Kaggle 입문
&
1~2 주차

2017010698
수학과 오서영

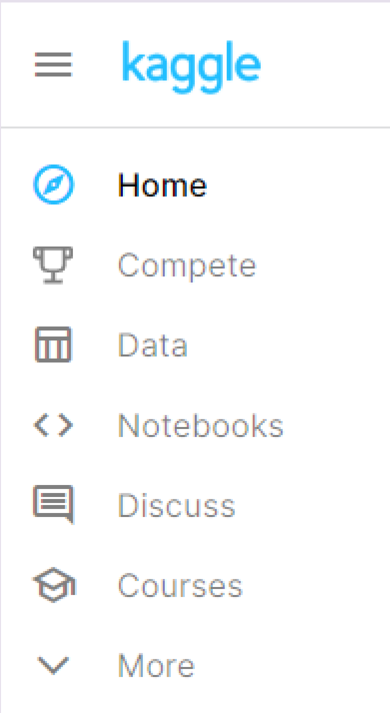
Contents

1. Kaggle 입문

Kaggle : 예측모델 및 분석 대회를 하는 플랫폼

https://www.kaggle.com/

1. Kaggle 입문



Compete (Competition)

: 현재 진행중인 또는 완료된 대회들을 볼 수 있다. 대회에 참가는 이 메뉴에서 한다.

Data

: 다른 공개된 데이터 셋

Notebooks (Kernel)

: 온라인 데이터 분석 환경 제공

Discuss

: 분석 관련 의견을 공유

Course

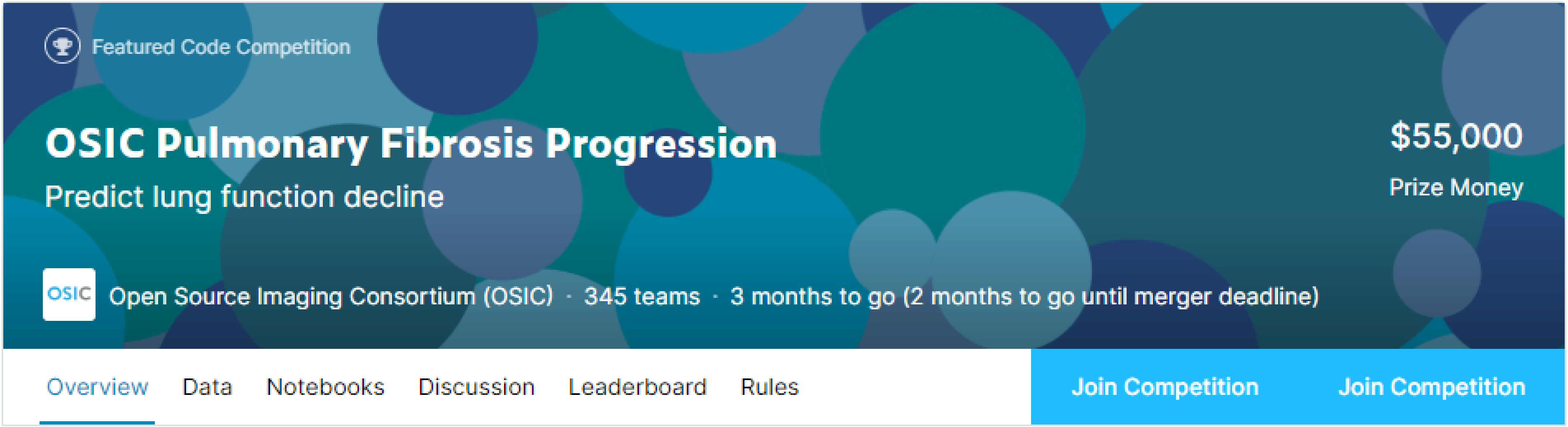
: 데이터 분석, 머신러닝 관련된 교육

1. Kaggle 입문

Competitions

1. 데이터를 다운받아서 내 PC에서 작업
2. 클라우드 서비스(Kernel)를 이용하는 것처럼 서버에 접속해서 작업

-> Overview, Description 확인



1. Kaggle 입문

Public Learderboard
Private Learderboard

: Competition이 종료되기 전에는
Public Learderboard에서 내 모델이 예측한 결과의
50%를 기준으로 Accuracy를 계산하고, 랭킹을 산정
-> Competition이 종료되면 나머지 50%의 결과까지 포함해서 랭킹을 산정하게 된다.

2. Titanic Tutorial

Titanic: Machine Learning from Disaster

타이타닉에 탑승한 사람들의 신상정보를 활용하여, 승선한 사람들의 생존여부를 예측하는 모델을 생성

https://www.kaggle.com/c/titanic

2. Titanic Tutorial

Titanic Tutorial

- Exploratory data analysis, visualization, machine learning
- Reference : [EDA To Prediction (DieTanic)], https://www.kaggle.com/ash316/eda-to-prediction-dietanic

1. Import Packages & Setting

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn')
sns.set(font_scale=1) # font scale
import missingno as msno  # install !

#ignore warnings
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

2. Titanic Tutorial

2. Explore dataset

```
df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
```

```
df_train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

check !

- feature : Pclass, Age, SibSp, Parch, Fare
    1. pclass : Ticket class (1>>3)
    2. sibsp : # of siblings
    3. parch : # of parents
    4. fare : Passenger fare
- target label to predict: Survived

2. Titanic Tutorial

```
df_train.describe()   # Generate descriptive statistics.
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

2. Titanic Tutorial

3. Null Data

```
for col in df_train.columns:
    msg = 'column: {:>10}₩t Percent of NaN value: {:.2f}%'.format(col, 100 *
    print(msg)                          (df_train[col].isnull().sum() / df_train[col].shape[0]))
```

```
column: PassengerId      Percent of NaN value: 0.00%
column:   Survived       Percent of NaN value: 0.00%
column:     Pclass       Percent of NaN value: 0.00%
column:       Name       Percent of NaN value: 0.00%
column:        Sex       Percent of NaN value: 0.00%
column:        Age       Percent of NaN value: 19.87%
column:      SibSp       Percent of NaN value: 0.00%
column:      Parch       Percent of NaN value: 0.00%
column:     Ticket       Percent of NaN value: 0.00%
column:       Fare       Percent of NaN value: 0.00%
column:      Cabin       Percent of NaN value: 77.10%
column:   Embarked       Percent of NaN value: 0.22%
```

2. Titanic Tutorial

Test Data

```
column: PassengerId     Percent of NaN value: 0.00%
column:      Pclass     Percent of NaN value: 0.00%
column:        Name     Percent of NaN value: 0.00%
column:         Sex     Percent of NaN value: 0.00%
column:         Age     Percent of NaN value: 20.57%
column:       SibSp     Percent of NaN value: 0.00%
column:       Parch     Percent of NaN value: 0.00%
column:      Ticket     Percent of NaN value: 0.00%
column:        Fare     Percent of NaN value: 0.24%
column:       Cabin     Percent of NaN value: 78.23%
column:    Embarked     Percent of NaN value: 0.00%
```

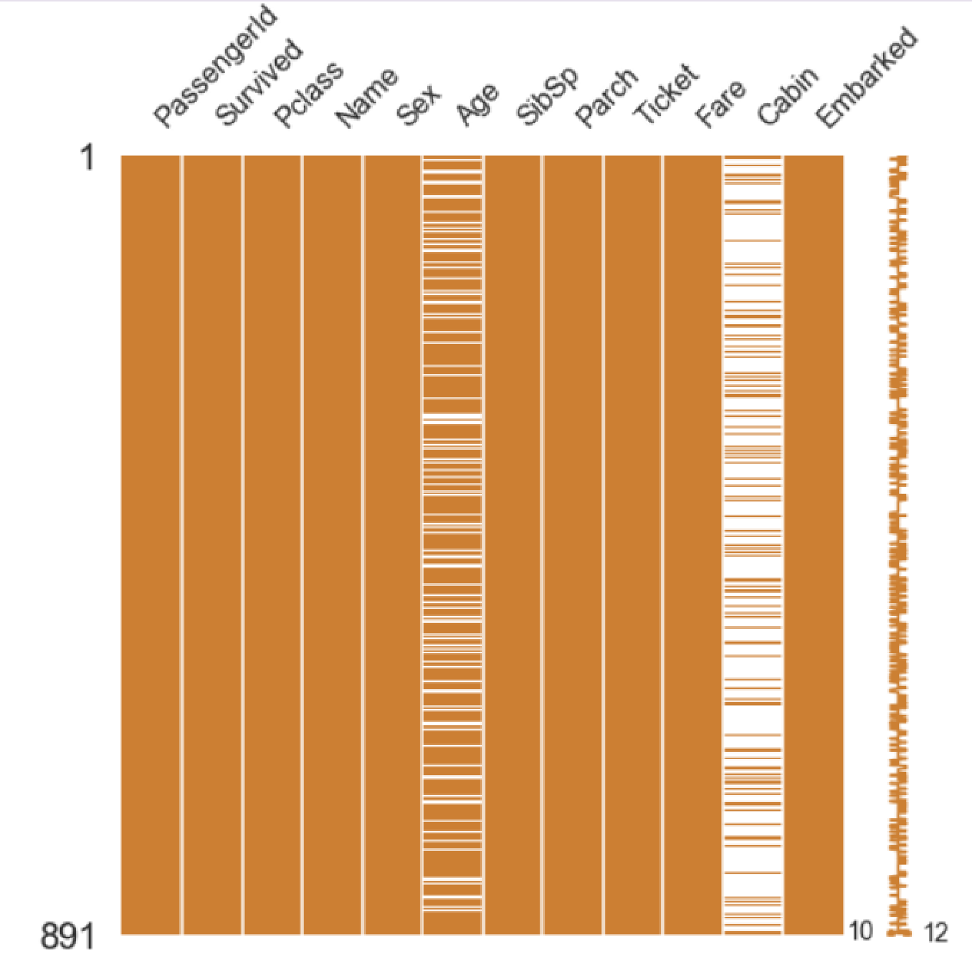
check! : percent of NAN value

- Age : 20% (both)
- Cabin : 80% (both)
- Embarked : 0.22% (only train)

