Titanic
Machine Learning from Disaster

(Kaggle—타이타닉 생존자 예측하기)

과목 : 데이터마이닝 (이주홍 교수님)
학번 : 12161532
이름 : 김난영

[모델 만들기]

1. 목표

: 머신러닝을 사용하여 타이타닉 난파선에서 살아남은 승객을 예측하는 모델을 만든다. (분류 문제 - Classification)

2. 데이터 정의하기

Data Dictionary

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

- pclass : 티켓 클래스 -> 사회적 지위와 관련이 있음
- sibsp : 함께 탑승한 자녀 / 배우자의 수
- parch : 함께 탑승한 부모님 / 아이들의 수
- cabin : 선실(좌석) 번호
- embarked : 선착장

3. 데이터 분포 살펴보기

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

4. 생존여부와 관련성이 높은 속성 구하기

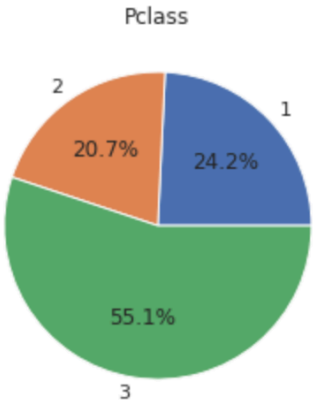
<1. Passenger Id>

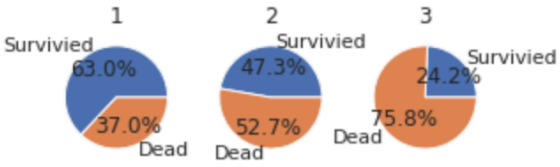
- 일련의 숫자를 단지 부여한 것으로 생존여부와 관련이 없다.

<2. Pclass>

- 티켓 클래스로서 1등급, 2등급, 3등급이 있다.
- 등급이 높을수록 요금(Fare)이 높다.
- 등급이 높을수록 생존율이 높다.

```
pie_chart('Pclass')
```



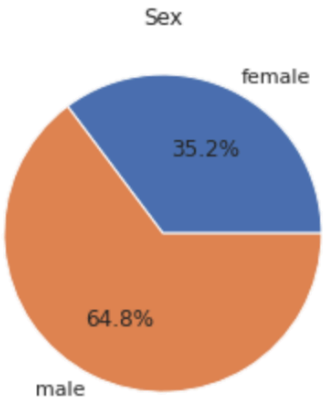


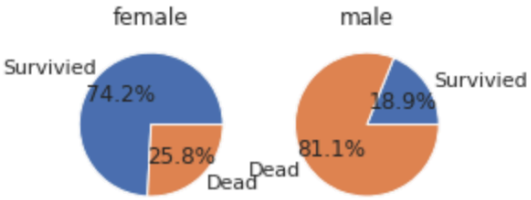
<3. Name>

- 이름에 ‘Mr’, ‘Miss’, “Mrs” 등 성별을 나타내는 단어가 있다.
- ‘Sex’ 속성을 사용하면 ‘Name’ 속성은 굳이 필요할 것 같지 않다.

<4. Sex>

- 승객 중 남성의 비율이 많다.
- 여성의 생존율이 높고 남성의 생존율이 낮다.
- 영화에서처럼 여성을 먼저 구조했을 것이라고 예측할 수 있다.



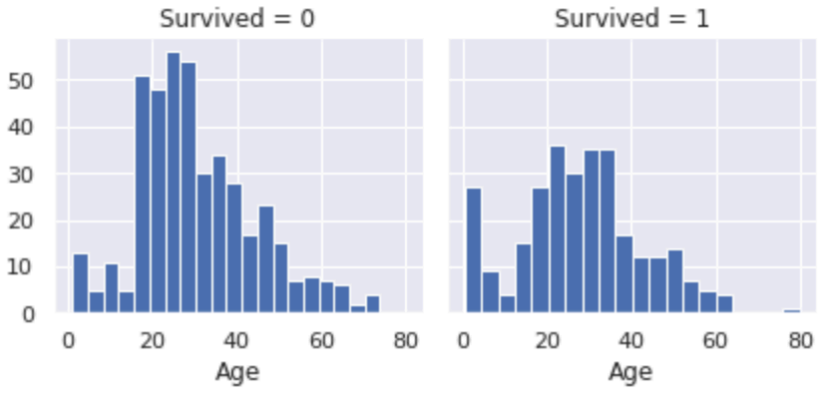


<5. Age>

- 20~30대의 생존비율이 특히 낮은 것을 확인할 수 있다.

```
g = sns.FacetGrid(train_data, col='Survived')
g.map(plt.hist, 'Age', bins=20)
```

