

타이타닉 생존 예측 프로젝트

최종발표

데이터 사당행

김도빈, 염수민, 조건우, 최세현

목차

01

프로젝트 소개

02

EDA

03

전처리

04

모델링

05

보완점

PART 1. 프로젝트 소개

01 프로젝트 소개

타이타닉 생존 예측 프로젝트

정형, 분류, 초급

📁 초급 프로젝트

🕒 3 시간 - <> 6 스테이지

👤 1622 명



실제 타이타닉호에 승선한 승객 데이터를 가지고 파이썬 기반의 데이터 분석 및 인공지능 기술을 활용하여 타이타닉호의 생존자를 예측하는 프로젝트

PART 2. EDA

변수 확인

PassengerID	승객 고유의 아이디	Sibsp	함께 탑승한 형제자매, 아내, 남편의 수
Survival	탑승객 생존 유무	Parch	함께 탑승한 부모, 자식의 수
Pclass	등실의 등급	Ticket	티켓 번호
Name	이름	Fare	티켓의 요금
SEX	성별	Cabin	객실번호
Age	나이	Embarked	배에 탑승한 항구의 이름

결측치 확인

```
[8] train.isna().sum()
```

PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2
dtype: int64	

Age : 177개

Cabin : 687개

Embarked: 2개

```
test.isna().sum()
```

PassengerId	0
Pclass	0
Name	0
Sex	0
Age	86
SibSp	0
Parch	0
Ticket	0
Fare	1
Cabin	327
Embarked	0
dtype: int64	