2. Titanic Tutorial

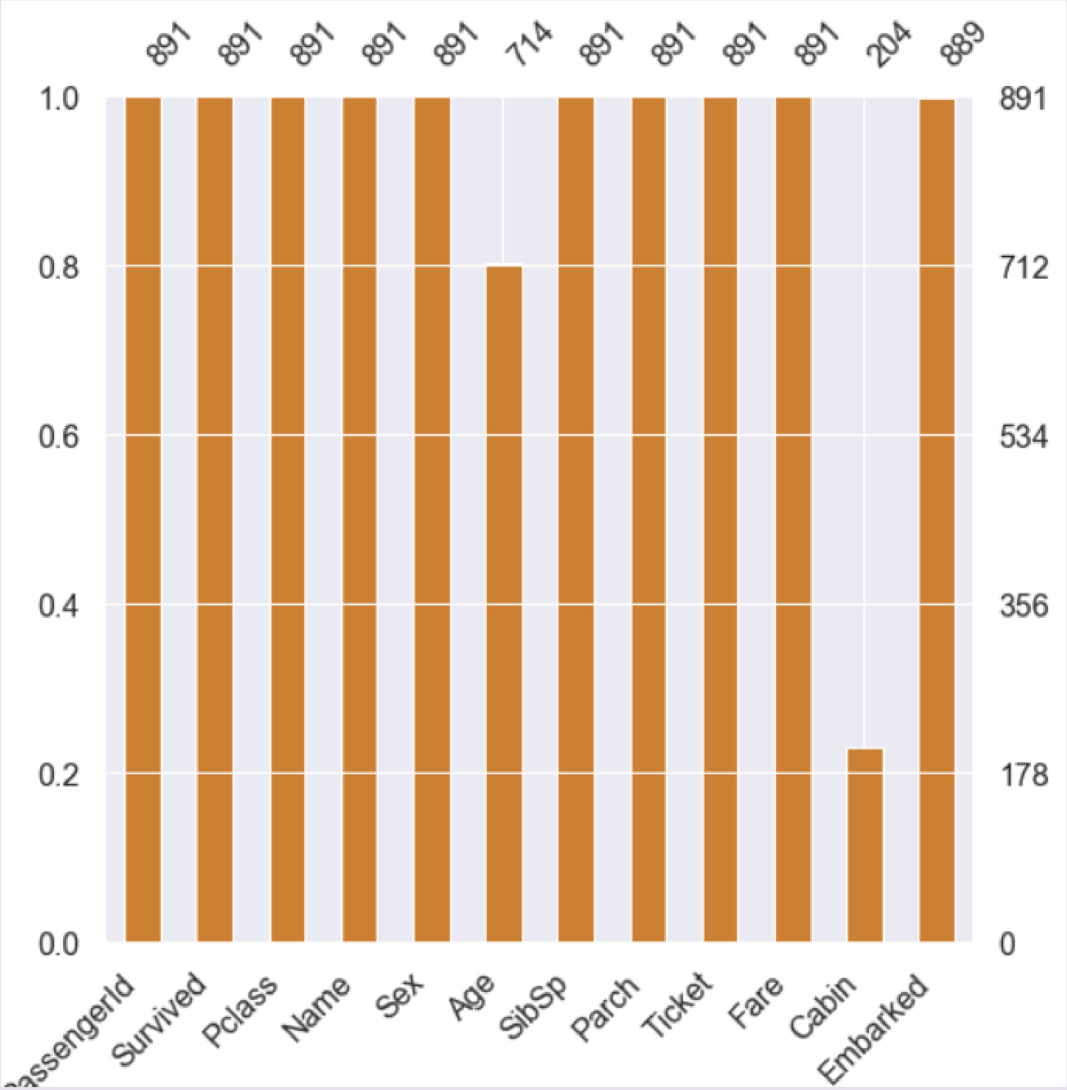
3-1. Visualization of NAN Value

```
msno.matrix(df=df_train.iloc[:, :], figsize=(8, 8), color=(0.8, 0.5, 0.2))
```



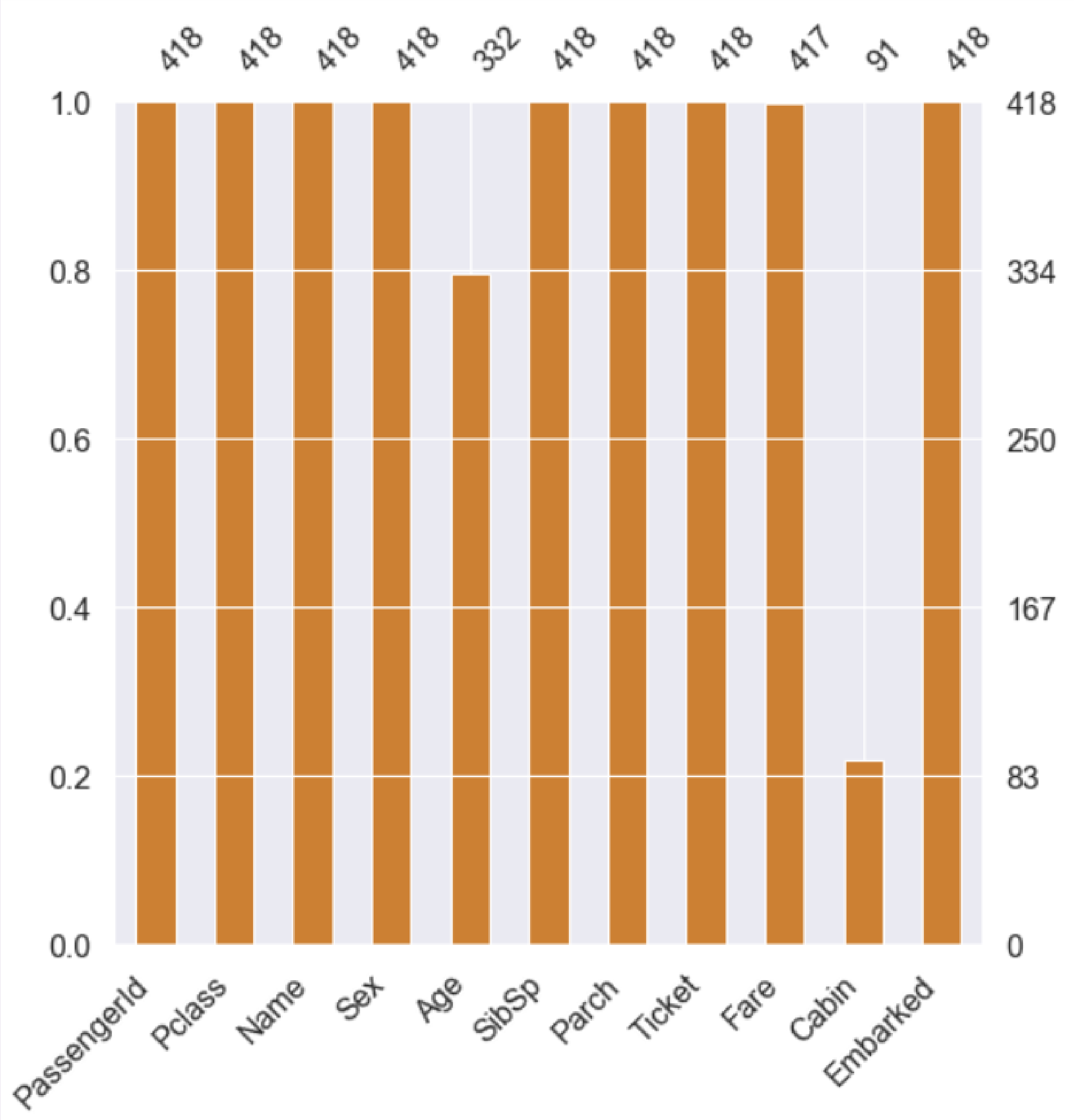
2. Titanic Tutorial

```
msno.bar(df=df_train.iloc[:, :], figsize=(8, 8), color=(0.8, 0.5, 0.2))
```



2. Titanic Tutorial

Test Data



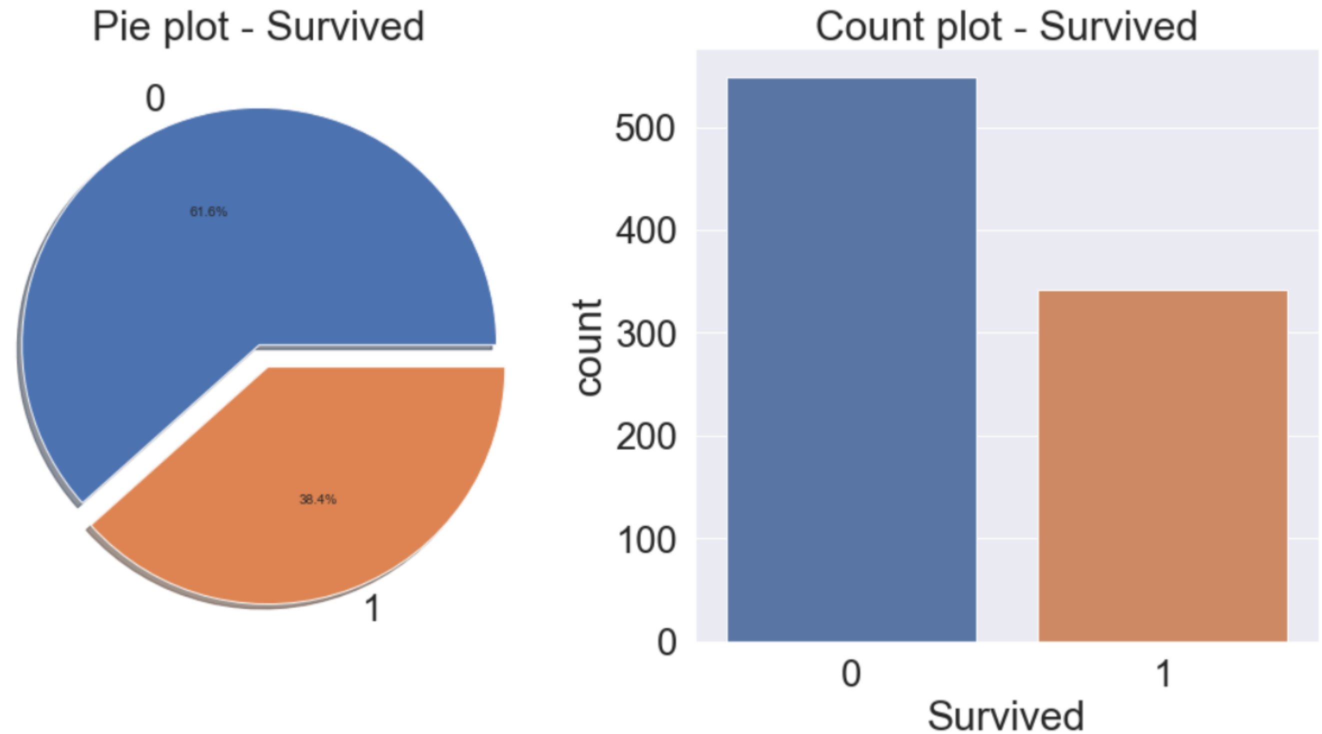
2. Titanic Tutorial

4. Explore "Target Label"

```
f, ax = plt.subplots(1, 2, figsize=(18, 8))

df_train['Survived'].value_counts().plot.pie(explode=[0, 0.1], autopct='%1.1f%%', ax=ax[0], shadow=True)
ax[0].set_title('Pie plot - Survived')
ax[0].set_ylabel('')
sns.countplot('Survived', data=df_train, ax=ax[1])
ax[1].set_title('Count plot - Survived')

plt.show()
```



