모델 개선하기

학습 목표

- 모델을 개선하는 여러가지 것들에 대해서 알아본다.
  - 교차 검증
  - 하이퍼 파라미터 튜닝
  - 교차 검증
  - 피처 엔지니어링

목차

In [47]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objs as go
```

01. 피처 엔지니어링을 통한 모델 개선

목차로 이동하기

In [48]:
```
train = pd.read_csv("./data/train.csv")
test = pd.read_csv("./data/test.csv")
sub = pd.read_csv("./data/gender_submission.csv")
```

데이터 결합

In [49]:
```
# 데이터 결합
all_data = pd.concat([train, test], sort=False).reset_index(drop=True)
```

결측값 처리

In [50]:
```
# Age 결측값 처리 - 평균으로 채움
all_data['Age'].fillna(all_data['Age'].mean(), inplace=True)

# Embarked 결측값 처리 - 최빈값으로 채움
all_data['Embarked'].fillna(all_data['Embarked'].mode()[0], inplace=True)

# Fare 결측값 처리 - 중간값으로 채움
all_data['Fare'].fillna(all_data['Fare'].median(), inplace=True)
```

범주형 변수 인코딩

In [51]:
```
# Sex 인코딩
all_data['Sex'] = all_data['Sex'].map({'male': 0, 'female': 1})

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

새로운 피처 생성

In [52]:
```
# 가족 크기 피처 생성
all_data['FamilySize'] = all_data['SibSp'] + all_data['Parch'] + 1

# 홀로 승선 여부 피처 생성
all_data['IsAlone'] = 1  # 기본 값은 1 (혼자)
all_data['IsAlone'].loc[all_data['FamilySize'] > 1] = 0  # 가족이 있으면 0
```

```
C:₩Users₩colab₩AppData₩Local₩Temp₩ipykernel_28424₩1385118688.py:6: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  all_data['IsAlone'].loc[all_data['FamilySize'] > 1] = 0  # 가족이 있으면 0
```

필요 없는 피처 제거

In [53]:
```
# 필요 없는 피처 제거
all_data.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
```

데이터 다시 나누기

- 전처리된 데이터를 다시 훈련 데이터와 테스트 데이터로 나눕니다.

In [54]:
```
from sklearn.model_selection import train_test_split
```

In [55]:
```
# 데이터 다시 나누기
train_processed = all_data[:len(train)]
test_processed = all_data[len(train):]

# 훈련 데이터와 타겟 변수 분리
X_train_all = train_processed.drop('Survived', axis=1)
y_train_all = train_processed['Survived']
X_test_last = test_processed.drop('Survived', axis=1)
```

In [56]:
```
# 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, random_state=42)
```

02. 모델 구축 및 학습, 평가

목차로 이동하기

In [57]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [58]:
```
# 모델 리스트
models = {
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "DecisionTree": DecisionTreeClassifier(),
    "Gradient Boosting": GradientBoostingClassifier()
}
```

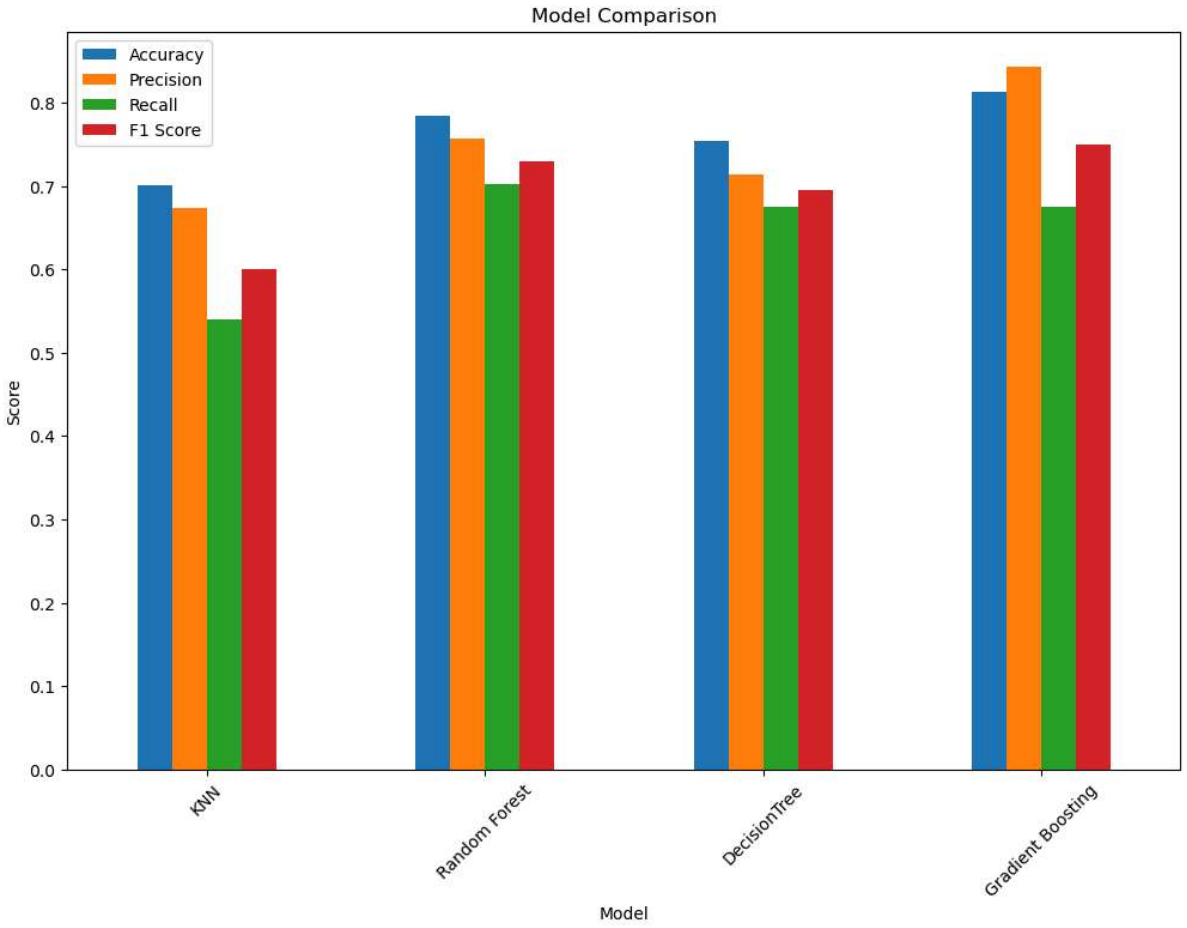
In [59]:
```
# 결과 저장을 위한 리스트
results = []

# 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', 'Recal
print(results_df)
```