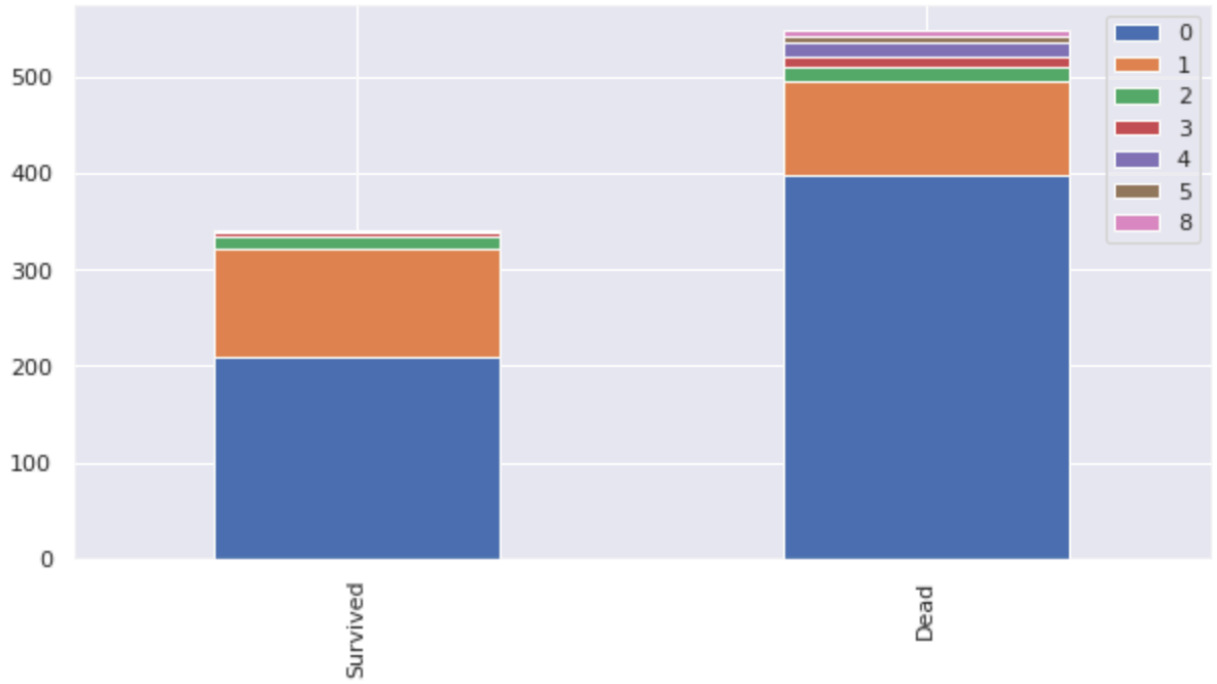
```
<seaborn.axisgrid.FacetGrid at 0x7fe81006cad0>
```



<6. SibSp>

- 자녀, 배우자와 함께 탑승하지 않은 사람들(파란색)의 생존 비율이 상대적으로 낮은 것을 볼 수 있다.