2. Titanic Tutorial

5. Explore "Pclass" -> Ordinal Feature

```
df_train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=True).count()   ## all passenger
```

	Survived
Pclass	
1	216
2	184
3	491

```
df_train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=True).sum()   ## survived passenger (1)
```

	Survived
Pclass	
1	136
2	87
3	119

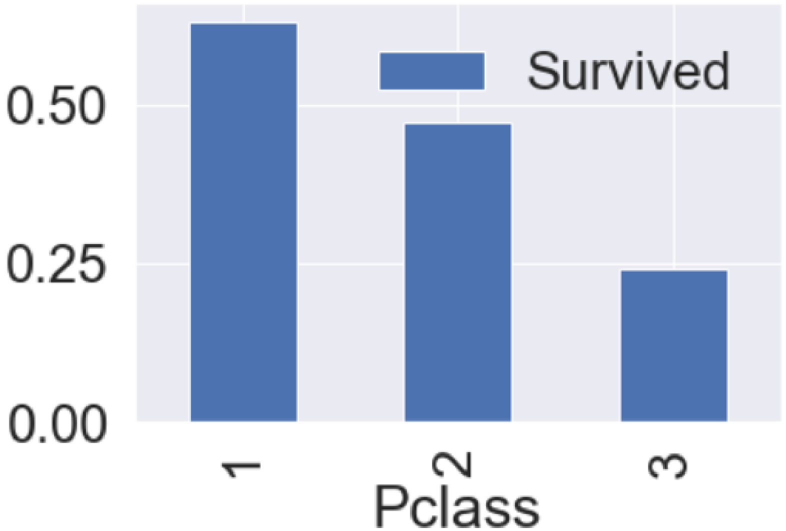
2. Titanic Tutorial

```
pd.crosstab(df_train['Pclass'], df_train['Survived'], margins=True).style.background_gradient(cmap='summer_r')
```

Survived	0	1	All
Pclass			
1	80	136	216
2	97	87	184
3	372	119	491
All	549	342	891

```
df_train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=True).mean().sort_values(by='Survived', ascending=False).plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1def90cbfc8>
```



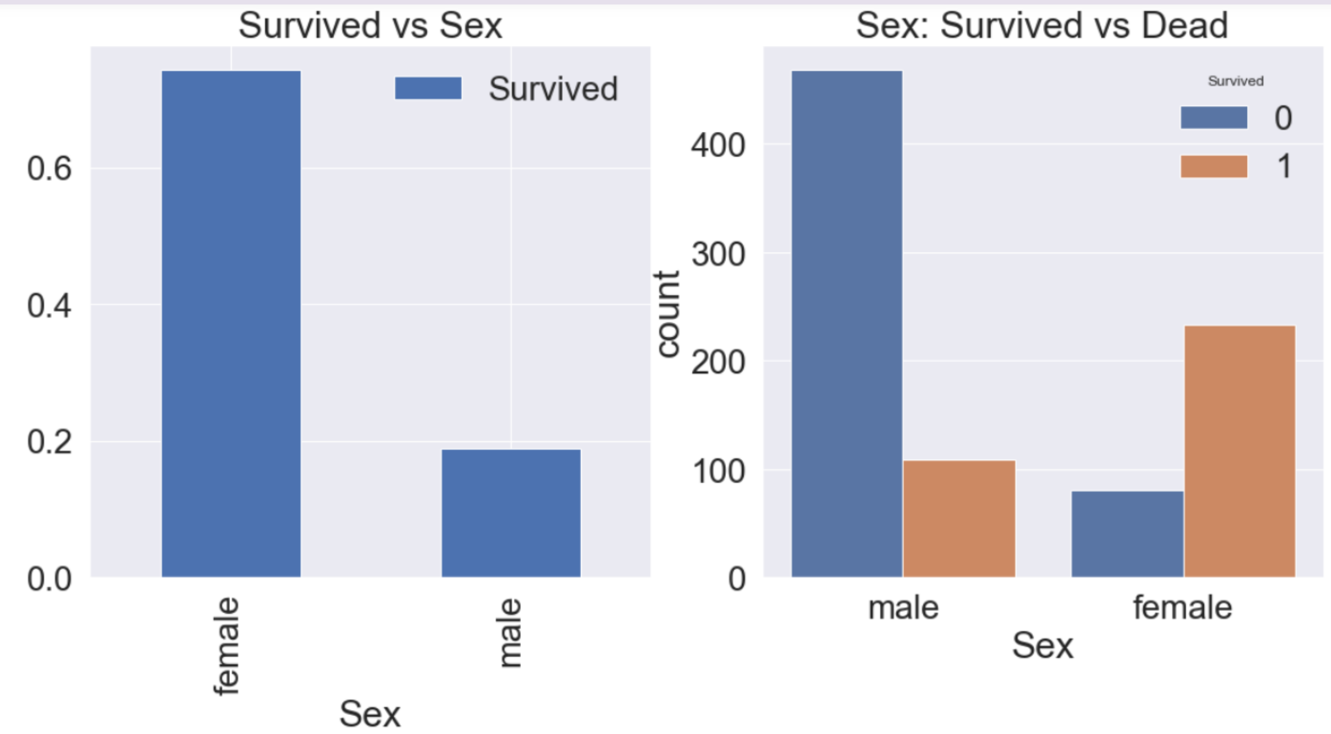
check!

- The better the Pclass, the higher the survival rate.

2. Titanic Tutorial

6. Explore "Sex" -> Categorical Feature

```
f, ax = plt.subplots(1, 2, figsize=(18, 8))
df_train[['Sex', 'Survived']].groupby(['Sex'], as_index=True).mean().plot.bar(ax=ax[0])
ax[0].set_title('Survived vs Sex')
sns.countplot('Sex', hue='Survived', data=df_train, ax=ax[1])
ax[1].set_title('Sex: Survived vs Dead')
plt.show()
```



2. Titanic Tutorial

```
df_train[['Sex', 'Survived']].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Sex	Survived
0	female	0.742038
1	male	0.188908

```
pd.crosstab(df_train['Sex'], df_train['Survived'], margins=True).style.background_gradient(cmap='summer_r')
```

Survived	**0**	**1**	**All**
Sex			
female	81	233	314
male	468	109	577
All	549	342	891

check!

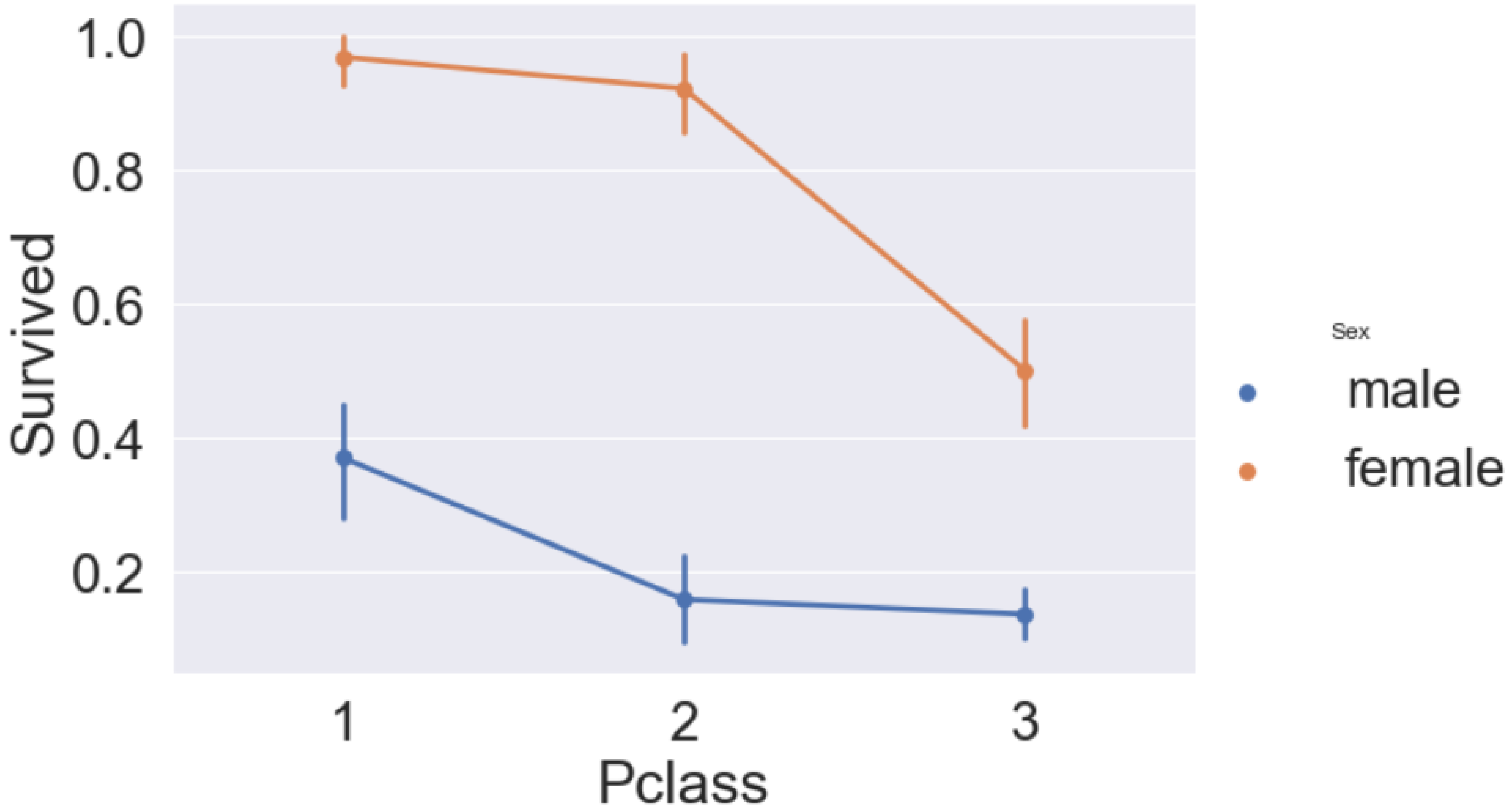
- Women are more likely to survive.
- "Pclass" and "Sex" are important features for the predictive model.

2. Titanic Tutorial

7. Both Sex and Pclass

