2. 사이킷런으로 시작하는 머신러닝

1. 사이킷런 소개와 특징

- 파이썬 기반의 다른 머신러닝 패키지도 사이킷런 스타일의 API를 지향할 정도로 쉽고 가장 파이썬스러운 API를 제공합니다.
- 머신러닝을 위한 매우 다양한 알고리즘과 개발을 위한 편리한 프레임워크와 AI기를 제공합니다.
- 오랜 기간 실전 환경에서 검증됐으며, 매우 많은 환경에서 사용되는 성숙한 라이브러리입니다.

2. 첫 번째 머신러닝 만들어 보기 - 붓꽃 품종 예측하기

머신러닝 모델로 붓꽃 데이터 세트로 붓꽃의 품종 분류(Classification)해보기

분류(Classification): 대표적인 지도학습(Supervised Learning) 방법

지도학습: 학습을 위한 다양한 피처(입력, 설명하는 변수들)와 분류 결정값인 레이블(Label) 데이터(출력, 우리가 예측하고 싶은 정)로 모델을 학습한 뒤, 별도의 테스트 데이터 세트에서 미지의 레이블을 예측. 즉 지도학습은 명확한 정답이 주어진 데이터를 먼저 학습한 뒤 미지의 정답을 예측하는 방식.

1. 사이킷런에서 사용할 모듈 임포트

사이킷런 패키지 내의 모듈

- sklearn.datasets 내의 모듈: 사이킷런에서 자체적으로 제공하는 데이터 세트를 생성하는 모듈의 모임.
- sklearn.tree 내의 모듈: 트리 기반 ML 알고리즘을 구현한 클래스의 모임
- sklearn.model_selection: 학습 데이터와 검증 데이터, 예측 데이터로 데이터를 분리하거나 최적의 하이퍼 파라미터로 평가하기 위한 다양한 모듈 의 모임
- 하이퍼 파라미터: 머신러닝 알고리즘별로 최적의 학습을 위해 직접 입력하는 파라미터들을 통칭하며, 하이퍼 파라미터를 통해 머신러닝 알고리즘의 성능을 튜닝할 수 있음

2. 학습용 데이터와 테스트용 데이터 분리

학습 데이터로 학습된 모델이 얼마나 뛰어난 성능을 가지는지 평가하려면 테스트 데이터 세트가 필요

이를 위해 train_test_split() API 제공

X_train, X_test, y_train, y_test = train_test_split(피처 데이터 세트, 레이블 데이터 세트, test_size= 전체 데이터 세트 중 테스트 데이터 세트의 비율, random_state=호출할 때마다 같은 학습/테스트 용 데이터 세트를 생성하기 위해 주어지는 난수 발생 값이며 seed와 같은 의미)

=> 학습용 피처 데이터 세트: X_train

테스트용 피처 데이터 세트: X_test

학습용 레이블 데이터 세트: y_train으로

테스트용 레이블 데이터 세트: y_test

3. 머신러닝 분류 알고리즘의 하나인 의사 결정 트리를 이용해 학습과 예측 수행 <학습> dt_clf = DecisionTreeClassifier(random_state=11) # DecisionTreeClassifier 객체 생성

dt_clf.fit(학습용 피처 데이터 속성, 학습용 결정값 데이터 세트) # 학습 수행

<예측> 예측은 반드 시 학습 데이터가 아닌 다른 데이터를 이용해야 하며, 일반적으로 테스트 데이터 세트를 이용

pred = dt_clf.predict(X_test) # 학습이 완료된 DecisionTreeClassifier 객체에서 테스트 데이터 세트로 예측 수행

4. 의사 결정 트리 기반의 DecisionTreeClassifier의 예측 성능 평가 정확도 측정(일반적으로 머신러닝 모델의 성능 평가 방법은 여러 가지가 있음) : 예측 결과가 실제 레이블 값과 얼마나 정확하게 맞는지를 평가하는 지표

accuracy_score(실제 레이블 데이터 세트, 예측 레이블 데이터 세트)

분류를 예측한 프로세스 정리

1. 데이터 세트 분리: 데이터를 학습 데이터와 테스트 데이터로 분리합니다.
2. 모델 학습: 학습 데이터를 기반으로 ML 알고리즘을 적용해 모델을 학습시킵니다.
3. 예측 수행: 학습된 ML 모델을 이용해 테스트 데이터의 분류(즉, 붓꽃 종류)를 예측합니다.
4. 평가: 이렇게 예측된 결괏값과 테스트 데이터의 실제 결괏값을 비교해 ML 모델 성능을 평가합니다.

