

# 데이터 탐색 전처리 후, 모델 구축

## 학습 목표

- 데이터를 다운로드 후, 데이터 로드부터 시각화, 전처리, 모델 구축의 전과정을 수행해 본다.

## 목차

- 데이터 불러오기
- 데이터 탐색 및 시각화
- 특징 선택 및 데이터 나누기
- 모델 구축 및 모델 학습 후, 평가
- 최종 모델 구축 후, 제출

## 01. 데이터 불러오기

목차로 이동하기

```
In [12]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objs as go
```

```
In [13]: train = pd.read_csv("./data/train.csv")
test = pd.read_csv("./data/test.csv")
sub = pd.read_csv("./data/gender_submission.csv")
```

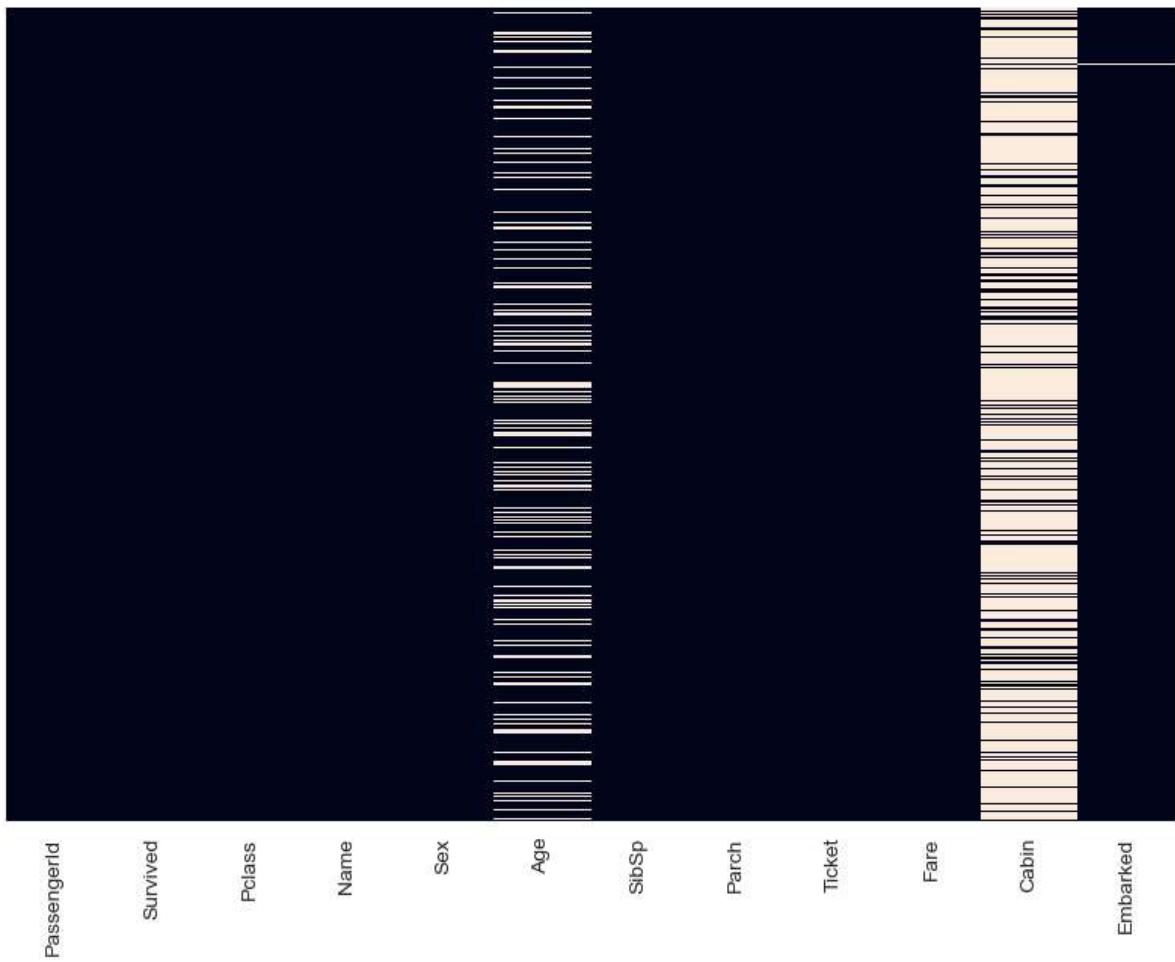
## 02. 데이터 탐색 및 시각화

목차로 이동하기

### Heatmap을 활용한 결측치 확인

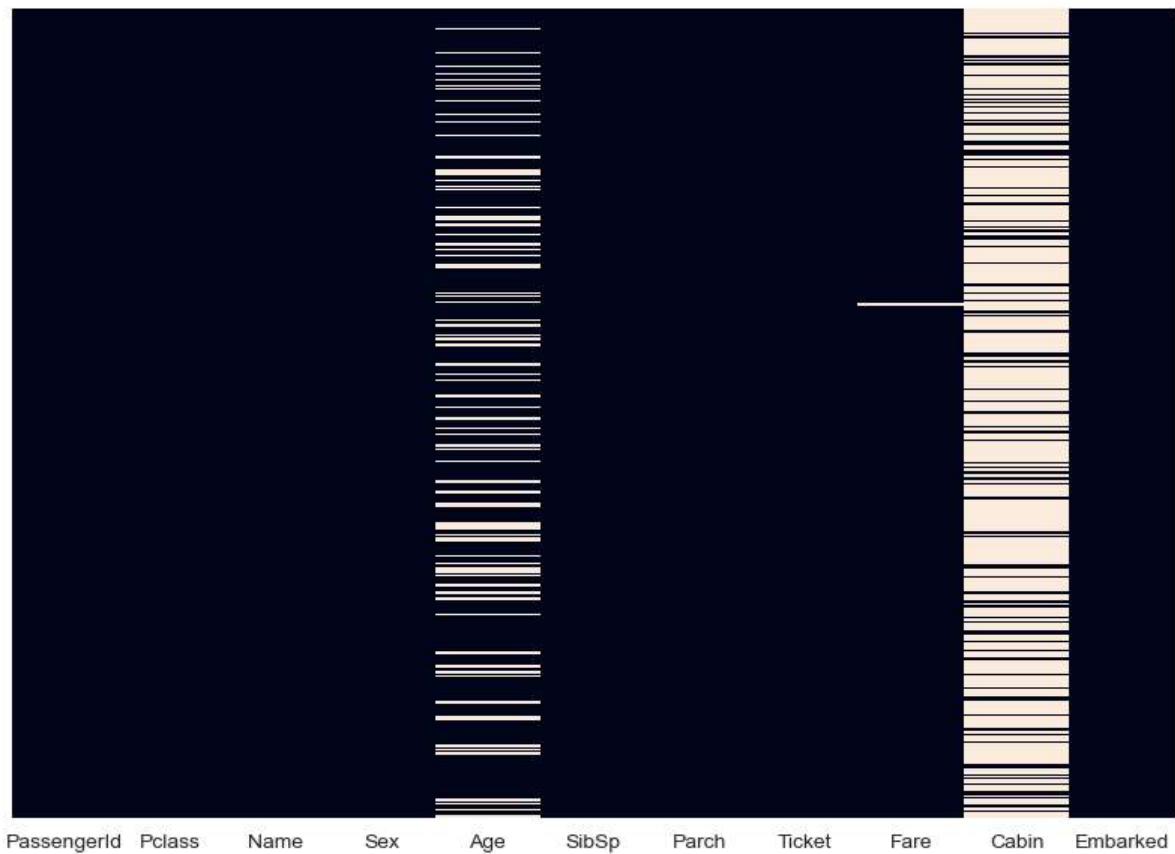
```
In [14]: plt.figure(figsize=(10,7))
sns.heatmap(train.isnull(), yticklabels=False, cbar=False) # cbar : colorbar를 그려보자
```