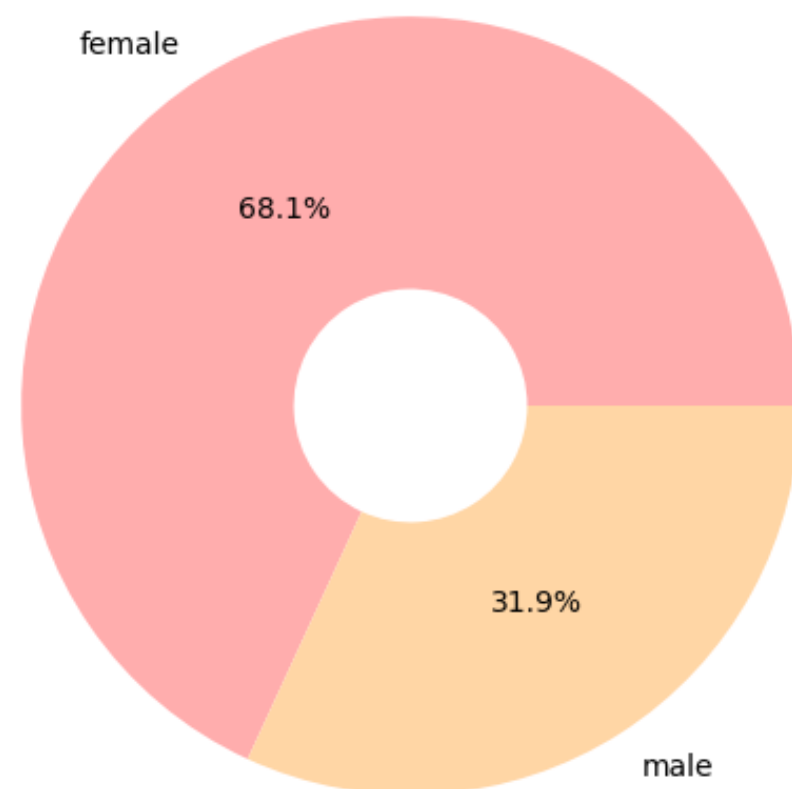
Age : 86개

Fare : 1개

Cabin : 327개

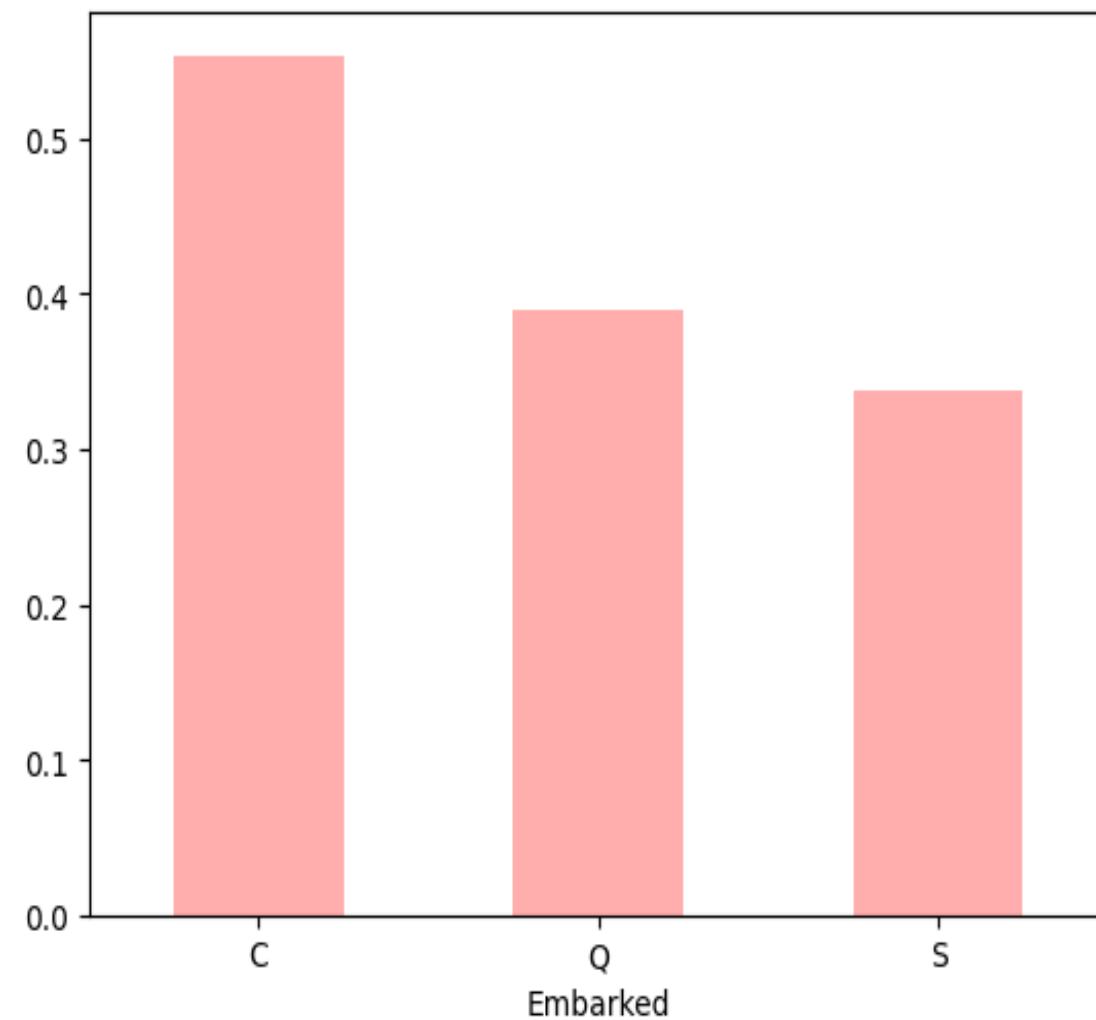
시각화

Survival Rate by Sex



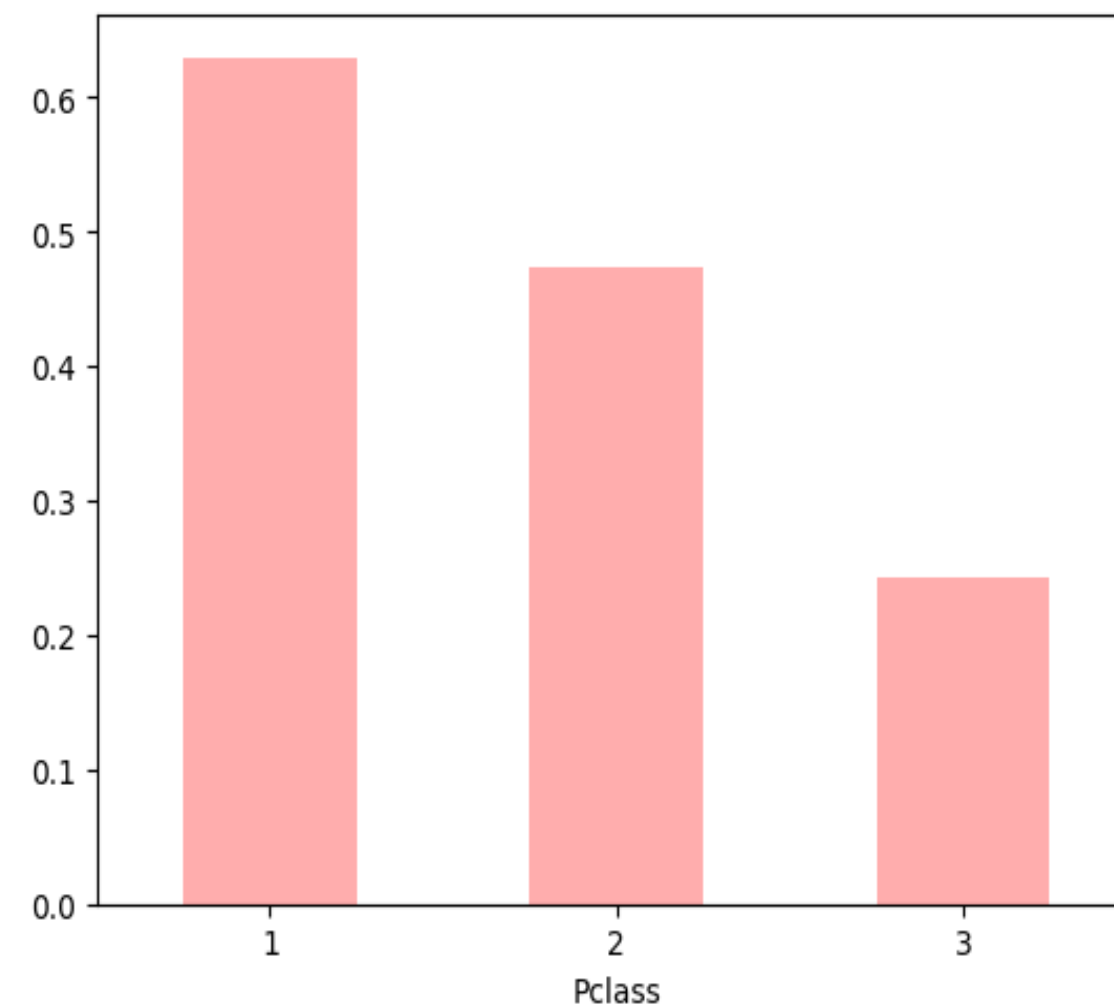
성별에 따른 생존 비율