3. 사이킷런의 기반 프레임워크 익히기

Estimator 이해 및 fit(), predict() 메서드

사이킷런은 ML 모델 학습을 위해서 fit()을, 학습된 모델의 예측을 위해 predict() 메서드를 제공

지도학습의 주요 두 축인 **분류(Classification)** (Classifier 클래스) 와 **회귀(Regression)** (Regressor 클래) 의 다양한 알고리즘을 구현한 모든 사이킷런 클래스는 fit()과 predict()만을 이용해 간단하게 학습과 예측 결과를 반환

지도학습의 모든 알고리즘을 구현한 클래스를 통칭: Estimator

사이킷런의 주요 모듈 p.92

머신러닝 모델을 구축하는 주요 프로세스는 1. 피처의 가공, 변경, 추출을 수행하는 피처 처리(feature processing), 2. ML 알고리즘 학습/예측 수행, 그리고 3.모델 평가의 단계를 반복적으로 수행 하는 것

내장된 예제 데이터 세트

분류와 클러스터링을 위한 표본 데이터 생성기

datasets.make_classifications( ): 분류를 위한 데이터 세트를 만듭니다. 특히 높은 상관도, 불필요한 속성 등의 노이즈 효과를 위한 데이터를 무작위로 생성해 줍니다.

datasets.make_blobs( ): 클러스터링을 위한 데이터 세트를 무작위로 생성해 줍니다. 군집 지정 개수에 따라 여러 가지 클러스터링을 위한 데이터 세트를 쉽게 만들어 줍니다.

분류나 회귀를 위한 연습용 예제 데이터의 구성

: 딕셔너리 형태. 키는 data, target, target_name, feature_names, DESCR로 구성

- data: 피처의 데이터 세트를 가리킵니다. 넘파이 배열(ndarray) 타입

- target: 분류 시 레이블 값 회귀일 때는 숫자 결괏값 데이터 세트입니다. 넘파이 배열(ndarray) 타입
- target_names: 개별 레이블의 이름을 나타냅니다. 넘파이 배열(ndarray) 타입 또는 파이썬 리스트(list) 타입
- feature_names: 피처의 이름을 나타냅니다. 넘파이 배열(ndarray) 타입 또는 파이썬 리스트(list) 타입
- DESCR: 데이터 세트에 대한 설명과 각 피처의 설명을 나타냅니다. 스트링 타입

load_iris() API의 반환 결과는 sklearn.utils.Bunch 클래스. Bunch 클래스는 파이썬 딕셔너리 자료형과 유사

load_iris() 데이터 세트의 key 값들 : 'data', 'target', 'target_names', 'feature_names' 등...

값을 추출하기 위해서는 데이터 세트.data(또는 데이터 세트['data'])를 이용

ex) load_iris.feature_names, iris_data.target

4. Model Selection 모듈 소개

사이킷런의 model_selection 모듈은 학습 데이터와 테스트 데이터 세트를 분리하거나 교차 검증 분할 및 평가, 그리고 Estimator의 하이퍼 파라미터를 튜닝하기 위한 다양한 함수와 클래스를 제공

학습/테스트 데이터 세트 분리 - train_test_split

예측을 수행하는 데이터 세트는 학습을 수행한 학습용 데이터 세트가 아닌 전용의 테스트 데이터 세트여야 함

train_test_split()를 이용해 붓꽃 데이터 세트를 학습 및 테스트 데이터 세트로 분리

1. sklearn.model_selection 모듈에서 train_test_split을 로드
2. train_test_split(피처 데이터 세트, 레이블 데이터 세트, ...)
   선택적으로 입력받는 파라미터들

- test.size : 전체 데이터에서 테스트 데이터 세트 크기를 얼마로 샘플링할 것인가를 결정합니다. 디폴트는 0.25, 즉 25%
- train_size : 전체 데이터에서 학습용 데이터 세트 크기를 얼마로 샘플링할 것인가를 결정합니다. test_size parameter 를 통상적으로 사용하기 때문에 train_size는 잘 사용되지 않습니다.
- shuffle : 데이터를 분리하기 전에 데이터를 미리 섞을지를 결정합니다. 디폴트는 True입니다. 데이터를 분산시켜서 좀 더 효율적인 학습 및 테스트 데이터 세트를 만드는 데 사용됩니다.
- random_state : random_state는 호출할 때마다 동일한 학습/테스트용 데이터 세트를 생성하기 위해 주어지는 난수 값입니다. train_test_split()는 호출 시 무작위로 데이터를 분리하므로 random_state를 지정하지 않으면 수행할 때마다 다른 학습/테스트용 데이터를 생성합니다.