Out[14]: <Axes: >



```
In [15]: plt.figure(figsize=(10,7))
sns.heatmap(test.isnull(), yticklabels=False, cbar=False) # cbar : colorbar를 없애기
```

```
Out[15]: <Axes: >
```

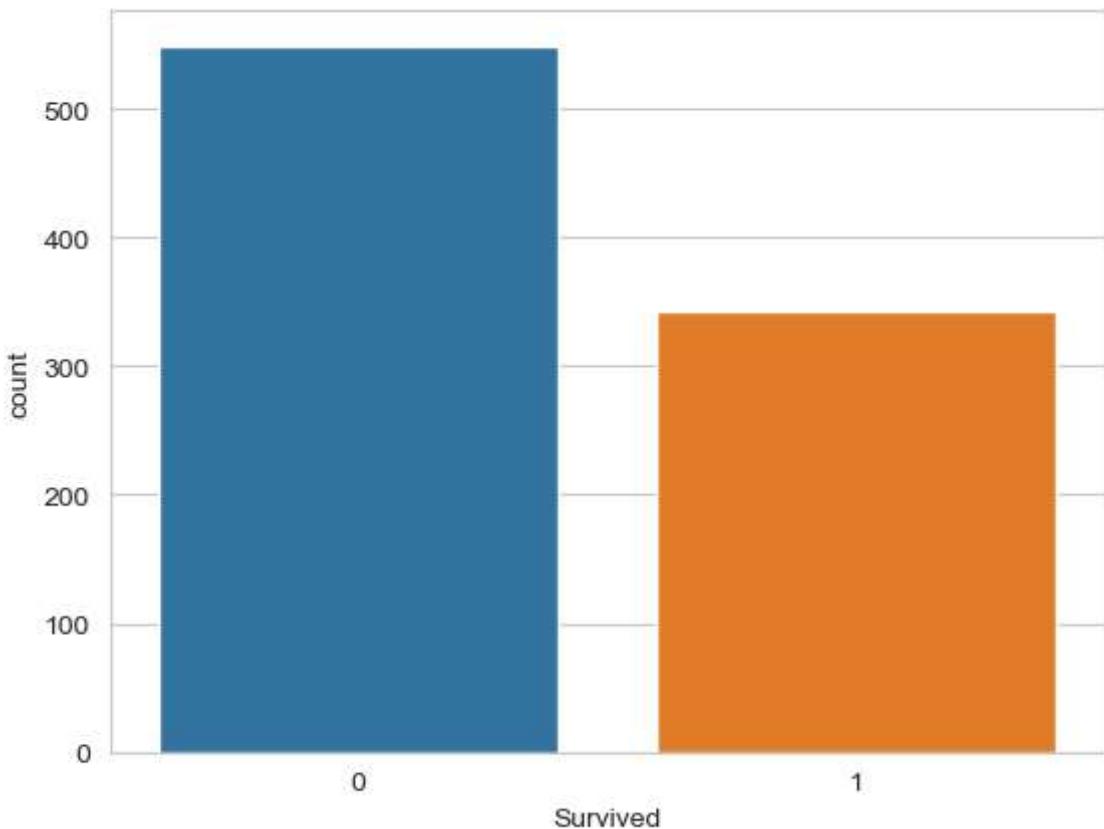


## 예측하고자 하는 특징(Survived)에 대해 살펴보기

생존자와 사망자의 수는 어느정도일까?

```
In [16]: sns.set_style('whitegrid')
sns.countplot(x='Survived', data=train)
```

```
Out[16]: <Axes: xlabel='Survived', ylabel='count'>
```

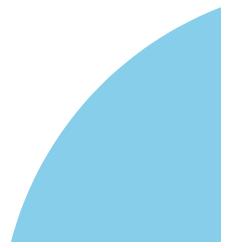


```
In [17]: # 생존자와 사망자 수 계산
survival_counts = train['Survived'].value_counts()

# Plotly 파이 차트 시각화
labels = ['Died', 'Survived']
values = [survival_counts[0], survival_counts[1]]

fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=0.3,
                             marker=dict(colors=['salmon', 'skyblue']))])

fig.update_layout(title_text='Survival Rates on Titanic', title_x=0.5)
fig.show()
```

