미래에셋생명 하반기 빅데이터 분석 전문가 레벨 3

데이터 분석을 위한 고급 전처리

박상수 강사

2019.11.19
2주차 강의

목차

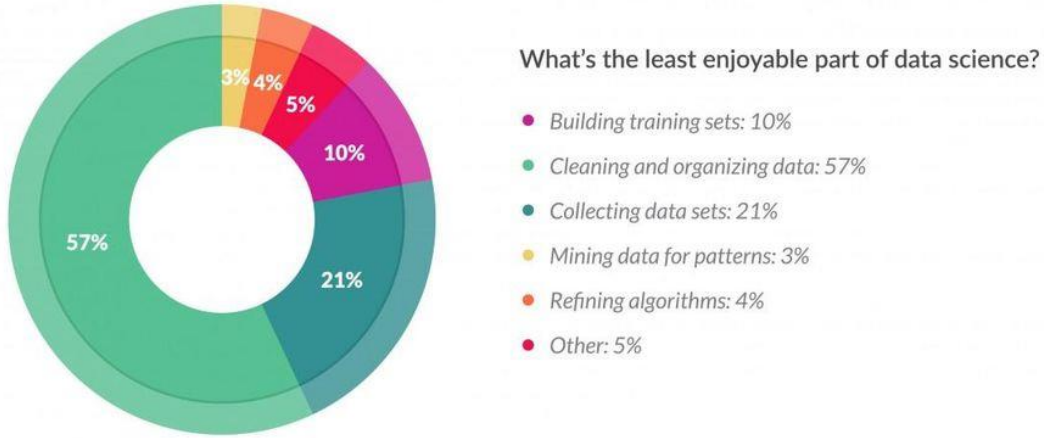
- 빅데이터의 전처리
- 데이터 전처리 (정제, 통합, 정리, 변환)
- 데이터 전처리 예제
- 프로젝트 논

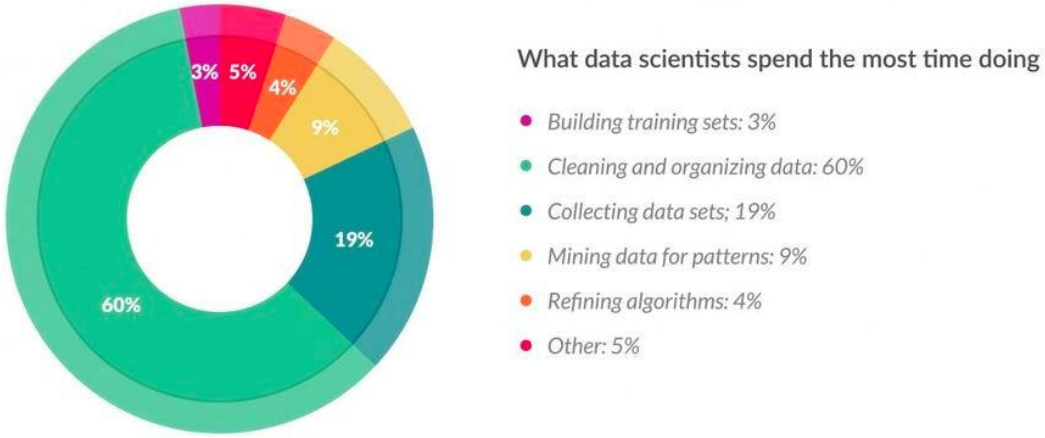
빅데이터의 전처리

전처리는 무엇이고, **왜 필요**할까 ?

데이터 전처리 (Data pre-processing) 특징

- **데이터를 분석하는 모든 프로젝트에서 반드시 필요한 과정**
  - 데이터 분석을 하는 사람이 가장 싫어하는 과정
  - 분석가가 **80%의 시간을 데이터 수집 및 전처리에 사용**[1]
  - 전처리 전의 데이터 셋은 **불필요한 정보**들이 포함되어 있음
  - **데이터가 분석이 가능한 상태로 만드는 과정**
    - **Pandas**: 정제된 데이터를 다루는 방법
    - **전처리**: 날것 (Raw)의 데이터를 분석을 위한 데이터로 만드는 과정

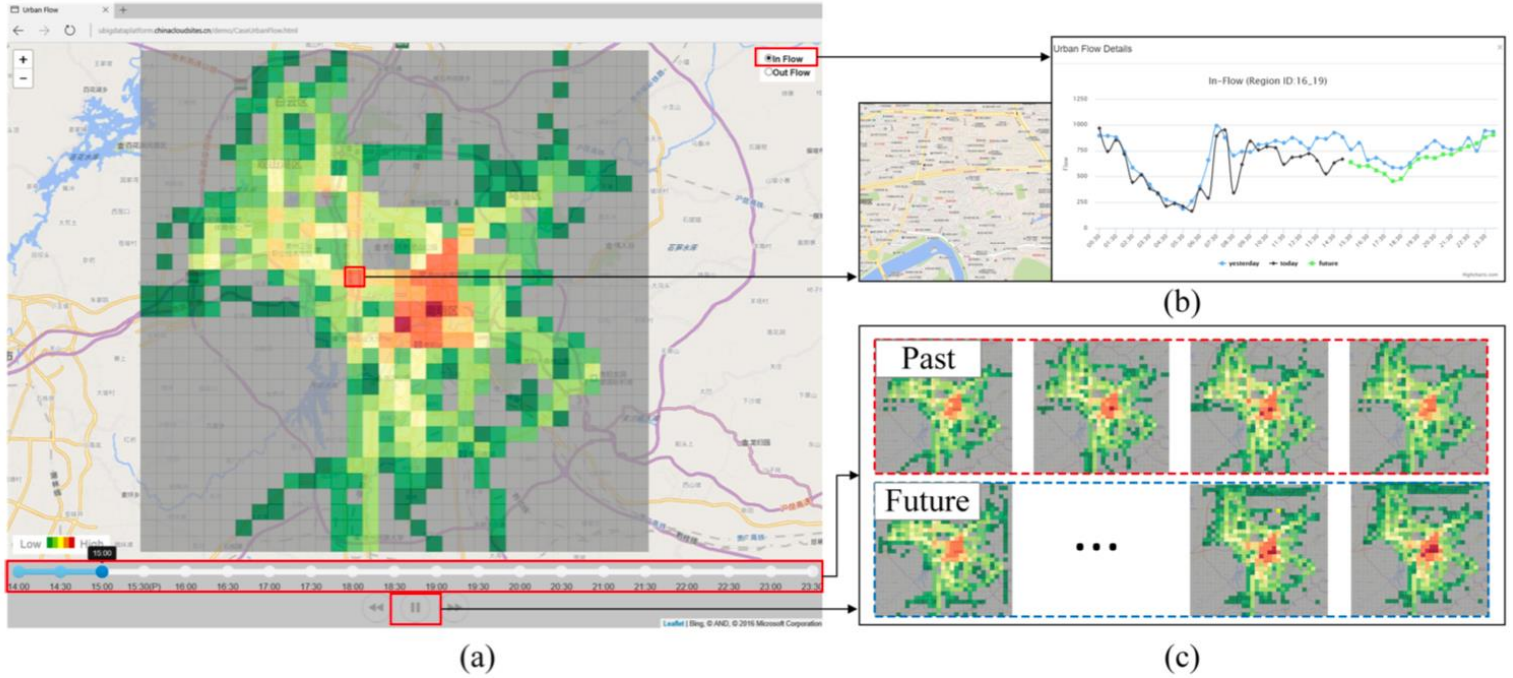




데이터 전처리 (Data pre-processing) 사례

카카오의 택시 수요 예측 모델[2]

- 우버, 디디, 리프트, 카카오 택시와 같은 서비스에서 **수요를 예측하는 모델**을 연구
- 하루에 수십만 건의 **사용자 이용 데이터 (시공간 정보)**가 축적되고 있음
- 모든 시공간 정보를 예측에 사용하는 것은 현실적으로 불가능
  - 지도 데이터를 **grid (일정한 간격)**로 나눠, 수요가 변하는 grid만 사용
  - 데이터를 그대로 사용할 경우, **원본 데이터는 100MB**이지만, **전처리의 결과는 수백 GB**



데이터 전처리 (Data pre-processing) 종류

데이터 전처리의 주요 작업

- **데이터 정제 (Cleaning)**
  - 결측값 (Missing value)를 채우거나, 잡음값 (Noisy data) 완화
- **데이터 통합 (Integration)**
  - 다양한 곳에서 얻은 데이터를 합쳐서 표현
- **데이터 정리 (Reduction)**
  - 크기는 더 작지만, 분석 결과는 동일한 결과로 표현
- **데이터 변환 (Transformation)**
  - 데이터 마이닝의 효과를 극대화 하기 위한 변형