생존자 중 여성의 비율이 68.1%로
남성보다 높음



탑승 항구 별 생존율

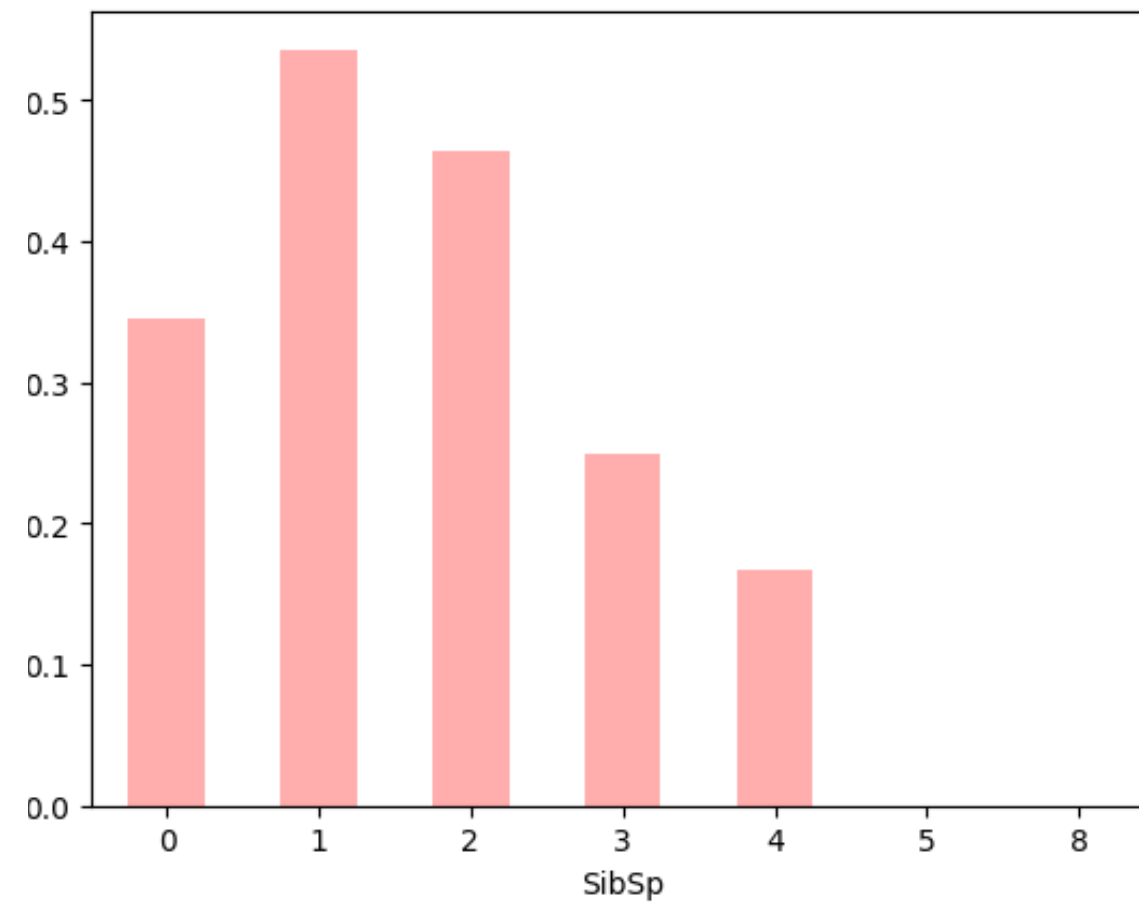
C (Cherbourg) 항구에서 탑승한 승객들의
생존율이 가장 높음



객실 등급 별 생존율

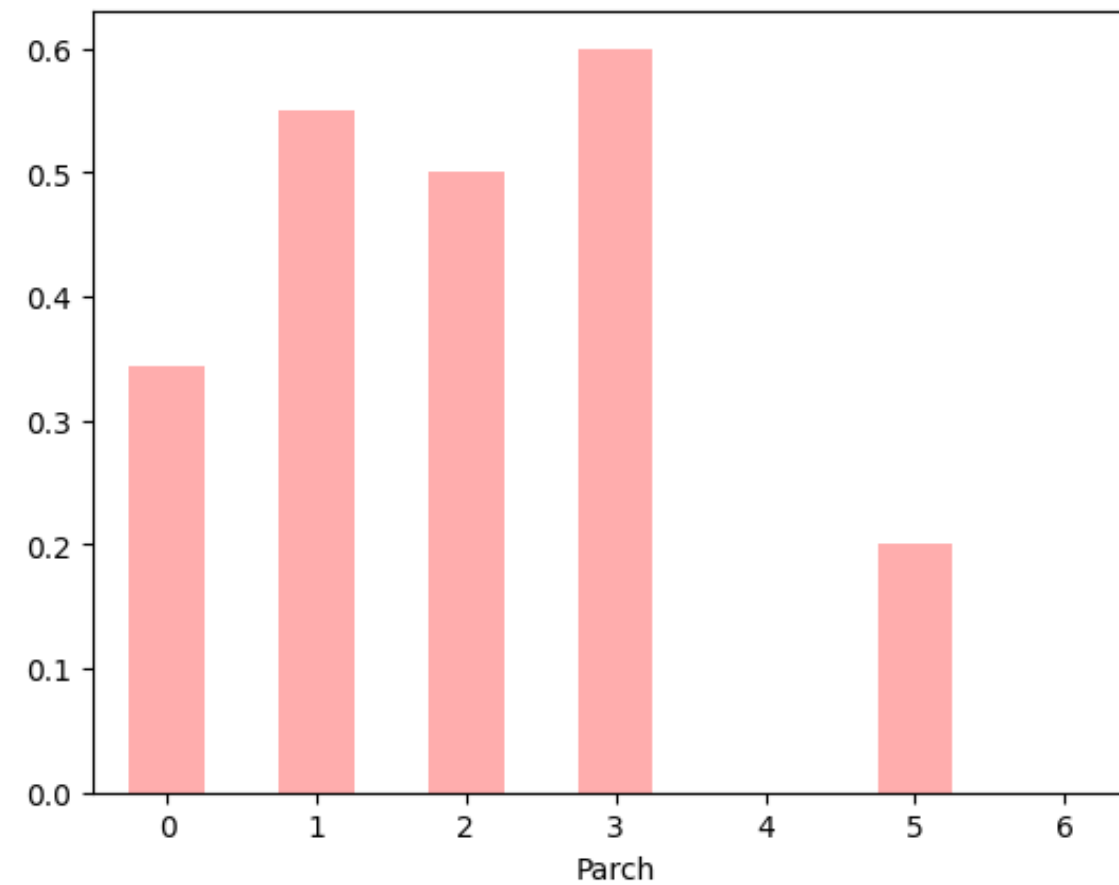
1등급, 2등급, 3등급 순으로 생존율이 높음

