데이터 전처리(Data Processing)

: 데이터를 가공하는 작업

구분	내용
데이터 클리닝	결측치 처리, 이상치 확인 및 정제 등
데이터 통합	다양한 데이터 파일의 결함 등
데이터 변환	스케일링, 요약 등
데이터 축소	변수 축소, 라벨링 등
불균형 데이터 처리	언더 샘플링, 오버 샘플링 등
데이터 분할	train, test 데이터 분할 등

결측 데이터

결측 데이터의 종류

- 완적 무작위 결측(MCAR) 결측 데이터가 관측 혹은 비관측 데이터와 아무런 연관이 없는 경우
- 무작위 결측(MAR) : 결측 데이터가 관측 변수와 관련 되어 있지만 비관측값들과는 연관되지 않은 경우
  ex) 여성(변수1)은 체중 공개를 꺼려함. 다른 변수(여성 여부)에 의해 체중 변수의 누락가능성이 생김
- 비 무작위 결측(NMAR) : 결측 변수가 모종의 이유에 의해 결측이 되는 데이터 ex) 적은 소득
  ex) 무겁거나 가벼운 사람들은 체종 공개를 꺼려함. 누락 가능성이 있는 변수가 해당 변수 자체에 관찰되지 않는 값에 달려있음.
- 일반적으로 완적 무작위 결측하에 처리, 즉 불완전 자료는 무시하고 완전히 관측된 자료만으로 표준 분석 시행

  문제점 : 효율성, 자료처리의 복잡성, 편향

결측값 탐색	
isna()	결측치일 경우 True 반환
notna()	결측치일 경우 False 반환
isna().sum()	열에서 결측치의 개수를 보여줌
isna().sum(1)	행에서 결측치의 개수를 보여줌

```
import numpy as np
import pandas as pd
```

```
#사용 데이터셋  https://www.kaggle.com/fedesoriano/heart-failure-prediction?select=heart.csv]
# na_values에 지정된 값들이 결측치로 변환되어 불러짐
df=read_csv("./DATA/archive/heart.csv", na_values=['', 'NA', -1, 9999])

#isna() 사용하면 결측치의 위치가 true로 표시됨.
#.sum()을 사용하면 각 열별 결측치 갯수 확인 가능
df.isna().sum()

#어떤 열에서 결측치가 포함된 행 출력
df[df['Oldpeak'].isnull()]

#결측치가 있는 행 제외
df[(df['Oldpeak'].notnull()) ]
```

1. 단순 대치법(Simple Imputation)

결측치를 MCAR, MAR로 판단하고 처리함.

- 완전분석(Completes Analysis)

  불완전 자료는 완전 무시, 분석이 용이하나 효율성 상실과 통계적 추론의 타당성에 문제 발생 가능성 있음.

- 평균 대치법(Mean Imputation)

  결측값을 데이터 평균으로 대치, 효율성은 향상되지만, 표준오차가 과서 추정됨, 비조건부 평균 대치법이라고도 함.

- 회귀 대치법(Regression Imputation)

  회귀분석에 의한 예측치로 결측치를 대치하는 방법, 조건부 평균 대치법

- 단순확률대치법(Single Stochastic Imputation)

  평균 대치법에서 추정량 표준오차의 과소 추정을 보완하는 방법, Hot-deck 방법, 확률추출에 의해서 전체 데이터 중 무작위로 대치하는 방법

- 최근접 대치법(Nearest-Neighbor Imputation)

  전체표본을 몇 개의 대체군으로 분류하여 각 층에서의 응답자료를 순서대로 정리한 후, 결측값 바로 이전의 응답으로 대치, 응답값이 여러 번 사용될 가능성 있음.

- 사회적 조사방법론

  조사단위 대치법, 콜드덱 대치법, 이월 대치법

2. 다중 대치법(Multiple Imputation)
   단순 대치법을 복수로 시행하여 통계적 효율성 및 일치성 문제를 보완, 복수 개의 단순대치를 통해 n개의 새로운 자료를 만들어 분석을 시행, 시행결과 얻어진 통계량에 대해 통계량 및 분산 결합을 통해 통합하는 방법

- 1단계, 대치 단계(Imputation Step) : 복수의 대치에 의한 결측을 대치한 데이터를 생성
- 2단계, 분석 단계(Analysis Step) : 복수 개의 데이터셋에 대한 분석을 시행
- 3단계, 결합 단계(Combination Step) : 복수 개의 분석결과에 대한 통계적 결합을 통해 결과를 도출