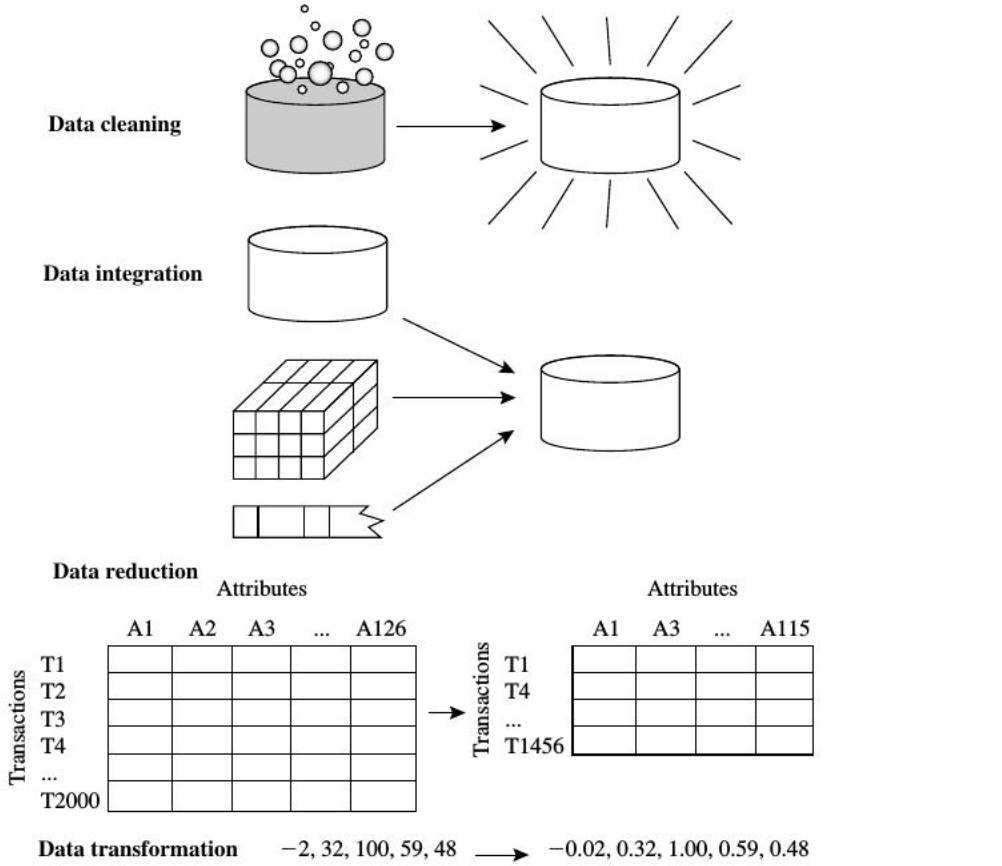


Figure 3.1 Forms of data preprocessing.

데이터 전처리 (Data pre-processing) 성능 지표

사용 목적에 따른 데이터 품질

- **정밀성 (Accuracy)**
  - 오류나 예상 치에서 벗어나는 값이 없음을 의미
- **완전성 (Completeness)**
  - 속성의 값이나 관심있는 속성이 모두 존재함을 의미
- **일관성 (Consistency)**
  - 값의 모순이 없음을 의미
- **해석성 (Accuracy)**
  - 데이터를 이해하기가 얼마나 쉬운지를 의미

데이터 전처리 (Data pre-processing) 예시 #1

데이터의 정제

- **정제되지 않은 데이터는 결측값을 포함**
  - 결측값: 데이터가 없는 값 (null)
- **is.null () 함수**
  - 값이 있으면 False, 없으면 True를 반환
- **dropna () 함수**
  - 하나라도 결측 값이 있으면 열/행을 기준으로 모두 제거 (how='any')
  - 열/행이 모두 결측값일때 해당 열/행을 제거 (how='all')
  - 행을 기준 (axis=0), 열을 기준으로는 (axis=1)

```
1    import numpy as np
2    import pandas as pd
3    import matplotlib.pyplot as plt
4    import seaborn as sns
5    color = sns.color_palette()
6    sns.set_style('darkgrid')
7
8    df=pd.read_csv('./Data USA Cart.csv')
9    df.apply(lambda x: any(          ))
10   df_2=df.      (      , how='any')
```

데이터 품질과 정제

데이터 분석의 첫 걸음, 데이터 정제 !

데이터 품질 개요

데이터 품질과 관련된 예시

- 데이터 분석에서 **데이터 품질**과 관련되어 발생할 수 있는 문제
- 만일 **다른 학교와의 비교**를 통해 B학교의 문제를 파악하지 못했다면
- B학교 학생들의 비만도가 다른 학교에 비해 높다라는 **잘못된 분석 결과**를 내놓을 수 있음
  - 빅데이터 분석의 시작은 **데이터의 이해**로부터 시작
  - 그 처음 단계로서 **데이터의 품질에 관한 분석**이 선행되어야 함

A연구소는 중학교 학생들의 체중 분포를 조사하기 위해 **각 학교로부터 학생들의 체중 자료**를 제공받았다. 그리고 분석을 수행하기 전 각 학교별 평균을 살펴보았는데 유독 **B학교**만 **평균보다 5kg 가까이 차이**가 남을 확인하였다. 이를 이상히 여겨 **B학교**에 문의를 한 결과 **해당 학교 체중계에 문제가 있음을 확인**할 수 있었다.

데이터 품질 문제 정의

데이터 품질 문제

- 데이터 품질 문제는 **측정과 수집하는 과정**에서 발생
  - **측정 오류**: 측정 과정에서 발생하는 문제, 기록된 값과 참값과의 수치적 차이
  - **수집 오류**: 데이터 객체나 속성 값을 제외하여 데이터를 부적절하게 포함하는 오류
- **데이터 품질 문제를 해결**하기 위해서는
  - 데이터 품질 문제의 검출과 수정 (데이터 정제)
  - 낮은 데이터 품질을 감내할 수 있는 알고리즘 사용

데이터 품질 문제 (데이터 정제)

데이터 정제와 관련된 데이터 품질 문제

- **데이터 정제**: 결측값을 채우거나, 잡음값 완화, 이상점 발견 및 제거 등 **데이터 자체에 대한 신뢰도를 높이는 작업**
- **결측값**: 값이 기재되어 있지 않은 속성이나 데이터
  - 해결방법: 해당 데이터 개체 또는 속성 **제거** 또는 결측값 **추정/무시**
- **잡음값**: 측정된 값에서 임의의 오류나 변화가 발생하는 것
  - **데이터 평활화**: 근접한 다른 값을 참고하여 정렬한 데이터 값을 평활화
- **이상점**: 관측된 데이터의 범위에서 많이 벗어난 아주 작은 값이나 아주 큰 값
  - 유사한 값으로 그룹을 묶는 방법인 군집화로 이상치를 찾아낼 수 있음

데이터 품질 (데이터 정제) 예시 #1

데이터 병합과정에서 발생하는 결측값

- Pandas의 **DataFrame**을 사용하여 분석을 하다 보면 **결측값 (Missing value)**를 해결 하는 것이 어려움
  - 데이터 수집 혹은 측정, 다수의 DataFrame을 병합하는 과정, index를 재 설정하는 과정에서 발생
- 결측값은 분석 오류를 발생시키거나, 왜곡 시킬 수 있음
- 따라서 분석할 **DataFrame**을 생성했으면 **결측값의 여부를 확인**해야 함
- **Pandas.merge (df1, df2, how='', on='') 함수**
  - df1/df2: (DataFrame), on (공통적인 열 이름, merge의 기준), how (inner/outer, left/right)

```
1   import pandas as pd
2   from pandas import DataFrame
3
4   df_left = DataFrame({'KEY': ['K0', 'K1', 'K2', 'K3'],
5    'A': ['A0', 'A1', 'A2', 'A3'],
6    'B': [0.5, 2.2, 3.6, 0.4]})
7
8   df_right = DataFrame({'KEY': ['K2', 'K3', 'K4', 'K5'],
9   'C': ['C2', 'C3', 'C4', 'C5'],
10  'D': ['D2', 'D3', 'D4', 'D5']})
11
12  df_all = pd.merge(                                    )
13  df_all
```