시각화



**함께 탑승한 형제자매,
아내, 남편의 수와 생존율**

1~2명과 함께 탑승했을 때 생존율이 높음

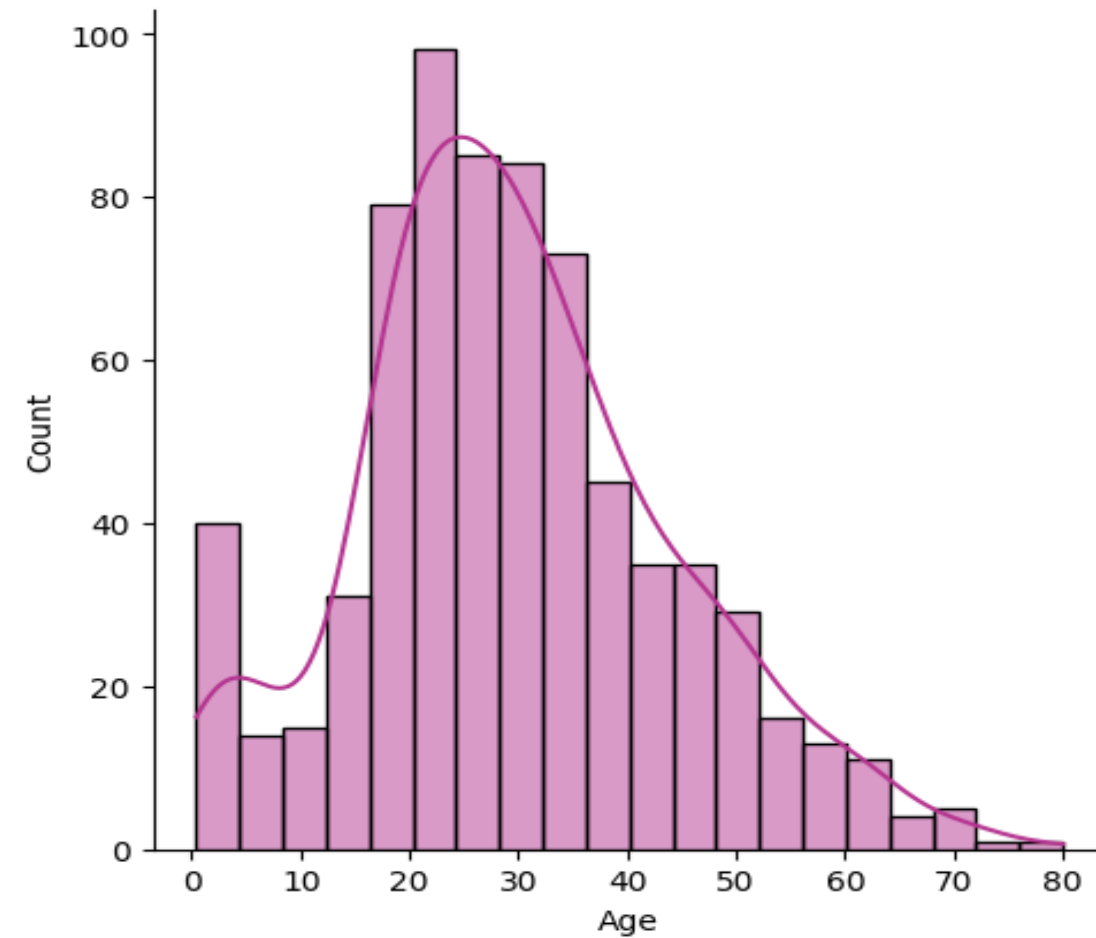


**함께 탑승한 부모,
자식의 수와 생존율**

1~3명과 함께 탑승했을 때 생존율이 높음

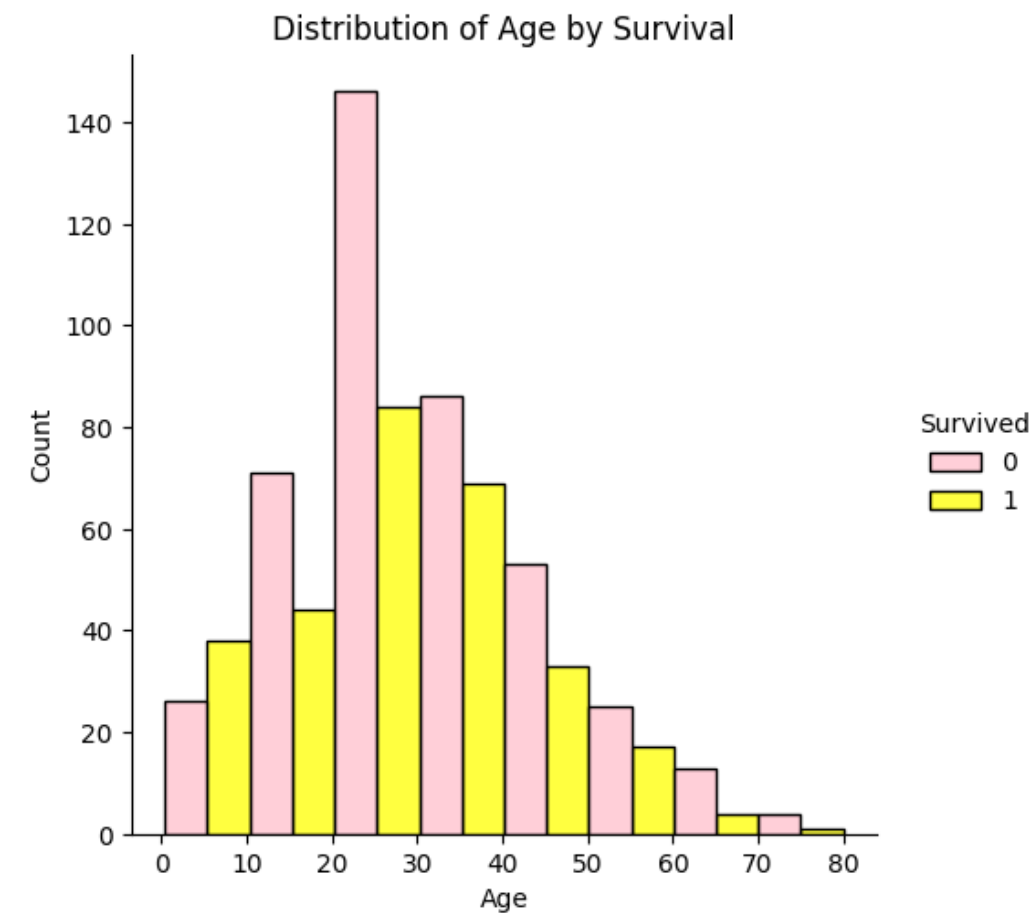