결측값 제거 함수
DataFrame.dropna(axis=0, how='any',thresh=None,subset=None,inplace=False)

axis	0 또는 'index' 결측값이 포함된 행 삭제, 1 또는 'column'이면 결측값이 포함된 열 삭제
how	any'면 결측값이 존재하는 모든 행/열 삭제, all이면 모든 값이 결측값일 때 삭제
thresh	결측값이 아닌 값(정상값)이 지정된 숫자보다 많을 때 행 또는 열을 유지
subset	어떤 레이블에 결측값이 존재하면 삭제할 지 정의
inplace	True이면 제자리에서 작업을 수행하고 None 반환, False면 복사값 리턴

중복행 제거 데이터 프레임에 모든 컬럼값이 중복된 행이 있는 경우 제거
DataFrame.drop_duplicates()

```
import numpy as np
from pandas import read_csv
import pandas as pd

##결측치 삭제
#NaN 값이 1개라도 포함된 행이 모두 제거
testdf.dropna()

#모든 값이 전부 NaN인 행만 제거하고 싶을 땐 how 인자를 all 로 지정
testdf.dropna(how = 'all')

#특정 열의 위치에서 결측치가 등장할 경우에만 행을 제거
testdf.dropna(subset = ['Oldpeak'])

#정상 데이터가 1개 이하인 행 제거
testdf.dropna(thresh  = 1)

#결측치를 포함하고 있는 행 대신 열 방향으로 제거하고 싶은 경우
#axis 인자를 1 혹은 'columns'로 지정
testdf.dropna(axis = 1)
```

결측값 대체 함수
DataFrame.fillna(value=None, method=None, axis=None, inplace=False, limit=None)

value	단일 값 혹은 dict/Series/DataFrame 형식으로 대체할 값을 직접 입력
method	pad', 'ffill' 이전 값으로 채우고, 'backfill', 'bfill'은 다음에 오는 값으로 채움
axis	0 또는 'index'이면 행 방향으로 채우고 1또는 'columns'이면 열 방향으로 채움
limit	method 인자를 지정한 경우 limit으로 지정한 개수만큼만 대체할 수 있음.
inplace	True이면 제자리에서 작업을 수행하고 None 반환, False면 복사값 리턴

```
import numpy as np
from pandas import read_csv
import pandas as pd
from sklearn.impute import SimpleImputer

##결측치 대체
```

```
testdf['Oldpeak'] = testdf['Oldpeak'].fillna(1)
print(testdf.isna().sum())
testdf[(testdf['Oldpeak'].isnull())]

#Scikit Learn의 SimpleImputer 함수를 이용해 결측치를 평균, 중앙값, 최빈값으로 대치
testdf=read_csv("./DATA/archive/heart.csv")
testdf2 = pd.DataFrame()
testdf2['Age']=testdf['Age']
testdf2['Oldpeak']=testdf['Oldpeak']
print(testdf2)

SI_mean = SimpleImputer(missing_values = np.nan, strategy='mean') #median
SI_mean.fit(testdf2)
testdf[testdf2['Oldpeak'].isnull()]
testdf2 = SI_mean.transform(testdf2)
testdf2
```

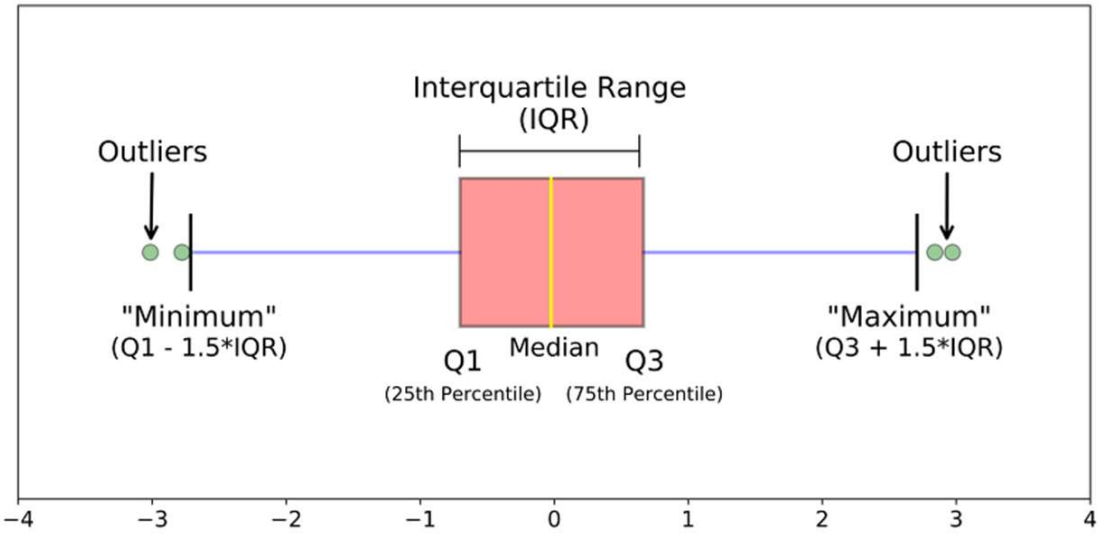
이상치

이상치의 종류

1. 관측값의 형식과는 다른 형식의 값으로 표시된 결측치 > 컬럼 형식으로 확인 가능
2. 관측값의 형식과 같은 형식의 값으로 표시된 결측치
3. 자료 수집의 오류로 발생한 이상치 > 2,3번의 경우 극단값으로 평균 연산에 전혀 다른 값 도출 될 수 있다.
4. 다른 관측치들과는 현저히 차이나는 실제 관측치 > 실제 추출된 값이지만 아주 독특한 값으로, 정제하지 않을 때는 모델 전체에 악영향 줄 수 있다. 하지만 해당 이상치를 탐색하는 경우는 삭제하면 안됨.