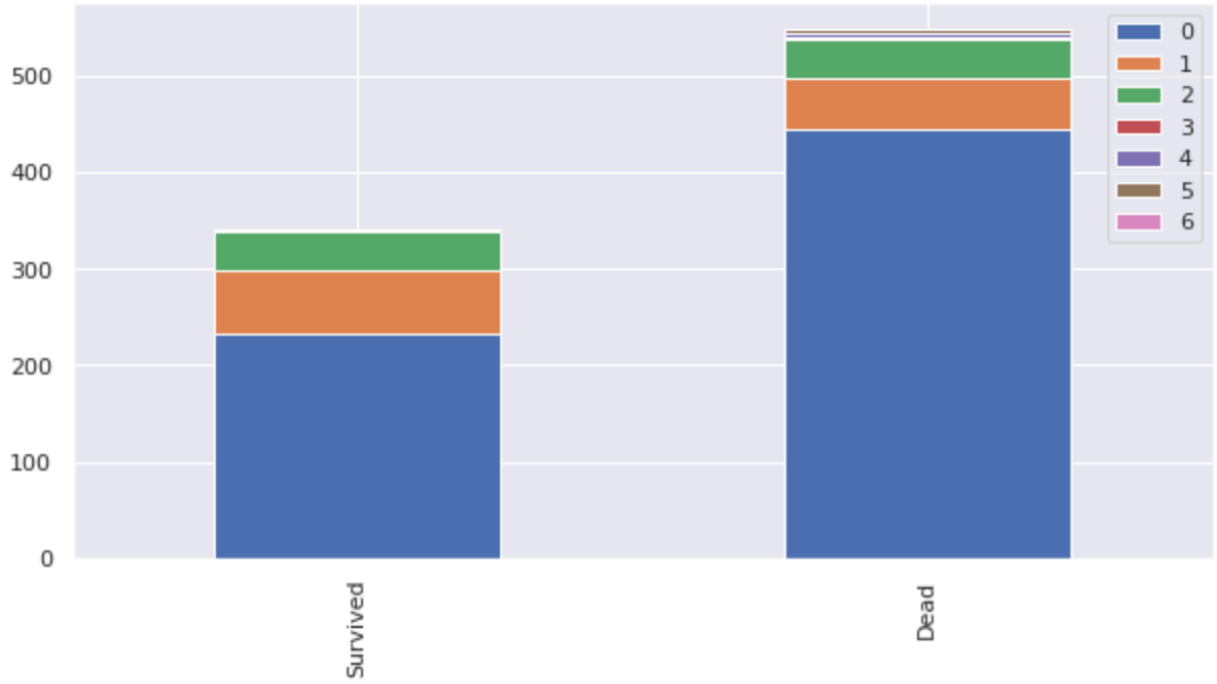
```
bar_chart("SibSp")
```



<7. Parch>

- Sibsp와 같이 부모, 자녀들과 함께 탑승하지 않은 사람들(파란색)의 생존 비율이 상대적으로 낮은 것을 볼 수 있다.

```
bar_chart("Parch")
```



<8. Ticket>

- 단순히 티켓 숫자를 나타낸 것이므로 생존 여부와 큰 연관이 없어 보인다.

<9. Fare>

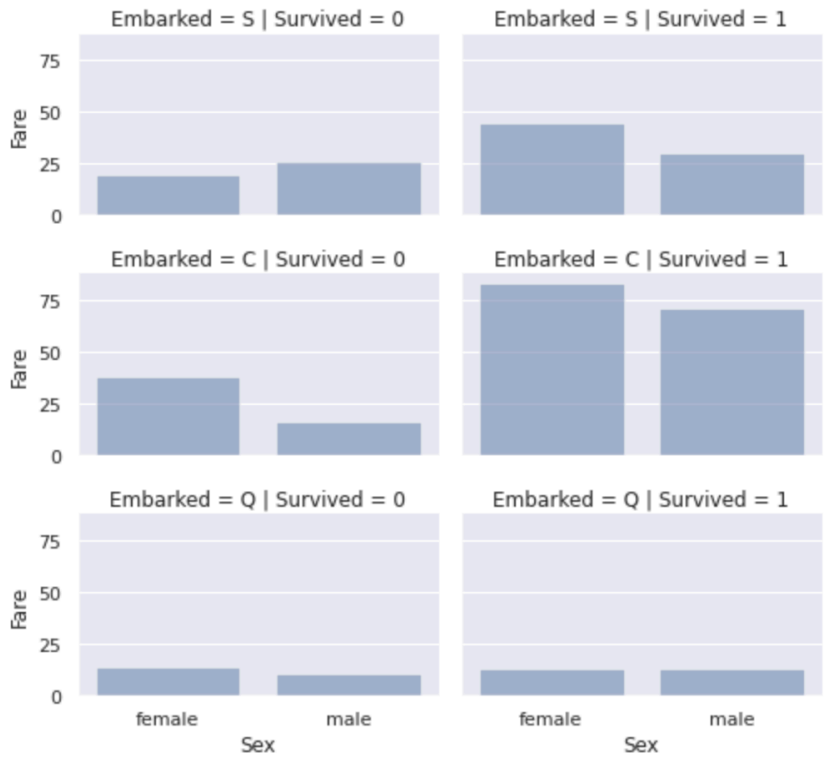
- Pclass가 높을수록 Fare가 높으므로 생존 여부와 관련이 있다.

