타이타닉 EDA

2017010715허지혜

0. 캐글이란 ?

Kaggle : 예측 모델 및 분석 대회를 하는 플랫폼
https://www.kaggle.com/

0. 캐글이란 ?

Titanic: Machine Learning from Disaster

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

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

1. 데이터 분석 진행 과정

1. 데이터셋 확인
2. 탐색적 데이터 분석
(Exploratory Data Analysis)
3. Feature Engineering
4. Model 만들기
5. 만든 Model 학습 및 예측
6. Model 평가

2. 데이터셋 확인

```
In [2]: df_train = pd.read_csv('../input/train.csv')
        df_test = pd.read_csv('../input/test.csv')
```

```
In [3]: df_train.head()
```

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

데이터 변수에 저장하기
⇒ 확인 .head()

2. 데이터셋 확인

변수(feature, variable)	정의	설명	타입
survival	생존여부	target label 임. 1, 0 으로 표현됨	integer
Pclass	티켓의 클래스	1 = 1st, 2 = 2nd, 3 = 3rd 클래스로 나뉘며 categorical feature	integer
sex	성별	male, female 로 구분되며 binary	string
Age	나이	continuous	integer
sibSp	함께 탑승한 형제와 배우자의 수	quantitative	integer
parch	함께 탑승한 부모, 아이의 수	quantitative	integer
ticket	티켓 번호	alphabat + integer	string
fare	탑승료	continuous	float
cabin	객실 번호	alphabat + integer	string
embared	탑승 항구	C = Cherbourg, Q = Queenstown, S = Southampton	string

데이터 전체 타입 확인
=>.info()

2. 데이터셋 확인

```
In [4]: df_train.describe()
```

	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

```
In [5]: df_test.describe()
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200

데이터 통계치 확인
⇒ .describe()

옵션 include = 'all'을 통해
범주형 데이터도 분석이 가능

2. 데이터셋 확인

```
In [6]: for col in df_train.columns:
            msg = 'column: {:>10}\t Percent of NaN value: {:.2f}%'.format(col, 100
        * (df_train[col].isnull().sum() / df_train[col].shape[0]))
            print(msg)
```

```
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%
```

```
In [7]: for col in df_test.columns:
            msg = 'column: {:>10}\t Percent of NaN value: {:.2f}%'.format(col, 100
        * (df_test[col].isnull().sum() / df_test[col].shape[0]))
            print(msg)
```

```
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%
```

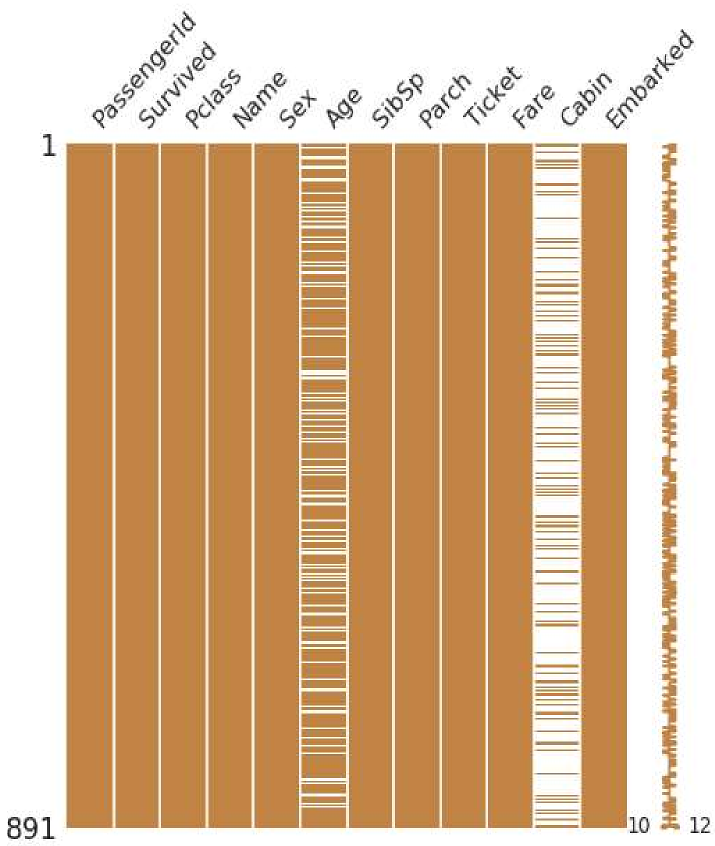
데이터 빈 값 확인하기
⇒ .isnull().sum()

MANO 라이브러리로
빈 값의 위치들을 살펴볼 수 있음

2. 데이터셋 확인

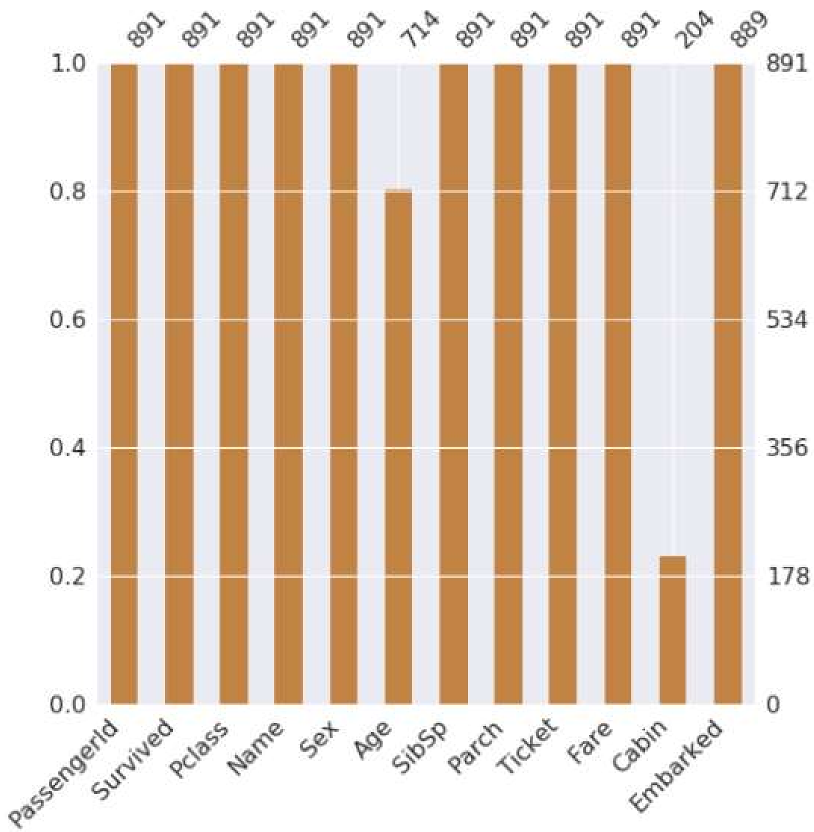
```
In [8]: msno.matrix(df=df_train.iloc[:, :], figsize=(8, 8), color=(0.8, 0.5, 0.2))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbb2bca5c0>
```



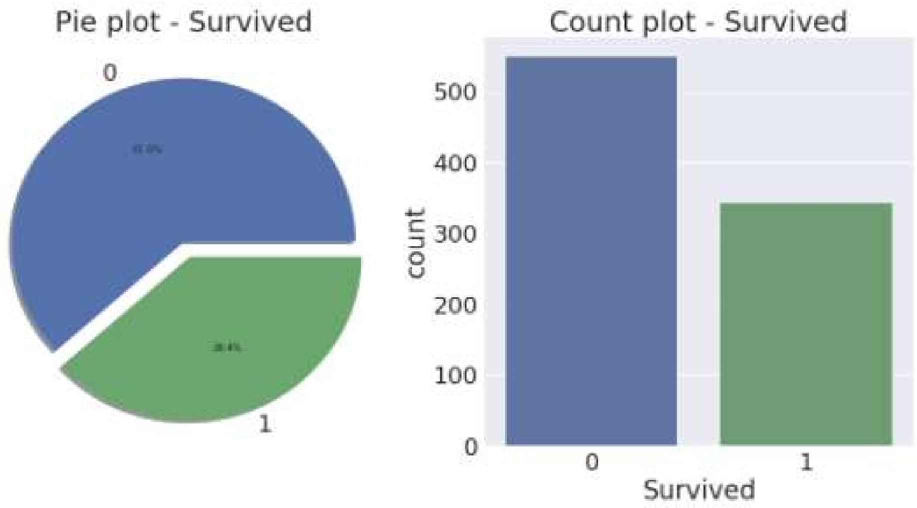