이상치 확인

1. IQR(Inter Quantile Range) Box Plot의 이상치 결정 방법을 그대로 이용하는 것
   Box Plot 데이터를 오름차순하여 정확히 4등분 했을 때 구분되는 선 3개를 차례대로 1,2,3사분위라고 한다.
   IQR은 제3사분위수(Q3)값에서 제1사분위수(Q1)값을 뺀 것이다.
   IQR 방법은 Q1-1.5*IQR부터 Q3+1.5*IQR 를 제외한 값은 이상치로 간주한다.



범주형 변수 처리

범주형 변수는 몇 개의 부류나 범우, 서열 등으로 구분하여 수집된 변수로 질적변수라고도 한다.
범주형 변수로 수학적 처리를 하기 위해서는 더미변수화하여 모델이 이해할 수 있는 헝태로 만드는 것이 중요
더미변수 : 범주형 변수에 있는 범주 각각을 컬럼으로 변경하고, 원본 컬럼의 값이 해당 범주에 속하는 지 여부에 따라 1 혹은 0으로 채운 변수

```
#범주형 변수를 더미변수로 변환
pd.get_dummies(data, columns =['범주형1','범주형2'])

#범주형 데이터를 더미변수로 생성
import pandas as pd
from sklearn.datasets import load_iris
iris=load_iris()
iris=pd.DataFrame(iris.data, columns=iris.feature_names)
iris['Class']=load_iris().target
iris['Class']=iris['Class'].map({0:'Setosa',1:'Versicolour',2:'Virginica'})

iris_dummy=pd.get_dummies(iris, columns=['Class'])
iris_dummy
```

데이터 분할

데이터를 Train, Test 세트로 나누고 독립변수와 종속변수를 분리해야 함.
머신러닝 등 분석 방법에 따라서는 Train과 Validation, Test세트로 분리하기도 함.
전통적으로 머신러닝에서는 Train : Test = 7:3, 혹은 Train : Validation : Test = 6,2,2 정도의 비율로 나누는 것이 적절

from sklearn.mode_selection import train_test_split
X_Train, X_test, y_train, y_test = train_test_split(arrays,test_size=None, train_size=None,random_state=None,shuffle=True,stratify=None)

파라미터	설명
X	독립변수 테이블
Y	종속변수 테이블
test_size	테스트 사이즈 비율
random_state	임의의 번호 지정, 같은 숫자를 사용하면 같은 출력이 나옴
shuffle	True이면 추출 전에 데이터를 섞음. False이면 섞지 않음.
stratify	None이 아닌 경우 데이터는 지정한 변수를 기준으로 계층화되어 해당 변수의 비율을 유지하도록 추출

```
import pandas as pd
from sklearn.datasets import load_iris
loaded_iris = load_iris()
iris = pd.DataFrame(loaded_iris, columns = loaded_iris.feature_names)
iris['class'] = loaded_iris.target
iris['class'] = iris['class'].map({0:'Setosa',1:'Versicolour',2:'Virginica'})

from sklearn.model_selection import train_test_split
#test_size=0.2로 train:test=8:2 비율로 쪼갬
#random_State에 수를 지정하면 다음번에 같은 수를 random_state에 지정했을 때 똑같은 데이터를 얻을 수 있음. 튜닝할 때나 분석결과를 비교할 때 유용

X_train, X_test, y_train, y_test =
train_test_split(iris.drop(columns='class'),iris['class'],test_size=0.2,random_state=1004)
print("X_train : ",X_train.shape, " X_test : ", X_test.shape)
print("y_train : ",y_train.shape, " y_test : ", y_test.shape)

''' 결과
X_train :  (120, 4)  X_test :  (30, 4)
y_train :  (120,)  y_test :  (30,)
'''

#stratify를 지정하면 층화임의추출이 일어나 각 계층을 대표하도록 모든 계층에서 원소를 임의추출함.
#test, train에서 원본 데이터와 같은 비율로 분포되어야 할 때, 변수 개수 차이가 크게 날때 종종 사용
X_train, X_test, y_train, y_test =
train_test_split(iris.drop(columns='class'),iris['class'],test_size=0.2,random_state=1004
                                                    ,stratify=iris['class'])
y_train.value_counts()
'''결과
Setosa         40
Virginica      40
Versicolour    40
Name: class, dtype: int64'''
```

데이터 스케일링(Data Scaling)

컬럼 간 데이터의 범위가 크게 날 경우 분석 알고리즘이 잘 작동하지 않을 수 있으므로, 스케일링을 통해 모든 컬럼의 값의 범위를 같게 만들어줌.