두 그래프가 비슷한 모양을 띠

시각화



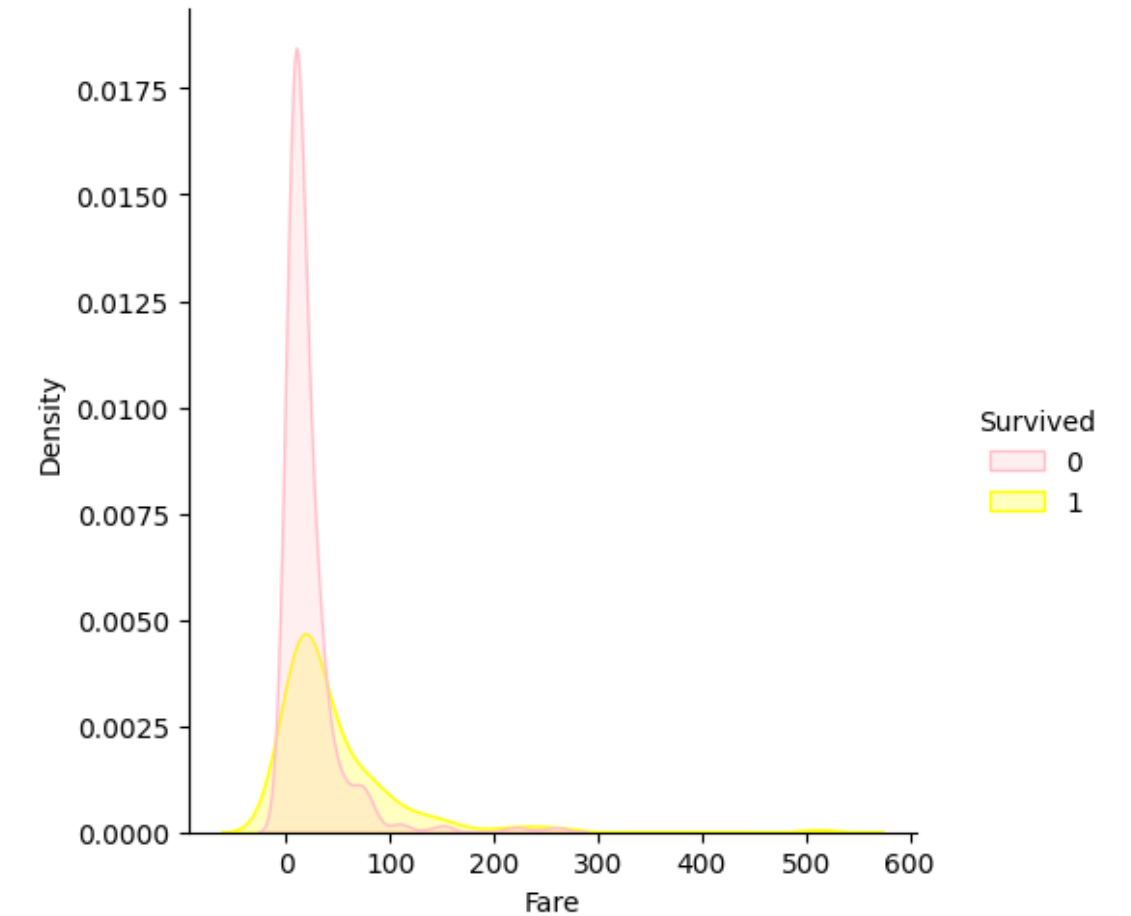
나이 별 탑승객의 수

20대~30대 탑승객이 가장 많은 것을
볼 수 있음



나이대 별 생존자 수

어린이와 노인을 제외한 나이대는
생존자 수보다 사망자 수가 더 높은 것을 볼
수 있음



티켓 요금과 생존율

티켓 요금이 높을 수록 사망률에 비해 생존율
이 높은 것을 볼 수 있음

PART 3. 전처리

03 전처리

데이터 정제

결측치 처리

1. Age 변수: 모델별로 이름과 등급별 평균으로 채움
2. Fare 변수: 결측치 행 확인 후 3등급 평균으로 채움
3. Embarked 변수: 최빈값인 'S'로 채움
4. 이상치 처리: Age와 Fare 모두 잘못된 값이라고 보기 어려움