```
sns.factorplot('Pclass', 'Survived', hue='Sex', data=df_train,
               size=6, aspect=1.5)
```

<seaborn.axisgrid.FacetGrid at 0x1def9178f88>



2. Titanic Tutorial

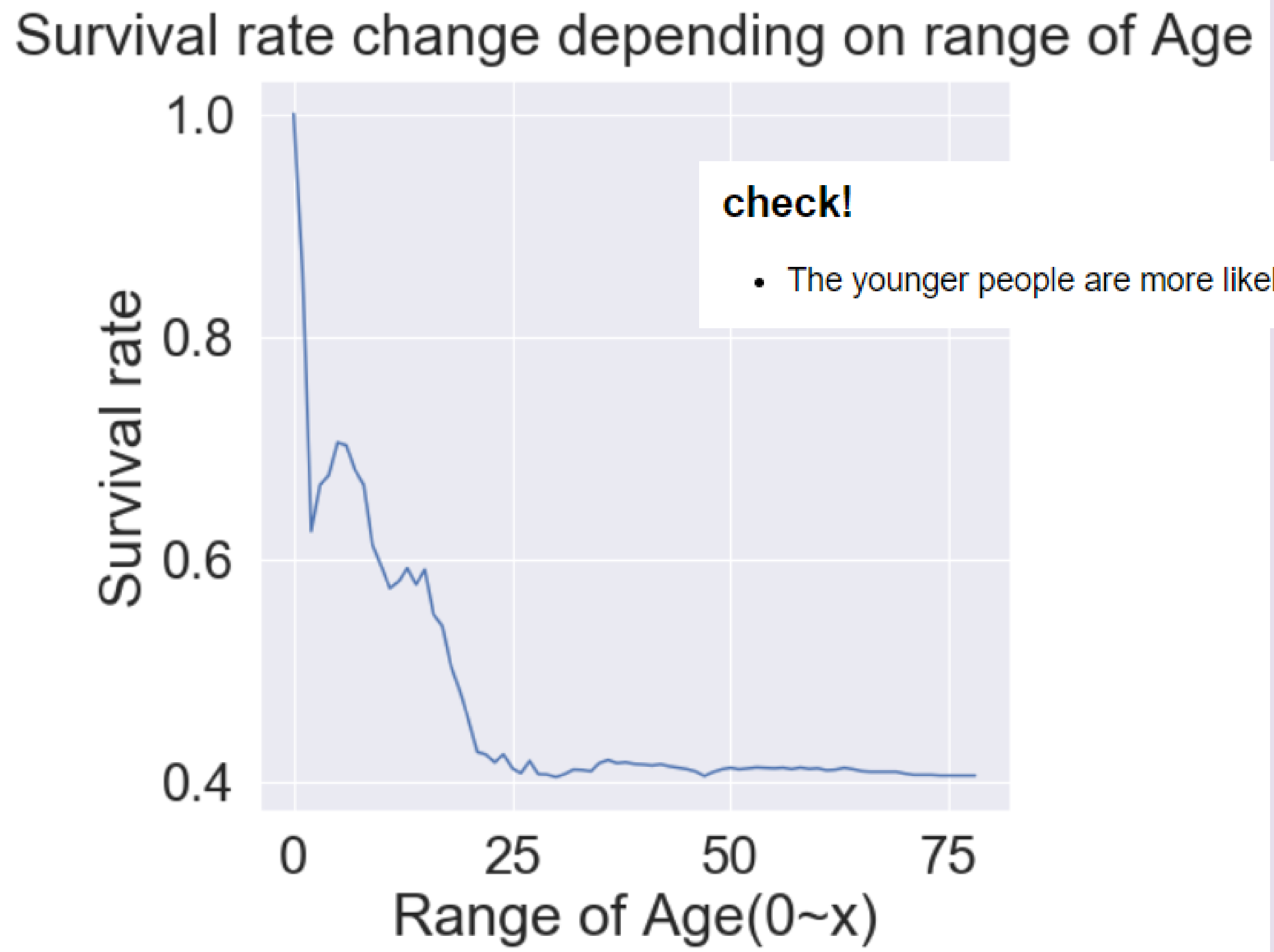
8. Explore "Age" -> Continous Feature

```
print('Oldest Passenger was of : {:.1f} Years'.format(df_train['Age'].max()))
print('Youngest Passenger was of: {:.1f} Years'.format(df_train['Age'].min()))
print('Average Age on the ship: {:.1f} Years'.format(df_train['Age'].mean()))
```

```
Oldest Passenger was of : 80.0 Years
Youngest Passenger was of: 0.4 Years
Average Age on the ship: 29.7 Years
```

```
cummulate_survival_ratio = []
for i in range(1, 80):
    cummulate_survival_ratio.append(df_train[df_train['Age'] < i]['Survived'].sum()
                                    / len(df_train[df_train['Age'] < i]['Survived']))
plt.figure(figsize=(7, 7))
plt.plot(cummulate_survival_ratio)
plt.title('Survival rate change depending on range of Age', y=1.02)
plt.ylabel('Survival rate')
plt.xlabel('Range of Age(0~x)')
plt.show()
```

2. Titanic Tutorial



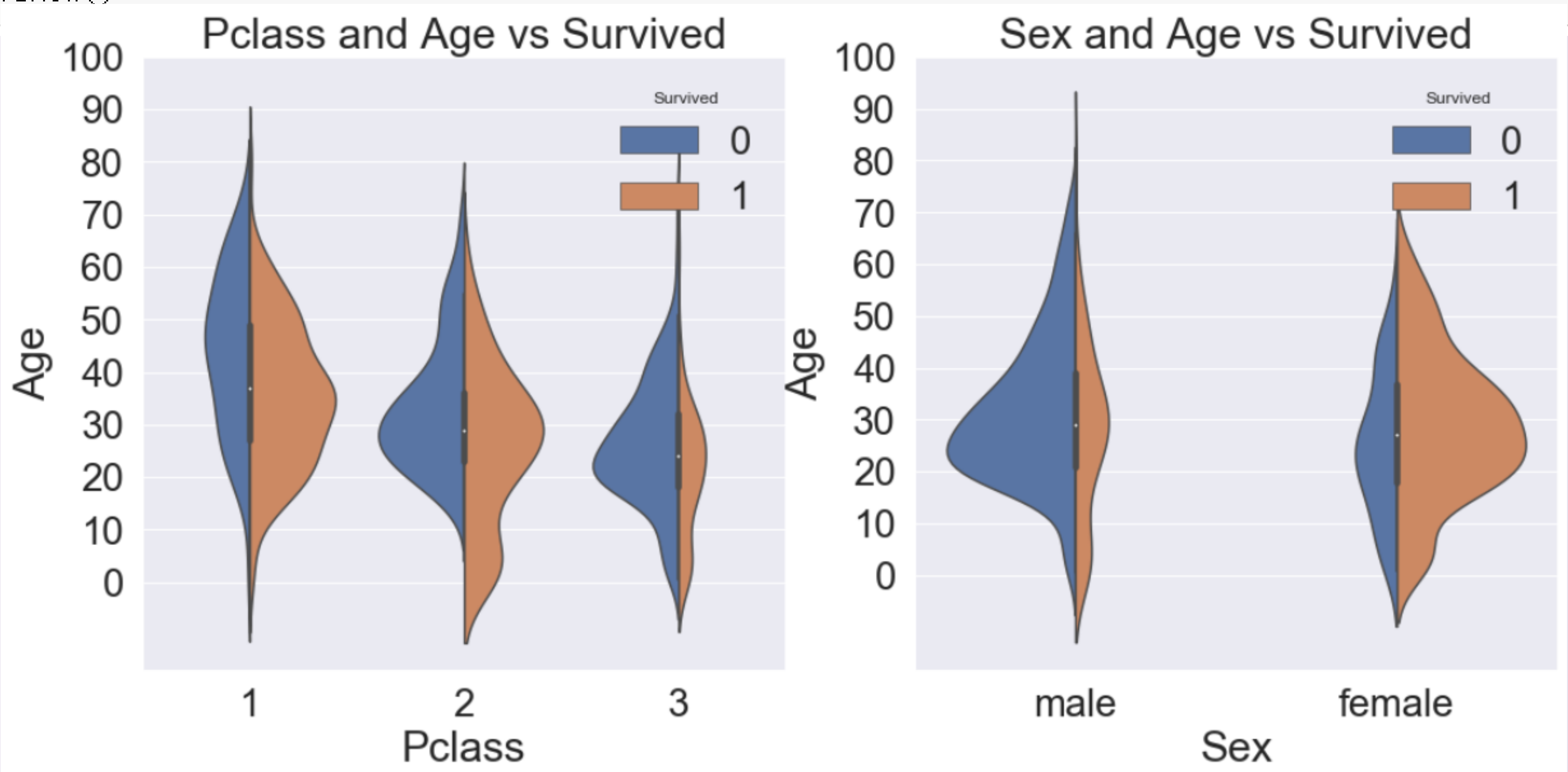
check!

- The younger people are more likely to survive.

2. Titanic Tutorial

9. Pclass, Sex, Age

```
f,ax=plt.subplots(1,2,figsize=(18,8))
sns.violinplot("Pclass","Age", hue="Survived", data=df_train, scale='count', split=True,ax=ax[0])
ax[0].set_title('Pclass and Age vs Survived')
ax[0].set_yticks(range(0,110,10))
sns.violinplot("Sex","Age", hue="Survived", data=df_train, scale='count', split=True,ax=ax[1])
ax[1].set_title('Sex and Age vs Survived')
ax[1].set_yticks(range(0,110,10))
plt.show()
```



2. Titanic Tutorial

10. Explore "Family - SibSp + Parch"

```
df_train['FamilySize'] = df_train['SibSp'] + df_train['Parch'] + 1 # 자신을 포함해야하니 1을 더합니다
df_test['FamilySize'] = df_test['SibSp'] + df_test['Parch'] + 1 # 자신을 포함해야하니 1을 더합니다
```

```
print("Maximum size of Family: ", df_train['FamilySize'].max())
print("Minimum size of Family: ", df_train['FamilySize'].min())
```