데이터 스케일링 분류

1. 표준화 스케일링
   각 컬럼의 평균을 0, 분산이 1인 정규분포로 만드는 방법
2. 정규화 스케일링
   각 컬럼들의 값이 특정범위 (주로 0~1) 안에 들어가도록 스케일링

데이터 스케일링 방법

1. Scaler 선택 및 import
2. Scaler 객체 생성
3. train 데이터의 분포 저장 : scaler.fit(X_train)
4. train 데이터 스케일링 : scaler.transform(X_train)
5. test 데이터 스케일링 : scaler.transform(X_test)
6. 원래 스케일로 변환 : scaler.inverse_transform(X_train_scaled)

데이터 스케일링 기법

1. Standard Scaler
   표준화 방식, 이상치에 매우 민감하여 이상치 정제 후 사용. 분류분석에 많이 사용

```
#Standard Scaler : 표준화 방식
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_iris
import pandas as pd

iris = pd.DataFrame(load_iris().data, columns = load_iris().feature_names)
iris['class'] = load_iris().target
X_train, X_test, y_train, y_test =
train_test_split(iris.drop(columns='class')
                     ,iris['class'],test_size=0.2,random_state=1004)

'''★중요★'''
staardScaler = StandardScaler()
staardScaler.fit(X_train)
X_train_sc = staardScaler.transform(X_train)
X_test_sc = staardScaler.transform(X_test)

print("x_train 데이터 ")
print(f"평균 : {X_train_sc.mean()}, 분산 : {X_train_sc.var()}, 최소값 :
{X_train_sc.mean()} , 최대값 : {X_train_sc.max()}")
print("x_test 데이터 ")
print(f"평균 : {X_test_sc.mean()}, 분산 : {X_test_sc.var()}, 최소값 :
{X_test_sc.mean()} , 최대값 : {X_test_sc.max()}")

'''
x_train 데이터
평균 : 6.883382752675971e-16, 분산 : 1.0, 최소값 : 6.883382752675971e-16 , 최대
값 : 3.0244041288130474
```

```
x_test 데이터
평균 : 0.07523060109391032, 분산 : 1.0707626582288745, 최소값 :
0.07523060109391032 , 최대값 : 2.330749306858225
'''
```

2. Min-max Scaler
   정규화 방식, 컬럼들을 0과 1 사이의 값으로 스케일링. 최솟값이 0 최댓값이 1, 이상치에 매우 민감, 회귀에 많이 사용

```
#MinMaxScaler : 정규화 방식
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_iris
import pandas as pd

iris = pd.DataFrame(load_iris().data, columns = load_iris().feature_names)
iris['class'] = load_iris().target
X_train, X_test, y_train, y_test =
train_test_split(iris.drop(columns='class')
                     ,iris['class'],test_size=0.2,random_state=1004)

'''★중요★'''
MmScaler = MinMaxScaler()
MmScaler.fit(X_train)
X_train_sc = MmScaler.transform(X_train)
X_test_sc = MmScaler.transform(X_test)

print("x_train 데이터 ")
print(f"평균 : {X_train_sc.mean()}, 분산 : {X_train_sc.var()}, 최소값 :
{X_train_sc.mean()} , 최대값 : {X_train_sc.max()}")
print("x_test 데이터 ")
print(f"평균 : {X_test_sc.mean()}, 분산 : {X_test_sc.var()}, 최소값 :
{X_test_sc.mean()} , 최대값 : {X_test_sc.max()}")

'''
x_train 데이터
평균 : 0.444757827212806, 분산 : 0.06734581070849766, 최소값 :
0.444757827212806 , 최대값 : 1.0
x_test 데이터
평균 : 0.4644342435655994, 분산 : 0.07448989656309342, 최소값 :
0.4644342435655994 , 최대값 : 0.9661016949152543
'''
```