train_test_split()의 반환값은 튜플 형태
=> 순차적으로 학습용 데이터의 피처 데이터 세트, 테스트용 데이터의 피처 데이터 세트, 학습용 데이터의 레이블 데이터 세트, 테스트용 데이터의 레이블 데이터 세트가 반환

교차 검증

과적합: 모델이 학습 데이터에만 과도하게 최적화되어, 실제 예측을 다른 데이터로 수행할 경우에는 예측 성능이 과도하게 떨어지는 것.
해당 테스트 데이터에만 과적합되는 학습 모델이 만들어져 다른 테스트용 데이터가 들어올 경우에는 성능이 저하될 수 있

교차 검증: 별도의 여러 세트로 구성된 학습 데이터 세트와 검증 데이터 세트에서 학습과 평가를 수행하는 것. 각 세트에서 수행한 평가 결과에 따라 하이퍼 파라미터 튜닝 등의 모델 최적화를 더욱 손쉽게 할 수 있음

대부분의 ML 모델의 성능 평가: 교차 검증 기반으로 1차 평가를 한 뒤에 최종적으로 테스트 데이터 세트에 적용해 평가하는 프로세스

ML에 사용되는 데이터 세트를 세분화해서 **학습, 검증, 테스트 데이터 세트**로 나눌 수 있음

K 폴트 교차 검증

먼저 K개의 데이터 폴드 세트를 만들어서 K번만큼 각 폴트 세트에 학습과 검증 평가를 반복적으로 수행하는 방법

ex) 5 폴드 교차 검증(K=5)

1. 데이터 세트를 K등분（5등분）
2. 첫 번째 반복에서는 처음부터 4개 등분을 학습 데이터 세트, 마지막 5번째 등분 하나를 검증 데이터 세트로 설정하고 학습 데이터 세트에서 학습 수행, 검증 데이터 세트에서 평가를 수행
3. 첫 번째 평가를 수행하고 나면 이제 두 번째 반복에서 다시 비슷한 학습과 평가 작업을 수행. 단, 이번에는 학습 데이터와 검증 데이터를 변경.（처음부터 3개 등분까지, 그리고 마지막 5번째 등분을 학습 데이터 세트로, 4번째 등분 하나를 검증 데이터 세트로 설정
4. 학습 데이터 세트와 검증 데이터 세트를 점진적으로 변경하면서 마지막 5번째（K번째）까지 학습과 검증을 수행
5. 5개（K개）의 예측 평가를 구했으면 이를 평균해서 K 폴드 평가 결과로 반영

사이킷런에서는 KFold와 StratifiedKFold 클래스를 제공
ex) kfold = KFold(n_splits=5) # 5개의 폴드 세트로 분리하는 KFold객체 생성
kfold.split(features) # 학습용/검증용 데이터로 분할할 수 있는 인덱스를 반환

Stratified K 폴드

불균형한(imbalanced) 분포도(특정 레이블 값이 특이하게 많거나 매우 적어서 값의 분포가 한쪽으로 치우치는 것)를 가진 레이블(결정 클래스) 데이터 집합을 위한 K 폴드 방식

K 폴드가 레이블 데이터 집합이 원본 데이터 집합의 레이블 분포를 학습 및 테스트 세트에 제대로 분배하지 못하는 경우의 문제를 해결해줌

StratifiedKFold는 레이블 데이터 분포도에 따라 학습/검증 데이터를 나누기 때문에 split() 메서드에 인자로 피처 데이터 세트뿐만 아니라 레이블 데이터 세트도 반드시 필요 ex)skf = StratifiedKFold(n_splits=3)
skf.split(iris_df, iris_df['label']) # KFold와 다르게 iris_df['label'] 추가

회귀(Regression)에서는 Stratified K 폴드가 지원되지 않음
회귀의 결정값은 이산값 형태의 레이블이 아니라 연속된 숫자값이기 때문에 결정값별로 분포를 결정하는 의미가 없기 때문

교차 검증을 보다 간편하게 - cross_val_score()