```
In [9]: msno.bar(df=df_train.iloc[:, :], figsize=(8, 8), color=(0.8, 0.5, 0.2))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbb2b2feb8>
```



2. 데이터셋 확인

In [11]:
```
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()
```



데이터 분포 시각화로 보기

3. 탐색적 데이터 분석

```
In [12]: df_train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=True).count
()
```

Pclass	Survived
1	216
2	184
3	491

```
In [13]: df_train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=True).sum()
```

Pclass	Survived
1	136
2	87
3	119

각 열에 있는 수 확인
⇒ .count()

3. 탐색적 데이터 분석

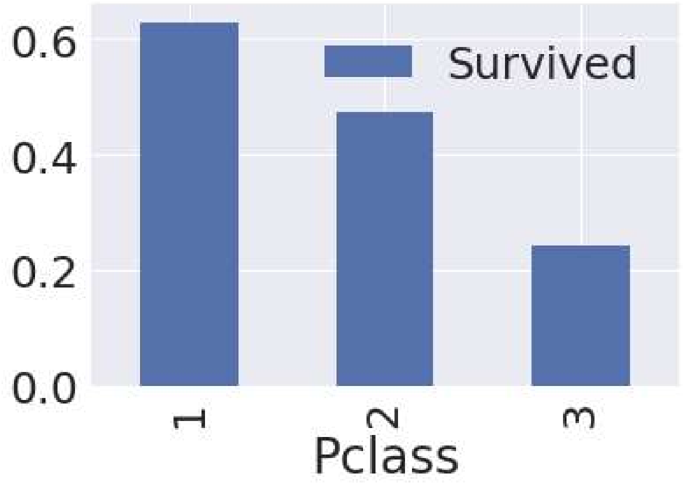
```
In [14]: pd.crosstab(df_train['Pclass'], df_train['Survived'], margins=True).style.
         background_gradient(cmap='summer_r')
```

Survived	0	1	All
Pclass			
1	80	136	216
2	97	87	184
3	372	119	491
All	[illegible]	[illegible]	[illegible]

데이터 빈도표 만들기
⇒ pd.crosstab()

```
In [15]: df_train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=True).mean
         ().sort_values(by='Survived', ascending=False).plot.bar()
```

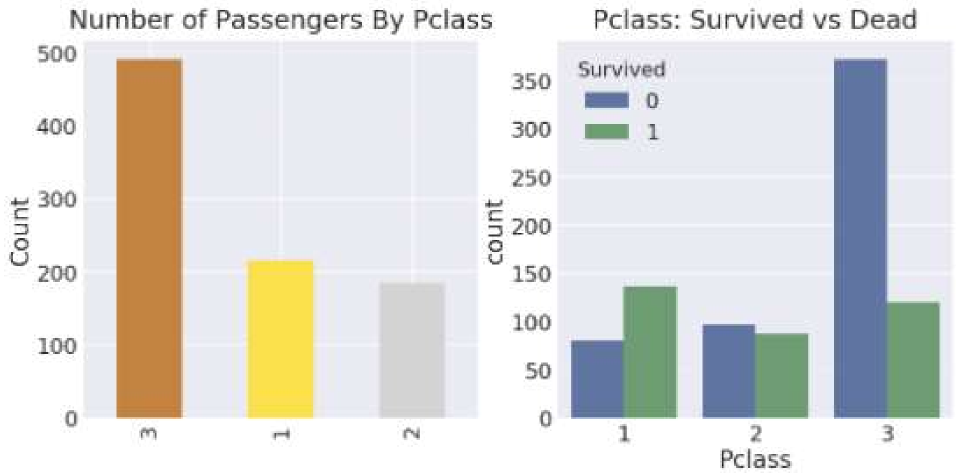
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbb221d518>
```



컬럼 값으로 데이터 정렬하기
=> sort_values()

3. 탐색적 데이터 분석

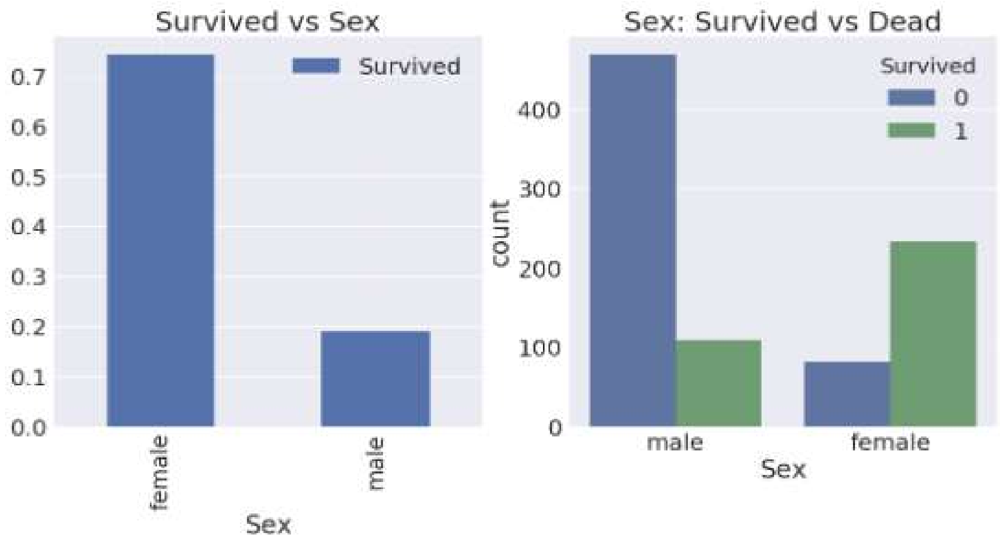
```
In [16]: y_position = 1.02
f, ax = plt.subplots(1, 2, figsize=(18, 8))
df_train['Pclass'].value_counts().plot.bar(color=['#CD7F32','#FFDF00','#D3
D3D3'], ax=ax[0])
ax[0].set_title('Number of Passengers By Pclass', y=y_position)
ax[0].set_ylabel('Count')
sns.countplot('Pclass', hue='Survived', data=df_train, ax=ax[1])
ax[1].set_title('Pclass: Survived vs Dead', y=y_position)
plt.show()
```



1. 클래스가 높을수록 생존 확률이 높다.
2. 생존에 Pclass 가 영향을 미친다.