<10. Cabin>

- 선실 번호를 뜻하는 속성이다.
- null 값이 많은 것으로 보아 그닥 중요한 값은 아닌 것 같다.

<11. Embarked>

- S, C 선착장에서 탄 사람들의 생존 비율이 높은 것을 확인할 수 있다.



5. 생존여부와 관련이 없는 column 제거

- ‘PassengerId’, ’Name’, ‘Ticket’, ‘Cabin’ column을 제거하였다.

```
train_data = train_data.drop(['PassengerId','Name','Ticket', 'Cabin'], axis=1)
test_data = test_data.drop(['PassengerId','Name','Ticket', 'Cabin'], axis=1)
combine = [train_data, test_data]
```

```
train_data.head()
```

|:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S

```
test_data.head()
```

:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	34.5	0	0	7.8292	Q
1	3	female	47.0	1	0	7.0000	S
2	2	male	62.0	0	0	9.6875	Q
3	3	male	27.0	0	0	8.6625	S
4	3	female	22.0	1	1	12.2875	S

6. 데이터 전처리

데이터를 training 시키기 위해 결측값을 채우고 데이터를 정제해야한다.

6-1. Sex

-Dtype이 object 이므로 string로 바꿔준다.

```
for dataset in combine:
    dataset['Sex'] = dataset['Sex'].astype(str)
```

6-2. Age

결측값을 KNN을 통해 채워주었다.

```
from sklearn.impute import KNNImputer
train_knn = train_data.copy(deep=True)

knn_imputer = KNNImputer(n_neighbors=2, weights="uniform")
train_knn['Age'] = knn_imputer.fit_transform(train_knn[['Ag
e']])
```

```
train_knn['Age'].isnull().sum()
```

```
test_knn = test_data.copy(deep=True)

knn_imputer = KNNImputer(n_neighbors=2, weights="uniform")
test_knn['Age'] = knn_imputer.fit_transform(test_knn[['Ag
e']])
```

```
test_knn['Age'].isnull().sum()
```

그 후 age 범위를 총 5개로 나누어서 0~4까지의 정수로 나타내었다.

```
train_data['AgeBand'] = pd.cut(train_data['Age'], 5)
train_data[['AgeBand', 'Survived']].groupby(['AgeBand'], as_i
ndex=False).mean().sort_values(by='AgeBand', ascending=True)
```

	AgeBand	Survived
0	(0.34, 16.336]	0.550000
1	(16.336, 32.252]	0.344168
2	(32.252, 48.168]	0.404255
3	(48.168, 64.084]	0.434783
4	(64.084, 80.0]	0.090909

```
for dataset in combine:
    dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
    dataset.loc[ dataset['Age'] > 64, 'Age'] = 4
train_data.head()
```

6-3. SibSp + Parch

가족과 함께 탑승한 사람의 생존 비율을 따지기 위해 FamilySize와 IsAlone 속성을 정의한다.

]:
```
for dataset in combine:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1

train_data[['FamilySize', 'Survived']].groupby(['FamilySize'], as_index=Fa
lse).mean().sort_values(by='Survived', ascending=False)
```

]:

	FamilySize	Survived
3	4	0.724138
2	3	0.578431
1	2	0.552795
6	7	0.333333
0	1	0.303538
4	5	0.200000
5	6	0.136364
7	8	0.000000
8	11	0.000000

```
for dataset in combine:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1

train_data[['IsAlone', 'Survived']].groupby(['IsAlone'], as_index=False).m
ean()
```