1. 폴드 세트를 설정하고 2. for 루프에서 반복으로 학습 및 테스트 데이터의 인덱스를 추출한 뒤 3. 반복적으로 학습과 예측을 수행하고 예측 성능을 반환 하는 과정을 한꺼번에 수행해주는 API

cross_val_score(estimator, X, y=None, scoring=None, cv=Nonez n_jobs=1, verbose=0, fit_params=None, pre_dispatch='2*n_jobs')

주요 파라미터

- estimator(사이킷런의 분류 알고리즘 클래스인 Classifier 또는 회귀 알고리즘 클래스인 Regressor를 의미)
- X(피처 데이터 세트)
- y(레이블 데이터 세트)
- scoring(예측 성능 평가 지표)
- cv(교차 검증 폴드 수)

GridSearchCV - 교차 검증과 최적 하이퍼 파라미터 튜닝을 한 번에

사이킷런은 GridSearchCV API를 이용해 Classifier나 Regressor와 같은 알고리즘에 사용되는 하이퍼 파라미터를 순차적으로 입력하면서 편리하게 최적의 파라미터를 도출할 수 있는 방안을 제공

ex)grid_parameters = {'max_depth': [1, 2, 3],
'min_samples_split': [2, 3]
} # 총 6회에 걸쳐 파라미터를 순차적으로 바꿔 실행하며 최적의 파라미터와 수행 결과 도출할 수 있음

GridSearchCV 클래스의 생성자로 들어가는 주요 파라미터

- estimator : classifier, regressor, pipeline이 사용될 수 있습니다.
- param_grid : key + 리스트 값을 가지는 딕셔너리가 주어집니다. estimator의 튜닝을 위해 파라미터명과 사용될 여러 파라미터 값을 지정합니다.
- scoring : 예측 성능을 측정할 평가 방법을 지정합니다. 보통은 사이킷런의 성능 평가 지표를 지정하는 문자열 (예: 정확도의 경우 'accuracy') 로 지정하나 별도의 성능 평가 지표 함수도 지정할 수 있습니다.
- cv : 교차 검증을 위해 분할되는 학습/테스트 세트의 개수를 지정합니다.
- refit : 디폴트가 True이며 True로 생성 시 가장 최적의 하이퍼 파라미터를 찾은 뒤 입력된 estimator 객체를 해당 하이퍼 파라미터로 재학습시킵니다

1. 학습 데이터 세트를 GridSearchCV 객체의 fit(학습 데이터 세트) 메서드에 인자로 입력
2. GridSearchCV 객체의 fit(학습 데이터 세트) 메서드를 수행하면 학습 데이터를 cv에 기술된 폴딩 세 트로 분할해 param_grid에 기술된 하이퍼 파라미터를 순차적으로 변경하면서 학습/평가를 수행하고 그 결과를 cv_results_ 속성에 기록
3. cv_results_는 gridsearchcv의 결과 세트로서 딕셔너리 형태로 key 값과 리스트 형태의 value 값을 가짐 cv_results를 df으로 변환

- params 칼럼에는 수행할 때마다 적용된 개별 하이퍼 파라미터값을 나타냄
- rank_test_socre는 하이퍼 파라미터별로 성능이 좋은 score 순위를 나타냄. 1이 가장 뛰어난 순위이며 이때의 파라미터가 최적의 하이퍼 파라미터
- mean_test_score는 개별 하이퍼 파라미터별로 CV의 폴딩 테스트 세트에 대해 총 수행한 평가 평균값

ex) grid_dtree = GridSearchCV(dtree, param_grid=parameters, cv=3, refit=True)
grid_dtree.fit(X_train, y_train)
scores_df = pd.DataFrame(grid_dtree.cv_results_)

최고 성능을 나타낸 하이퍼 파라미터의 값과 그때의 평가 결과 값이 각각 best_params_, best_score_ 속성에 기록

5. 데이터 전처리

<사이킷런의 ML 알고리즘을 적용하기 전에 데이터에 대해 미리 처리해야 할 기본 사항>

1. 결손값, 즉 NaN, Null 값은 허용되지 않음
2. 문자열 값을 입력값으로 허용하지 않음

데이터 인코딩

1. 레이블 인코딩(Label encoding)

:카테고리 피처를 코드형 숫자 값으로 변환. *LabelEncoder 클래스*

LabelEncoder를 객체로 생성한 후 fit()과 transform()을 호출해 레이블 인코딩을 수행
ex) items=[' ',' ',' ']
encoder = LabelEncoder()
encoder.fit(items) labels = encoder.transform(items)

무슨 문자열이 어떤 숫자값으로 인코딩 됐는지 알려면 -> LabelEncoder 객체의 classes_속성값 확인.
classes_ 속성은 0번부터 순서대로 변환된 인코딩 값에 대한 원본값을 가지고 있음 ex) encoder.classes_

인코딩된 값을 다시 디코딩 -> inverse_transform() ex) encoder.inverse_transform([1,2,3...])