3. 탐색적 데이터 분석

```
In [17]: 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()
```

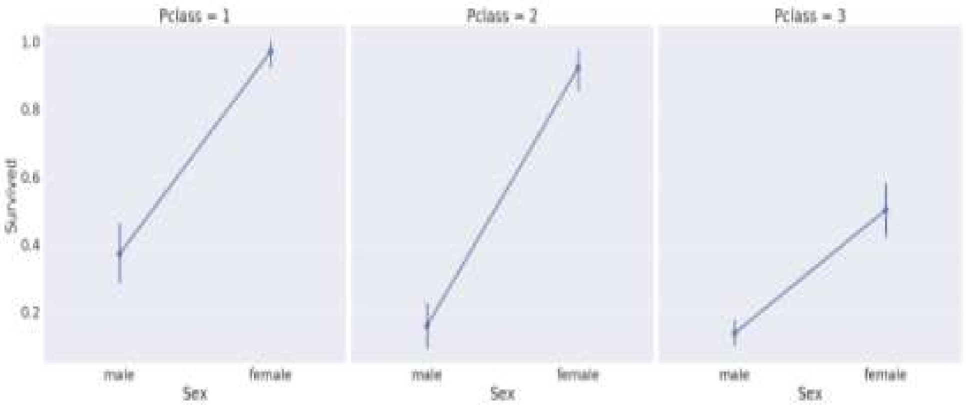


1. 여자가 생존할 확률이 높다.

3. 탐색적 데이터 분석

```
In [21]: sns.factorplot(x='Sex', y='Survived', col='Pclass',
                        data=df_train, satureation=.5,
                         size=9, aspect=1
                        )
```

```
<seaborn.axisgrid.FacetGrid at 0x7fbbb20ee438>
```



1. Female이 살 확률이 Male 보다 높다.

2. Pclass 가 높을수록 살 확률이 높다.

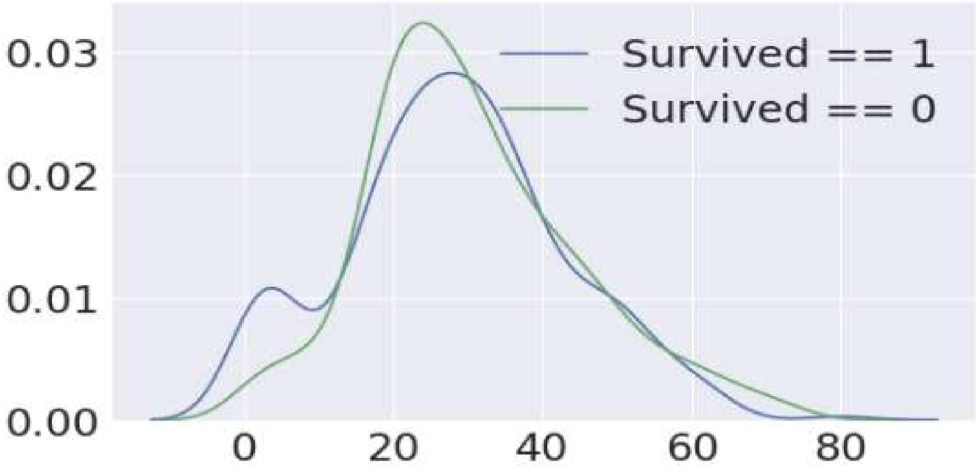
3. 탐색적 데이터 분석

```
In [22]: print('제일 나이 많은 탑승객 : {:.1f} Years'.format(df_train['Age'].max()))
         print('제일 어린 탑승객 : {:.1f} Years'.format(df_train['Age'].min()))
         print('탑승객 평균 나이 : {:.1f} Years'.format(df_train['Age'].mean()))
```

```
제일 나이 많은 탑승객 : 80.0 Years
제일 어린 탑승객 : 0.4 Years
탑승객 평균 나이 : 29.7 Years
```

- 생존에 따른 Age의 histogram 을 그려보겠습니다.

```
In [23]: fig, ax = plt.subplots(1, 1, figsize=(9, 5))
         sns.kdeplot(df_train[df_train['Survived'] == 1]['Age'], ax=ax)
         sns.kdeplot(df_train[df_train['Survived'] == 0]['Age'], ax=ax)
         plt.legend(['Survived == 1', 'Survived == 0'])
         plt.show()
```

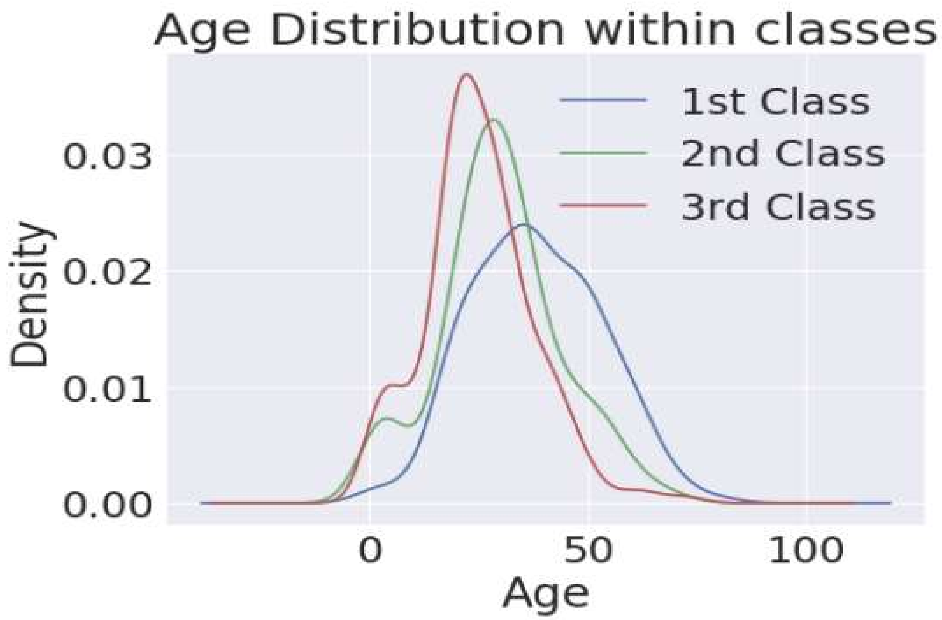


생존자 중 나이 어린 사람이 많다.