```
Maximum size of Family:  11
Minimum size of Family:  1
```

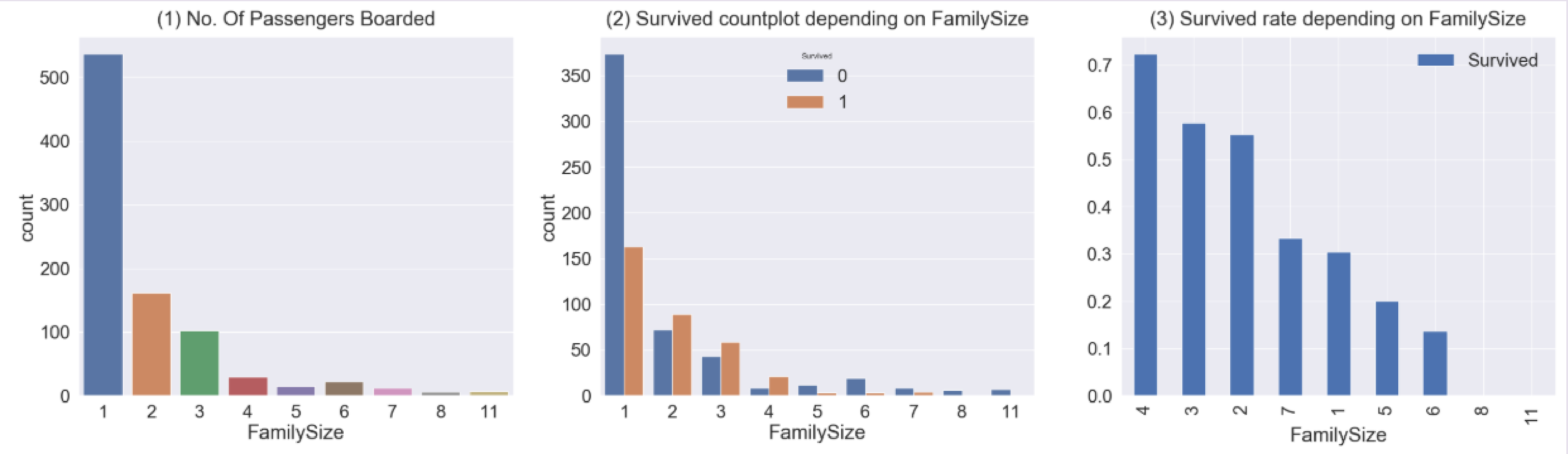
```
f,ax=plt.subplots(1, 3, figsize=(40,10))
sns.countplot('FamilySize', data=df_train, ax=ax[0])
ax[0].set_title('(1) No. Of Passengers Boarded', y=1.02)

sns.countplot('FamilySize', hue='Survived', data=df_train, ax=ax[1])
ax[1].set_title('(2) Survived countplot depending on FamilySize',  y=1.02)

df_train[['FamilySize', 'Survived']].groupby(['FamilySize'],
                                             as_index=True).mean().sort_values(by='Survived', ascending=False).plot.bar(ax=ax[2])
ax[2].set_title('(3) Survived rate depending on FamilySize',  y=1.02)

plt.subplots_adjust(wspace=0.2, hspace=0.5)
plt.show()
```

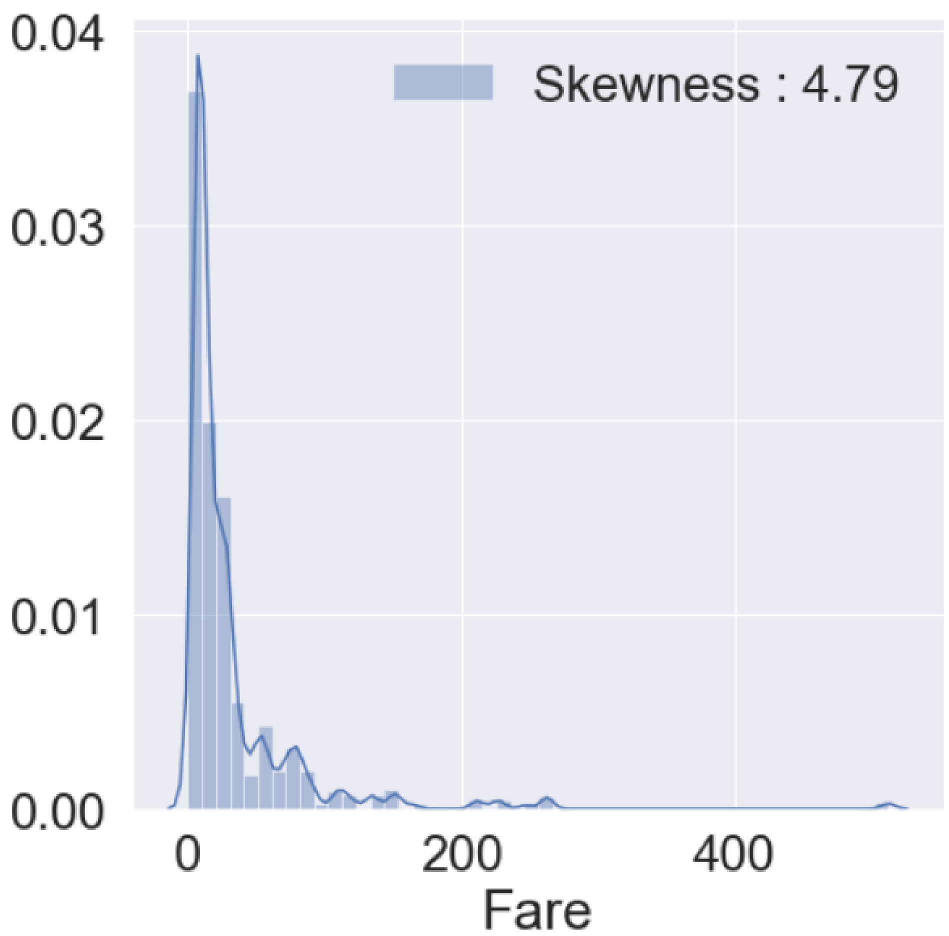
2. Titanic Tutorial



2. Titanic Tutorial

11. Explore "Fare"

```
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
g = sns.distplot(df_train['Fare'], color='b', label='Skewness : {:.2f}'.format(df_train['Fare'].skew()), ax=ax)
g = g.legend(loc='best')
```



check!

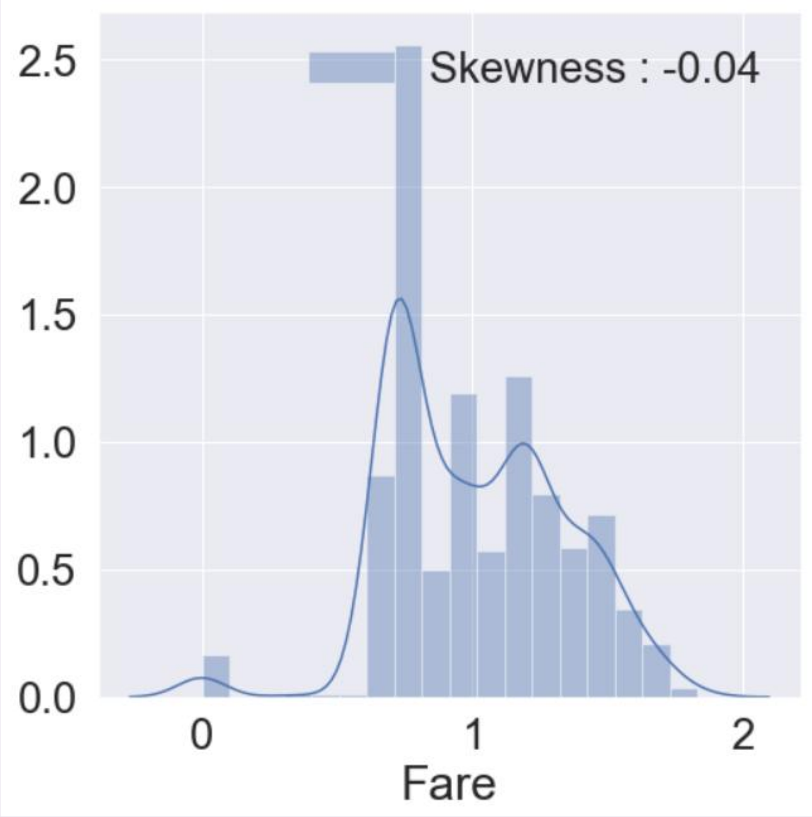
- high skewness -> outlier
- log function !

2. Titanic Tutorial

```
df_test.loc[df_test.Fare.isnull(), 'Fare'] = df_test['Fare'].mean() # NAN Value -> Mean Value

df_train['Fare'] = df_train['Fare'].map(lambda i: np.log(i) if i > 0 else 0)
df_test['Fare'] = df_test['Fare'].map(lambda i: np.log(i) if i > 0 else 0)
```

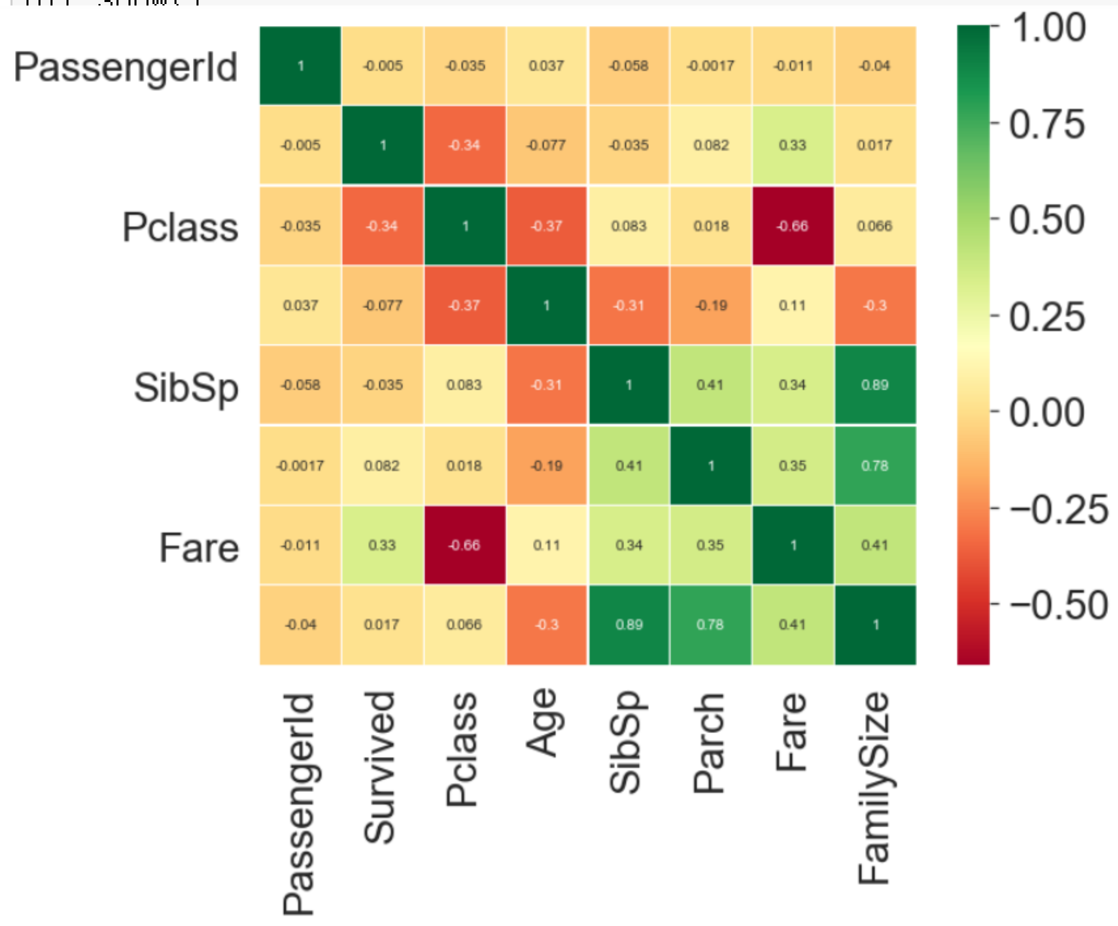
```
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
g = sns.distplot(df_train['Fare'], color='b', label='Skewness : {:.2f}'.format(df_train['Fare'].skew()), ax=ax)
g = g.legend(loc='best')
```



3. EDA to Prediction (DieTanic)

12. Correlation Between The Features

