Titanic Dataset

KARUT 1회차
한규탁

Content

CHAPTER

01

Dataset Info

Titanic Dataset Colums

—

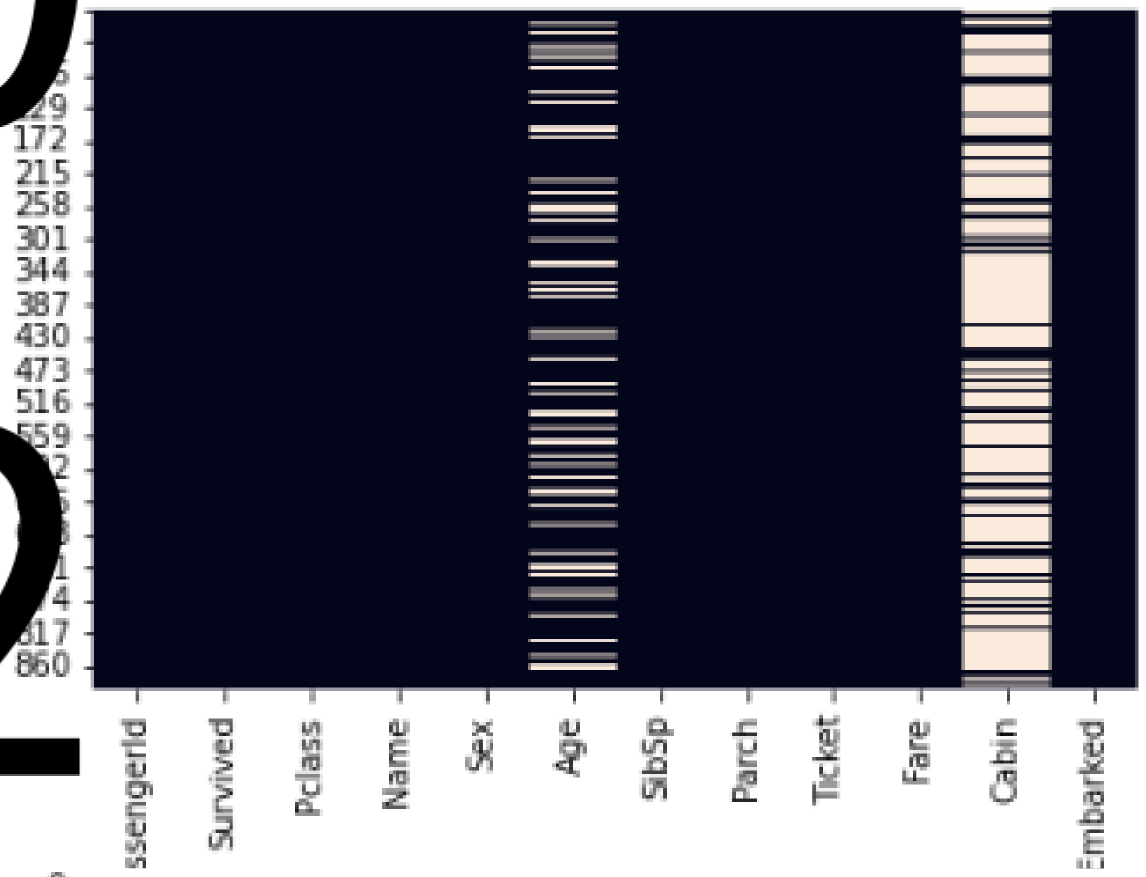
Survived : 생존여부
Pclass : 좌석 등급
Sex : 성별
Age : 당시 나이
Sibsp : 형제자매 수
Parch : 자녀 수
Fare : 좌석 요금
Cabin : 선실
Embarked : 승선 항구