	KEY	A	B	C	D
0	K0	A0	0.5	NaN	NaN
1	K1	A1	2.2	NaN	NaN
2	K2	A2	3.6	C2	D2
3	K3	A3	0.4	C3	D3
4	K4	NaN	NaN	C4	D4
5	K5	NaN	NaN	C5	D5

데이터 품질 (데이터 정제) 예시 #2

결측값의 확인

- DataFrame의 **결측값 확인 함수**
  - pandas.isnull ()/isnull (df), pandas.notnull/notnull (df)
  - **Isnull ()**은 관측치가 결측이면 True, 아니면 False
  - **notnull ()**은 **isnull ()**과 반대의 의미

	KEY	A	B	C	D
0	False	False	False	True	True
1	False	False	False	True	True
2	False	False	False	False	False
3	False	False	False	False	False
4	False	True	True	False	False
5	False	True	True	False	False

```
15    pd.isnull(df_all)
16
17
18    df_all.notnull()
```

데이터 품질 (데이터 정제) 예시 #3

결측값의 갯수 확인

- **행/열 단위**로 결측값의 개수를 확인하기 위해서는
  - **열단위**: df.isnull().sum()
  - **행단위**: df.isnull.sum(1)
- DataFrame**의 새로운 열 추가**
  - DataFrame['**새로운 열 이름**']=cell **내용**

```
KEY     0
A       2
B       2
C       2
D       2
NaN_
dtyp
```

	KEY	A	B	C	D	NotNull_cnt	NaN_cnt
0	K0	A0	0.5	NaN	NaN	3	2
1	K1	A1	2.2	NaN	NaN	3	2
2	K2	A2	3.6	C2	D2	5	0
3	K3	A3	0.4	C3	D3	5	0
						3	2
						3	2

```
20    df_all.isnull().sum()
21    df_all.isnull().sum(1)
22    df_all[              ]
23    df_all[         ]
```

데이터 품질 문제 (데이터 정제)

- **결측값 문제를 해결하는 방법 방법**
  - **결측값의 위치와 갯수를 확인**하고, **채우는 과정/제거하는 과정**을 통해 **데이터 전처리** 진행
  - **비슷한 값으로 채우는 방법** 또는 결측값을 제거 하는 방법으로 데이터 전처리
  - **결측값을 채우는 방법**
    - 특정 값, 앞 방향 혹은 뒷 방향의 값,을 변수별 평균으로 대처하는 방법을 사용
  - **결측값을 제거하는 방법**
    - 결측값이 포함된 특정 속성을 지우거나, 연관된 모든 속성을 제거

데이터 품질 (데이터 정제) 예시 #4

결측값 생성 및 특정 값으로 채우기

- Padnas.loc [행, 열]
  - 특정 행과 열의 데이터를 추출하거나 수정할 때 사용
- **Pandas.fillna (value) 함수**
  - 결측값을 특정 값으로 채우는 방법
  - Value에 입력된 숫자 값으로 결측 값을 변환

```
1   import pandas as pd
2   import numpy as np
3
4   df = pd.DataFrame(np.random.randn(5, 3),
5   columns=['C1', 'C2', 'C3'])
6   df
7
8                           = np.nan
9   df
10
11  df_0 =
12  df_0
```

	C1	C2	C3
0	0.100850	0.809966	0.718741
1	NaN	-0.303963	NaN
2	0.728262	1.457807	-0.425956
3	-0.580532	1.414201	-0.237947
4	-0.591523	-0.208767	0.607579

	C1	C2	C3
0	0.100850	0.809966	0.718741
1	0.000000	-0.303963	0.000000
2	0.728262	1.457807	-0.425956
3	-0.580532	1.414201	-0.237947
4	-0.591523	-0.208767	0.607579

데이터 품질 (데이터 정제) 예시 #5

결측값을 앞/뒤 값으로 채우기

- 결측값을 **앞 방향**으로 채우려면
  - fillna(method='**ffill**')혹은 fillna (method='**pad**')를 사용
- 결측값을 **뒤 방향**으로 채우려면
  - fillna(method='**bfill**')혹은 fillna (method='**backfill**')를 사용

	C1	C2	C3
0	0.100850	0.809966	0.718741
1	NaN	-0.303963	NaN
2	0.728262	1.457807	-0.425956
3	-0.580532	1.414201	-0.237947
4	-0.591523	-0.208767	0.607579

	C1	C2	C3
0	0.100850	0.809966	0.718741
1	0.100850	-0.303963	0.718741
2	0.728262	1.457807	-0.425956
3	-0.580532	1.414201	-0.237947
4	-0.591523	-0.208767	0.607579

	C1	C2	C3
0	0.100850	0.809966	0.718741
1	0.728262	-0.303963	-0.425956
2	0.728262	1.457807	-0.425956
3	-0.580532	1.414201	-0.237947
4	-0.591523	-0.208767	0.607579

```
14    df_0 = df.fillna(                )
15    df_0
16
17    df_0 = df.fillna(                )
18    df_0
```

데이터 품질 (데이터 정제) 예시 #6

결측값을 평균 값으로 채우기

- **평균값**을 계산하는 함수
  - pandas.mean()를 사용하여 **pandas.fillna(pandas.mean())**
  - 결측값의 위치를 파악하여 채우는 것도 가능함: **padnas.where**(pd.notnull(df), df.mean(), axis=**'columns**')
  - where()함수는 **위치를 판별하는 기준** (notnull), **결측값을 채우는 방법 또는 숫자** (mean), **계산 기준** (axis)으로 입력

	C1	C2	C3
0	0.100850	0.809966	0.718741
1	NaN	-0.303963	NaN
2	0.728262	1.457807	-0.425956
3	NaN	NaN	NaN
4	-0.591523	-0.208767	0.607579

	C1	C2	C3
0	0.100850	0.809966	0.718741
1	0.079196	-0.303963	0.300121
2	0.728262	1.457807	-0.425956
3	0.079196	0.438761	0.300121
4	-0.591523	-0.208767	0.6

```
20    df.loc[1, ['C1', 'C3']] = np.nan
21    df.loc[3, ['C1', 'C2','C3']] = np.nan
22    df
23
24    df.mean()
25    df.fillna(df.mean())
26    df.where(                                    )
```

```
C1    0.079196
C2    0.438761
C3    0.300121
dtype: float64
```

데이터 품질 (데이터 정제) 예시 #7

결측값을 제거하는 방법

결측값이 포함되어 있는 행/열 전체 삭제

- pandas.dropna()를 사용하여 **결측값이 포함되어 있는 행 또는 열 삭제**
- **행: pandas.dropna(), 열: pandas.dropna(axis=1)**

	C1	C2	C3
0	NaN	-2.243674	1.430261
1	NaN	-0.964276	NaN
2	-0.631549	-0.966691	-1.193602
3		1.148105	0.118415
4		0.686199	NaN