그러나! 레이블 인코딩은 몇몇 ML 알고리즘에는 이를 적용할 경우 예측 성능이 떨어질 수 있음
냉장고가 1, 믹서가 2로 변환되면, 1보다 2가 더 큰 값이므로 특정 ML 알고리즘에서 가중치가 더 부여되거나 더 중요하게 인식할 가능성이 발생
선형 회귀와 같은 ML 알고리즘에는 적용하지 않아야 함

원-핫 인코딩(One—Hot Encoding)

피처 값의 유형에 따라 새로운 피처를 추가해 고유 값에 해당하는 칼럼에만 1을 표시하고 나머지 칼럼에는 0을 표시하는 방식
행 형태로 돼 있는 피처의 고유 값을 열 형태로 차원을 변환한 뒤, 고유 값에 해당하는 칼럼에만 1을 표시하고 나머지 칼럼에는 0을 표시 # 1차원이 2차원이 됨

주의점(LabelEncoder와 차이)

- 입력값으로 2차원 데이터가 필요함
- OneHotEncoder를 이용해 변환한 값이 희소 행렬(Sparse Matrix) 형태이므로 이를 다시 toarray() 메서드를 이용해 밀집 행렬(Dense Matrix)로 변환해야 함

<적용>

1. 2차원 ndarray로 변환
   items = [' ',' ',' ']
   items = np.array(items).reshape(-1, 1)

2. 원-핫 인코딩을 적용 oh_encoder = OneHotEncoder()
   oh_encoder.fit(items)
   oh_labels = oh_encoder.transform(items)

3. OneHotEncoder로 변환한 결과는 희소행렬이므로 toarray()를 이용해 밀집 행렬로 변환.
   oh_labels.toarray()

*판다스*에 원-핫 인코딩을 더 쉽게 지원하는 API:
get_dummies()
ex) df = pd.DataFrame({'item':['TV', '냉장고', '전자레인지', '컴퓨터', '선풍기', '선풍기', '믹서', '믹서']})
pd.get_dummies(df)

피처 스케일링과 정규화

서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업

1. 표준화 : 데이터의 피처 각각이 평균이 0이고 분산기 1 인 가우시안 정규 분포를 가진 값으로 변환 *원래 값에서 피처의 평균을 뺀 값을 피처의 표준편차로 나눈 값*
2. 정규화 : 서로 다른 피처의 크기를 통일하기 위해 크기를 변환해주는 개념. 개별 데이터의 크기를 모두 똑 같은 단 위로 변경

원래 값에서 피처x의 최솟값을 뺀 값을 피처x의 최댓값과 최솟값의 차이로 나눈 값으로 변환

사이킷런의 Normalizer 모듈 : 선형대수에서의 정규화 개념이 적용. 개별 벡터의 크기를 맞추기 위해 변환하는 것을 의미
개별 벡터를 모든 피처 벡터의 크기로 나눠줌
ex) 세 개의 피처 x, y, z가 있다고 하면 새로운 데이터 xi_new는 원래 값에서 세 개의 피처의 i번째 피처 값에 해당하는 크기를 합한 값으로 나눠줌

StandardScaler

: 표준화를 쉽게 지원하기 위한 클래스. 개별 피처를 평균이 0이고, 분산이 1 인 값으로 변환.