	IsAlone	Survived
0	0	0.505650
1	1	0.303538

6-4. Fare

test 데이터에 결측값이 하나 있는데, 이는 Pclass=3인 데이터이다.
Fare는 몇등석인지에 따라 값이 변하는 경향이 있으므로 3등석의 평균 요금으로 결측값을 채워준다.

```
print (train_data[['Pclass', 'Fare']].groupby(['Pclass'], as_index=False).
mean())
print("")
print(test_data[test_data["Fare"].isnull()]["Pclass"])
```

```
   Pclass       Fare
0       1  84.154687
1       2  20.662183
2       3  13.675550

152    3
Name: Pclass, dtype: int64
```

|스트 데이터의 fare 결측값 채우기 (3클래스의 평균)

```
for dataset in combine:
    dataset['Fare'] = dataset['Fare'].fillna(13.675)
```

그 후 Age 와 마찬가지로 범위를 나누어서 정수화 시킨다.

```
train_data['FareBand'] = pd.qcut(train_data['Fare'], 4)
train_data[['FareBand', 'Survived']].groupby(['FareBand'], as_index=Fals
e).mean().sort_values(by='FareBand', ascending=True)
```

	FareBand	Survived
0	(-0.001, 7.91]	0.197309
1	(7.91, 14.454]	0.303571
2	(14.454, 31.0]	0.454955
3	(31.0, 512.329]	0.581081

```
for dataset in combine:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'F
are'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Far
e']   = 2
    dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)

train_data = train_data.drop(['FareBand'], axis=1)
combine = [train_data, test_data]
```

6-5. Embarked

'S' 에서 탄 승객이 가장 많으므로 결측값을 S로 채워준다.

```
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].fillna('S')
    dataset['Embarked'] = dataset['Embarked'].astype(str)
```

7. 트레이닝

로지스틱 회귀, SVM, KNN, 랜덤 포레스트, 네이브 베이지안 방법을 사용한 결과 랜덤 포레스트의 정확도가 가장 높았다.

```
from sklearn.utils import shuffle
train_data, train_label = shuffle(train_data, train_label, random_state =
5)
```

정렬이 되어 있으면 트레이닝에 영향을 줄 수 있기 때문에 섞었다.

```
def train_and_test(model):
    model.fit(train_data, train_label)
    prediction = model.predict(test_data)
    accuracy = round(model.score(train_data, train_label) * 100, 2)
    print("Accuracy : ", accuracy, "%")
    return prediction
```

모델링 함수이다.

```
# Logistic Regression
log_pred = train_and_test(LogisticRegression())
# SVM
svm_pred = train_and_test(SVC())
#kNN
knn_pred_4 = train_and_test(KNeighborsClassifier(n_neighbors = 4))
# Random Forest
rf_pred = train_and_test(RandomForestClassifier(n_estimators=100))
# Navie Bayes
nb_pred = train_and_test(GaussianNB())
```

```
Accuracy :  79.57 %


/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.p
y:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)


Accuracy :  71.16 %
Accuracy :  85.52 %
Accuracy :  93.49 %
Accuracy :  77.22 %
```

8. 결과

Best Submission
✔ Successful
Submitted by NanyoungKim 20 hours ago

Public Score
0.71291

[성능 향상시키기]

1. Age 의 결측값 채우는 방법 바꾸기

Age의 결측값이 가장 많았기 때문에 이 결측값을 채우는 방식이 결과에 가장 큰 영향을 미쳤을 것이라고 판단하였다. 그래서 결측값을 채우는 방법을 KNN에서 MICE 로 바꾼 후 제출을 해보았다.

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
train_mice = train_data.copy(deep=True)

mice_imputer = IterativeImputer()
train_mice['Age'] = mice_imputer.fit_transform(train_mice[['Age']])
```

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
test_mice = test_data.copy(deep=True)

mice_imputer = IterativeImputer()
test_mice['Age'] = mice_imputer.fit_transform(test_mice[['Age']])
```

1-1. 결과

Best Submission
✔ Successful
Submitted by NanyoungKim 17 hours ago

Public Score
0.72727

성능이 약간 향상되었다.

2.첫번째 모델 오류 해결

Age를 5개의 범위로 나누어서 0~4의 정수를 부여했을 때 적용되지 않은 것을 확인하였다.