	C2
0	-2.243674
1	-0.964276
2	-0.966691
3	1.148105
4	0.686199

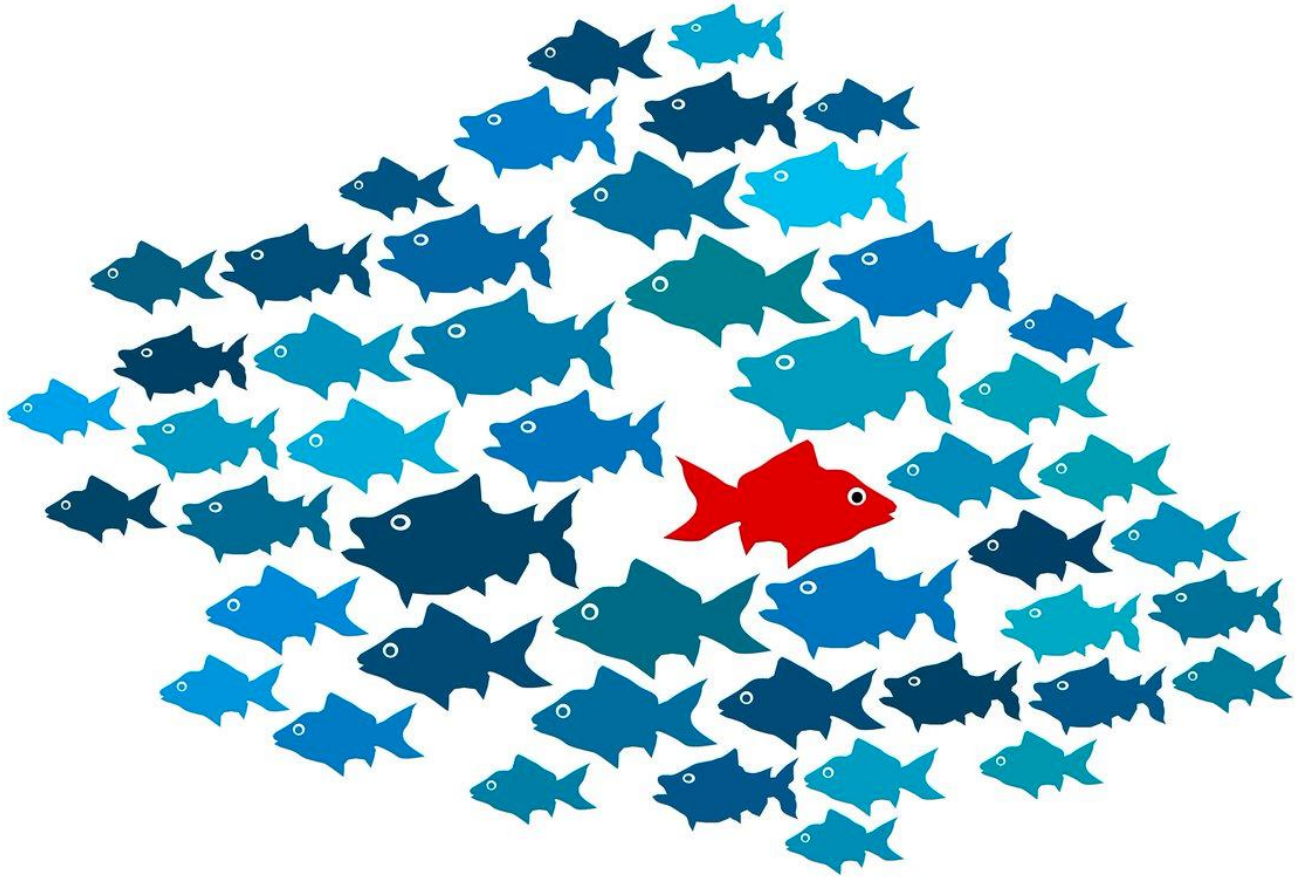
	C1	C2	C3
2	-0.631549	-0.966691	-1.193602
3	-1.025338	1.148105	0.118415

```
 8   df.loc[1, ['C1', 'C3']] = np.nan
 9   df.loc[4, ['C1', 'C3']] = np.nan
10   df.loc[0, ['C1']] = np.nan
11   df
12
13   df_dop_row =
14   df_dop_row
15
16   df_dop_col =
17   df_dop_col
```

데이터 품질 (데이터 정제)

이상한 데이터 (이상치)

- 이상치는 **정상 범주에서 크게 벗어난 값**을 의미
  - 데이터 수집 과정에서 오류가 발생
  - 분석 결과가 왜곡되는 문제점의 원인



데이터 품질 (데이터 정제) 예시 #8

- **보스턴 집값 데이터를 사용한 이상치 예측 (그래프)**
  - **데이터명: Boston Houtsing Price (보스턴 주택 가격 데이터)**
  - **레코드수 (506개), 필드 개수 (14개)**
    - 여러 개의 측정지표 (범죄율, 학생/교사 비율)
    - 14개의 필드는 입력 변수로 사용

[01] CRIM	자치시 (town) 별 1인당 범죄율
[02] ZN	25,000 평방피트를 초과하는 거주지역의 비율
[03] INDUS	비소매상업지역이 점유하고 있는 토지의 비율
[04] CHAS	찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
[05] NOX	10ppm 당 농축 일산화질소
[06] RM	주택 1가구당 평균 방의 개수
[07] AGE	1940년 이전에 건축된 소유주택의 비율
[08] DIS	5개의 보스턴 직업센터까지의 접근성 지수
[09] RAD	방사형 도로까지의 접근성 지수
[10] TAX	10,000 달러 당 재산세율
[11] PTRATIO	자치시 (town)별 학생/교사 비율
[12] B	1000(Bk-0.63)^2, 여기서 Bk는 자치시별 흑인의 비율을 말함.
[13] LSTAT	모집단의 하위계층의 비율(%)
[14] MEDV	본인 소유의 주택가격(중앙값) (단위: $1,000)

데이터 품질 (데이터 정제) 예시 #8

- **보스턴 집값 데이터를 사용한 이상치 예측 (그래프)**
  - **해당 데이터는 딕셔너리 형태로 구성**
    - 딕셔너리: 대응 관계로 구별하는 파이썬의 자료형
    - 예) "이름"="홍길동", "생일"="몇 월 몇 일"
    - 해당 데이터는 data, target, feature_names, DESCR의 딕셔너리 형태가 존재
    - 딕셔너리의 대응 관계를 확인하기 위해서는 **dictionary 객체.keys() 함수 사용**

```
1   import numpy as np
2   import pandas as pd
3   from sklearn.datasets import load_boston
4
5   #Load the data
6   boston = load_boston()
7
8   #Find features and target
9   x = boston.data
10  y = boston.target
11
12  #Find the dic keys
13  print(              )
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR'])
```

데이터 품질 (데이터 정제) 예시 #8

- **보스턴 집값 데이터를 사용한 이상치 예측 (그래프)**
  - **해당 데이터는 딕셔너리 형태로 구성**
    - 딕셔너리: 대응 관계로 구별하는 파이썬의 자료형
    - 예) "이름"="홍길동", "생일"="몇 월 몇 일"
    - 해당 데이터는 data, target, feature_names, DESCR의 딕셔너리 형태가 존재

```
Boston House Prices dataset
===========================

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 19
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

```
15    columns =
16    columns
17
18    print(              )
```

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

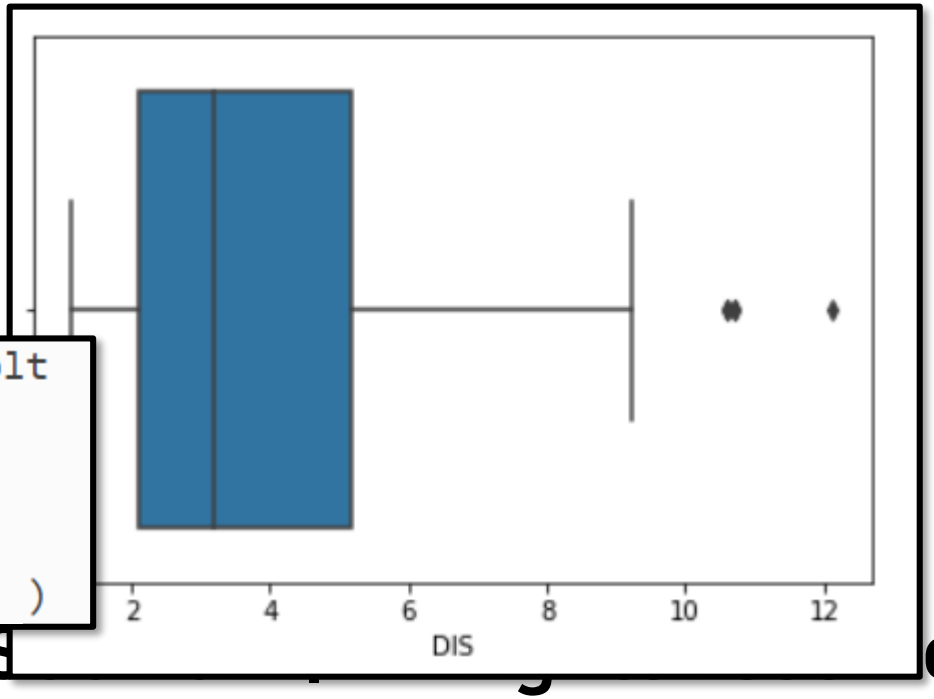
데이터 품질 (데이터 정제) 예시 #8

보스턴 집값 데이터를 사용한 이상치 예측 (그래프)

Boxplot를 활용하여 데이터의 분포 확인

- Boxplot (상자수염 그림): 통계학에서 수치적 자료를 표현하는 그래프
- 이 방법에서는 자료에서 얻는 **다섯 수치 요약**을 사용하여 그래프를 작성
- 최소값 (Q1에서 1.5 IQR을 뺀 위치), 제1~3사 분위 (Q1, Q2, Q2), 최대값 (Q3에서 1.5 IQR을 더한 위치)
- **IQR (Interquartile range, Q1~Q3의 값)**
- **Seaborn.boxplot** (x='x축 기준값')를 사용하여 그래프 작성