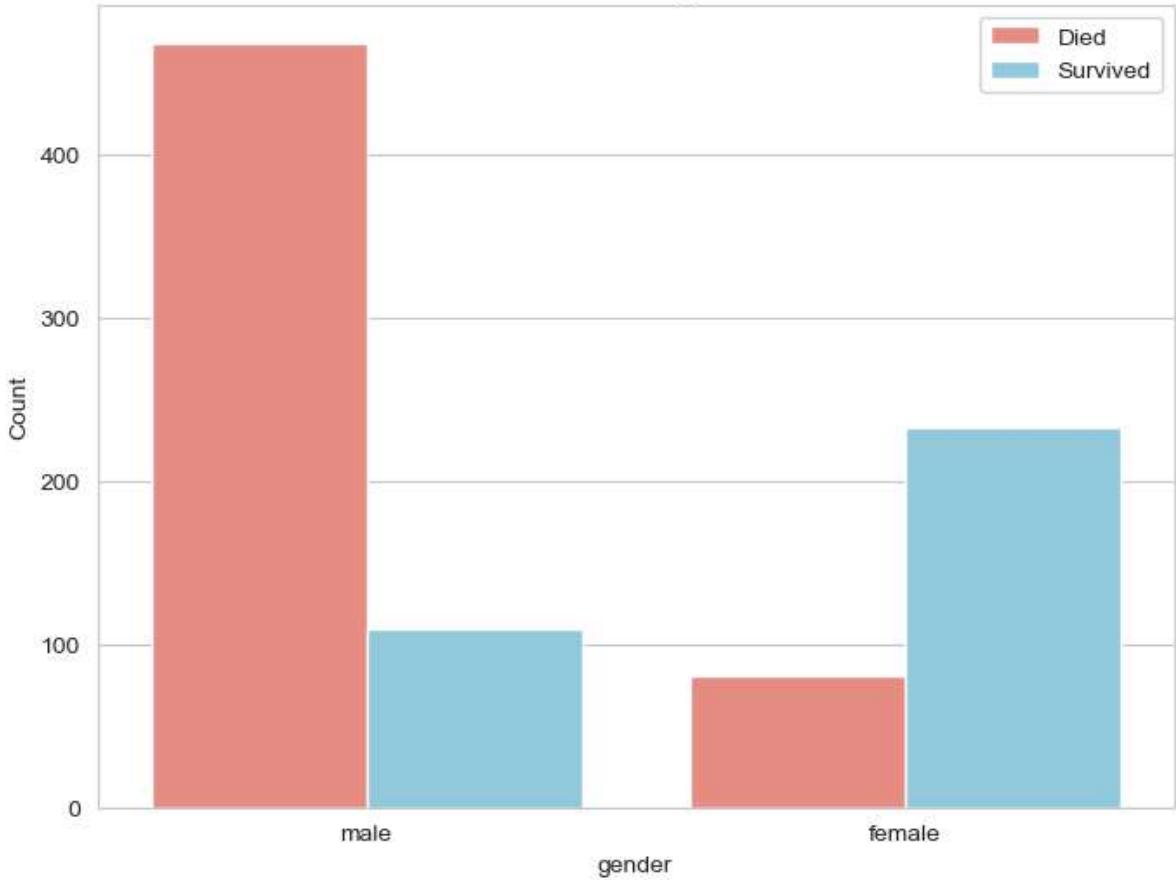


## 성별에 따른 생존자의 비율은 어떻게 될까?

In [21]:

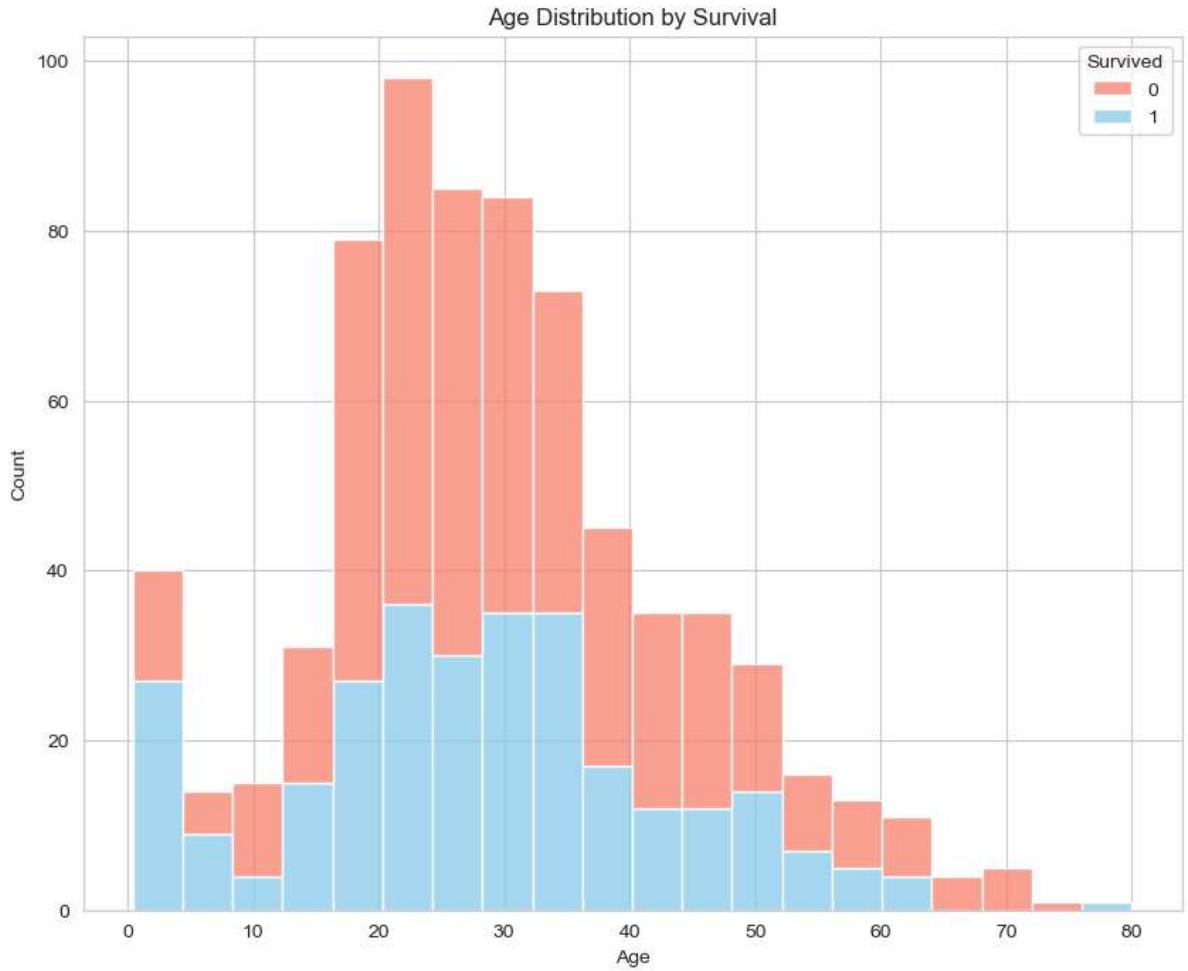
```
# 성별에 따른 생존율
plt.figure(figsize=(8, 6))
sns.countplot(data=train, x='Sex', hue='Survived', palette=['salmon', 'skyblue'])
plt.title('Survival Count by Gender')
plt.xlabel('gender')
plt.ylabel('Count')
plt.legend(['Died', 'Survived'])
plt.show()
```

Survival Count by Gender



나이 분포에 따른 생존율은 어떻게 될까?