다시 정수 0~4를 적용하니 성능이 향상되었다.

```
for dataset in combine:
    dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
    dataset.loc[ dataset['Age'] > 64, 'Age'] = 4

train_data.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeBand
0	0	3	male	1	1	0	7.2500	S	(16.0, 32.0]
1	1	1	female	2	1	0	71.2833	C	(32.0, 48.0]
2	1	3	female	1	0	0	7.9250	S	(16.0, 32.0]
3	1	1	female	2	1	0	53.1000	S	(32.0, 48.0]
4	0	3	male	2	0	0	8.0500	S	(32.0, 48.0]

2-1 .결과

```
for dataset in combine:
    dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
    dataset.loc[ dataset['Age'] > 64, 'Age'] = 4
train_data.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeBand
0	0	3	male	22.0	1	0	7.2500	S	(16.336, 32.252]
1	1	1	female	38.0	1	0	71.2833	C	(32.252, 48.168]
2	1	3	female	26.0	0	0	7.9250	S	(16.336, 32.252]
3	1	1	female	35.0	1	0	53.1000	S	(32.252, 48.168]
4	0	3	male	35.0	0	0	8.0500	S	(32.252, 48.168]

* 첫번째 모델의 코드 : Age 컬럼을 보면 0~4까지의 정수로 분할되지 않았었다.

3. 결측값 채우는 방법 디테일하게 바꾸기

3-1. Name 속성 활용하기

첫번째 모델에서는 Name 열을 drop 시켰었는데 Name이 Age 의 결측값을 채울 때 도움이 될 수도 있을 것 같아 drop 시키지 않고 이용해보기로 했다.

```
for dataset in combine:
    dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=F
alse)

pd.crosstab(train_data['Title'], train_data['Sex'])
```

이름은

Name
Braund, Mr. Owen Harris
Cumings, Mrs. John Bradley (Florence Briggs Thayer)
Heikkinen, Miss. Laina
Futrelle, Mrs. Jacques Heath (Lily May Peel)
Allen, Mr. William Henry
Moran, Mr. James
McCarthy, Mr. Timothy J

이므로 가운데의 Mr / Mrs / Miss 등을 추출하였다.

Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1

- Don / Dona: = sir(상류층)
- Mme : Madame
- Capt : Captain
- Lady : 상류층 숙녀
- the Countess : Countess(여자 백작)
- mlle : Mademoiselle
- Jonkheer : 낮은 직급의 귀족
- Rev : 목사/신부
- Col : 군인 관련
- Master : Master는 뜻이 많다.
- Major : 소령

사회적 지위와 나이를 유추할 수 있다. 종류가 많으므로 비슷한 뜻끼리 그룹을 만든다.

```
Title_Dict.update(dict.fromkeys(['Capt', 'Col', 'Major', 'Dr', 'Rev'], 'Officer'))
Title_Dict.update(dict.fromkeys(['Don', 'Sir', 'the Countess', 'Dona', 'Lady'], 'Royalty'))
Title_Dict.update(dict.fromkeys(['Mme', 'Ms', 'Mrs'], 'Mrs'))
Title_Dict.update(dict.fromkeys(['Mlle', 'Miss'], 'Miss'))
Title_Dict.update(dict.fromkeys(['Mr'], 'Mr'))
Title_Dict.update(dict.fromkeys(['Master','Jonkheer'], 'Master'))

all_data['Title'] = all_data['Title'].map(Title_Dict)
sns.barplot(x="Title", y="Survived", data=all_data)
```

3-2. Age 결측값 채우기

Pclass, Sex, Title을 기반으로 Random Forest를 이용하여 Age 의 결측값을 채웠다.

```
from sklearn.ensemble import RandomForestRegressor
age_df = all_data[['Age', 'Pclass','Sex','Title']]
age_df=pd.get_dummies(age_df)
known_age = age_df[age_df.Age.notnull()].iloc[:,:].values
unknown_age = age_df[age_df.Age.isnull()].iloc[:,:].values
y = known_age[:, 0]
X = known_age[:, 1:]
rfr = RandomForestRegressor(random_state=0, n_estimators=100, n_jobs=-1)
rfr.fit(X, y)
predictedAges = rfr.predict(unknown_age[:, 1::])
all_data.loc[ (all_data.Age.isnull()), 'Age' ] = predictedAges
```