```
20    import matplotlib.pyplot as plt
21    import seaborn as sns
22
23    %matplotlib inline
24    sns.boxplot(x=                )
```



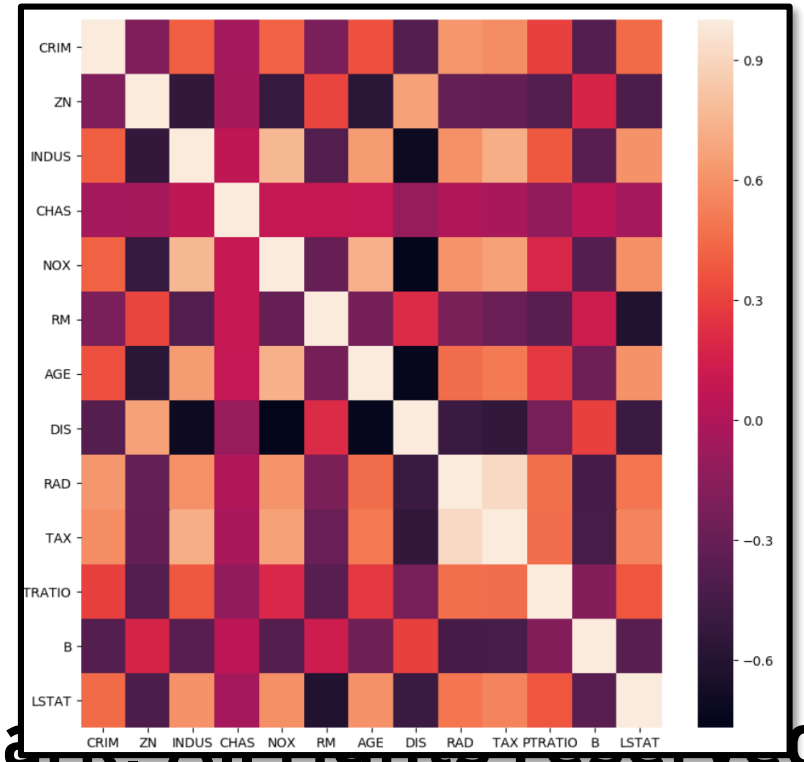
데이터 품질 (데이터 정제) 예시 #8

보스턴 집값 데이터를 사용한 이상치 예측 (그래프)

Headmap를 활용하여 데이터의 상관 관계 확인

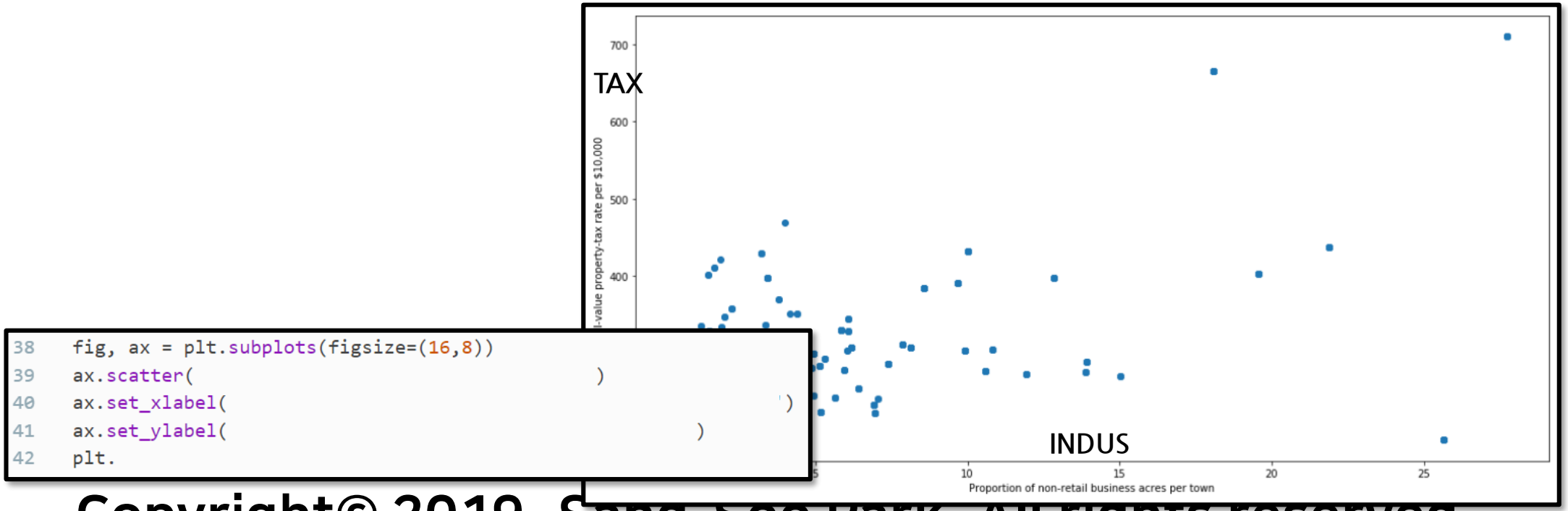
- Headmap: 열을 뜻하는 Heat와 지도를 뜻하는 Map을 결합한 단어
- 색상으로 표현할 수 있는 다양한 정보를 일정한 이미지 위에 열분포 형태로 출력하는 그래프더한 위치)
- **Seaborn.headmap** (상관관계 데이터)를 사용하여 그래프 작성
- 상관관계 데이터: **data.corr()** 함수를 사용

```
31    import matplotlib.pyplot as plt
32    import seaborn as sns
33    %matplotlib inline
34
35    plt.figure(figsize= (10,10), dpi=100)
36    sns.heatmap(                    )
```



데이터 품질 (데이터 정제) 예시 #8

- **보스턴 집값 데이터를 사용한 이상치 예측 (그래프)**
  - **산점도를 활용하여 데이터의 이상치 관측**
    - **Matplotlib.scatter** (x축 데이터, y축 데이터)
    - **Matplotlib.set_xlabel/set_ylabel** (라벨 데이터)
    - 그래프를 그리기 위해서는 **matplotlib.show ()**

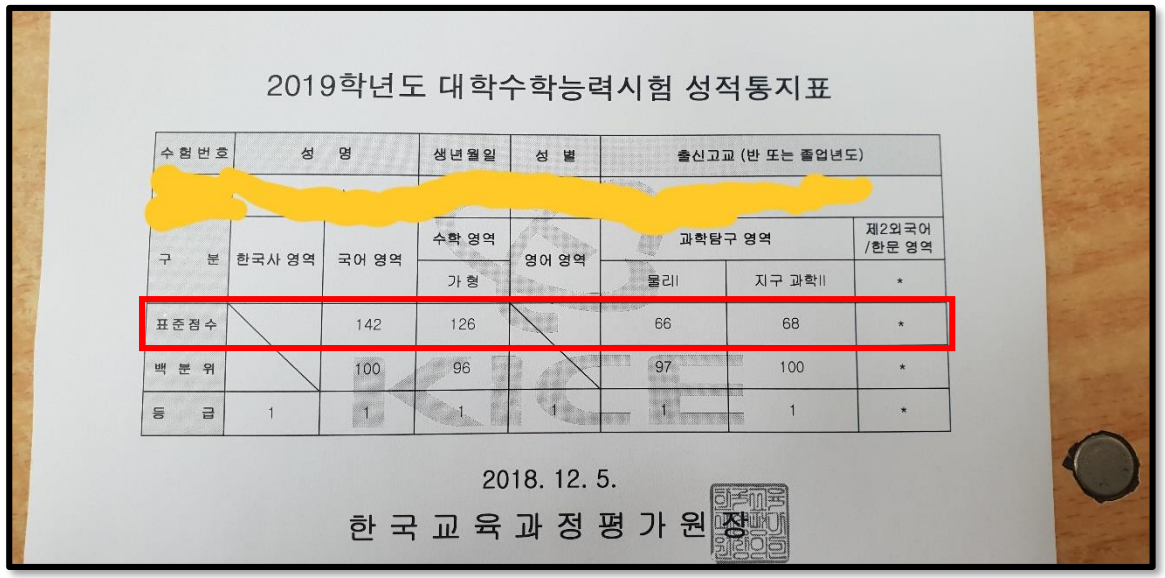