3. SMax Abs Scaler
   최대 절댓값과 0이 각각 1,0이 되도록 스케일링하는 정규화 방식. 모든값은 -1과 1 사이에 표현. 데이터가 양수인 경우 Min-max Scaler와 동일
   이상치에 매우 민감, 회귀분석에 많이 사용

```
from sklearn.preprocessing import MaxAbsScaler
from sklearn.datasets import load_iris
import pandas as pd
```

```
iris = pd.DataFrame(load_iris().data, columns = load_iris().feature_names)
iris['class'] = load_iris().target
X_train, X_test, y_train, y_test =
train_test_split(iris.drop(columns='class')
                     ,iris['class'],test_size=0.2,random_state=1004)

'''★중요★'''
maxAbsScaler = MaxAbsScaler()
maxAbsScaler.fit(X_train)
X_train_sc = maxAbsScaler.transform(X_train)
X_test_sc = maxAbsScaler.transform(X_test)

print("x_train 데이터 ")
print(f"평균 : {X_train_sc.mean()}, 분산 : {X_train_sc.var()}, 최소값 :
{X_train_sc.mean()} , 최대값 : {X_train_sc.max()}")
print("x_test 데이터 ")
print(f"평균 : {X_test_sc.mean()}, 분산 : {X_test_sc.var()}, 최소값 :
{X_test_sc.mean()} , 최대값 : {X_test_sc.max()}")

'''
x_train 데이터
평균 : 0.611639629300712, 분산 : 0.05524187557926485, 최소값 :
0.611639629300712 , 최대값 : 1.0
x_test 데이터
평균 : 0.6270439160454295, 분산 : 0.05782332684744166, 최소값 :
0.6270439160454295 , 최대값 : 0.9746835443037974
'''
```

4. Robuster SCaler

   평균과 분산 대신 중앙값과 사분위 값을 활용하는 방식, 중앙값을 0으로 설멍, IQR을 사용하여 이상치의 영향을 최소화함.

   quantile_range 파라미터(default [0.25, 0.75])를 조정하여 더 넓거나 좁은 범위의 값을 이상치로 설정하여 정제

```
from sklearn.preprocessing import RobustScaler
from sklearn.datasets import load_iris
import pandas as pd

iris = pd.DataFrame(load_iris().data, columns = load_iris().feature_names)
iris['class'] = load_iris().target
X_train, X_test, y_train, y_test =
train_test_split(iris.drop(columns='class')
,iris['class'],test_size=0.2,random_state=1004)

'''★중요★'''
robustScaler = RobustScaler()
robustScaler.fit(X_train)
X_train_sc = robustScaler.transform(X_train)
X_test_sc = robustScaler.transform(X_test)

print("x_train 데이터 ")
print(f"평균 : {X_train_sc.mean()}, 분산 : {X_train_sc.var()}, 최소값 :
{X_train_sc.mean()} , 최대값 : {X_train_sc.max()}")
```

```
print("x_test 데이터 ")
print(f"평균 : {X_test_sc.mean()}, 분산 : {X_test_sc.var()}, 최소값 :
{X_test_sc.mean()} , 최대값 : {X_test_sc.max()}")

'''
x_train 데이터
평균 : -0.027444762684124355, 분산 : 0.4271695957200582, 최소값 :
-0.027444762684124355 , 최대값 : 2.8000000000000007
x_test 데이터
평균 : 0.018691580287324976, 분산 : 0.4203515070082836, 최소값 :
0.018691580287324976 , 최대값 : 1.7999999999999998
'''
```

5. 원본 스케일로 변경하기

```
pd.DataFrame(X_train_sc).head(3)
'''
    0   1   2   3
0   0.384615    -1.4    0.028369    0.000000
1   0.000000    -0.6    -0.056738   -0.200000
2   1.615385    1.6 0.595745    0.466667
'''

X_Original = robustScaler.inverse_transform(X_train_sc)
pd.DataFrame(X_Original).head(3)
'''
    0   1   2   3
0   6.3 2.3 4.4 1.3
1   5.8 2.7 4.1 1.0
2   7.9 3.8 6.4 2.0
'''
```

차원 축소

차원의 저주 : 설명변수가 아주 많은 경우 성능이 저하되는 현상,
공간 섬김 현상(Sparsity) : 차원이 늘어남에 따라 데이터 간의 거리가 멀어지는 현상 또는
각 차원별 같은 영역의 자료를 갖고 있을 때 전체 영역에서 설명할 수 있는 데이터의 비율은 줄어드는 현상
데이터의 차원이 증가한다는 것은 새로운 패턴 발생 가능성이 있지만, 데이터가 없는 공간을 늘리는 행위이다

설명변수 선택

유용하지 않거나 상관관계가 높은 컬럼은 제거하는 방법
ex) 같은 값으로 이루어진 컬럼을 삭제한다거나, 일자, 코드 정보들을 삭제한다.
설명 변수의 해석이 쉽고 수행 과정이 간단하나 설명변수간의 고차원적인 상관관계는 고려하기 어렵다.