```
sns.heatmap(df_train.corr(),annot=True,cmap='RdYlGn',linewidths=0.2) #data.corr()-->correlation matrix
fig=plt.gcf()
fig.set_size_inches(10,8)
plt.show()
```



3. EDA to Prediction (DieTanic)

13. Predictive Modeling

1. Logistic Regression
2. Support Vector Machines(Linear and radial)
3. Random Forest
4. K-Nearest Neighbours
5. Naive Bayes
6. Decision Tree

3. EDA to Prediction (DieTanic)

13-1. Logistic Regression

```
model = LogisticRegression()
model.fit(train_X,train_Y)
prediction3 = model.predict(test_X)
print('The accuracy of the Logistic Regression is',metrics.accuracy_score(prediction3,test_Y))
```

```
The accuracy of the Logistic Regression is 0.8134328358208955
```

13-2. Support Vector Machines(Linear and radial)

```
model=svm.SVC(kernel='rbf',C=1,gamma=0.1)
model.fit(train_X,train_Y)
prediction1=model.predict(test_X)
print('Accuracy for rbf SVM is ',metrics.accuracy_score(prediction1,test_Y))
```

```
Accuracy for rbf SVM is  0.835820895522388
```

```
model=svm.SVC(kernel='linear',C=0.1,gamma=0.1)
model.fit(train_X,train_Y)
prediction2=model.predict(test_X)
print('Accuracy for linear SVM is',metrics.accuracy_score(prediction2,test_Y))
```

```
Accuracy for linear SVM is 0.8171641791044776
```

3. EDA to Prediction (DieTanic)

13-3. Random Forest

```
model=RandomForestClassifier(n_estimators=100)
model.fit(train_X,train_Y)
prediction7=model.predict(test_X)
print('The accuracy of the Random Forests is',metrics.accuracy_score(prediction7,test_Y))
```

```
The accuracy of the Random Forests is 0.8171641791044776
```

3. EDA to Prediction (DieTanic)

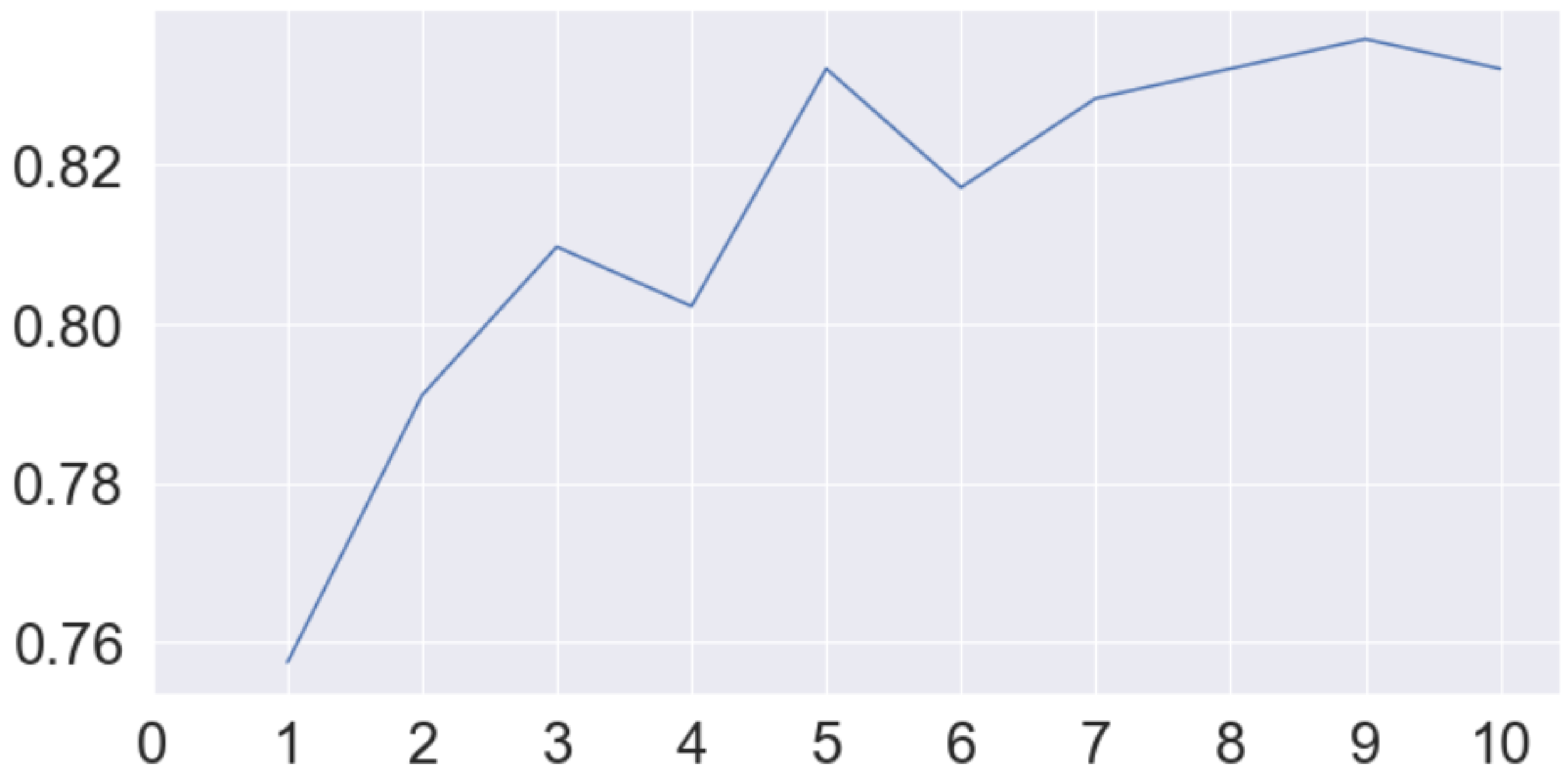
13-4. K-Nearest Neighbours

```
model=KNeighborsClassifier()
model.fit(train_X,train_Y)
prediction5=model.predict(test_X)
print('The accuracy of the KNN is',metrics.accuracy_score(prediction5,test_Y))
```

```
The accuracy of the KNN is 0.832089552238806
```

```
a_index=list(range(1,11))
a=pd.Series()
x=[0,1,2,3,4,5,6,7,8,9,10]
for i in list(range(1,11)):
    model=KNeighborsClassifier(n_neighbors=i)
    model.fit(train_X,train_Y)
    prediction=model.predict(test_X)
    a=a.append(pd.Series(metrics.accuracy_score(prediction,test_Y)))
plt.plot(a_index, a)
plt.xticks(x)
fig=plt.gcf()
fig.set_size_inches(12,6)
plt.show()
print('Accuracies for different values of n are:',a.values,'with the max value as ',a.values.max())
```

3. EDA to Prediction (DieTanic)




```
Accuracies for different values of n are: [0.75746269 0.79104478 0.80970149 0.80223881 0.83208955 0.81716418
 0.82835821 0.83208955 0.8358209  0.83208955] with the max value as  0.835820895522388
```

3. EDA to Prediction (DieTanic)

13-5. Naive Bayes

```
model=GaussianNB()
model.fit(train_X,train_Y)
prediction6=model.predict(test_X)
print('The accuracy of the NaiveBayes is',metrics.accuracy_score(prediction6,test_Y))
```

```
The accuracy of the NaiveBayes is 0.813432835820895 5
```

13-6. Decision Tree

```
model=DecisionTreeClassifier()
model.fit(train_X,train_Y)
prediction4=model.predict(test_X)
print('The accuracy of the Decision Tree is',metrics.accuracy_score(prediction4,test_Y))
```