3. 탐색적 데이터 분석

```
In [24]: # Age distribution withing classes
         plt.figure(figsize=(8, 6))
         df_train['Age'][df_train['Pclass'] == 1].plot(kind='kde')
         df_train['Age'][df_train['Pclass'] == 2].plot(kind='kde')
         df_train['Age'][df_train['Pclass'] == 3].plot(kind='kde')

         plt.xlabel('Age')
         plt.title('Age Distribution within classes')
         plt.legend(['1st Class', '2nd Class', '3rd Class'])
```

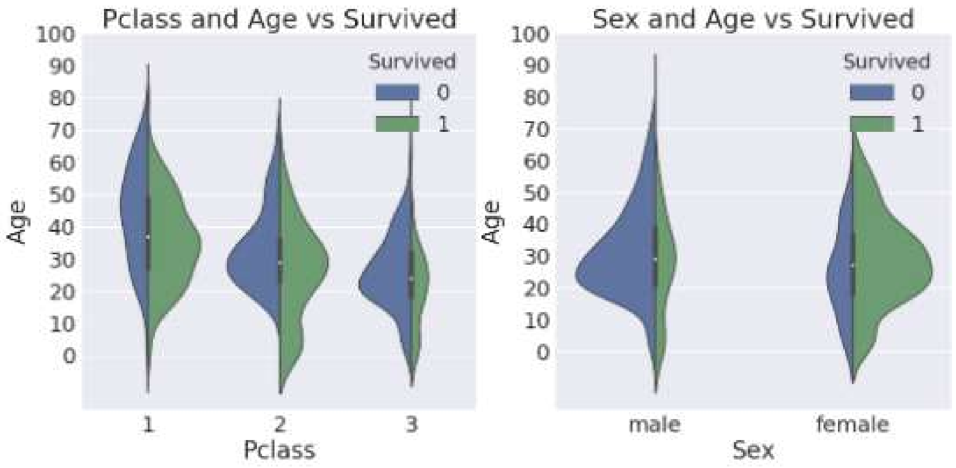
```
<matplotlib.legend.Legend at 0x7fbbb20f9400>
```



Class가 높을 수록 나이 많은 사람의 비중이 커진다.

3. 탐색적 데이터 분석

```
In [26]: 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()
```

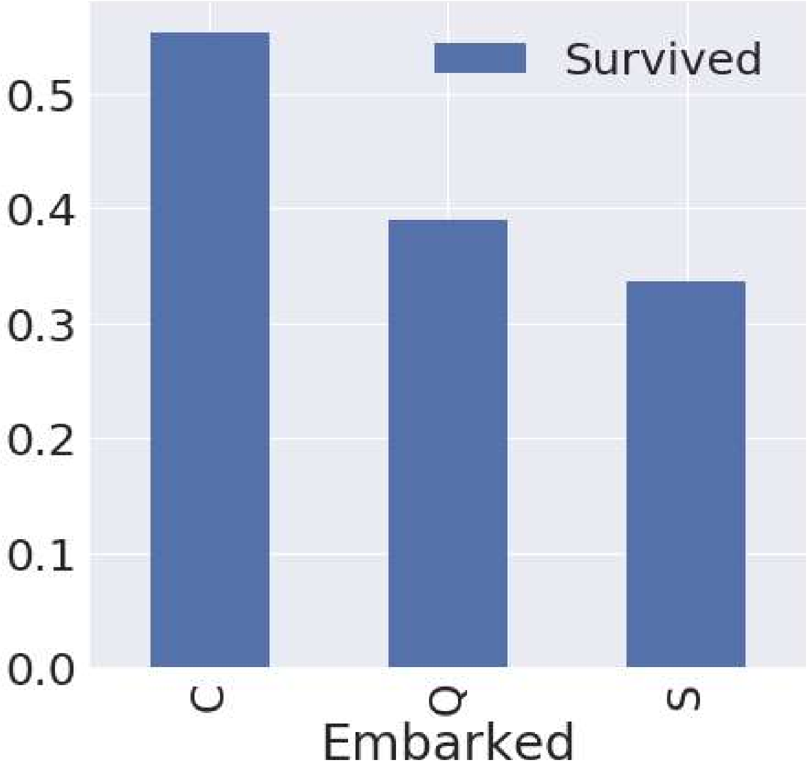


Plcass, Sex, Age 관계

Violinplot

3. 탐색적 데이터 분석

```
In [27]: f, ax = plt.subplots(1, 1, figsize=(7, 7))
         df_train[['Embarked', 'Survived']].groupby(['Embarked'], as_index=True).me
         an().sort_values(by='Survived', ascending=False).plot.bar(ax=ax)

<matplotlib.axes._subplots.AxesSubplot at 0x7fbbb1dc0a90>
```



C가 가장 생존율이 높다.

3. 탐색적 데이터 분석

In [35]:

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

상관 관계 살펴보기
⇒ .corr()

상관 관계 시각화
⇒ .heatmap()

4. Feature Engineering

데이터 셋을 받을 때 모든 열을 다 사용할 필요가 없고 제거할 행들이 있고, 추출할 행들이 있다.
따라서 예측 모델링에 적합한 형태로 변환을 시켜야 한다.

4. Feature Engineering

Age

```
data['Age_band']=0
data.loc[data['Age']<=16,'Age_band']=0
data.loc[(data['Age']>16)&(data['Age']<=32),'Age_band']=1
data.loc[(data['Age']>32)&(data['Age']<=48),'Age_band']=2
data.loc[(data['Age']>48)&(data['Age']<=64),'Age_band']=3
data.loc[data['Age']>64,'Age_band']=4
data.head(2)
```

Out[36]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fa	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85

나이 = 연속형

=> 범주형으로 바꿔 주기

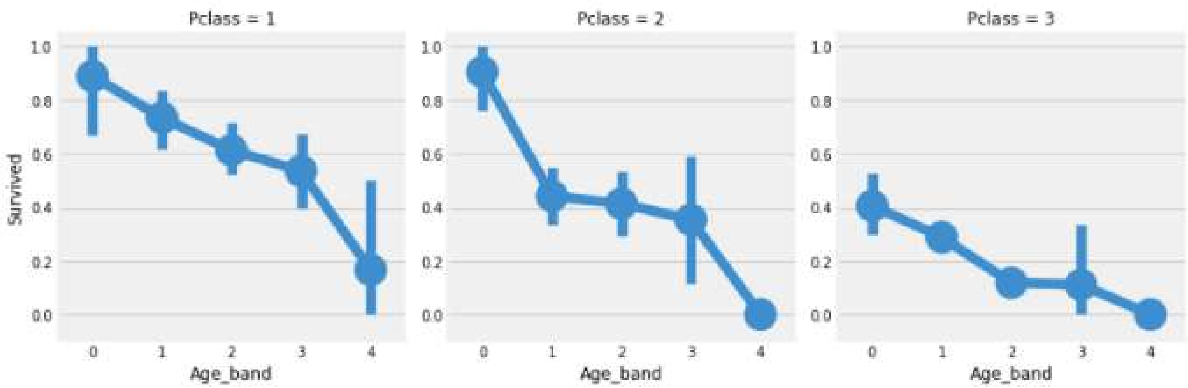
4. Feature Engineering

Age

```
data['Age_band'].value_counts().to_frame().style.background_gradient(cmap='summe
r')#checking the number of passenegers in each band
```

	Age_band
1	382
2	325
0	104
3	69
4	11

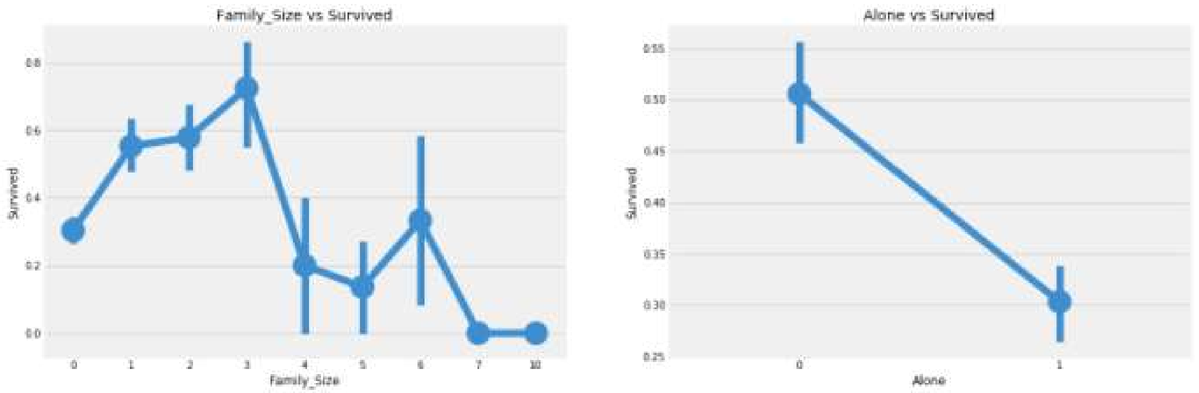
```
sns.factorplot('Age_band','Survived',data=data,col='Pclass')
plt.show()
```