주성분 분석(PCA : principal Component Analysis)

차원 추출은 기존의 컬럼을 새롭게 해석하여 저차원의 초평면에 투영하는 것.
여러 변수 간에 존재하는 상관관계를 이용하는 차원 축소 방법으로 데이터를 축에 사영했을 때 분산이 가장 높은 축을 찾아 그 축을 새로운 주성분으로 결정하는 방법
먼저 가장 큰 분산을 기반으로 첫번째 축을 생성, 이 벡터 축에 직각이 되는 벡터를 두번째 축으로 선택, 세번째 축은 다시 두번째 축과 직각이 되는 벡터를 축으로 설정

주성분 분석과정

1. PCA를 위한 전처리 : 변수 간 스케일의 차이가 주성분 선정에 영향을 주는 것을 방지하기 위해 이상치를 제거하고 스케일링을 수행
   #수치형 데이터만 추출
   fetrues = ['수치형변수1','수치형변수2']
   x = data[features]

   #수치형 변수 정규화(StandardScaler 참고)
   from sklearn.preprocessing import StandardScaler
   x = StandardScaler().fit_transform(x)

```
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
import pandas as pd

iris = pd.DataFrame(load_iris().data, columns = load_iris().feature_names)
standardScaler = StandardScaler()

#Train 데이터의 fitting과 스케일링
standardScaler.fit(iris)
iris_sc = standardScaler.transform(iris)

print("iris_sc 데이터 ")
print(f"평균 : {iris_sc.mean()}, 분산 : {iris_sc.var()}, 최소값 :
{iris_sc.mean()} , 최대값 : {iris_sc.max()}")
'''
iris_sc 데이터
평균 : -1.4684549872375404e-15, 분산 : 1.0, 최소값 : -1.4684549872375404e-15 ,
최대값 : 3.0907752482994253
'''
```

2. 주성분 추출 : 생성할 주성분 개수를 임의로 설정하여 추출

   pca.singular_values_ (고유값): 전체 데이터에서 각 주성분이 설명할 수 있는 분산의 비율
   pca.explained_variance_ratio_ (분산설명력) : 전체 데이터에서 선택된 주성분 개수로 설명할 수 있는 분산의 비율

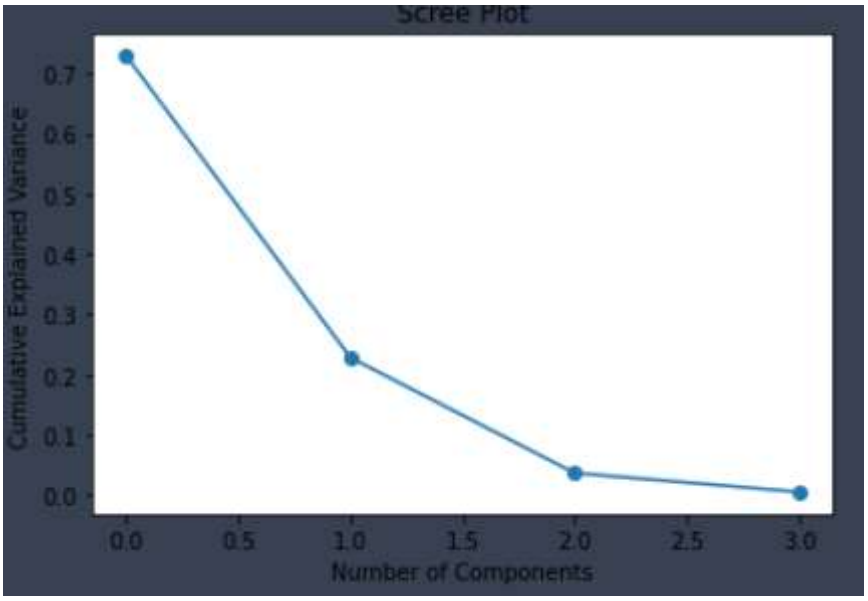
```
from sklearn.decomposition import PCA

pca = PCA(n_components = 4 ) #n_components : 생성할 주성분의 개수
pca_fit = pca.fit(iris_sc)

#전체 데이터에서 선택된 주성분 개수로 설명할 수 있는 분산의 비율
print("고유 값 : ", pca.singular_values_)
```

```
#전체 데이터에서 각 주성분이 설명할 수 있는 분산의 비율
print("분산 설명력 : ", pca.explained_variance_ratio_)

'''
고유 값 :  [20.92306556 11.7091661   4.69185798  1.76273239]
분산 설명력 :  [0.72962445 0.22850762 0.03668922 0.00517871]
'''
```

3. Scree Plot으로 사용할 주성분의 개수 정하기

   주성분 각각으로 설명할 수 있는 분산의 정도를 점으로 표시하고 각 점들을 이은 선
   주성분 개수가 증가할수록 Scree Plot의 기울기(분산의 변화 정도)는 감소
   플롯의 기울기가 급격히 감소하는 지점의 직전까지를 주성분으로 선택

```
import matplotlib.pyplot as plt
plt.title('Scree Plot')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.plot(pca.explained_variance_ratio_,'o-')
plt.show()
```



