column missing values : {}')
          print('\n')

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

1.2.1 Age.

df_all_corr = df_all.corr().abs().unstack().sort_values(ko-
kind='quicksort', ascending=False).reset_index()

df_all_corr.rename(columns={'level_0': 'Feature 1', --- 3, inplace=T)

df_all_corr[df_all_corr['Feature 1'] == 'Age']

index 사용
median값 ← age_by_pclass_sex = df_all.groupby(['Sex', 'Pclass']).median()['Age']
채워 넣음.

df_all['Age'] = df_all.groupby(['Sex', 'Pclass'])['Age'].apply(lambda x:
x.fillna(x.median()))

1.22 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][1]

['Fare']

1.2.4 Cabin.

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

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

NO.

```
def get_pclass_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):
                    DataFrame
    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='#b5ffb9', edgecolor='white',
            width=bar_width, label='passenger class 1')
    plt.bar(  "              bottom=pclass1      "  )
    plt.bar(  "              bottom=pclass1+pclass2
                                    각 축의 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) → x축 label 변경
    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_pclss_dist(df_all_decks)
display_pclss_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()   ↳ 기존 column.  새로운 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][0]

    df_surv = pd.DataFrame(surv_counts)
    surv_percentages = { }

    for col in df_surv.columns:
        surv_percentages = [(count / df_surv[col].sum()) * 100
                            for count in df_surv[col]]
```

return surv_counts, surv_percentages.

def display_surv_dist (percentages):

위와 거의 동일.

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

value 바꾸기. 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=T)

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

```
sns.heatmap(df_train.drop(['PassengerID'], axis=1).corr(),
            ax=axs[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('')  } x축 label None. 생략.
[1]  "

for j in range(2):
    axs[i][j].tick_params(axis='x', labelsize=20)
                          'y'
```

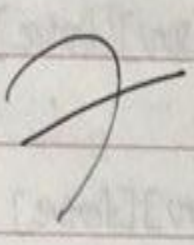
NO.

1.5.2 Categorical Features

```
cat_features = [      ~      ]

fig, axs = plt.subplots(ncols=2, nrow=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)          ← Categorial label
    sns.countplot(x=feature, hue='Survived', data=df_train)
```



1.6 Conclusion

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

NO.

2. Feature Engineering

2.1 Binning the continuous features

2.1.1 Fare

데이터의 갯수의 여러 / 13개의 구간으로 나눔.

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

fig, axes = plt.subplots(figsize=(22,9))
sns.countplot(x='Fare', hue='Survived', data=df_all)
```

Option

2.1.2 Age

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

fig, axs = plt.subplots(figsize=(22,9))
sns.countplot(x='Age', hue='Survived', data=df_all)
```

2.2 Family size & Ticket frequency.

```
df_all['Family_size'] = df_all['SibSp'] + df_all['Parch'] + 1

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20,20))
plt.subplots_adjust(right=1.5)      # Subplot 간 간격 조절.

sns.barplot(x=df_all['Family_Size'].value_counts.index,
            y=      "               .values,
            ax=axs[0][0])
sns.countplot(x='Family_Size', y='Survived',
              data=df_all, ax=axs[0][1])

axs[0][0].set_title('Family Size feature Value counts', size=20, y=1.05)
axs[0][1]    "    ('Survival counts in family size', size=20, y=1.05)

family_map = {1:'Alone', 2:'Alone', ... }
df_all['Family_Size_group'] = df_all['Family_Size'].map(family_map)
```

NO.

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'].loc[df_all['Title'] == 'Mrs'] = 1
```

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 length(data):
        name = data.iloc[i]

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

```
        family = name_no_bracket.split(',')[0]
        title = "                    [1].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.loc[:890]
df_test = "      [891:]
dfs = [df_train, df_test]

# Creating a list of families and tickets
      that are occuring 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'] "  [,]]

                    group
df_family_survival_rate = df_train['Family'][['Survived', 'Family', 'Family_size']]
                          .median()
df_ticket_survival_rate = "  = "('Ticket)  'Ticket'
```

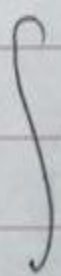
NO.

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_~.index[i]] = df_~.iloc[i,

Ticket → for



mean_survival_rate = np.mean(df_train['Survived'])
→ 전체 승객 중 몇 퍼센트의 승객이 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_survived_rate.append(family_rates[ ])
        train_family_ "    _NA.append(1)
    else:
        train_ _.append(mean_survival_rate)
            "    _NA.append(0)
```

df_test 도

```
df_train['Family_survival_rate'] = train_family_survival_rate
df_train['    "    _NA'] =  "      _NA
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 dfs:
    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, *encoded_features[:6]], axis=1)
df_test
```

Unpacking the [ ]

NO.

3. Machine Learning.

평균이 0, 표준편차 1이 되도록 변환.

X_train = StandardScaler().fit_transform(df_train.drop(columns = drop_cols))

y_train = df_train['Survived'].values

X_test = "

print('X_train shape : {}'.format(X_train.shape))
print('y_train shape : {}'.format(y_train.shape))
print('X_test shape : {}'.format(X_test.shape))

3.1 Models. (RF.)

single_best_model = RandomForestClassifier(criterion = 'gini',
n_estimators = 1100, # 생성할 트리 갯수
max_depth = 5,
노드를 분할하기 위한 최소 샘플 데이터의 수 ← min_samples_split = 4,
min_samples_leaf = 5,
max_features = 'auto',
'Out of Bag' ← oob_score = True,
random_state = SEED,
CPU 의 병렬 작업 갯수 ← n_jobs = -1,
printing 'wordy' informations ← verbose = 1)

all works.

```
N = 5    ~~ K-fold 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)])
importance = 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 = [], [], []
                                        (Split 안의 구성개수 설정.)
skf = StratifiedKFold(n_splits = N, random_state = N, shuffle = True)
                                        idx Start num
for fold, (trn_idx, val_idx) in enumerate(skf.split(X_train, y_train), 1):
    print('Fold_{}'.format(fold))        ('Stratified'의 중요!!)

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

    # Computing Train AUC ~~score~~.                    (중요!)
    trn_fpr, trn_tpr, trn_thresholds = roc_curve(y_train[trn_idx],
                                                 leaderboard_model.predict_proba
    trn_auc_score = auc(trn_fpr, trn_tpr)        (X_train[trn_idx])[:, 1])
    # Computing Validation AUC score.
```

?

```
scores.append((tm_auc_score, val_auc_score))
fprs.append(val_fpr)
tprs.append(val_tpr)

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

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

oob += leaderboard_model.oob_score_ / N
print('Fold_{} OOB score: {}\n'.
                format(fold, leaderboard_model.oob_score_))

print('Average OOB score: {}'.format(oob))
```