4. Feature Engineering

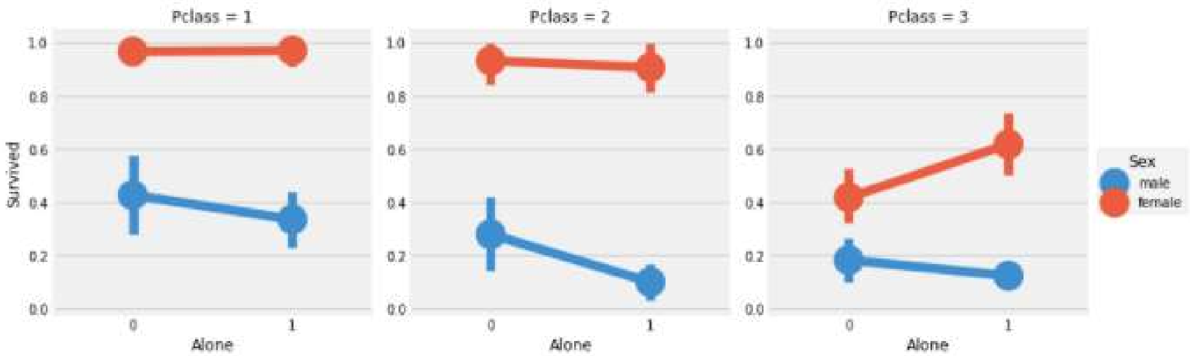
```
data['Family_Size']=0
data['Family_Size']=data['Parch']+data['SibSp']#family size
data['Alone']=0
data.loc[data.Family_Size==0,'Alone']=1#Alone

f,ax=plt.subplots(1,2,figsize=(18,6))
sns.factorplot('Family_Size','Survived',data=data,ax=ax[0])
ax[0].set_title('Family_Size vs Survived')
sns.factorplot('Alone','Survived',data=data,ax=ax[1])
ax[1].set_title('Alone vs Survived')
plt.close(2)
plt.close(3)
plt.show()
```

통행인 혼자 => 생존율 낮음



4. Feature Engineering

```
sns.factorplot('Alone','Survived',data=data,hue='Sex',col='Pclass')
plt.show()
```



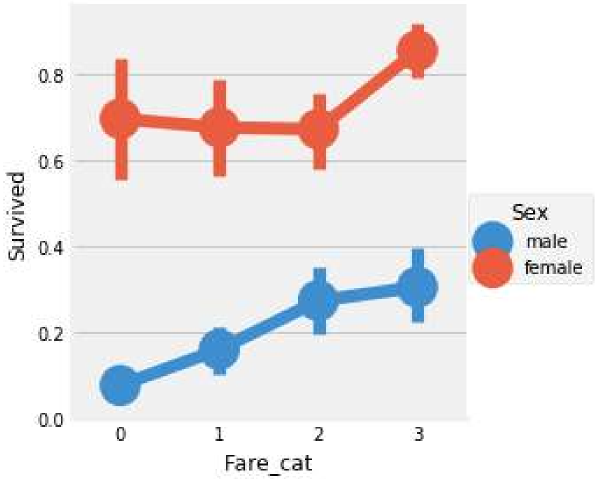
가족이 있는 여성보다 혼자 있는
여성이 더 생존 확률이 높다.

4. Feature Engineering

Fare

```
data['Fare_cat']=0
data.loc[data['Fare']<=7.91,'Fare_cat']=0
data.loc[(data['Fare']>7.91)&(data['Fare']<=14.454),'Fare_cat']=1
data.loc[(data['Fare']>14.454)&(data['Fare']<=31),'Fare_cat']=2
data.loc[(data['Fare']>31)&(data['Fare']<=513),'Fare_cat']=3
```

```
sns.factorplot('Fare_cat','Survived',data=data,hue='Sex')
plt.show()
```



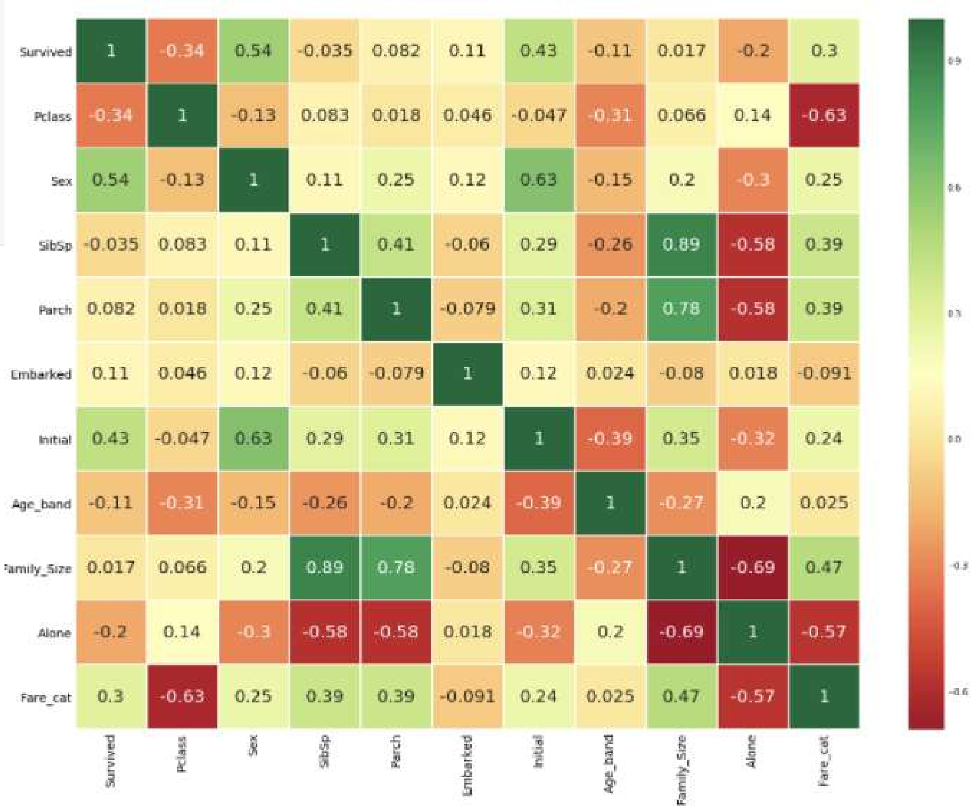
4. Feature Engineering

```
data['Sex'].replace(['male','female'],[0,1],inplace=True)
data['Embarked'].replace(['S','C','Q'],[0,1,2],inplace=True)
data['Initial'].replace(['Mr','Mrs','Miss','Master','Other'],[0,1,2,3,4],inplace
=True)
```