```
38    fig, ax = plt.subplots(figsize=(16,8))
39    ax.scatter(                                    )
40    ax.set_xlabel(                                  )
41    ax.set_ylabel(                                )
42    plt.
```

데이터 품질 (데이터 정제)

■ 이상치를 수치적으로 파악하는 방법

- 이상치 파악을 위해서 **일변량**, **이변량**, **다변량** 기법을 사용
  - **일변량: 표준점수 (Z-score)가 2.5~4.0 이상인 표본 (기준은 표본에 따라 상이함)**
  - 표준점수는 상대적 서열, 정해진 표준과 편차를 갖도록 변환한 분포에서 **원점수가 해당하는 위치를 나태는 점수**
  - **이변량:** 산포도를 이용하여 독립변수와 종속변수의 관계성 테스트, 특정 신뢰도 구간에 포함되지 않는 표본

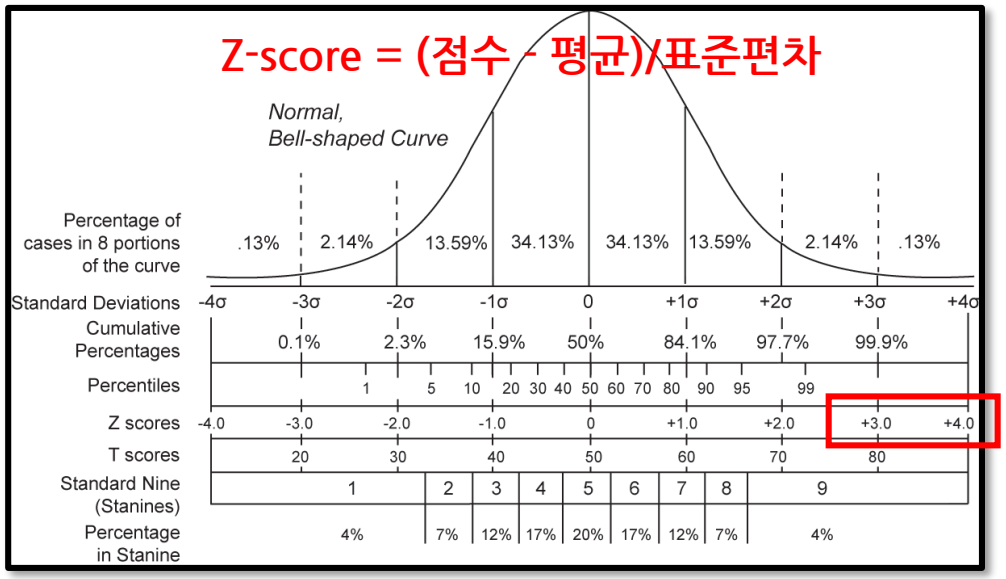


2019학년도 대학수학능력시험 성적통지표

수험번호	성 명		생년월일	성 별	출신고교 (반 또는 졸업년도)		
구 분	한국사 영역	국어 영역	수학 영역 가형	영어 영역	과학탐구 영역 물리Ⅰ	과학탐구 영역 지구 과학Ⅱ	제2외국어/한문 영역
표준점수		142	126		66	68	*
백 분 위		100	96		97	100	*
등 급	1	1	1	1	1	1	*

2018. 12. 5.

한 국 교 육 과 정 평 가 원 장



데이터 품질 (데이터 정제) 예시 #8

■ 이상치를 수치적으로 파악하는 방법

- **Numpy**와 **일변량** 방법을 사용하는 **이상치 파악**
  - 일변량에서는 표준점수를 사용, 따라서 표준점수로 변환 필요
  - Scipy에서는 표준점수로 변환하는 함수 제공: **scipy.zscore (data)**

```
[[0.41771335 0.28482986 1.2879095  ... 1.45900038 0.44105193 1.0755623 ]
 [0.41526932 0.48772236 0.59338101 ... 0.30309415 0.44105193 0.49243937]
 [0.41527165 0.48772236 0.59338101 ... 0.30309415 0.39642699 1.2087274 ]
 ...
 [0.41137448 0.48772236 0.11573841 ... 1.17646583 0.44105193 0.98304761]
 [                                  ... 1.17646583 0.4032249  0.86530163]
 [                                  ... 1.17646583 0.44105193 0.66905833]]
```

```
44    from scipy import stats
45    import numpy as np
46    z = np.abs(                    )
47    print(z)
```

데이터 품질 (데이터 정제) 예시 #8

이상치를 수치적으로 파악하는 방법

- Numpy와 **일변량** 방법을 사용하는 **이상치 파악**
  - **표준점수의 값이 3보다 클 경우** 이상치라고 가정할 때, 이때의 **임계값 (threshold)**는 3
  - **Numpy.where (조건)**을 사용하여 이상치 파악 가능

```
(array([ 55,  56,  57, 102, 141, 142, 152, 154, 155, 160, 162, 163, 199,
       200, 201, 202, 203, 204, 208, 209, 210, 211, 212, 216, 218, 219,
       220, 221, 222, 225, 234, 236, 256, 257, 262, 269, 273, 274, 276,
       277, 282, 283, 283, 284, 347, 351, 352, 353, 353, 354, 355, 356,
       357, 358, 363, 364, 364, 365, 367, 369, 370, 372, 373, 374, 374,
       380, 398, 404, 405, 406, 410, 410, 411, 412, 412, 414, 414, 415,
       416, 418, 418, 419, 423, 424, 425, 426, 427, 427, 429, 431, 436,
       , 456, 457, 466], dtype=int64), array([ 1,  1,  1, 11, 12,  3,  3,  3,

 3,  3,  3,  3,  3,  5,  3,  3,  1,  5,
 1,  3,  1,  1,  7,  7,  1,  7,  7,  7,
 5,  3,  3,  3, 12,  5, 12,  0,  0,  0,
 0, 12, 11, 11,  0, 11, 11, 11, 11, 11,
11, 11, 11, 11, 11, 11, 11, 11],
```

```
44  from scipy import stats
45  import numpy as np
46  z = np.abs(stats.zscore(boston_df))
47  print(z)
48
49  threshold = 3
50  print(np.where(      )
```

데이터 품질 (데이터 정제) 예시 #8

- **이상치 제거: 표준 점수의 값이 3 이상인 경우 이상치**
  - **이상치를 제외한 구간 계산**
    - Q1, Q3와 IQR (Q3-Q1)의 값을 계산하고, 이를 바탕으로 이상치 제거
    - **Z값이 3보다 작은 것은 표준값이 ±3 아래로 있으며, 99.9% 신뢰구간 내에 있는 것을 의미**
  - **Numpy.quantilze** (percent): 백분위수를 구하는 함수
    - 이상치 제거 전후의 데이터 셋 크기의 변화는 ?
    - **Q1과 Q3를 계산하지 않고, 쉽게 하는 방법:** data = data[data['컬럼이름'].between(-3,3)]

```
59   boston_df_o = boston_df_o[(z < 3).all(axis=1)]
60   boston_df.shape
61
62   boston_df_o.shape
63   boston_df_o1 = boston_df
64
65   Q1 = boston_df_o1.
66   Q3 = boston_df_o1.
67   IQR =
68   print(IQR)
69
70   boston_df_out = boston_df_o1[~((boston_df_o1 < (Q1 - 1.5 * IQR)) |(boston_df_o1 > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
boston_df_out =
```

데이터 품질 (데이터 정제)

표준화 (Standardization)와 정규화 (Normalization)

- **데이터 표준화**를 통해 이상치를 제거하고, **정규화**를 통해 상대적 크기에 대한 영향력 감소

	표준화(standardization)	정규화(normalization)
공통점	데이터 rescaling	
정의 &목적	데이터가 평균으로부터 얼마나 떨어져있는지 나타내는 값으로, 특정 범위를 벗어난 데이터는 outlier로 간주, 제거	데이터의 상대적 크기에 대한 영향을 줄이기 위해 데이터범위를 0~1로 변환
값의 범위	±1.96(또는 ±2) 데이터만 선택	0~1
공식	$z = \frac{x - \bar{x}}{\sigma}$ (분모가 표준편차)	$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$ (분모가 max값)
파이썬 코드	from scipy import stats df['new컬럼명'] = stats.zscore(df['Z값 구할 컬럼명'] df = df[df['Z값 넣은 new 컬럼명']. between(-2,2)]	from sklearn.preprocessing import MinMaxScaler scaler = MinMaxScaler() df[:] = scaler.fit_transform(df[:])
코딩 결과	1개 컬럼만 표준화 시킬 수 있음	여러 컬럼을 한꺼번에 정규화 시킬 수 있음

데이터 통합

다양한 데이터를 합쳐 단순화 하는 과정 !

데이터 통합 (Data integration)

- **많은 데이터를 통합하고 이를 단순화하여 표현하는 방법**
  - **단순화**: **중복**을 제거하는 과정
  - 예) 고객 A의 DB에서 **custom_id** 속성, 고객 B의 DB에서 **cust_id**
  - 불필요한 중복성이 존재함으로, **하나의 표현**으로 재표현
  - **Pandas의 DataFrame을 통합하는 과정에서는 2가지 통합 방법이 존재**
    - **Concatenating (연결)**: 단순하게 하나의 DataFrame에 다른 DataFrame을 연속적으로 붙이는 방법
    - **Merging (병합)**: 두 DataFrame의 공통적으로 포함되어 있는 하나의 열을 기준으로, 해당 열의 값이 동일한 두개의 행들을 하나로 합치는 방법
    - 아래의 그림은 **어떤 경우**일까요 ?

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannuaj	Phd

		ge	Address	Qualification
4	Abhi	17	Nagpur	Btech
5	Ayushi	14	Kanpur	B.A
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuaj	B.hons

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannuaj	Phd
4	Abhi	17	Nagpur	Btech
5	Ayushi	14	Kanpur	B.A
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuaj	B.hons

데이터 통합 예시 #1

간단한 연결 (Concatenating)

- 데이터 연결은 **동일한 index나 columns**을 가지고 있는 경우 **연속적으로 붙이는 작업**
- 서로 다른 DataFrame의 연결을 위해서는 **pandas.concat ()** 함수 사용
- Pandas.concat(DataFrame 1, DataFrame 2, axis)
  - **Axis의 값이 0이면 row bind, 1이면 column bind**

	A	B	C	D
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5

Row bind

	A	B	C	D	E	F	G	H
0	A0	B0	C0	D0	A6	B6	C6	D6
1	A1	B1	C1	D1	A7	B7	C7	D8
2	A2	B2	C2	D2	A8	B8	C8	D8

Column bind