전처리

```
print('Mr의 평균 나이:', train[train['Title']==0]['Age'].mean())
print('Miss의 평균 나이:', train[train['Title']==1]['Age'].mean())
print('Mrs의 평균 나이:', train[train['Title']==2]['Age'].mean())
print('Master의 평균 나이:', train[train['Title']==3]['Age'].mean())
```

```
Mr의 평균 나이: 32.98441247002398
Miss의 평균 나이: 22.02
Mrs의 평균 나이: 35.88288288288288
Master의 평균 나이: 4.574166666666667
```

```
train.loc[train['Title']==0, ['Age']] = train.loc[train['Title']==0, ['Age']].fillna(32.98)
train.loc[train['Title']==1, ['Age']] = train.loc[train['Title']==1, ['Age']].fillna(22.02)
train.loc[train['Title']==2, ['Age']] = train.loc[train['Title']==2, ['Age']].fillna(35.88)
train.loc[train['Title']==3, ['Age']] = train.loc[train['Title']==3, ['Age']].fillna(4.57)
```

```
✓ [1369] print('1등급의 평균 나이:', train[train['Pclass']==1]['Age'].mean())
0초 print('2등급의 평균 나이:', train[train['Pclass']==2]['Age'].mean())
print('3등급의 평균 나이:', train[train['Pclass']==3]['Age'].mean())
```

```
1등급의 평균 나이: 38.233440860215055
2등급의 평균 나이: 29.87763005780347
3등급의 평균 나이: 25.14061971830986
```

```
✓ [1370] train.loc[train['Pclass']==1, ['Age']] = train.loc[train['Pclass']==1, ['Age']].fillna(38.23)
0초 train.loc[train['Pclass']==2, ['Age']] = train.loc[train['Pclass']==2, ['Age']].fillna(29.88)
train.loc[train['Pclass']==3, ['Age']] = train.loc[train['Pclass']==3, ['Age']].fillna(25.14)
```

```
test.loc[test['Pclass']==3, ['Fare']] = test.loc[test['Pclass']==3, ['Fare']].fillna(12.46)
```

```
✓ [14] train['Embarked'] = train['Embarked'].fillna('S')
0초
```

03 전처리

데이터 변환

변수 변환

1. Sex, Embarked 문자형 데이터를 수치형 데이터로 변환

2. 데이터 카테고리화

-Age 변수: Age_band로 새로운 변수 생성 후 카테고리화

-Fare 변수: Fare_band로 새로운 변수 생성 후 4분위수를 기준으로 카테고리화

-Name 변수: Title 변수로 카테고리화

전처리

```
[15] train['Sex'] = train['Sex'].map({'male':0, 'female':1})
```

```
[16] test['Sex'] = test['Sex'].map({'male':0, 'female':1})
```

```
[17] train['Embarked'] = train['Embarked'].map({'S':0, 'C':1, 'Q':2})
```

```
[18] test['Embarked'] = test['Embarked'].map({'S':0, 'C':1, 'Q':2})
```

```
[31] train['Fare_band'] = 0
      train.loc[train['Fare'] < 7.91, 'Fare_band'] = 0
      train.loc[(train['Fare'] >= 7.91) & (train['Fare'] < 14.45), 'Fare_band'] = 1
      train.loc[(train['Fare'] >= 14.45) & (train['Fare'] < 31), 'Fare_band'] = 2
      train.loc[train['Fare'] >= 31, 'Fare_band'] = 3
```

```
[33] train['Age_band'] = 0
      train.loc[train['Age'] < 6, 'Age_band'] = 'Baby'
      train.loc[(train['Age'] >= 6) & (train['Age'] < 13), 'Age_band'] = 'Child'
      train.loc[(train['Age'] >= 13) & (train['Age'] < 19), 'Age_band'] = 'Teenager'
      train.loc[(train['Age'] >= 19) & (train['Age'] < 65), 'Age_band'] = 'Adult'
      train.loc[train['Age'] >= 65, 'Age_band'] = 'Elderly'
```

```
data_list = [train, test]
for data in data_list:
    data['Title'] = data['Name'].str.extract('([A-Za-z]+)\.', expand=False)
    data['Title'] = data['Title'].replace(['Ms', 'Lady', 'Mlle'], 'Miss')
    data['Title'] = data['Title'].replace(['Mme', 'Dona', 'Countess'], 'Mrs')
    data['Title'] = data['Title'].replace(['Don', 'Capt', 'Rev', 'Jonkheer', 'Major'], 'Mr')
    data['Title'] = data['Title'].replace(['Col', 'Sir', 'Dr'], 'Mr')
train[['Title', 'Survived']].groupby(['Title']).mean()
test['Title'].value_counts()
```

데이터 통합

파생 변수

1. Sibsp와 Parch를 통합해
FamilySize라는 새로운
변수 생성

전처리

```
train['FamilySize'] = train['SibSp'] + train['Parch'] + 1
```

```
test['FamilySize'] = test['SibSp'] + test['Parch'] + 1
```

데이터 축소

변수 삭제

1. 카테고리화를 통해 Name, Sibsp, Parch 변수 삭제
2. PassengerId, Ticket, Cabin 변수는 생존여부에 영향을 미치지 않는다고 판단하여 삭제

전처리

```
9] drop_list = ['SibSp', 'Parch', 'Name']  
for data in data_list:  
    data.drop(drop_list, inplace=True, axis=1)
```

```
0] drop_list = ['PassengerId', 'Ticket', 'Cabin']  
for data in data_list:  
    data.drop(drop_list, inplace=True, axis=1)
```