Name : 카테고리 값으로 변환이 불가능 하므로 필요 없음
Age : 범주형으로 바꿔줌
Ticket : 분류할 수 없는 임의의 문자열임
Fare : 바꿔줌
PassengerId : 분류 불가능

4. Feature Engineering

```
data.drop(['Name','Age','Ticket','Fare','Cabin','Fare_Range','PassengerId'],axis
=1,inplace=True)
sns.heatmap(data.corr(),annot=True,cmap='RdYlGn',linewidths=0.2,annot_kws={'siz
e':20})
fig=plt.gcf()
fig.set_size_inches(18,15)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



5. Model 만들기

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

5. Model 만들기

1) Logistic Regression

2) Support Vector Machine

3) Random Forest

4) K-Nearest Neighbors

5) Naïve Bayes

6) Decision Tree

5. Model 만들기

패키지 불러오기

```
#importing all the required ML packages
from sklearn.linear_model import LogisticRegression #logistic regression
from sklearn import svm #support vector Machine
from sklearn.ensemble import RandomForestClassifier #Random Forest
from sklearn.neighbors import KNeighborsClassifier #KNN
from sklearn.naive_bayes import GaussianNB #Naive bayes
from sklearn.tree import DecisionTreeClassifier #Decision Tree
from sklearn.model_selection import train_test_split #training and testing data s
plit
from sklearn import metrics #accuracy measure
from sklearn.metrics import confusion_matrix #for confusion matrix
```

5. Model 만들기

데이터 분리

```
train,test=train_test_split(data,test_size=0.3,random_state=0,stratify=data['Sur
vived'])
train_X=train[train.columns[1:]]
train_Y=train[train.columns[:1]]
test_X=test[test.columns[1:]]
test_Y=test[test.columns[:1]]
X=data[data.columns[1:]]
Y=data['Survived']
```

5. Model 만들기

Model : SVM

Radial Support Vector Machines(rbf-SVM)

```
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  0.835820895522
```

Linear Support Vector Machine(linear-SVM)

```
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 0.817164179104
```

5. Model 만들기

Model : Logistic Regression

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(predic
tion3,test_Y))
```

```
The accuracy of the Logistic Regression is 0.817164179104
```

5. Model 만들기

Model : K-Nearest Neighbors

K-Nearest Neighbours(KNN)

```
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 0.832089552239
```

5. Model 만들기

Model : Decision Tree

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 0.798507462687
```

5. Model 만들기

Model : Gaussian Naïve Bayes

Gaussian 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,tes
t_Y))
```

```
The accuracy of the NaiveBayes is 0.813432835821
```

5. Model 만들기

Model : Random Forest

Random Forests

```
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(prediction
7,test_Y))
```

```
The accuracy of the Random Forests is 0.820895522388
```

5. Model 만들기

Cross Validation

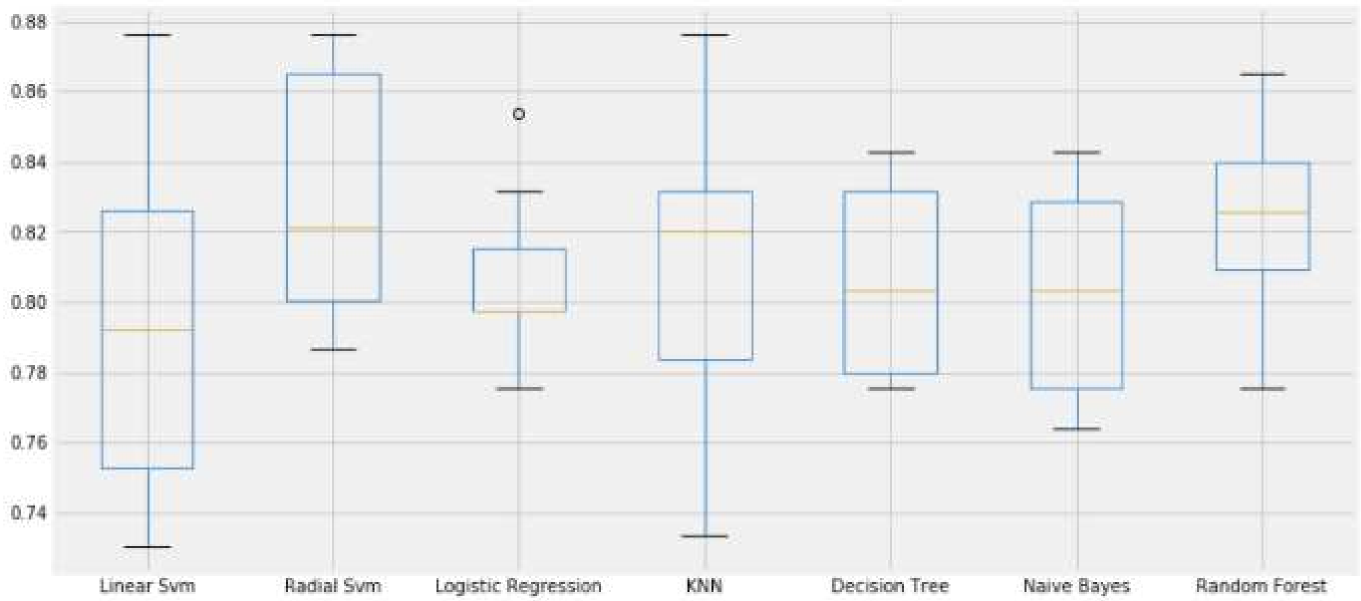
	CV Mean	Std
Linear Svm	0.793471	0.047797
Radial Svm	0.828290	0.034427
Logistic Regression	0.805843	0.021861
KNN	0.813783	0.041210
Decision Tree	0.805868	0.025361
Naive Bayes	0.801386	0.028999
Random Forest	0.822684	0.026868

```
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
```

5. Model 만들기

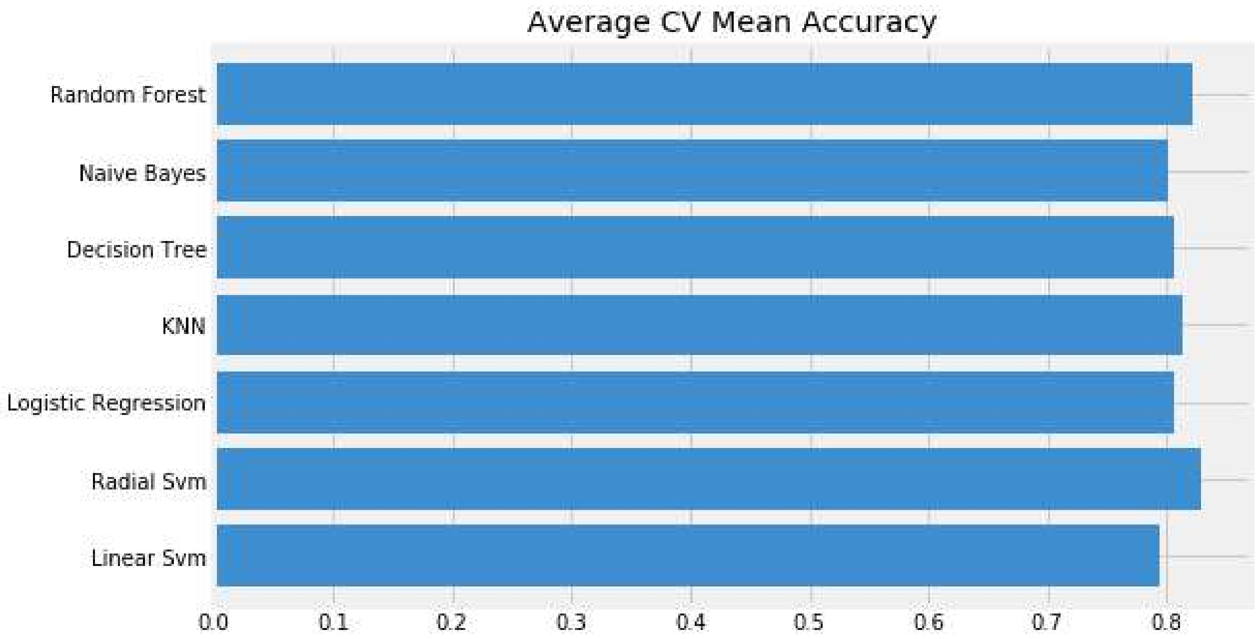
```
plt.subplots(figsize=(12,6))
box=pd.DataFrame(accuracy,index=[classifiers])
box.T.boxplot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb162de2048>
```