```
The accuracy of the Decision Tree is 0.8059701492537313
```

3. EDA to Prediction (DieTanic)

14. Cross Validation

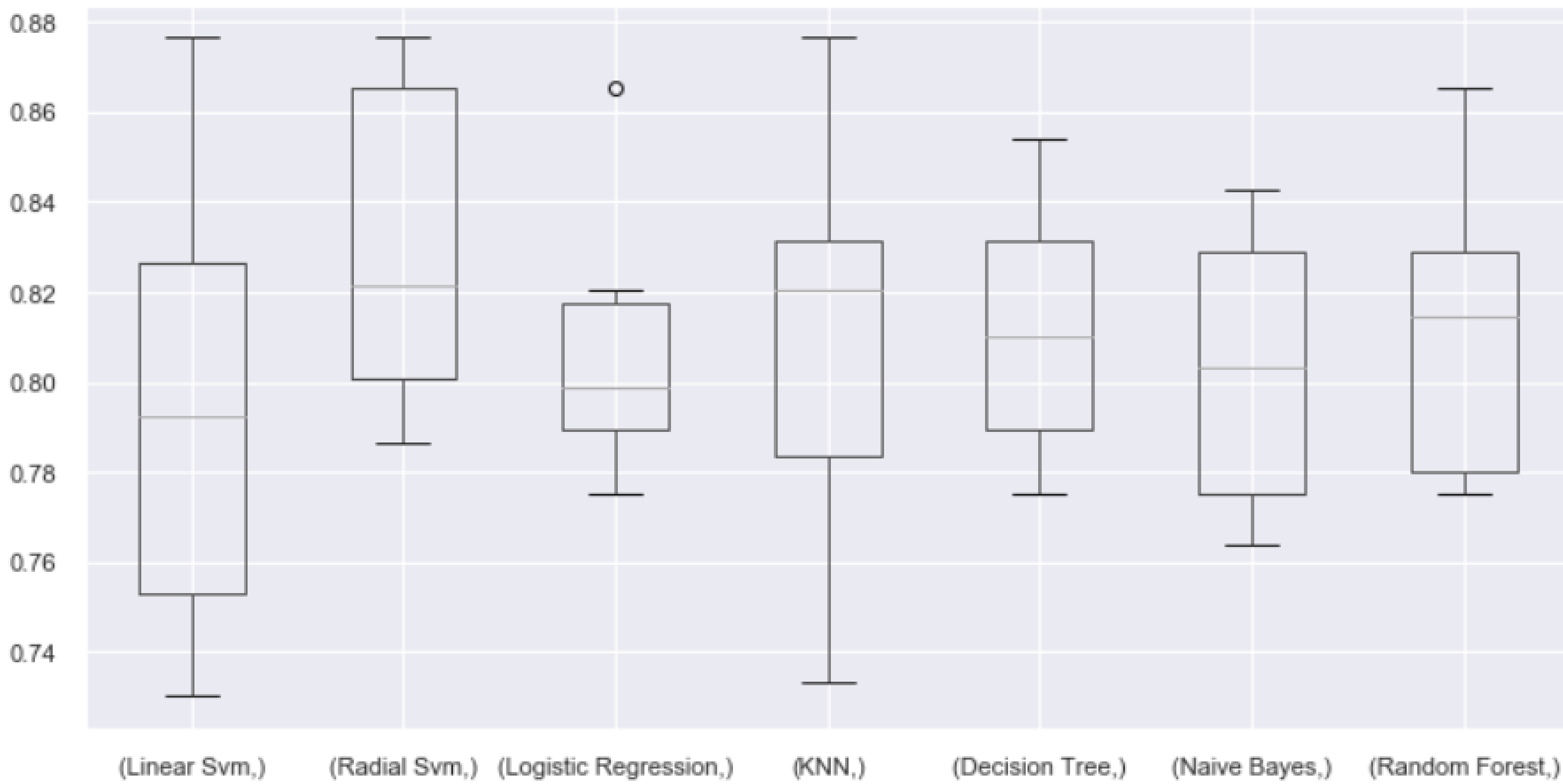
```
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.model_selection import cross_val_score #score evaluation
from sklearn.model_selection import cross_val_predict #prediction
kfold = KFold(n_splits=10, random_state=22) # k=10, split the data into 10 equal parts
xyz=[]
accuracy=[]
std=[]
classifiers=['Linear Svm','Radial Svm','Logistic Regression','KNN','Decision Tree','Naive Bayes','Random Forest']
models=[svm.SVC(kernel='linear'),svm.SVC(kernel='rbf'),LogisticRegression(),
        KNeighborsClassifier(n_neighbors=9),DecisionTreeClassifier(),GaussianNB(),RandomForestClassifier(n_estimators=100)]
for i in models:
    model = i
    cv_result = cross_val_score(model,X,Y, cv = kfold,scoring = "accuracy")
    cv_result=cv_result
    xyz.append(cv_result.mean())
    std.append(cv_result.std())
    accuracy.append(cv_result)
new_models_dataframe2=pd.DataFrame({'CV Mean':xyz,'Std':std},index=classifiers)
new_models_dataframe2
```

	CV Mean	Std
Linear Svm	0.793471	0.047797
Radial Svm	0.828290	0.034427
Logistic Regression	0.805843	0.024061
KNN	0.813783	0.041210
Decision Tree	0.811461	0.026862
Naive Bayes	0.801386	0.028999
Random Forest	0.810362	0.029635

3. EDA to Prediction (DieTanic)

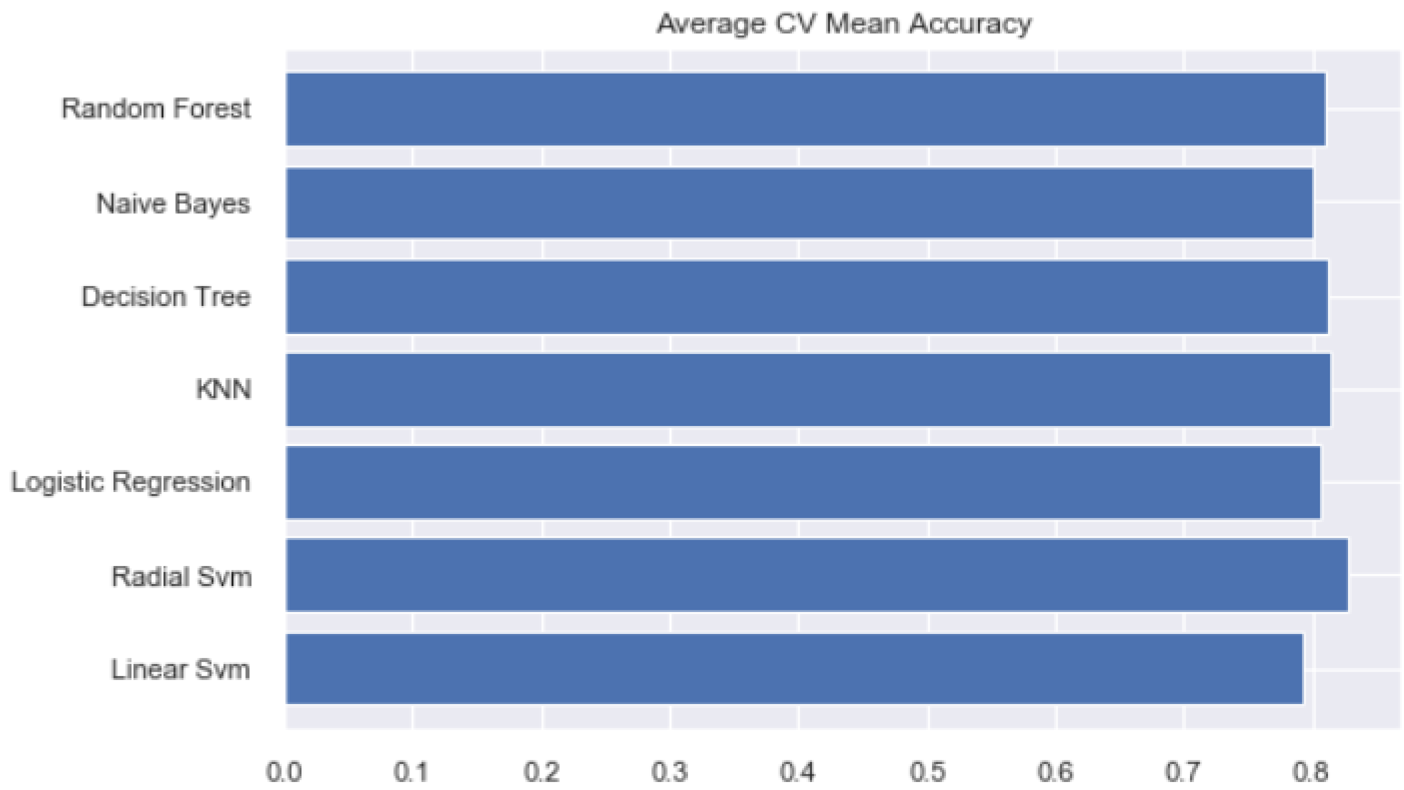
```
plt.subplots(figsize=(12,6))
box=pd.DataFrame(accuracy,index=[classifiers])
box.T.boxplot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1def97dcc48>
```



3. EDA to Prediction (DieTanic)

```
new_models_dataframe2['CV Mean'].plot.barh(width=0.8)
plt.title('Average CV Mean Accuracy')
fig=plt.gcf()
fig.set_size_inches(8,5)
plt.show()
```

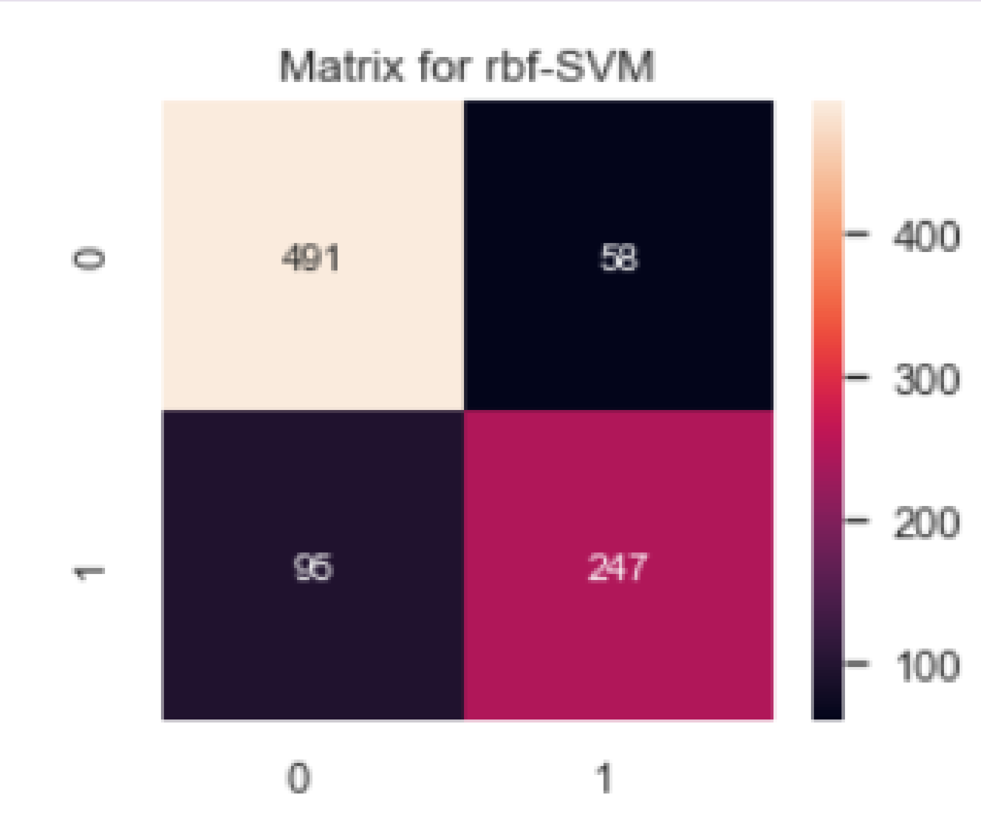


3. EDA to Prediction (DieTanic)

15. Confusion Matrix

```
f,ax=plt.subplots(3,3,figsize=(12,10))
y_pred = cross_val_predict(svm.SVC(kernel='rbf'),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,0],annot=True,fmt='2.0f')
ax[0,0].set_title('Matrix for rbf-SVM')
y_pred = cross_val_predict(svm.SVC(kernel='linear'),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,1],annot=True,fmt='2.0f')
ax[0,1].set_title('Matrix for Linear-SVM')
y_pred = cross_val_predict(KNeighborsClassifier(n_neighbors=9),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,2],annot=True,fmt='2.0f')
ax[0,2].set_title('Matrix for KNN')
y_pred = cross_val_predict(RandomForestClassifier(n_estimators=100),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,0],annot=True,fmt='2.0f')
ax[1,0].set_title('Matrix for Random-Forests')
y_pred = cross_val_predict(LogisticRegression(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,1],annot=True,fmt='2.0f')
ax[1,1].set_title('Matrix for Logistic Regression')
y_pred = cross_val_predict(DecisionTreeClassifier(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,2],annot=True,fmt='2.0f')
ax[1,2].set_title('Matrix for Decision Tree')
y_pred = cross_val_predict(GaussianNB(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[2,0],annot=True,fmt='2.0f')
ax[2,0].set_title('Matrix for Naive Bayes')
plt.subplots_adjust(hspace=0.2,wspace=0.2)
plt.show()
```

3. EDA to Prediction (DieTanic)



check!

- correct predictions are 491(for dead) + 247(for survived)
- Errors--> Wrongly Classified 58 dead people as survived and 95 survived as dead

3. EDA to Prediction (DieTanic)

16. Hyper-Parameters Tuning