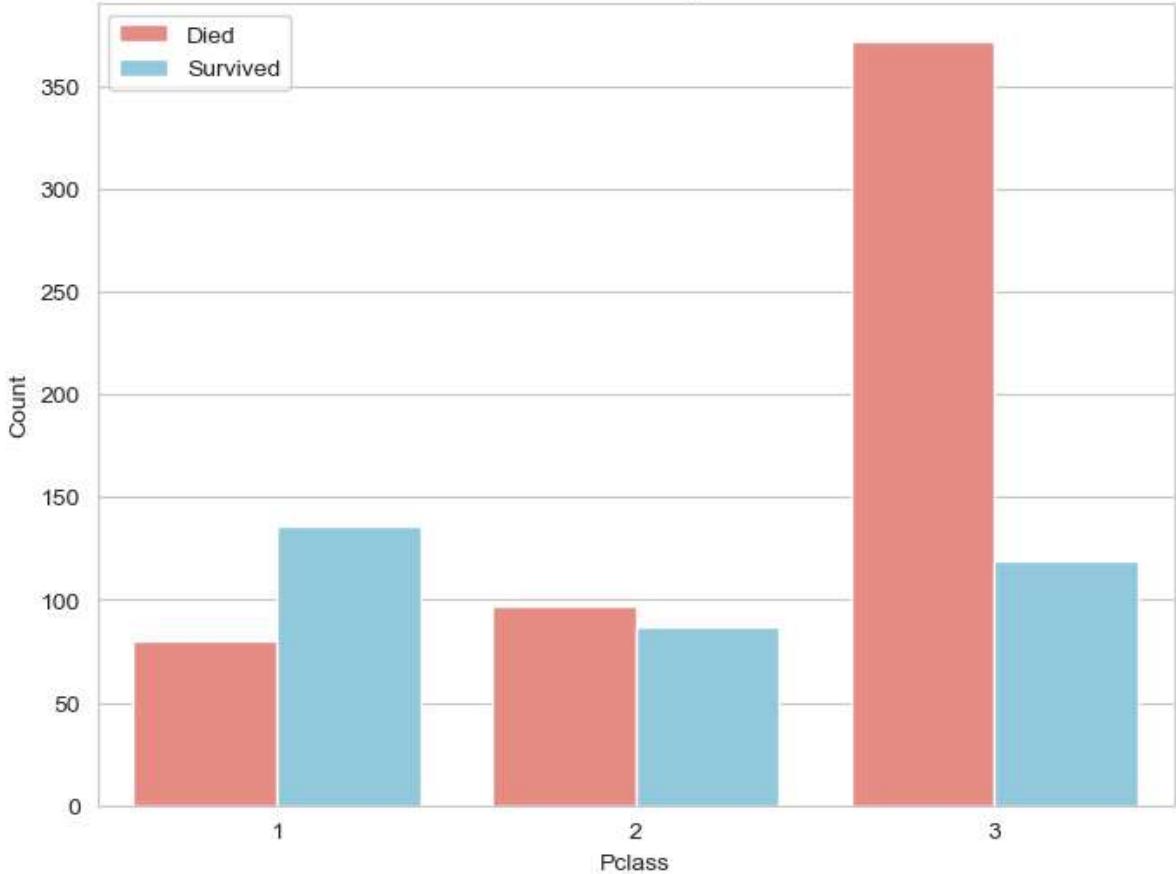
```
In [27]: # 나이 분포와 생존율
plt.figure(figsize=(10, 8))
sns.histplot(data=train, x='Age', hue='Survived', multiple='stack', palette=['salmon', 'steelblue'])
plt.title('Age Distribution by Survival')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```



객실 등급별에 따른 생존율은 어떠할까?

```
In [29]: # 객실 등급별 생존율
plt.figure(figsize=(8, 6))
sns.countplot(data=train, x='Pclass', hue='Survived', palette=['salmon', 'skyblue'])
plt.title('Survival Count by Pclass')
plt.xlabel('Pclass')
plt.ylabel('Count')
plt.legend(['Died', 'Survived'])
plt.show()
```

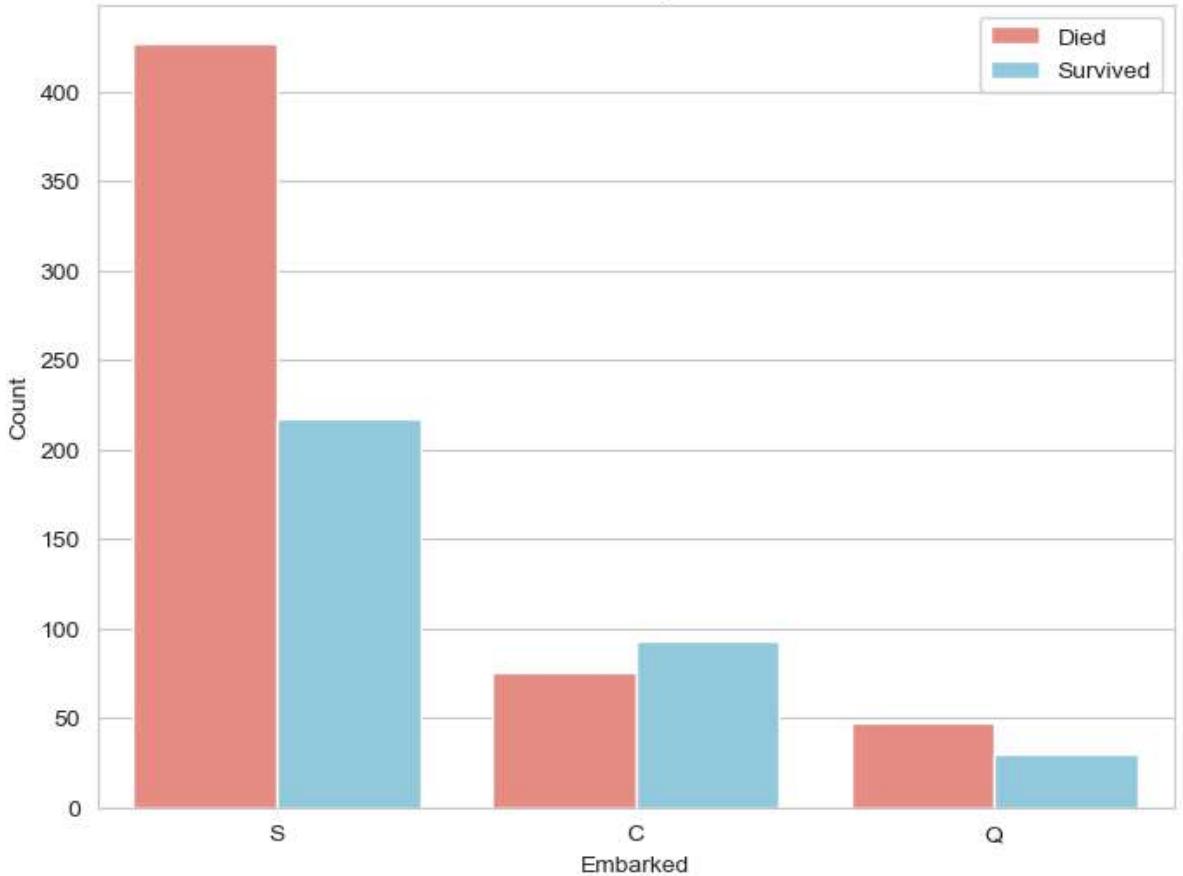
Survival Count by Pclass



승선 항구별 생존율은 어떠할까?

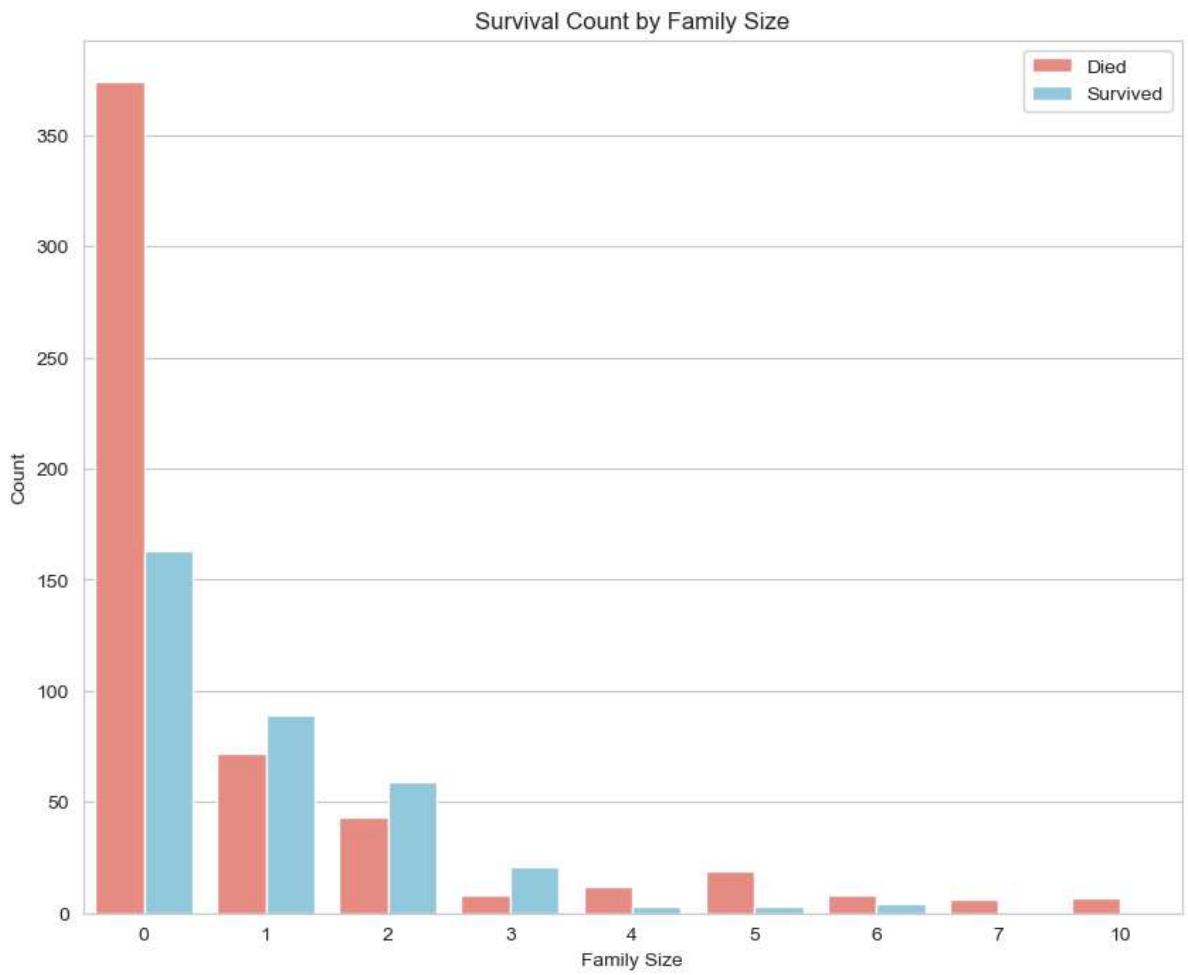
```
In [30]: # 승선 항구별 생존율
plt.figure(figsize=(8, 6))
sns.countplot(data=train, x='Embarked', hue='Survived', palette=['salmon', 'skyblue'])
plt.title('Survival Count by Embarked')
plt.xlabel('Embarked')
plt.ylabel('Count')
plt.legend(['Died', 'Survived'])
plt.show()
```

Survival Count by Embarked



가족 구성에 따른 생존율은 어떠할까?