```
2   import pandas as pd
3   from pandas import DataFrame
4
5   df_1 = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
6   'B': ['B0', 'B1', 'B2'],
7   'C': ['C0', 'C1', 'C2'],
8   'D': ['D0', 'D1', 'D2']},
9   index=[0, 1, 2])
10
11  df_2 = pd.DataFrame({'A': ['A3', 'A4', 'A5'],
12  'B': ['B3', 'B4', 'B5'],
13  'C': ['C3', 'C4', 'C5'],
14  'D': ['D3', 'D4', 'D5']},
15  index=[3, 4, 5])
16
17  # row bind : axis = 0
18  df_12_axis0 = pd.concat(            )
19  df_12_axis0
20
21  df_3 = pd.DataFrame({'E': ['A6', 'A7', 'A8'],
22  'F': ['B6', 'B7', 'B8'],
23  'G': ['C6', 'C7', 'C8'],
24  'H': ['D6', 'D8', 'D8']},
25  index=[0, 1, 2])
26
27  # colmun bind : axis = 1
28  df_12_axis0 = pd.concat(            )
29  df_12_axis0
```

데이터 통합 예시 #2

합집합과 교집합의 연결 (Concatenating)

- 서로 다른 DataFrame의 연결을 위해서는 **pandas.concat ()** 함수 사용
- Pandas.concat(DataFrame 1, DataFrame 2, join='option')
  - **합집합 (outer), 교집합 (inner)**

	A	B	C	D	E
0	A0	B0	C0	D0	NaN
1	A1	B1	C1	D1	NaN
2	A2	B2	C2	D2	NaN
0	A0	B0	C0	NaN	E0
1	A1	B1	C1	NaN	E1
3	A2	B2	C2	NaN	E2

Outer bind

	A	B	C
0	A0	B0	C0
1	A1	B1	C1
2	A2	B2	C2
0	A0	B0	C0
1	A1	B1	C1
3	A2	B2	C2

Inner bind

```
32    df_4 = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
33    'B': ['B0', 'B1', 'B2'],
34    'C': ['C0', 'C1', 'C2'],
35    'E': ['E0', 'E1', 'E2']},
36    index=[0, 1, 3])
37
38    # union
39    df_14_outer = pd.concat([df_1, df_4],              )
40    df_14_outer
41
42    # intersection
43    df_14_inner = pd.concat([df_1, df_4],              )
44    df_14_inner
```

데이터 통합 예시 #3

Index 중복 여부 점검 (Concatenating)

- 서로 다른 DataFrame의 연결을 위해서는 **pandas.concat ()** 함수 사용
- Pandas.concat(DataFrame 1, DataFrame 2, verify_integrity)
  - verify_integrity: index의 중복 여부를 점검하는 옵션
  - False일 경우 아무런 문제 없이 잘 합쳐지지만, Ture인 경우에는 ?

	A	B	C	D
r0	A0	B0	C0	D0
r1	A1	B1	C1	D1
r2	A2	B2	C2	D2
r2	A2	B2	C2	D2
r3	A3	B3	C3	D3
r4	A4	B4	C4	D4

옵션이 False 인 경우

```
48  df_7 = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
49  'B': ['B0', 'B1', 'B2'],
50  'C': ['C0', 'C1', 'C2'],
51  'D': ['D0', 'D1', 'D2']},
52  index=['r0', 'r1', 'r2'])
53
54  df_8 = pd.DataFrame({'A': ['A2', 'A3', 'A4'],
55  'B': ['B2', 'B3', 'B4'],
56  'C': ['C2', 'C3', 'C4'],
57  'D': ['D2', 'D3', 'D4']},
58  index=['r2', 'r3', 'r4'])
59
60  # False
61  df_78_F_verify_integrity = pd.concat([df_7, df_8],                    )
62  df_78_F_verify_integrity
63
64  # True
65  df_78_F_verify_integrity = pd.concat([df_7, df_8]                     )
```

데이터 통합 예시 #4

간단한 병합 (Merging)

- 데이터 병합은 두 DataFrame의 공통 열/인덱스를 기준으로 두개의 테이블을 병합
- 이때, **기준이 되는 열/행의 데이터는 키 (Key)**
- Pandas.merge (DataFrame 1, DataFrame 2, how, on)
  - **how (병합을 하는 방식), how (left/right)는 첫번째 혹은 두번째 DataFrame의 키 값을 모두 나타냄**
  - **on (공통적인 열의 이름)**

	KEY	A	B	C	D
0	K0	A0	B0	NaN	NaN
1	K1	A1	B1	NaN	NaN
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

Left merge

	KEY	A	B	C	D
0	K2	A2	B2	C2	D2
1	K3	A3	B3	C3	D3
2	K4	NaN	NaN	C4	D4
3	K5	NaN	NaN	C5	D5

Right merge

```
1   import pandas as pd
2   from pandas import DataFrame
3
4   df_left = DataFrame({'KEY': ['K0', 'K1', 'K2', 'K3'],
5   'A': ['A0', 'A1', 'A2', 'A3'],
6   'B': ['B0', 'B1', 'B2', 'B3']})
7
8   df_right = DataFrame({'KEY': ['K2', 'K3', 'K4', 'K5'],
9   'C': ['C2', 'C3', 'C4', 'C5'],
10  'D': ['D2', 'D3', 'D4', 'D5']})
11
12  f_merge_how_left = pd.merge(                                    )
13  f_merge_how_left
14
15  f_merge_how_right = pd.merge(                                     )
16  f_merge_how_right
```

데이터 통합 예시 #5

합집합과 교집합을 사용하는 병합 (Merging)

- 데이터 병합은 두 DataFrame의 공통 열/인덱스를 기준으로 두개의 테이블을 병합
- Pandas.merge (DataFrame 1, DataFrame 2, how, on)
  - **how (병합을 하는 방식), how (left/right)는 첫번째 혹은 두번째 DataFrame의 키 값을 모두 나타냄**
  - **합집합 (outer), 교집합 (inner)**

	KEY	A	B	C	D
0	K2	A2	B2	C2	D2
1	K3	A3	B3	C3	D3

Inner merge

	KEY	A	B	C	D
0	K0	A0	B0	NaN	NaN
1	K1	A1	B1	NaN	NaN
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3
4	K4	NaN	NaN	C4	D4
5	K5	NaN	NaN	C5	D5

Outer merge

```
19    # Inner
20    df_merge_how_inner = pd.merge(                    )
21    df_merge_how_inner
22
23
24    # Outer
25    df_merge_how_outer = pd.merge(                    )
26    df_merge_how_outer
```

데이터 통합 예시 #6

변수 이름이 중복되는 경우 (Merging)

- 데이터 병합은 두 DataFrame의 공통 열/인덱스를 기준으로 두개의 테이블을 병합
- Pandas.merge (DataFrame 1, DataFrame 2, how, on, suffixes=('_x','_y'))
  - **Suffixes는 중복되는 변수 이름에 대해 접두사 부여**