3-3. Embarked 결측값 채우기

```
all_data[all_data['Embarked'].isnull()]
```

ed	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	FamilyS
	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0	B28	NaN	1
	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0	0	0	113572	80.0	B28	NaN	1

이 두 사람은 1등석을 탔으며 80의 요금을 지불하였다. 선착장이 다르면 운항 거리가 달라지므로 요금 또한 달라질 것이며 등급 또한 당연히 요금에 영향이 미치므로 이들의 관계를 적용하여 어디 선착장에서 탔는지 유추해볼 수 있다.

```
all_data.groupby(by=["Pclass","Embarked"]).Fare.median()
```

```
Pclass  Embarked
1       C           76.7292
        Q           90.0000
        S           52.0000
2       C           15.3146
        Q           12.3500
        S           15.3750
3       C            7.8958
        Q            7.7500
        S            8.0500
Name: Fare, dtype: float64
```

```
all_data['Embarked'] = all_data['Embarked'].fillna('C')
```

1등석의 요금 중 80과 가장 가까운 평균 요금 76.7292는 선착장 C에서 탔을 때의 요금이므로 결측값을 C로 채워준다.

3-4. Fare 결측값 채우기

```
all_data[all_data['Fare'].isnull()]
```

Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Fa
NaN	3	Storey, Mr. Thomas	male	60.5	0	0	3701	NaN	Unknown	S	1

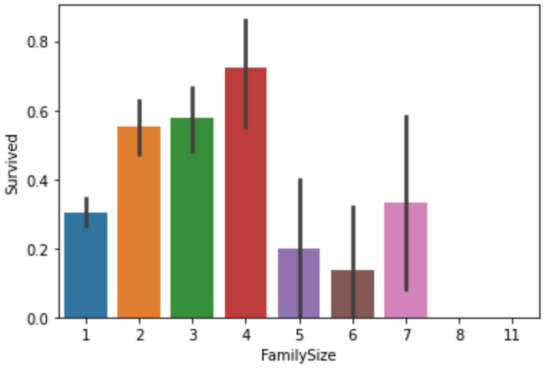
이 사람은 'S' 선착장에서 3등급 티켓을 끊은 사람이다. 위에서 말했던 것과 같이 Fare는 선착장과 등급과 관련이 있으므로 'S' 선착장에서 3등급 티켓을 끊었을 때의 요금의 평균값을 계산하여 결측값을 채웠다.

```
fare=all_data[(all_data['Embarked'] == "S") & (all_data['Pclass'] == 3)].F
are.median()
all_data['Fare']=all_data['Fare'].fillna(fare)
```

3-5. 기존 모델의 IsAlone 속성은 함께 탑승한 가족의 유무만을 판단했는데, SibSp와 Parch 속성을 이용하여 FamilySize를 구하고 이를 정수로 카테고리화 하였다.

```
all_data['FamilySize']=all_data['SibSp']+all_data['Parch']+1
sns.barplot(x="FamilySize", y="Survived", data=all_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc7fa16750>
```



가족의 수가 2,3,4 명인 경우가 비교적 많으므로 첫번째 그룹, 나머지 1,5,6,7인 경우에는 두번째 그룹, 8이상인 경우가 세번째 그룹으로 지정하였다.

3-6. 트레이닝

랜덤 포레스트와 교차 검증을 적용하였다.

```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import SelectKBest
select = SelectKBest(k = 20)
clf = RandomForestClassifier(random_state = 10, warm_start = True,
                                  n_estimators = 26,
                                  max_depth = 6,
                                  max_features = 'sqrt')
pipeline = make_pipeline(select, clf)
pipeline.fit(X, y)
```

```
from sklearn import model_selection, metrics
cv_score = model_selection.cross_val_score(pipeline, X, y, cv= 10)
print("CV Score : Mean - %.7g | Std - %.7g " % (np.mean(cv_score), np.std(cv_scor
e)))
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/feature_selection/_univariate_se
lection.py:114: UserWarning: Features [23] are constant.
  UserWarning)
/opt/conda/lib/python3.7/site-packages/sklearn/feature_selection/_univariate_se
lection.py:115: RuntimeWarning: invalid value encountered in true_divide
  f = msb / msw


CV Score : Mean - 0.8462422 | Std - 0.03623982
```

3-7. 결과

결측값을 채울때 디테일하게 고려했더니 가장 높은 점수가 나온것을 확인할 수 있었다.