```
# SVM
from sklearn.model_selection import GridSearchCV
C=[0.05,0.1,0.2,0.3,0.25,0.4,0.5,0.6,0.7,0.8,0.9,1]
gamma=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
kernel=['rbf','linear']
hyper={'kernel':kernel,'C':C,'gamma':gamma}
gd=GridSearchCV(estimator=svm.SVC(),param_grid=hyper,verbose=True)
gd.fit(X,Y)
print(gd.best_score_)
print(gd.best_estimator_)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits
0.8282593685267716
SVC(C=0.4, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.3, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
[Parallel(n_jobs=1)]: Done 1200 out of 1200 | elapsed:   15.6s finished
```

3. EDA to Prediction (DieTanic)

```
# Random Forests
n_estimators=range(100,1000,100)
hyper={'n_estimators':n_estimators}
gd=GridSearchCV(estimator=RandomForestClassifier(random_state=0),param_grid=hyper,verbose=True)
gd.fit(X,Y)
print(gd.best_score_)
print(gd.best_estimator_)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

```
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:   31.9s finished
```

```
0.819327098110602
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=300,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

check! -> best hyperparameter

- Rbf-Svm is 82.82% with C=0.05 and gamma=0.1
- RandomForest, score is abt 81.8% with n_estimators=900.

3. EDA to Prediction (DieTanic)

17. Ensembling

- combination of various simple models to create a single powerful model -> increase accuracy

17-1. Voting Classifier

- simplest way of combining predictions from many different simple machine learning models
- average prediction result based on the prediction of all the submodels

```
from sklearn.ensemble import VotingClassifier
ensemble_lin_rbf=VotingClassifier(estimators=[('KNN',KNeighborsClassifier(n_neighbors=10)),
                                              ('RBF',svm.SVC(probability=True,kernel='rbf',C=0.5,gamma=0.1)),
                                              ('RFor',RandomForestClassifier(n_estimators=500,random_state=0)),
                                              ('LR',LogisticRegression(C=0.05)),
                                              ('DT',DecisionTreeClassifier(random_state=0)),
                                              ('NB',GaussianNB()),
                                              ('svm',svm.SVC(kernel='linear',probability=True))
                                             ],
                       voting='soft').fit(train_X,train_Y)
print('The accuracy for ensembled model is:',ensemble_lin_rbf.score(test_X,test_Y))
cross=cross_val_score(ensemble_lin_rbf,X,Y, cv = 10,scoring = "accuracy")
print('The cross validated score is',cross.mean())
```

```
The accuracy for ensembled model is: 0.8246268656716418
The cross validated score is 0.8249188514357053
```

3. EDA to Prediction (DieTanic)

17-2. Bagging

- It works by applying similar classifiers on small partitions of the dataset and then taking the average of all the predictions.
- Due to the averaging,there is reduction in variance.

```
from sklearn.ensemble import BaggingClassifier
model=BaggingClassifier(base_estimator=KNeighborsClassifier(n_neighbors=3),random_state=0,n_estimators=700)
model.fit(train_X,train_Y)
prediction=model.predict(test_X)
print('The accuracy for bagged KNN is:',metrics.accuracy_score(prediction,test_Y))
result=cross_val_score(model,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for bagged KNN is:',result.mean())
```

```
The accuracy for bagged KNN is: 0.835820895522388
The cross validated score for bagged KNN is: 0.8160424469413232
```

```
model=BaggingClassifier(base_estimator=DecisionTreeClassifier(),random_state=0,n_estimators=100)
model.fit(train_X,train_Y)
prediction=model.predict(test_X)
print('The accuracy for bagged Decision Tree is:',metrics.accuracy_score(prediction,test_Y))
result=cross_val_score(model,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for bagged Decision Tree is:',result.mean())
```

```
The accuracy for bagged Decision Tree is: 0.8246268656716418
The cross validated score for bagged Decision Tree is: 0.8227590511860174
```

3. EDA to Prediction (DieTanic)

17-3. Boosting

- ensembling technique which uses sequential learning of classifiers -> step by step enhancement of a weak model
- A model is first trained on the complete dataset.
- the learner will focus more on the wrongly predicted instances or give more weight to it
- Thus it will try to predict the wrong instance correctly

```
# AdaBoost(Adaptive Boosting)
from sklearn.ensemble import AdaBoostClassifier
ada=AdaBoostClassifier(n_estimators=200,random_state=0,learning_rate=0.1)
result=cross_val_score(ada,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for AdaBoost is:',result.mean())
```

```
The cross validated score for AdaBoost is: 0.8249188514357055
```

```
# Stochastic Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
grad=GradientBoostingClassifier(n_estimators=500,random_state=0,learning_rate=0.1)
result=cross_val_score(grad,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for Gradient Boosting is:',result.mean())
```

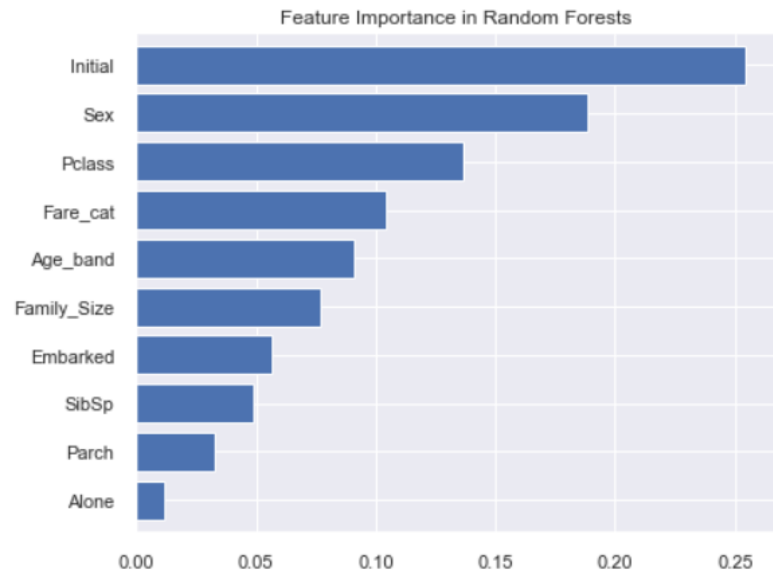
```
The cross validated score for Gradient Boosting is: 0.8115230961298376
```

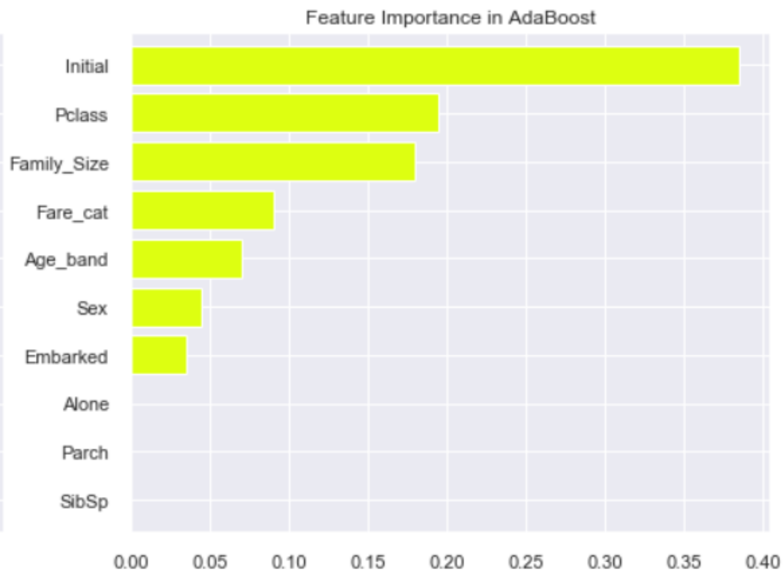
3. EDA to Prediction (DieTanic)

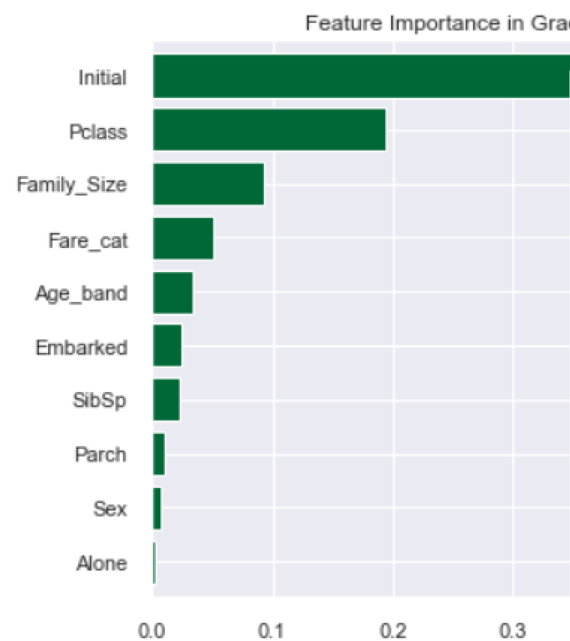
18. Feature Importance

```
f,ax=plt.subplots(2,2,figsize=(15,12))
model=RandomForestClassifier(n_estimators=500,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[0,0])
ax[0,0].set_title('Feature Importance in Random Forests')
model=AdaBoostClassifier(n_estimators=200,learning_rate=0.05,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[0,1],color='#ddff11')
ax[0,1].set_title('Feature Importance in AdaBoost')
model=GradientBoostingClassifier(n_estimators=500,learning_rate=0.1,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[1,0],cmap='RdYlGn_r')
ax[1,0].set_title('Feature Importance in Gradient Boosting')
plt.show()
```

3. EDA to Prediction (DieTanic)







check!

- Some of the common important features are Initial,Fare_cat,Pclass,Family_Size.
- The Sex feature doesn't seem to give any importance.
  Sex looks to be important only in RandomForests.
- we can see the feature Initial, which is at the top in many classifiers.
  We had already seen the positive correlation between Sex and Initial,
  so they both refer to the gender.
- Similarly the Pclass and Fare cat refer to the status of the passengers
  and Family_Size with Alone,Parch and SibSp.

끝