4. 새로운 데이터프레임 확인

   주성분의 수를 정하고 2를 다시 수행하여 주성분 객체를 생성하고, fit_transform()을 수행하여 원하는 개수의 주성분을 가진 데이터프레임을 만든다.

```
pca = PCA(n_components = 2) #주성분 개수는 2개
principalComponents = pca.fit_transform(iris)
principal_iris = pd.DataFrame(data = principalComponents, columns =
['pca1','pca2'])

principal_iris.head()
'''
    pca1    pca2
0   -2.684126   0.319397
1   -2.714142   -0.177001
2   -2.888991   -0.144949
3   -2.745343   -0.318299
4   -2.728717   0.326755
'''
```

5. 주성분 산포도 확인

   주성분 데이터 프레임의 산포도를 다시 확인하면 원본 데이터프레임으로 그린 산포도보다 종속변수를 더 잘 설명하는 산포도를 확인할 수 있다.

```
import matplotlib.pyplot as plt
import seaborn as sns
principal_iris['class'] = load_iris().target
principal_iris['class'] = principal_iris['class'] .map({0:'Setosa',
1:'Versicolour', 2:'Virginica'})

plt.title("2 component PCA")
sns.scatterplot(x='pca1',y='pca2',hue='class', data = principal_iris)
plt.show()
```

데이터 불균형 문제

언더 샘플링(Under Sampling)

다수의 레이블을 가진 데이터를 샘플링하여 소수의 데이터세트가 가진 레이블의 수 수준으로 감소시키는 기법
전체 데이터의 수가 급격하게 줄어들어 오히려 학습 성능을 떨어뜨릴 수 있음.

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from collections import Counter
from imblearn.under_sampling import RandomUnderSampler

#95:1의 불균형 데이터 생성
x,y = make_classification(n_samples=2000,n_features=6,weights=[0.95],flip_y=0)
print(Counter(y))

#Random Under Sampling
underSample = RandomUnderSampler(sampling_strategy='majority')
x_under, y_under = underSample.fit_resample(x,y)
print(Counter(y_under))
```

```
#비율을 0.5로 지정하면 1:0.5의 비율로 undersampling
#Random Under Sampling은 다수를 차지하는 레이블에서 무작위로 데이터를 제거하는 방법
underSample=RandomUnderSampler(sampling_strategy=0.5)
x_under2,y_under2 = underSample.fit_resample(x,y)
print(Counter(y_under2))

'''
Counter({0: 1900, 1: 100})
Counter({0: 100, 1: 100})
Counter({0: 200, 1: 100})
'''
```

오버 샘플링(Over Sampling)

소수의 레이블을 지닌 데이터세트를 다수 레이블을 지난 데이터세트의 수만큼 증식시켜 학습에 사용하기 위한 충분한 양과 비율의 데이터를 확보하는 기법, 데이터의 손실이 없어 일반적으로 언더 샘플링보다 성능이 유리하여 주로 사용됨.

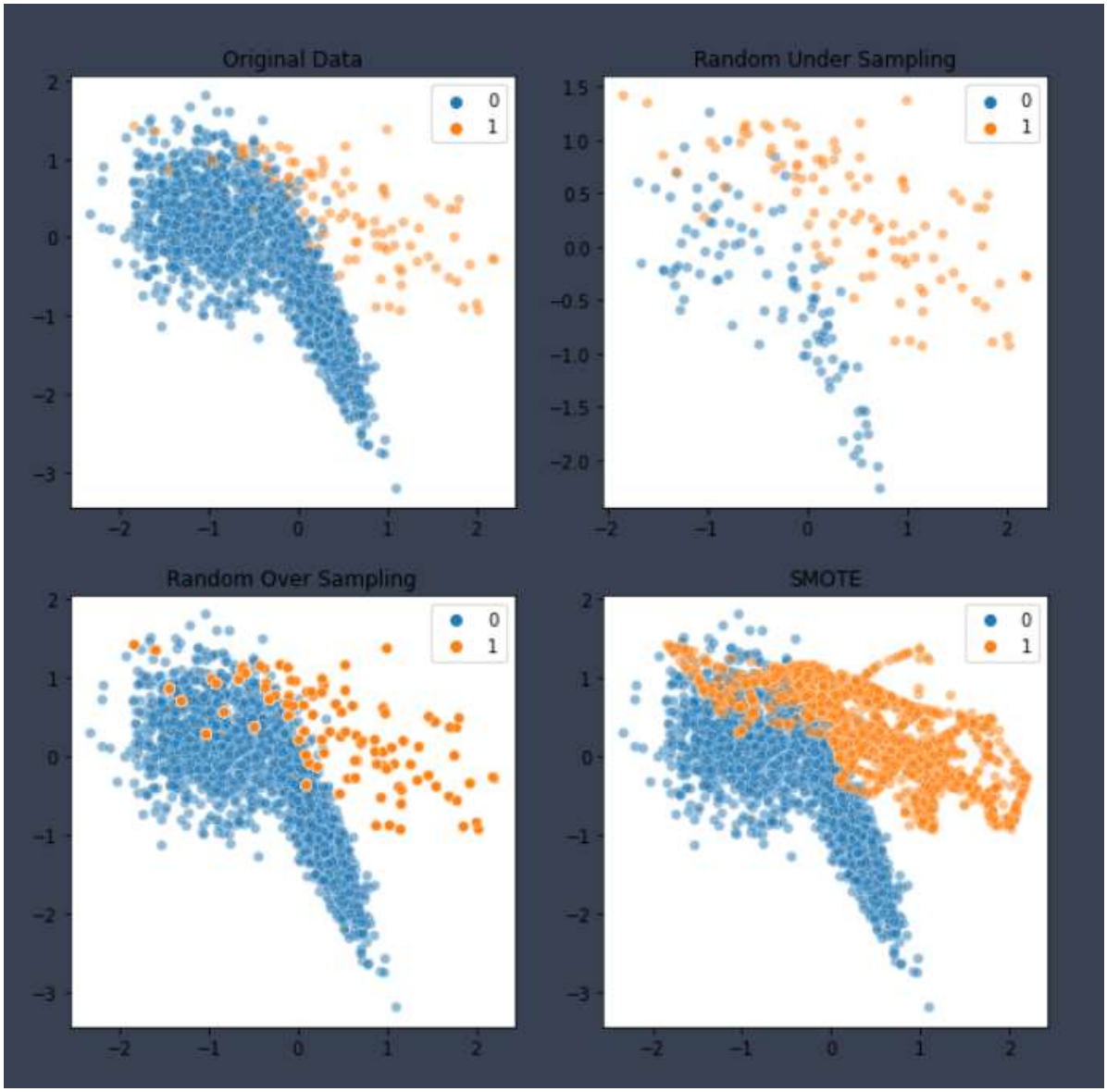
Random Over Sampling
소수의 레이블을 지난 데이터세트를 단순 복제하여 다수의 레이블과 비율을 맞추는 방법