이렇게 가우시안 정규 분포를 가질 수 있도록 데이터를 변환하는 것은 몇몇 알고리즘에서 매우 중요
=> 사이킷런에서 구현한 RBF 커널을 이용하는 서포트 벡터 머신(Support Vector Machine)이나 선형 회귀(Linear Regression), 로지스틱 회귀(Logistic Regression)는 데이터가 가우시안 분포를 가지고 있다고 가정하고 구현됐기 때문

1. StandardScaler 객체를 생성한 후에 fit()과 transform() 메서드에 변환 대상 피처 데이터 세트를 입력하고 호출
2. transform( ) 시 스케일 변환된 데이터 세트가 NumPy ndarray로 반환돼 이를 DataFrame으로 변환

MinMaxScalaer

데이터값을 0과 1 사이의 범위 값으로 변환합니다(음수 값이 있으면 -1 에서 1값으로 변환)
데이터의 분포가 가우시안 분포가 아닐 경우에 Min, Max Scale을 적용해 볼 수 있음

1. MinMaxScaler 객체를 생성한 후에 MinMaxScaler로 데이터 세트 변환. fit()과 transformO 호출.
2. transform( )시 스케일 변환된 데이터 세트가 NumPy ndarray로 반환돼 이를 DataFrame으로 변환

학습 데이터와 테스트 데이터의 스케일링 변환 시 유의점

StandardScaler나 MinMaxScaler와 같은 Scaler 객체를 이용해 데이터의 스케일링 변환 시 fit( ), transform(), fit_transform() 메서드를 이용

- fit(): 데이터 변환을 위한 기준 정보 설정(예를 들어 데이터 세트의 최댓값/최솟값 설정 등)을 적용
- transform(): 이렇게 설정된 정보를 이용해 데이터를 변환
- fit_transform(): fit()과 transform()을 한 번에 적용하는 기능을 수행

! 주의점 !
Scaler 객체를 이용해 학습 데이터 세트로 fit()과 transform()을 적용하면 *테스트 데이터 세트로는 다시 fit()을 수행하지 않고* 학습 데이터 세트로 fit()을 수행한 결과를 이용해 transform() 변환을 적용해야 한다는 것.
학습 데이터로 fit()이 적용된 스케일링 기준 정보를 그대로 테스트 데이터에 적용해야 함
(테스트 데이터로 다시 새로운 스케일링 기준 정보를 만들게 되면 학습 데이터와 테스트 데이터의 스케일링 기준 정보가 서로 달라지기 때문에 올바른 예측 결과를 도출 하지 못 함)

! 기억할 점!

1. 가능하다면 전체 데이터의 스케일링 변환을 적용한 뒤 학습과 테스트 데이터로 분리
2. 1이 여의치 않다면 테스트 데이터 변환 시에는 fit()이나 fit_transform()을 적용하지 않고 학습 데이터로 이미 fit()된 Scaler 객체를 이용해 transform()으로 변환

6. 사이킷런으로 수행하는 타이타닉 생존자 예측

타이타닉 탑승자 데이터

- Passengerid : 탑승자 데이터 일련번호
- survived : 생존 여부, 0 = 사망, 1 = 생존
- pdass : 티켓의선실 등급, 1 = 일등석,2 = 이등석,3 = 삼등석
- sex : 탑승자 성별
- name : 탑승자 이름
- Age : 탑승자 나이
- sibsp : 같이 탑승한 형제자매 또는 배우자 인원수
- parch: 같이 탑승한 부모님 또는 어린이 인원수
- ticket: 티켓 번호
- fare : 요금
- cabin : 선실번호
- embarked : 중간 정착 항구 C = Cherbourg, Q = Queenstown, S = Southampto

사이킷런 머신러닝 알고리즘은 Null 값을 허용하지 않으므로 Null 값을 어떻게 처리할지 결정해야 함
-> DataFrame의 fillna() 함수를 사용해 간단하게 Null 값을 평균 또는 고정 값으로 변경

<타이타닉 데이터 전처리>

- Null값 처리
  titanic_df['Age'].fillna(titanic_df['Age'].mean(), inplace=True)
  titanic_df['Cabin'].fillna('N', inplace=True)
  titanic_df['Embarked'].fillna('N', inplace=True)

- 문자열 카테고리 피처를 숫자형 카테고리 피처로 변환
  인코딩은 사이킷런의 LabelEncoder 클래스를 이용해 레이블 인코딩을 적용
  LabelEncoder 객체는 카테고리 값의 유형 수에 따라 0 ~ (카테고리 유형 수-1)까지의 숫자 값으로 변환

- 머신러닝 알고리즘에 불필요한 피처 제거
  df.drop(['Passengerld', 'Name', 'Ticket'], axis=1, inplace=True)

시본(Seaborn) 패키지를 이용하여 시각화 시본은 기본적으로 맷플롯립에 기반하고 있지만, 좀 더 세련된 비주얼과 쉬운 API, 편리한 판다스 DataFrame 과의 연동 등으로 데이터 분석을 위한 시각화로 애용되는 패키지

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