	Model	Accuracy	Precision	Recall	F1 Score
0	KNN	0.701493	0.674157	0.540541	0.600000
1	Random Forest	0.783582	0.757282	0.702703	0.728972
2	DecisionTree	0.753731	0.714286	0.675676	0.694444
3	Gradient Boosting	0.813433	0.842697	0.675676	0.750000

In [60]:
```
# 시각화
results_df.set_index('Model').plot(kind='bar', figsize=(12, 8))
plt.title('Model Comparison')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.show()
```



03. 교차 검증을 이용한 모델 개선

목차로 이동하기

In [61]:
```
models
```

Out[61]:
```
{'KNN': KNeighborsClassifier(),
 'Random Forest': RandomForestClassifier(random_state=42),
 'DecisionTree': DecisionTreeClassifier(),
 'Gradient Boosting': GradientBoostingClassifier()}
```

In [62]:
```
from sklearn.model_selection import cross_val_score

# 교차 검증
cross_val_results = {}

for name, model in models.items():
    scores = cross_val_score(model, X_train_all, y_train_all, cv=5, scoring='accurac
    cross_val_results[name] = scores.mean()

# 교차 검증 결과 출력
for name, score in cross_val_results.items():
    print(f"{name}: {score:.2f}")
```

```
KNN: 0.71
Random Forest: 0.81
DecisionTree: 0.77
Gradient Boosting: 0.83
```

04. 모델 구축 및 모델 학습 후, 평가

목차로 이동하기

In [63]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
```

In [64]:
```
# 하이퍼파라미터 그리드 정의
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.05],
    'max_depth': [3, 4, 5],
    'subsample': [0.8, 0.9, 1.0],
    'min_samples_split': [2, 5, 10]
}
```

In [65]:
```
# 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, random_state=42)
```

In [66]:
```
# GridSearchCV 객체 생성
grid_search = GridSearchCV(GradientBoostingClassifier(random_state=42),
                           param_grid,
                           cv=5, scoring='accuracy', n_jobs=-1, verbose=1)

# 학습
grid_search.fit(X_train, y_train)

# 최적 하이퍼파라미터 출력
print("Best parameters found: ", grid_search.best_params_)

# 최적 모델로 예측
best_model = grid_search.best_estimator_
pred = best_model.predict(X_test)
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
Best parameters found:  {'learning_rate': 0.1, 'max_depth': 3, 'min_samples_split':
10, 'n_estimators': 200, 'subsample': 0.8}
```

In [67]:
```
# 최종 모델 평가
from sklearn.metrics import accuracy_score, classification_report

# 예측 및 평가
y_train_pred = best_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f'Training Accuracy: {train_accuracy:.2f}')

# 분류 보고서 출력
train_report = classification_report(y_train, y_train_pred)
print(train_report)
```

```
Training Accuracy: 0.94
              precision    recall  f1-score   support

         0.0       0.92      0.99      0.95       392
         1.0       0.98      0.86      0.91       231

    accuracy                           0.94       623
   macro avg       0.95      0.92      0.93       623
weighted avg       0.94      0.94      0.94       623
```

In [68]:
```
# 테스트 데이터에 대한 예측
test_pred = best_model.predict(X_test_last).astype("int")
```

```
# 제출 파일 생성
submission = pd.DataFrame({
    "PassengerId": test['PassengerId'],
    "Survived": test_pred
})

submission.shape
```

Out[68]:
```
(418, 2)
```

In [69]:
```
# 제출 파일 저장
submission.to_csv('third_submission_gbc.csv', index=False)
```