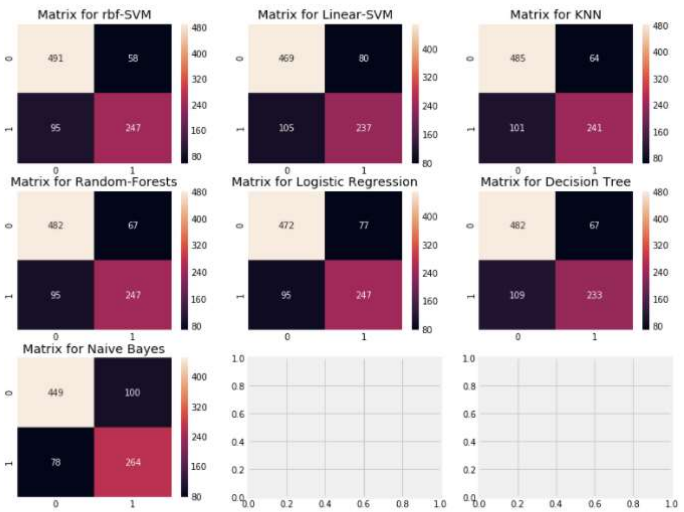
5. Model 만들기

```
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()
```



5. Model 만들기

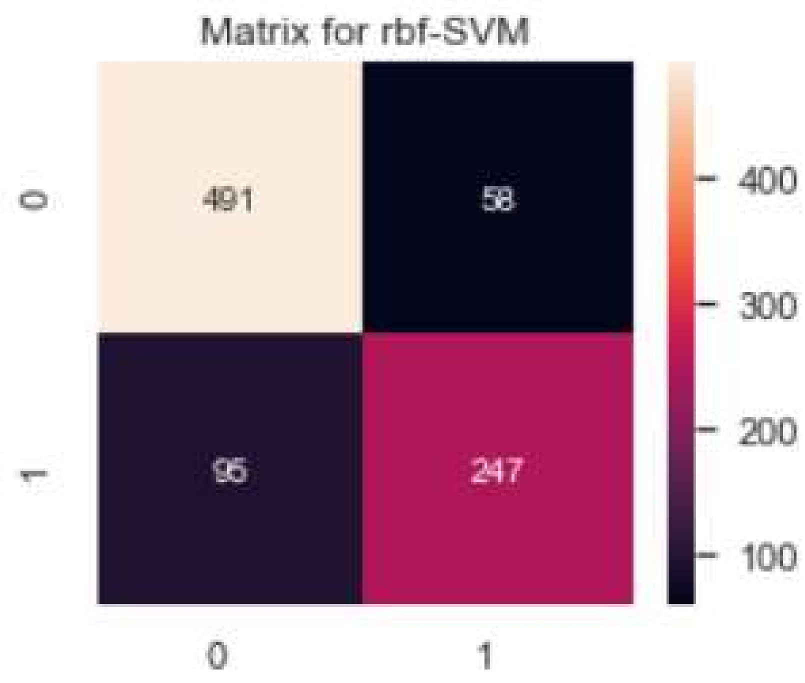
Confusion Matrix




```
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()
```

5. Model 만들기

Confusion Matrix



1) correct predictions = 491(죽음) + 247(생존)
평균 cv 정확도는 (291 + 247) / 891 = 82.8%

2) Errors 58명의 사망자를 살았다고 예측했고
95명의 산 사람을 사망자라고 예측

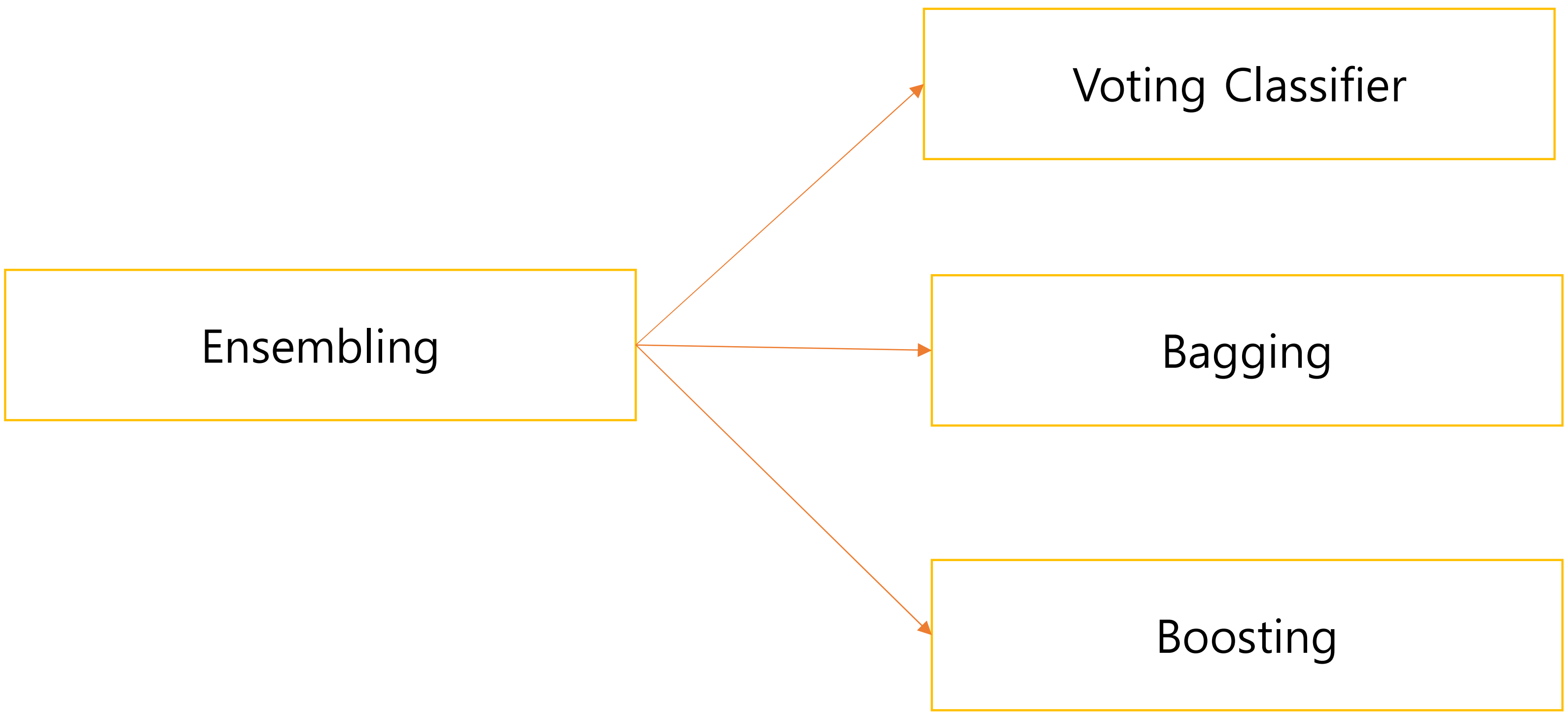
5. Model 만들기

Hyper-Parameters Tuning

```
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_)
```

```
Fitting 3 folds for each of 240 candidates, totalling 720 fits
0.828282828283
SVC(C=0.5, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)


[Parallel(n_jobs=1)]: Done 720 out of 720 | elapsed:   14.7s finished
```