```
In [35]: # 가족 구성 여부에 따른 생존율  
train['FamilySize'] = train['SibSp'] + train['Parch']  
plt.figure(figsize=(10, 8))  
sns.countplot(data=train, x='FamilySize', hue='Survived', palette=['salmon', 'skyblue'])  
plt.title('Survival Count by Family Size')  
plt.xlabel('Family Size')  
plt.ylabel('Count')  
plt.legend(['Died', 'Survived'])  
plt.show()
```

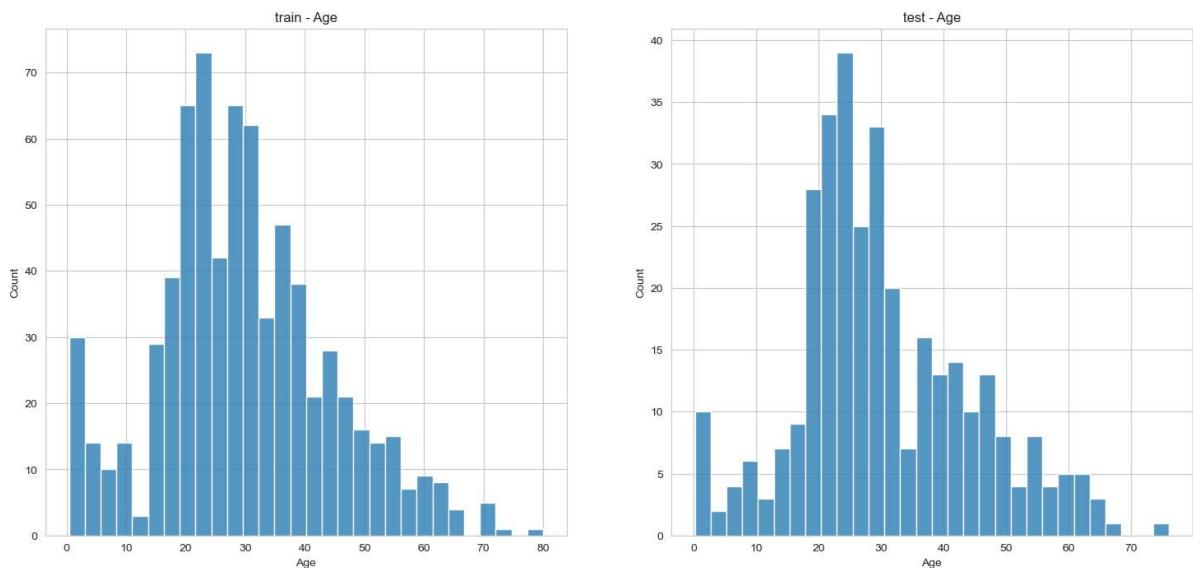


학습용(train), 테스트용(test)의 나이대 분포를 살펴보자.

```
In [37]: f,ax=plt.subplots(1,2,figsize=(18,8))

