```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
matplotlib inline ?
```
라이브러리 호출

```
titanic_df = pd.read_csv('./titanic_train.csv') → 데이터 읽어오기

titanic_df['Age'].fillna(titanic_df['Age'].mean(), inplace=True)
titanic_df['Cabin'].fillna('N', inplace = True)
titanic_df['Embarked'].fillna('N', inplace = True)
titanic_df.isnull().sum().sum() → 결측값 개수

titanic_df['Cabin'] = titanic_df['Cabin'].str[:1] → 앞 문자만 추출
print(titanic_df['Cabin'].head(3)) → 데이터 3개만 보기

titanic_df.groupby(['Sex', 'Survived'])['Survived'].count() → 성별에 따른 생존자수 확인
```
결측치

→ 시각화 (Seaborn 라이브러리)

```
sns.barplot(x='Sex', y='Survived', data=titanic_df)
sns.barplot(x='Pclass', y='Survived', hue='Sex', data=titanic_df)
```

```
def get_category(age):
    cat=''
    if age <= -1 : cat='Unknown'
    elif age <= 5 : cat='Baby'
    elif age <= 12 : cat='Child'
    elif age <=18 : cat='Teenager'
    elif age <= 25 : cat='Student'
    elif age <=35 : cat='Young Adult'
    elif age <= 60 : cat='Adult'
    else : cat='Elderly'
    return cat
```
Age 데이터 범주(구간) 정하기

```
# 막대그래프의 크기 figure를 더 크게 설정
plt.figure(figsize=(10,6))
# X축의 값을 순차적으로 표시하기 위한 설정
group_names=['Unknown','Baby','Child','Teenager','Student','Young Adult','Adult','Elderly']
# lamda 식에 위에서 생성한 get_category() 함수를 반환값으로 지정.
# get_category(X)는 입력값으로 'Age' 칼럼 값을 받아서 해당하는 cat 반환
titanic_df['Age_cat']=titanic_df['Age'].apply(lamda x: get_category(x))
sns.barplot(x='Age_cat', y='Survived', hue='Sex', data=titanic_df, order=group_names)
titanic_df.drop('Age_cat', axis=1, inplace=True)   → 데이터프레임 원상복귀
```

문자열 → 숫자열

(LabelEncoder → 인코딩 0 ~ (카테고리-1))

```
from sklearn.preprocessing import LabelEncoder
def encode_features(dataDF):
    features = ['Cabin', 'Sex', 'Embarked']
    for feature in features:
        le = LabelEncoder()
        le = le.fit(dataDF[feature])
        dataDF[feature] = le.transform(dataDF[feature])
    return dataDF
titanic_df = encode_features(titanic_df)
titanic_df.head()   → 5개
```

데이터 전처리 함수 만들기

```
# Null 처리함수
def fillna(df):
    df['Age'].fillna(df['Age'].mean(), inplace=True)
    df['Cabin'].fillna('N', inplace=True)
    df['Embarked'].fillna('N', inplace=True)
    df['Fare'].fillna(0, inplace=True)
    return df

# 머신러닝 알고리즘에 불필요한 피처제거
def drop_features(df):
    df.drop(['PassengerId', 'Name', 'Ticket'], axis=1, inplace=True)
    return df

# 카테고리 인코딩 수행
def format_features(df):
    features = ['Cabin', 'Sex', 'Embarked']
    for feature in features:
        le = LabelEncoder()
        le = le.fit(df[feature])
        df[feature] = le.transform(df[feature])
    return df

# 앞에서 설정한 데이터 전처리 함수 호출
def transform_features(df):
    df = fillna(df)
    df = drop_features(df)
    df = format_features(df)
    return df

# 원본데이터를 재로딩하고, 피처데이터 세트와 레이블 데이터 세트 추출
titanic_df = pd.read_csv('./titanic_train.csv')
y_titanic_df = titanic_df['Survived']
X_titanic_df = titanic_df.drop('Survived', axis=1)
X_titanic_df = transform_features(X_titanic_df)  # 피처데이터

from sklearn.model_selection import train_test_split  # 학습, 검증 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X_titanic_df, y_titanic_df, test_size=0.2,
                                                    random_state=11)
                                                    # 난수 발생값 지정
```

```python
from sklearn.tree import DecisionTreeClassifier   #결정트리
from sklearn.ensemble import RandomForestClassifier   # RandomForest
from sklearn.linear_model import LogisticRegression   #로지스틱회귀
from sklearn.metrics import accuracy_score   #정확성평가

# 결정트리, RandomForest, 로지스틱회귀를 위한 사이킷런 Classifier 클래스 생성
dt_clf = DecisionTreeClassifier(random_state=11)
rt_clf = RandomForestClassifier(random_state=11)
lr_clf = LogisticRegression(solver='liblinear')  # → 최적화알고리즘 (작은데이터 이진분류)

# DecisionTreeClassifier 학습/예측/평가
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf.predict(X_test)
print('DecisionTreeClassifier 정확도: {0:.4f}'.format(accuracy_score(y_test, dt_pred)))

#RandomForestClassifier 학습/예측/평가
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.pred(X_test)
print('RandomForestClassifier 정확도: {0:.4f}'.format(accuracy_score(y_test, rf_pred)))

#LogisticRegression 학습/예측/평가
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
print('LogisticRegression 정확도: {0:.4f}'.format(accuracy_score(y_test, lr_pred)))
```