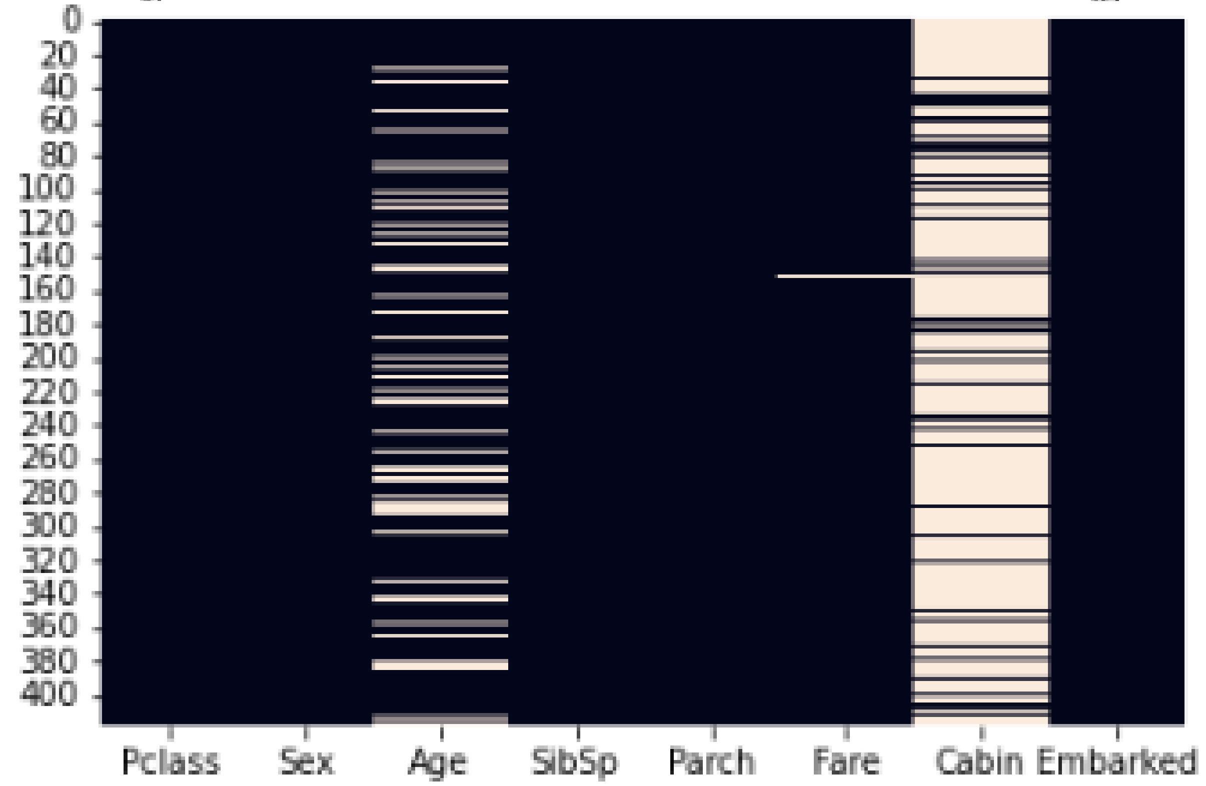
01

CHAPTER 02

Missing Value

02 Check Missing Value





— Train set

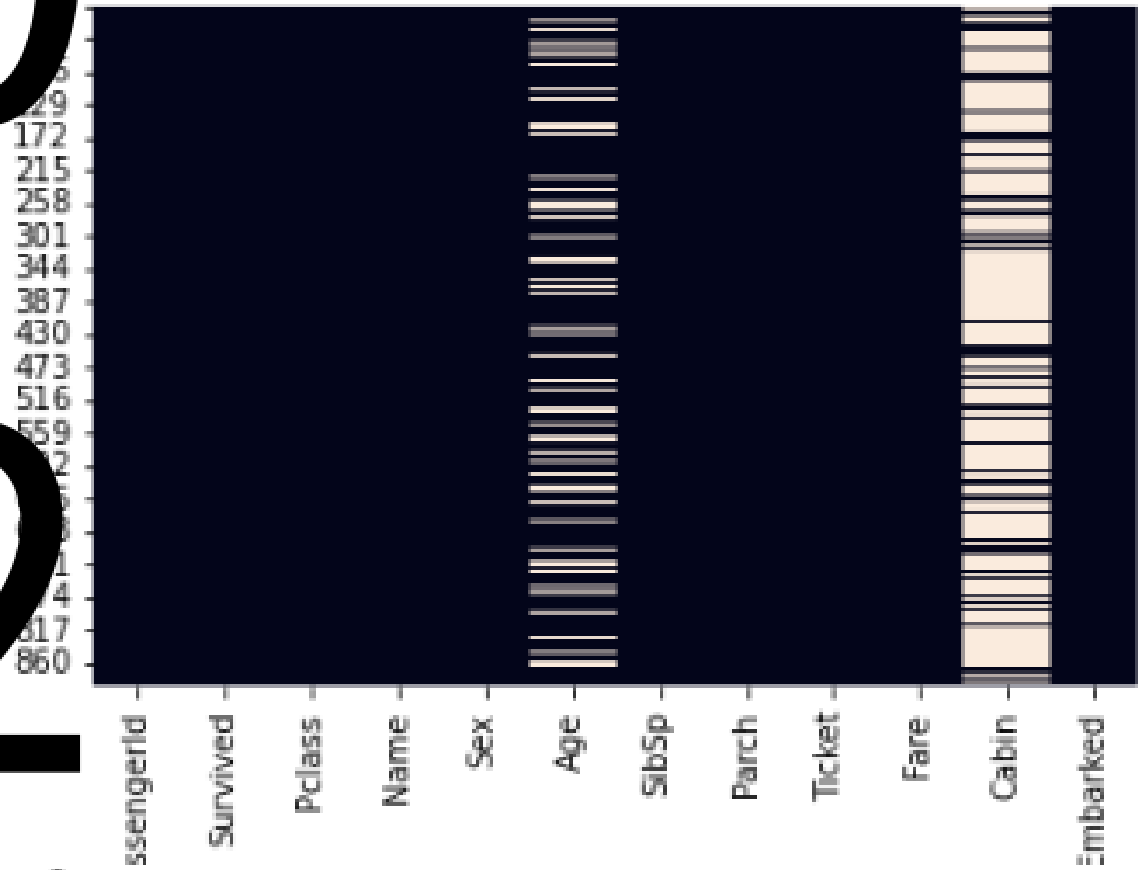
- Carbin, Age 데이터에서 결측치 다수 관측
- Embarked 데이터에서 소량 관측