# 첫번째 그래프
sns.histplot(train['Age'].dropna(), bins=30,ax=ax[0])
ax[0].set_title('train - Age')

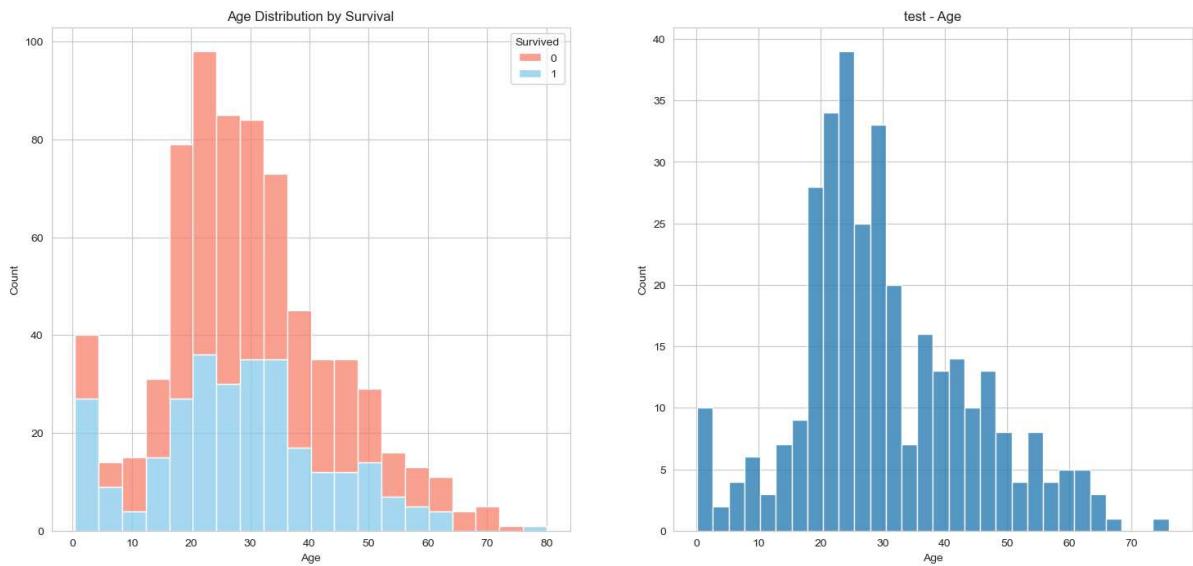
# 두번째 그래프
sns.histplot(test['Age'].dropna(), bins=30,ax=ax[1])
ax[1].set_title('test - Age')
plt.show()
```



```
In [40]: f,ax=plt.subplots(1,2,figsize=(18,8))

# 첫번째 그래프
plt.figure(figsize=(10, 8))
sns.histplot(data=train, x='Age', hue='Survived', multiple='stack', palette=['salmon','steelblue'])
ax[0].set_title('Age Distribution by Survival')
ax[0].set_xlabel('Age')
ax[0].set_ylabel('Count')

# 두번째 그래프
sns.histplot(test['Age'].dropna(), bins=30,ax=ax[1])
ax[1].set_title('test - Age')
plt.show()
```



<Figure size 1000x800 with 0 Axes>

데이터 전처리 후, 모델에 해당 특징을 추가해 보자.

## 03. 데이터 전처리

목차로 이동하기

### Age(나이)에 대한 데이터 전처리

```
In [41]: train['Age'] = train['Age'].fillna(train['Age'].mean())
test['Age'] = test['Age'].fillna(test['Age'].mean())
```

```
In [42]: print(train.isnull().sum())
print(test.isnull().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket          0
Fare             0
Cabin           687
Embarked        2
FamilySize       0
dtype: int64
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket          0
Fare             1
Cabin           327
Embarked        0
dtype: int64
```

## Embarked(승선항)에 대한 결측치 전처리

```
In [43]: val_Embarked = train['Embarked'].value_counts()
val_Embarked
```

```
Out[43]: Embarked
S    644
C    168
Q     77
Name: count, dtype: int64
```

- 승선항이 많은 값으로 결측치 처리를 수행한다.

```
In [44]: train['Embarked'] = train['Embarked'].fillna('S')
```

## Fare(요금)에 대한 결측치 처리

- 특징을 선택해서 사용해야 한다면, test 데이터 셋은 반드시 결측치 처리 후, 사용해야 한다.

```
In [45]: test['Fare'] = test['Fare'].fillna(test['Fare'].mean())
```

```
In [46]: print(train.isnull().sum())
print(test.isnull().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket          0
Fare             0
Cabin           687
Embarked        0
FamilySize       0
dtype: int64
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket          0
Fare             0
Cabin           327
Embarked        0
dtype: int64
```

## 성별에 대한 데이터 전처리 후, 모델에 사용

- 레이블 인코딩(Label Encoding) : 범주형 데이터를 수치형 데이터로 변환하는 과정입니다. 범주형 데이터는 고유한 값(예: 문자열 또는 범주)으로 이루어진 데이터이며, 이를 모델에 직접 입력할 수 없기 때문에 수치형 데이터로 변환해야 합니다.

```
In [47]: print( train['Sex'].value_counts() )
print( train['Embarked'].value_counts() )
```

```
Sex
male     577
female   314
Name: count, dtype: int64
Embarked
S       646
C       168
Q       77
Name: count, dtype: int64
```

## pandas의 map함수를 이용하여 레이블 인코딩

```
In [48]: train['Sex'] = train['Sex'].map( {'female': 0, 'male': 1} ).astype(int)
test['Sex'] = test['Sex'].map( {'female': 0, 'male': 1} ).astype(int)

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

```
In [49]: ## 나이에 대한 int 처리
train['Age'] = train['Age'].astype('int')
test['Age'] = test['Age'].astype('int')
```

```
In [51]: print(train.columns)
print(train.info())
print()
print(test.info())

Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'FamilySize'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 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    int32  
 5   Age          891 non-null    int32  
 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     891 non-null    int32  
 12  FamilySize   891 non-null    int64  
dtypes: float64(1), int32(3), int64(6), object(3)
memory usage: 80.2+ KB
None
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId 418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object  
 3   Sex          418 non-null    int32  
 4   Age          418 non-null    int32  
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object  
 8   Fare          418 non-null    float64 
 9   Cabin        91 non-null    object  
 10  Embarked     418 non-null    int32  
dtypes: float64(1), int32(3), int64(4), object(3)
memory usage: 31.2+ KB
None
```

## 03. 특징 선택 및 데이터 나누기

[목차로 이동하기](#)

### 특징 간의 관계 확인을 위해 상관계수 확인

```
In [69]: # 사용할 특징과 예측할 타겟. 상관계수 확인
sel_XY = ['PassengerId', 'Pclass', 'Sex', 'Age', 'SibSp', 'SibSp','Parch', 'Embarked'
train_all = train[sel_XY]
print( train_all.columns )
```

```
corr_XY = train_all.corr()  
corr_XY
```

```
Index(['PassengerId', 'Pclass', 'Sex', 'Age', 'SibSp', 'SibSp', 'Parch',  
       'Embarked', 'Survived'],  
      dtype='object')
```