5. Model 만들기



5. Model 만들기

Voting Classifier

다양한 머신 러닝의 예측을 결합하는 가장 간단한 방법

하위 모델의 예측을 기반으로 평균 예측 결과를 제공함

```
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: 0.824626865672
The cross validated score is 0.823766031097
```

5. Model 만들기

Bagging

일반적인 앙상블 방법

데이터 셋의 작은 파티션에 similar classifier를 적용한 다음 모든 예측의 평균을 구하는 방식

Bagged KNN

Bagging works best with models with high variance. An example for this can be Decision Tree or Random Forests. We can use KNN with small value of **n_neighbours**, as small value of n_neighbours.

```
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: 0.835820895522
The cross validated score for bagged KNN is: 0.814889342867
```

Bagged DecisionTree

```
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: 0.824626865672
The cross validated score for bagged Decision Tree is: 0.820482635342
```

5. Model 만들기

Boosting

순차적 학습을 사용하는 앙상블

데이터셋을 학습하다 잘못 예측한 데이터셋에 좀 더 많은 가중치를 부여해서 올바르게 예측하려는 방식

AdaBoost(Adaptive Boosting)

The weak learner or estimator in this case is a Decsion Tree. But we can change the dafault base_estimator to any algorithm of our choice.

```
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: 0.824952616048
```

5. Model 만들기

Boosting

순차적 학습을 사용하는 앙상블

데이터셋을 학습하다 잘못 예측한 데이터셋에 좀 더 많은 가중치를 부여해서 올바르게 예측하려는 방식

Stochastic Gradient Boosting

Here too the weak learner is a Decision Tree.

```
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: 0.818286233118
```

5. Model 만들기

Boosting

순차적 학습을 사용하는 앙상블

데이터셋을 학습하다 잘못 예측한 데이터셋에 좀 더 많은 가중치를 부여해서 올바르게 예측하려는 방식

XGBoost

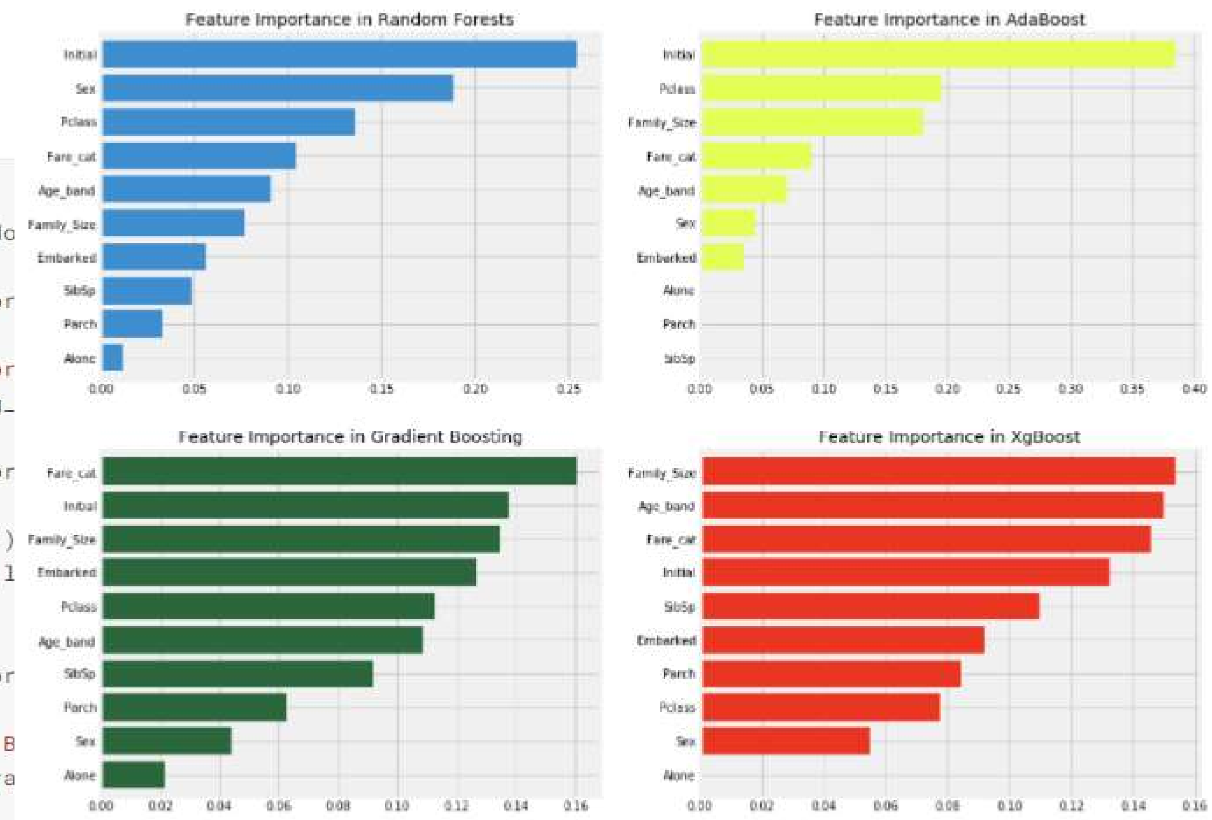
```
import xgboost as xg
xgboost=xg.XGBClassifier(n_estimators=900,learning_rate=0.1)
result=cross_val_score(xgboost,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for XGBoost is:',result.mean())
```

```
The cross validated score for XGBoost is: 0.810471002156
```

5. Model 만들기

Feature Importance

```
f,ax=plt.subplots(2,2,figsize=(15,12))
model=RandomForestClassifier(n_estimators=500,rando
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sor
t.barh(width=0.8,ax=ax[0,0])
ax[0,0].set_title('Feature Importance in Random For
model=AdaBoostClassifier(n_estimators=200,learning_
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sor
t.barh(width=0.8,ax=ax[0,1],color='#ddff11')
ax[0,1].set_title('Feature Importance in AdaBoost')
model=GradientBoostingClassifier(n_estimators=500,l
=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sor
t.barh(width=0.8,ax=ax[1,0],cmap='RdYlGn_r')
ax[1,0].set_title('Feature Importance in Gradient B
model=xg.XGBClassifier(n_estimators=900,learning_ra
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plo
t.barh(width=0.8,ax=ax[1,1],color='#FD0F00')
ax[1,1].set_title('Feature Importance in XgBoost')
plt.show()
```



END