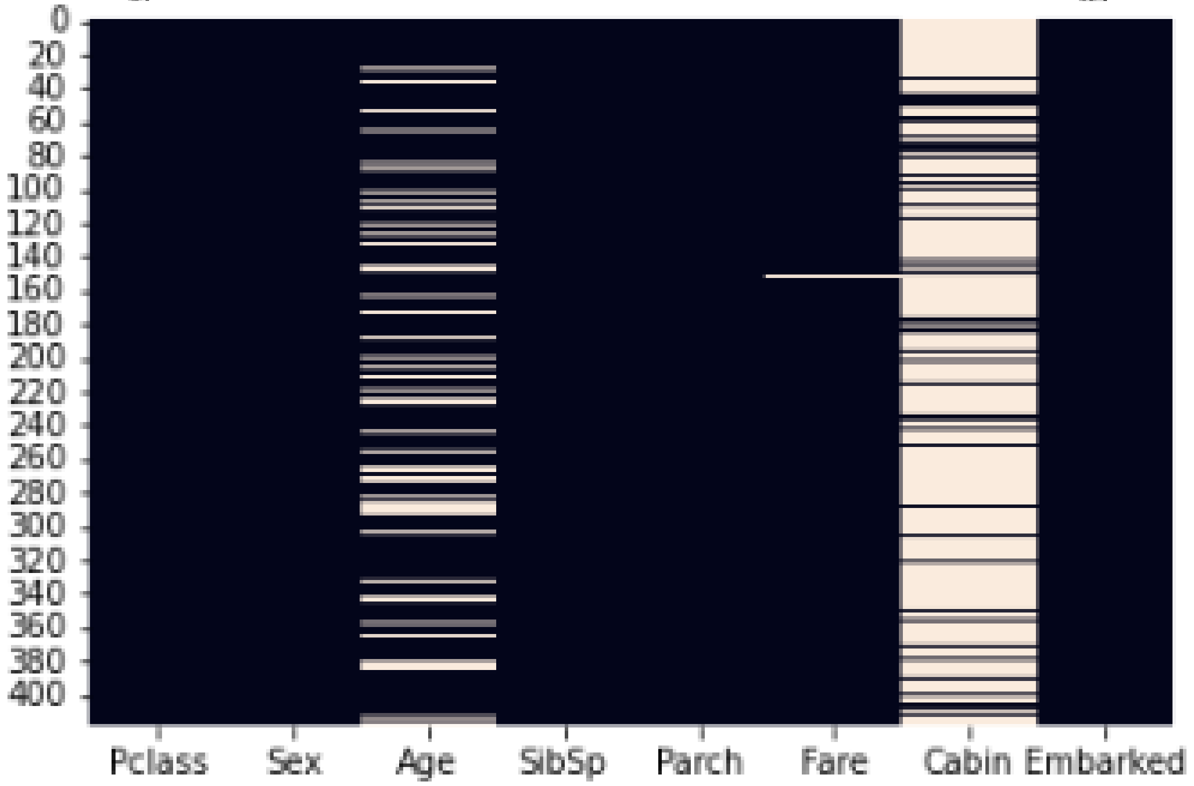
— Test set

- Carbin, Age 데이터에서 결측치 다수 관측
- Fare 데이터에서 소량 관측

02

Check Missing Value





— Train set

```
X = X.dropna(subset = ['Age','Embarked' ])
X.info()
```

— Test set

- Submit파일의 데이터수와 같음
- 삭제가 아닌 채우기

```python
def miss_zero():
    age_tmp = test['Age'].fillna(0)

    Fare_tmp = test['Fare'].fillna(0)

    return age_tmp, Fare_tmp
```

결측치를 모두 0으로 채워줌

```python
def miss_mean():
    age_tmp = test['Age'].fillna(test['Age'].mean())

    Fare_tmp = test['Fare'].fillna(test['Fare'].mean())

    return age_tmp, Fare_tmp
```

결측치를 각 column의 평균으로 채워줌

```python
def miss_linear():
    age_tmp = test['Age'].interpolate(method = 'linear', limit_direction = 'forward')

    Fare_tmp = test['Fare'].interpolate(method = 'linear', limit_direction = 'forward')

    return age_tmp, Fare_tmp
```

결측치를 보간을 통해 유추

CHAPTER

03

EDA & Preprocessing

EDA & Data preprocessing

LabelEncoding

```python
from sklearn.preprocessing import LabelEncoder

cols = ['Sex', 'Embarked']

for col in cols:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
    test[col] = le.transform(test[col])
```

EDA & Data preprocessing

Round

```
print(np.array(X['Age']))
```

```
[22.   38.   26.   35.   35.   54.    2.   27.   14.    4.   58.   20.
 39.   14.   55.    2.   31.   35.   34.   15.   28.    8.   38.   19.
 40.   66.   28.   42.   21.   18.   14.   40.   27.    3.   19.   18.