```
from imblearn.over_sampling import RandomOverSampler
oversample = RandomOverSampler(sampling_strategy=0.5)
x_over, y_over = oversample.fit_resample(x,y)
print(Counter(y_over))

oversample = RandomOverSampler(sampling_strategy = 'minority')
x_over, y_over = oversample.fit_resample(x,y)
print(Counter(y_over))

'''
Counter({0: 1900, 1: 950})
Counter({0: 1900, 1: 1900})
'''
```

SMOTE
소수 레이블을 지난 데이터 세트의 관측 값에 대한 K개의 최근접 이웃을 찾고, 관측 값과 이웃으로 선택도니 값 사이에 임의의 새로운 데이터를 생성하는 방법으로 샘플의 수를 늘리는 방법

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from collections import Counter
from imblearn.under_sampling import RandomUnderSampler

#95:1의 불균형 데이터 생성
x,y = make_classification(n_samples=2000,n_features=6,weights=[0.95],flip_y=0)
print(Counter(y))
```

```
#Random Under Sampling
underSample = RandomUnderSampler(sampling_strategy='majority')
x_under, y_under = underSample.fit_resample(x,y)
print(Counter(y_under))

underSample=RandomUnderSampler(sampling_strategy=0.5)
x_under2,y_under2 = underSample.fit_resample(x,y)
print(Counter(y_under2))


from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE

oversample = RandomOverSampler(sampling_strategy=0.5)
x_over, y_over = oversample.fit_resample(x,y)
print(Counter(y_over))

oversample = RandomOverSampler(sampling_strategy = 'minority')
x_over, y_over = oversample.fit_resample(x,y)
print(Counter(y_over))

smote_sample = SMOTE(sampling_strategy='minority')
x_sm, y_sm = smote_sample.fit_resample(x,y)
print(Counter(y_sm))

from matplotlib import pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(nrows=2,ncols=2,figsize=(10,10))

sns.scatterplot(x[:,1], x[:,2],hue=y,ax=axes[0][0],alpha=0.5)
sns.scatterplot(x_under[:,1], x_under[:,2],hue=y_under,ax=axes[0][1],alpha=0.5)
sns.scatterplot(x_over[:,1], x_over[:,2],hue=y_over,ax=axes[1][0],alpha=0.5)
sns.scatterplot(x_sm[:,1], x_sm[:,2],hue=y_sm,ax=axes[1][1],alpha=0.5)

axes[0][0].set_title('Original Data')
axes[0][1].set_title('Random Under Sampling')
axes[1][0].set_title('Random Over Sampling')
axes[1][1].set_title('SMOTE')
plt.show()
```


Original Data
Random Under Sampling
Random Over Sampling
SMOTE
0
1