model.selection 모듈로

모델 성능평가 → 교차검증 (KFold 클래스, cross_val_score(), GridSearchCV)

#1. KFold로 직접반복적 교차검증

```
from sklearn.model_selection import KFold

def exec_kfold(clf, folds=5):
    폴드세트를 5개인 KFold 객체를 생성, 폴드 수만큼 예측결과 저장을 위한 리스트객체 생성
    kfold = KFold(n_splits=folds)
    scores = []
    # KFold 교차 검증 수행
    for iter_count, (train_index, test_index) in enumerate(kfold.split(X_titanic_df)):
        # X_titanic_df 데이터에서 교차 검증별로 학습과 검증 데이터를 가리키는 index 생성
        X_train, X_test = X_titanic_df.values[train_index], X_titanic_df.values[test_index]
        y_train, y_test = y_titanic_df.values[train_index], y_titanic_df.values[test_index]
        # Classifier 학습, 예측, 정확도 계산
        clf.fit(X_train, y_train)
        predictions = clf.predict(X_test)
        accuracy = accuracy_score(y_test, predictions)
        scores.append(accuracy)
        print('교차검증 {0} 정확도: {1:.4f}'.format(iter_count, accuracy))
    # 5개 fold에서의 평균 정확도 계산
    mean_score = np.mean(scores)
    print("평균 정확도: {0:.4f}".format(mean_score))
#exec_kfold 호출
exec_kfold(dt_clf, folds=5)
```

#2. cross_val_score로 간편하게 교차검증

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(dt_clf, X_titanic_df, y_titanic_df, cv=5)
for iter_count, accuracy in enumerate(scores):
    print('교차검증 {0}, 정확도 {1:.4f}'.format(iter_count, accuracy))
print('평균정확도: {0:.4f}'.format(np.mean(scores)))
```

3. GridSearchCV를 통한 하이퍼파라미터 튜닝과 교차검증

```
from sklearn.model_selection import GridSearchCV

parameters = {'max_depth' : [2,3,5,10],
              'min_samples_split':[2,3,5], 'min_samples_leaf' : [1,5,8]}
grid_dclf = GridSearchCV(dt_clf, param_grid=parameters, scoring='accuracy', cv=5)
grid_dclf.fit(X_train, y_train)

print('GridSearchCV 최적 하이퍼파라미터 :', grid_dclf.best_params_)
print('GridSearchCV 최고정확도 : {0:.4f}'.format(grid_dclf.best_score_))
best_dclf = grid_dclf.best_estimator_

# GridSearchCV의 최적 하이퍼파라미터로 학습된 Estimator로 예측 및 정확도 측정
dpredictions = best_dclf.predict(X_test)
accuracy = accuracy_score(y_test, dpredictions)
print('테스트 세트에서의 DecisionTreeClassifier 정확도 : {0:.4f}'.format(accuracy))
```