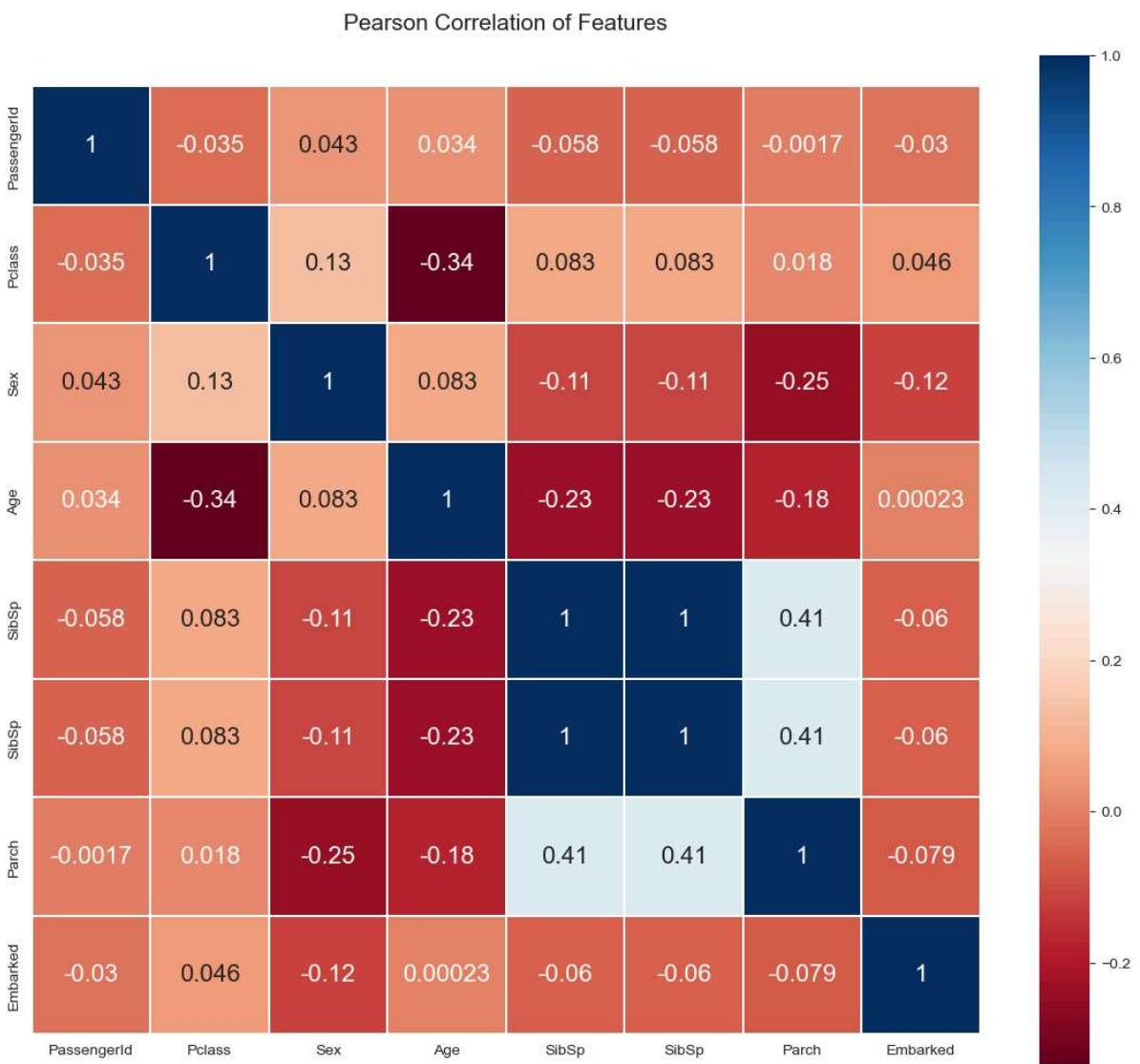
Out[69]:

|                    | <b>PassengerId</b> | <b>Pclass</b> | <b>Sex</b> | <b>Age</b> | <b>SibSp</b> | <b>SibSp</b> | <b>Parch</b> | <b>Embark</b> |
|--------------------|--------------------|---------------|------------|------------|--------------|--------------|--------------|---------------|
| <b>PassengerId</b> | 1.000000           | -0.035144     | 0.042939   | 0.033741   | -0.057527    | -0.057527    | -0.001652    | -0.0304       |
| <b>Pclass</b>      | -0.035144          | 1.000000      | 0.131900   | -0.335071  | 0.083081     | 0.083081     | 0.018443     | 0.0457        |
| <b>Sex</b>         | 0.042939           | 0.131900      | 1.000000   | 0.082533   | -0.114631    | -0.114631    | -0.245489    | -0.1165       |
| <b>Age</b>         | 0.033741           | -0.335071     | 0.082533   | 1.000000   | -0.232743    | -0.232743    | -0.176744    | 0.0002        |
| <b>SibSp</b>       | -0.057527          | 0.083081      | -0.114631  | -0.232743  | 1.000000     | 1.000000     | 0.414838     | -0.0599       |
| <b>SibSp</b>       | -0.057527          | 0.083081      | -0.114631  | -0.232743  | 1.000000     | 1.000000     | 0.414838     | -0.0599       |
| <b>Parch</b>       | -0.001652          | 0.018443      | -0.245489  | -0.176744  | 0.414838     | 0.414838     | 1.000000     | -0.0786       |
| <b>Embarked</b>    | -0.030467          | 0.045702      | -0.116569  | 0.000234   | -0.059961    | -0.059961    | -0.078665    | 1.0000        |
| <b>Survived</b>    | -0.005007          | -0.338481     | -0.543351  | -0.067809  | -0.035322    | -0.035322    | 0.081629     | 0.1068        |

In [62]:

```
colormap = plt.cm.RdBu  
plt.figure(figsize=(14, 12))  
plt.title('Pearson Correlation of Features', y=1.05, size=15)  
sns.heatmap(corr_XY, linewidths=0.1, vmax=1.0,  
            square=True, cmap=colormap, linecolor='white', annot=True, annot_kws={"s
```

Out[62]: <Axes: title={'center': 'Pearson Correlation of Features'}>



```
In [70]: from sklearn.model_selection import train_test_split
```

```
In [71]: # 'Name', 'Ticket' => 문자포함
sel = ['PassengerId', 'Pclass', 'Sex', 'Age', 'SibSp', 'SibSp', 'SibSp', 'Parch', 'Embarked']

# 학습에 사용될 데이터 준비 X_train, y_train
X_train_all = train[sel]
y_train_all = train['Survived']
```

```
In [72]: # 데이터를 학습 세트와 테스트 세트로 분할 train(70%), test(30%)
X_train, X_test, y_train, y_test = train_test_split(X_train_all, y_train_all, test_size=0.3)

# 분할된 데이터 출력
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

X_train shape: (623, 8)
X_test shape: (268, 8)
y_train shape: (623,)
y_test shape: (268,)
```

```
In [73]: # 최종 예측을 위한 test 데이터
X_test_last = test[sel]
```

## 04. 모델 구축 및 학습, 이후 모델 평가

목차로 이동하기

```
In [75]: from sklearn.neighbors import KNeighborsClassifier
```

```
# 모델 구축 및 학습
model = KNeighborsClassifier()
model.fit(X_train, y_train)
pred = model.predict(X_test)
pred
```

```
Out[75]: array([0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   .... 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
   .... 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
   .... 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
   .... 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
   .... 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
   .... 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
   .... 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
   .... 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
   .... 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
   .... 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
   .... 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1,
   .... 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
   .... 0, 1, 0, 1], dtype=int64)
```