4. AgeBand 적용하기

위의 3번 모델에서 Age의 구간을 나누어 모델링을 해보았다.

```
all_data = all_data.drop(['AgeBand'], axis=1)
all_data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	FamilySize	FamilyLabel	Title	Deck	TicketGroup
0	1	0.0	3	Braund, Mr. Owen Harris	male	1.0	1	0	A/5 21171	7.2500	Unknown	S	2	2	Mr	U	1
1	2	1.0	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	2.0	1	0	PC 17599	71.2833	C85	C	2	2	Mrs	C	2
2	3	1.0	3	Heikkinen, Miss. Laina	female	1.0	0	0	STON/O2. 3101282	7.9250	Unknown	S	1	1	Miss	U	1
3	4	1.0	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	2.0	1	0	113803	53.1000	C123	S	2	2	Mrs	C	2
4	5	0.0	3	Allen, Mr. William Henry	male	2.0	0	0	373450	8.0500	Unknown	S	1	1	Mr	U	1

4-1. 결과

점수가 3번에 비해 떨어졌다.

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission.csv	just now	0 seconds	0 seconds	0.80861

Complete

Jump to your position on the leaderboard

[결과 및 고찰]

유명한 영화인 타이타닉의 생존자를 예측한다는 주제덕에 처음에 흥미를 가지고 과제를 시작할 수 있었다.

우선 첫번째 모델을 만들 때에는 데이터 분포값을 보고 나름의 예측을 하여 큰 연관이 없을 것 같은 속성들 (‘PassengerId’, ’Name’, ‘Ticket’, ‘Cabin’)을 배제시키고 모델링을 하였다. 그러나 Name에는 Mrs, Mr, Miss 와 같은 성별을 알 수 있는 단어 뿐만이 아니라 Dr, Master, Lady(상류층 여성), Countess(여성 백작) 등 과거의 사회적 지위를 알 수 있는 단어들도 포함되어 drop 하지 않고 데이터를 활용하였더니 Score가 높아진 것을 확인할 수 있었다.

그리고 결측값을 채우는 과정이 중요했다. 단순히 평균으로만 채워넣는 것이 아니라 데이터의 분포를 그래프로 그려 파악하고 넣음으로써 Score를 높일 수 있었다.

또한 관련이 없어 보이지만 여러 속성들간에 연관성이 있는지를 확인하여 Age의 결측값을 채움으로써 Score를 높일 수 있었다.

아쉬웠던 점은 Ticket 속성 혹은 Cabin 속성이 정확히 데이터 분포에 어떤 영향을 끼치는지를 파악하지 못한 것이었다. 예상을 해보면 Ticket은 Fare와 관련이 있을 것이고 이는 class 와도 연관이 있을 것이다. 즉 사회적 지위가 높은 사람들은 값이 비싼 Ticket 종류를 구매했을 것이고 사회적 지위가 낮은 사람들은 class 가 낮고 fare 가 상대적으로 져렴한 Ticket을 구매했을 것이다.

또한 Cabin은 선실 위치를 나타내는데 결측값이 너무 많았다. 자리 위치가 어디였느냐에 따라 침수 피해 정도가 달라졌을 것인데 이 Cabin 값을 어떻게 활용해야 결과를 향상시킬 수 있을지 감이 잡히지 않아 제대로 활용하지 못한 것 같다.

이번 과제에서 891명의 training data만을 이용하여 418명의 생존 여부를 약 80%의 확률로 맞출 수 있었는데 생각보다 높은 정확도로 예측을 할 수 있어서 굉장히 흥미로웠던 과제였다. 이번 타이타닉 과제를 통해 빅데이터의 활용이 실생활에 적용되는 것을 새삼 확인할 수 있었다.