  7.   21.   49.   29.   65.   21.   28.5   5.   11.   22.   45.    4.
 29.   19.   17.   26.   32.   16.   21.   26.   32.   25.    0.83 30.
 22.   29.   28.   17.   33.   16.   23.   24.   29.   20.   46.   26.
 59.   71.   23.   34.   34.   28.   21.   33.   37.   28.   21.   38.
 47.   14.5  22.   20.   17.   21.   70.5  29.   24.    2.   21.   32.5
 32.5  54.   12.   24.   45.   33.   20.   47.   29.   25.   23.   19.
 37.   16.   24.   22.   24.   19.   18.   19.   27.    9.   36.5  42.
 51.   22.   55.5  40.5  51.   16.   30.   44.   40.   26.   17.    1.
  9.   45.   28.   61.    4.    1.   21.   56.   18.   50.   30.   36.
  9.    1.    4.   45.   40.   36.   32.   19.   19.    3.   44.   58.
 42.   24.   28.   34.   45.5  18.    2.   32.   26.   16.   40.   24.
 35.   22.   30.   31.   27.   42.   32.   30.   16.   27.   51.   38.
 22.   19.   20.5  18.   35.   29.   59.    5.   24.   44.    8.   19.
 33.   29.   22.   30.   44.   25.   24.   37.   54.   29.   62.   30.
 41.   29.   30.   35.   50.    3.   52.   40.   36.   16.   25.   58.
 35.   25.   41.   37.   63.   45.    7.   35.   65.   28.   16.   19.
 33.   30.   22.   42.   22.   26.   19.   36.   24.   24.   23.5   2.
 50.   19.    0.92 17.   30.   30.   24.   18.   26.   28.   43.   26.
```

```
X['Age'] = X['Age'].round(0).astype('int64')
test['Age'] = test['Age'].round(0).astype('int64')
```