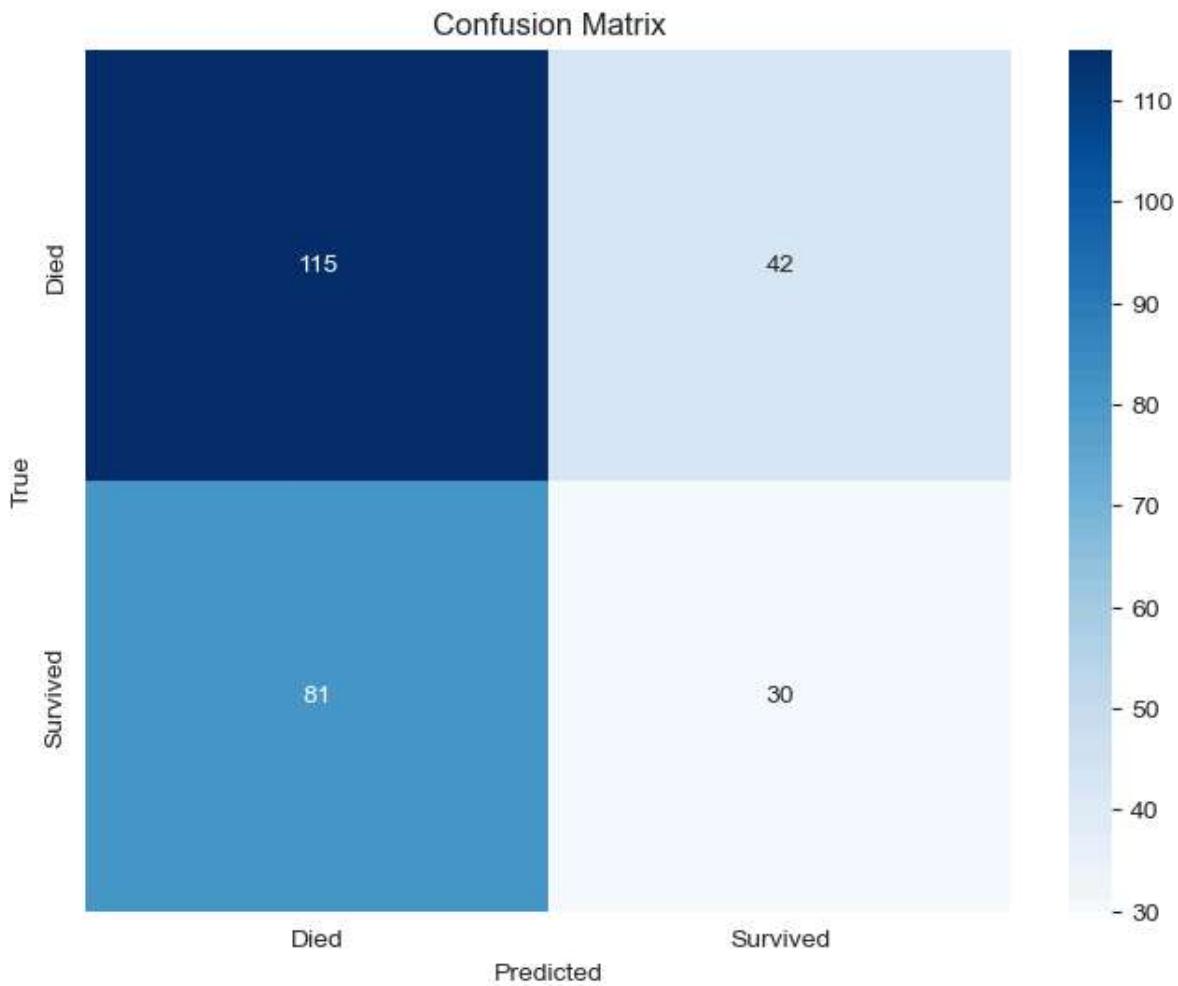
```
In [78]: from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
In [79]: # 정확도 계산
accuracy = accuracy_score(y_test, pred)
print(f'Accuracy: {accuracy:.2f}')
```

Accuracy: 0.54

```
In [81]: # 혼동 행렬 계산 및 시각화
conf_matrix = confusion_matrix(y_test, pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True,
            fmt='d', cmap='Blues', xticklabels=['Died', 'Survived'], yticklabels=['Died', 'Survived'])

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



좀 더 좋은 모델을 사용해 보자.

```
In [85]: from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
In [84]: # 모델 리스트
models = {
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "Logistic Regression": LogisticRegression()
}
```

```
In [86]: # 결과 저장을 위한 리스트
results = []

# 각 모델에 대해 학습, 예측 및 평가 수행
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    results.append([name, accuracy, precision, recall, f1])

# 결과 데이터프레임 생성
```

```

results_df = pd.DataFrame(results, columns=['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
print(results_df)

..... Model Accuracy Precision Recall F1 Score
0 ..... KNN 0.541045 0.416667 0.270270 0.327869
1 ..... Random Forest 0.817164 0.844444 0.684685 0.756219
2 ..... Logistic Regression 0.794776 0.797872 0.675676 0.731707

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:460: 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

```

## 참고를 위한 설명 - 분류 모델에 대한 평가 지표

- accuracy(정확도) : 정확도는 모델이 올바르게 예측한 비율을 나타냅니다. 전체 예측 중에서 얼마나 많은 예측이 정확했는지.
  - 계산 방법: (정확하게 예측한 샘플 수) / (전체 샘플 수)
  - 100개의 샘플 중 90개를 정확히 예측했다면 정확도는 90%입니다.
- precision(정밀도) : 정밀도는 모델이 양성(positive)이라고 예측한 것들 중에서 실제로 양성인 것의 비율을 나타냅니다. 즉, 양성 예측의 정확성.
  - 계산 방법: (정확히 양성으로 예측한 샘플 수) / (양성으로 예측한 전체 샘플 수)
  - 예를 들어, 모델이 10개를 양성으로 예측했는데 그 중 7개가 실제로 양성이라면, 정밀도는 70%입니다.
- recall(재현율) : 재현율은 실제 양성인 것들 중에서 모델이 얼마나 많은 양성을 올바르게 예측했는지를 나타냅니다. 즉, 실제 양성을 얼마나 잘 찾아냈는지 보여준다.
  - 계산 방법: (정확히 양성으로 예측한 샘플 수) / (전체 실제 양성 샘플 수)
  - 예를 들어, 실제로 20개의 양성이 있는데 모델이 그 중 15개를 양성으로 예측했다면, 재현율은 75%입니다.
- f1 score : F1점수
  - 계산 방법:  $2 \cdot (\text{정밀도} \cdot \text{재현율}) / (\text{정밀도} + \text{재현율})$
  - F1 점수는 정밀도와 재현율을 모두 고려하기 때문에, 특히 불균형 데이터셋(예: 긍정 샘플이 매우 적은 경우)에서 유용함.

## 05. 최종 모델 구축 후, 제출

목차로 이동하기

```
In [93]: from sklearn.ensemble import RandomForestClassifier
# 모델 구축
model = LogisticRegression()
```

```

# 학습
model.fit(X_train_all, y_train_all)

# 예측
y_pred = model.predict(X_test_last)

sub['Survived'] = y_pred
sub.head()

```

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear\_model\logistic.py:460: 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](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

Out[93]:

|   | PassengerId | Survived |
|---|-------------|----------|
| 0 | 892         | 0        |
| 1 | 893         | 1        |
| 2 | 894         | 0        |
| 3 | 895         | 0        |
| 4 | 896         | 1        |

In [94]:

## 제출 결과 확인

- Baseline 모델
  - knn 0.6220
  - Random Forest - 0.75119