PART 4. 모델링

로지스틱 회귀분석

```
[525] submission.to_csv('logistic_regression_pred_Age_N_Fare_NNN.csv', index = False)
```

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
```

```
score = cross_validate(model, X_train, y_train,
                        return_train_score=True, n_jobs=-1,
                        cv = StratifiedKFold())
print(np.mean(score['train_score']), np.mean(score['test_score']))
```

```
0.8356704591002991 0.8202107751403526
```

Age, Fare 카테고리화 X

Age 결측치 이름별 평균으로 채움

train data의 정확도: 83%

test data의 정확도: 82%

DACON 점수: 0.8153

제목

제출 일시

점수

제출선택

993078

logistic_regression_pred_Age_N_Fare_NNN.csv

Age, Fare X 나이 결측치 이름으로 edit

2024-02-27 14:34:28

0.8153237585



의사결정나무

```
# 의사결정나무 모델 학습
dt.fit(X_train, y_train)

# 학습 데이터에 대한 예측 결과 계산
y_pred_train = dt.predict(X_train)

# 학습 데이터에 대한 정확도 계산
accuracy_train = accuracy_score(y_train, y_pred_train)
print("학습 데이터 정확도:", accuracy_train)

# 테스트 데이터에 대한 예측 결과 계산
y_pred_test = dt.predict(X_test)

# 테스트 데이터에 대한 정확도 계산
submission['Survived'] = y_pred_test
accuracy_test = accuracy_score(pd.read_csv('/content/drive/MyDrive/데이터/submission.csv')['Survived'], y_pred_test)
print("테스트 데이터 정확도:", accuracy_test)

학습 데이터 정확도: 0.8432401795735129
테스트 데이터 정확도: 0.8851674641148325
```

decision_tree_pred (8).csv

edit

2024-02-28 19:29:27 0.806365628

Age, Fare 카테고리화 X

Age 결측치 이름별 평균으로 채움

train data의 정확도: 84%

test data의 정확도: 88%

DACON 점수: 0.8063

XGBoost

```
xgb = XGBClassifier(n_estimators=100, max_depth=4, random_state=32)
xgb.fit(X_train, y_train)
Y_pred = xgb.predict(X_test)
xgb.score(X_train, y_train)
submission.to_csv('xgb_boost_pred.csv', index = False)
```

```
model = XGBClassifier(n_estimators=100, learning_rate=0.2, max_depth=4, random_state=32)
score = cross_validate(model, train_input, train_target,
                      return_train_score=True, n_jobs=-1,
                      cv = StratifiedKFold())
print(np.mean(score['train_score']), np.mean(score['test_score']))
```

J 866 604436311203 0.8282907538760907

993222

Age o, Fare o, Name 결측치.csv

[edit](#)

2024-02-27 17:29:50

0.8036514119

Age, Fare 카테고리화 O

Age 결측치 이름별 평균으로 채움

트리의 수: 100 최대 한도 깊이: 4

train data의 정확도: 86%

test data의 정확도: 82%

DACON 점수: 0.8036

랜덤 포레스트

Age, Fare 카테고리화 X

Age 결측치 이름별 평균으로 채움

트리의 수: 100 최대 한도 깊이: 5

train data의 정확도: 85%

test data의 정확도: 82%

DACON 점수: 0.8206

```
[711] model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=0)
score = cross_validate(model, train_input, train_target,
                        return_train_score=True, n_jobs=-1,
                        cv = StratifiedKFold())
print(np.mean(score['train_score']), np.mean(score['test_score']))

0.8552240099595 0.8282781997363632
```


```
[712] from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=100, max_depth=5, random_state=0)
rf.fit(X_train, y_train)
rf_pred=rf.predict_proba(X_test)[:,-1]
submission['Survived']=rf_pred
submission.to_csv('n--*--_Forest_pred.csv', index=False)
```

993201 Random_Forest_pred (Age X, Fare X, 이름)(파라미터 100).csv
edit

2024-02-27 17:04:10 0.8206304771



최종 모델

993201	Random_Forest_pred (Age X, Fare X, 이름)(파라미터 100).csv edit	2024-02-27 17:04:10	0.8206304771	
--------	--	---------------------	--------------	---

최종 모델: 랜덤 포레스트로 결정

DACON 점수: 랜덤 포레스트 (0.8206)> 로지스틱 회귀분석 (0.8153)> 의사결정나무 (0.8063)> XGBoost (0.8036)

PART 5. 보완점

보완점

보완점 : -삭제된 변수, 이상치 처리 등 다양한 변수를 더 고려해서 모델링 해보지 못해 아쉬움

-모델 선택 전 모델에 대해 충분히 이해하기

-모델링 진행하면서 정확성과 데이콘 점수 중 어느 것을 우선적으로 생각해야 할 지 고민이 많았는데
그 부분에 대해 공부하기

- GridSearchCV에 대해 공부하기

Q & A