EDA & Data preprocessing

Scaling

```python
cols= ['Age', 'SibSp', 'Parch', 'Fare']

for col in cols:
    X[col] = np.log1p(X[col])
    test[col] = np.log1p(test[col])
```

np.log1p

▶ sklearn의 scaler 대신 사용

▶ log1p와 sklearn scaler 둘 다 사용해도 됨

▶ 각 데이터의 범위를 일괄적으로 맞추기 위함

▶ np.log 대신 np.log1p 를 사용하는 이유
: 0에 가까운 작은 양수의 경우 -∞ 가 되는것을 방지

CHAPTER 04

Model Training

Model Training

Data Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state=1, stratify =
Y)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(498, 7)
(498,)
(214, 7)
(214,)
```

Model Training

GridSearchCV

sklearn.model_selection.GridSearchCV

class sklearn.model_selection.**GridSearchCV**(*estimator, param_grid, *, scoring=None, n_jobs=None, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False*)

[source]

Model Training

LogisticRegression

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

para_grid = {'C' : [0.001, 0.01, 0.1, 1, 10 ,50],
             'solver' : ['sag', 'saga']}

Logit1 = GridSearchCV(LogisticRegression(penalty='l2' ,random_state=1), para_grid, cv = 3)

Logit1.fit(X_train, y_train)

y_test_logistic = Logit1.predict(X_test)
```

Model Training

KNN

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 2, p=1)

para_grid = {'n_neighbors' : [3,4,5,6,7,8]}

knn = GridSearchCV(KNeighborsClassifier(p=1), para_grid, cv = 3)

knn.fit(X_train,y_train)

y_test_knn = knn.predict(X_test)
```

Model Training

LDA 판별분석

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

cld =LinearDiscriminantAnalysis(store_covariance=True)

cld.fit(X_train, y_train)

y_test_lda = cld.predict(X_test)
```

Model Training

SVM

```
from sklearn.svm import LinearSVC

para_grid = {'loss' : ['hinge', 'squared_hinge'],
              'multi_class' : ['ovr', 'crammer_singer'],
             'C' : [0.001, 0.01, 0.1, 1, 10]}

svm = GridSearchCV(LinearSVC(class_weight='balanced'), para_grid, cv = 3)

svm.fit(X_train,y_train)
y_test_svm = svm.predict(X_test)
```

CHAPTER 05

Accuracy & Debate

Accuracy

```
from sklearn.metrics import accuracy_score

print("Logistic  :" , accuracy_score(y_test,y_test_logistic))
print("KNN  :" , accuracy_score(y_test,y_test_knn))
print("LDA  :" , accuracy_score(y_test,y_test_lda))
print("SVM  :" , accuracy_score(y_test,y_test_svm))
```

```
Logistic  : 0.8130841121495327
KNN  : 0.8130841121495327
LDA  : 0.8037383177570093
SVM  : 0.7663551401869159
```

THANK

YOU EVERYONE

PLEASE ENTER YOUR TEXT