```
29   df_left_2 = DataFrame({'KEY': ['K0', 'K1', 'K2', 'K3'],
30   'A': ['A0', 'A1', 'A2', 'A3'],
31   'B': ['B0', 'B1', 'B2', 'B3'],
32   'C': ['C0', 'C1', 'C2', 'C3']})
33
34   df_right_2 = DataFrame({'KEY': ['K0', 'K1', 'K2', 'K3'],
35   'B': ['B0_2', 'B1_2', 'B2_2', 'B3_2'],
36   'C': ['C0_2', 'C1_2', 'C2_2', 'C3_2'],
37   'D': ['D0_2', 'D1_2', 'D2_2', 'D3_3']})
38
39   pd.merge(df_left_2, df_right_2, how='inner', on='KEY',                    )
```

	KEY	A	B_left	C_left	B_right	C_right	D
0	K0	A0	B0	C0	B0_2	C0_2	D0_2
1	K1	A1	B1	C1	B1_2	C1_2	D1_2
2	K2	A2	B2	C2	B2_2	C2_2	D2_2
3	K3	A3	B3	C3	B3_2	C3_2	D3_3

데이터 통합

■ 데이터 중복

- 데이터를 수집하는 과정 중의 오류로 인해, 중복되지 않아야 할 데이터가 중복되는 경우가 발생
- 특히 Unique한 key 값을 관리하는 경우, 중복이 발생하면 심각한 영향을 줄 가능성이 있음
- 따라서 예외없이 중복 데이터를 확인하고 처리하는 전처리 작업이 필요
- **Pandas에는 중복 여부를 확인하는 함수와, 중복되는 값을 처리하는 함수가 존재**

	key1	key2	col
0	a	v	1
1	b	w	2
2	b	w	3
3	c	x	4
4	c	y	5

```
1   import pandas as pd
2
3   data = {'key1':['a', 'b', 'b', 'c', 'c'],
4   'key2':['v', 'w', 'w', 'x', 'y'],
5   'col':[1, 2, 3, 4, 5]}
6
7   df = pd.DataFrame(data, columns=['key1','key2','col'])
8   df
```

데이터 통합 예시 #7

데이터 중복

- **중복 여부를 확인하는 함수: pandas.duplicated(['key'])**
  - **Key**: 중복의 기준이 되는 키값
- **중복되는 값을 처리하는 함수가 존재: pandas.drop_duplicateds(['key'], keep)**
  - **Keep**: 중복이 있다면 처음과 마지막 값 중에 무엇을 남길 것인가 (first, last, false)

```
1   import pandas as pd
2
3   data = {'key1':['a', 'b', 'b', 'c', 'c'],
4   'key2':['v', 'w', 'w', 'x', 'y'],
5   'col':[1, 2, 3, 4, 5]}
6
7   df = pd.DataFrame(data, columns=['key1','key2
8   df
9
10  df.duplicated(        )
11  df
```

```
0    False
1    False
2     True
3    False
4     True
dtype: bool
```

```
13  df.duplicated(['key1'],                )
14  df
```

	key1	key2	col
0	a	v	1
1	b	w	2
3	c	x	4

데이터 축소

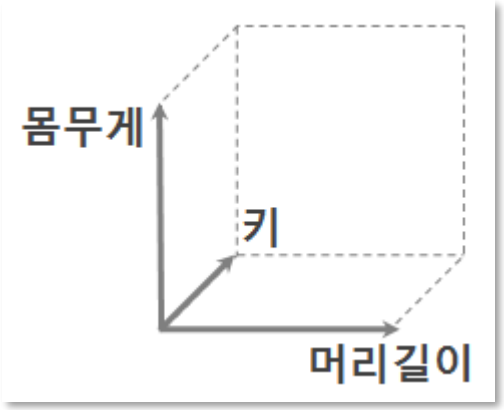
데이터 집합의 **부피**를 작게 만들면서, 동일한 결과를 얻는 방법 !

데이터 축소 (Data reduction)

차원을 축소하는 방법

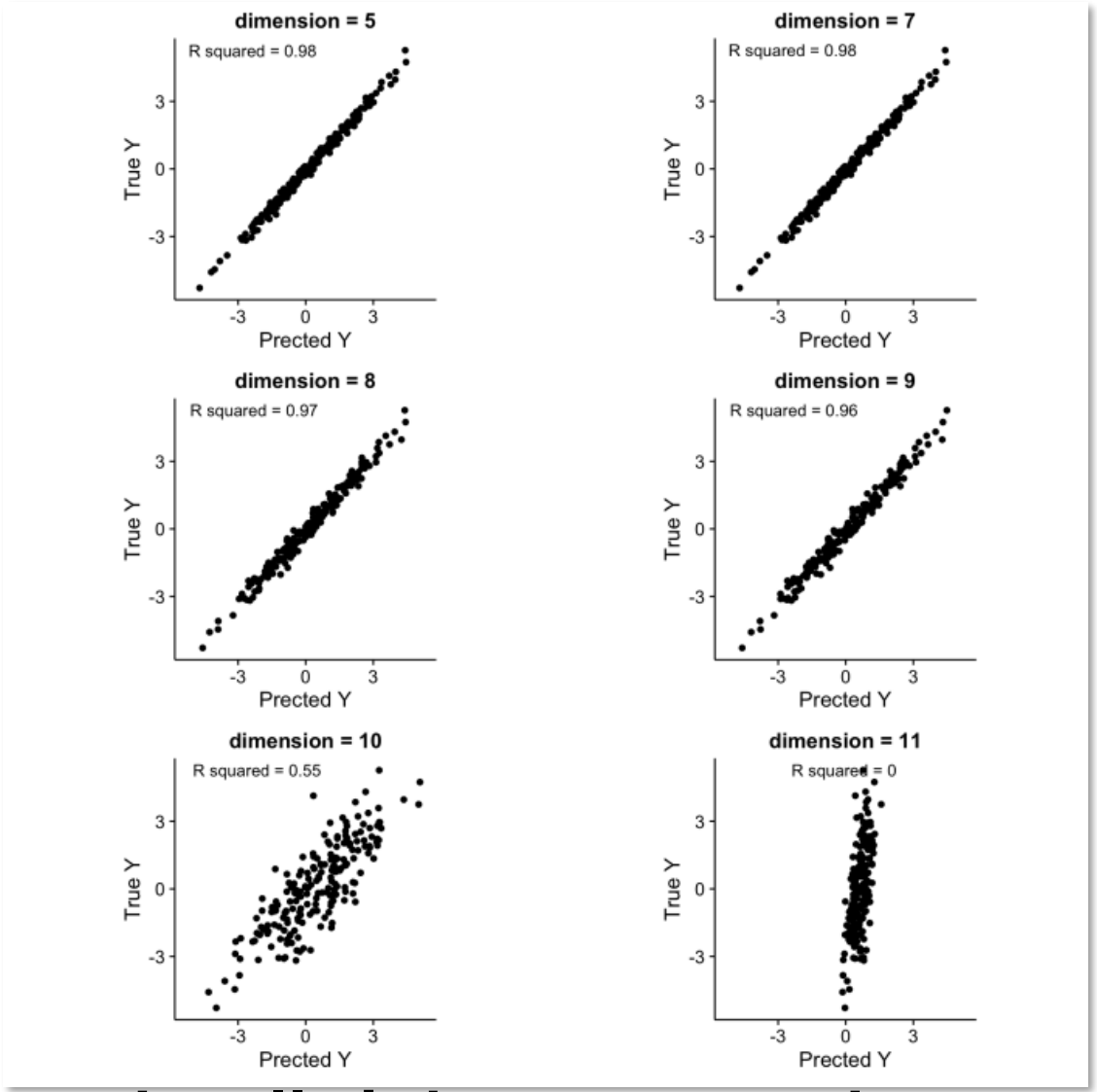
- **실무에서 분석하는 데이터는 많은 특성을 가지고 있음**
- **많은 특성**은 **높은 수준의 차원과 연관, 학습속도**가 느리고 **성능**이 좋지 않을 수 있음
- 차원: 수학에서 공간 내에 있는 점 등의 위치를 나타내기 위한 필요한 축의 갯수
- **아래의 데이터의 차원은 얼마일까 ?**
  - **축의 개수 = 변수의 수 = 차원**
  - **즉, 변수가 늘어나는 것은 차원의 증가, 그리고 데이터 공간이 커짐을 의미**

키 (cm)	몸무게 (kg)	머리 길이 (cm)
168	58	10
162	55	30
159	49	25
165	45	40



데이터 축소 (Data reduction)

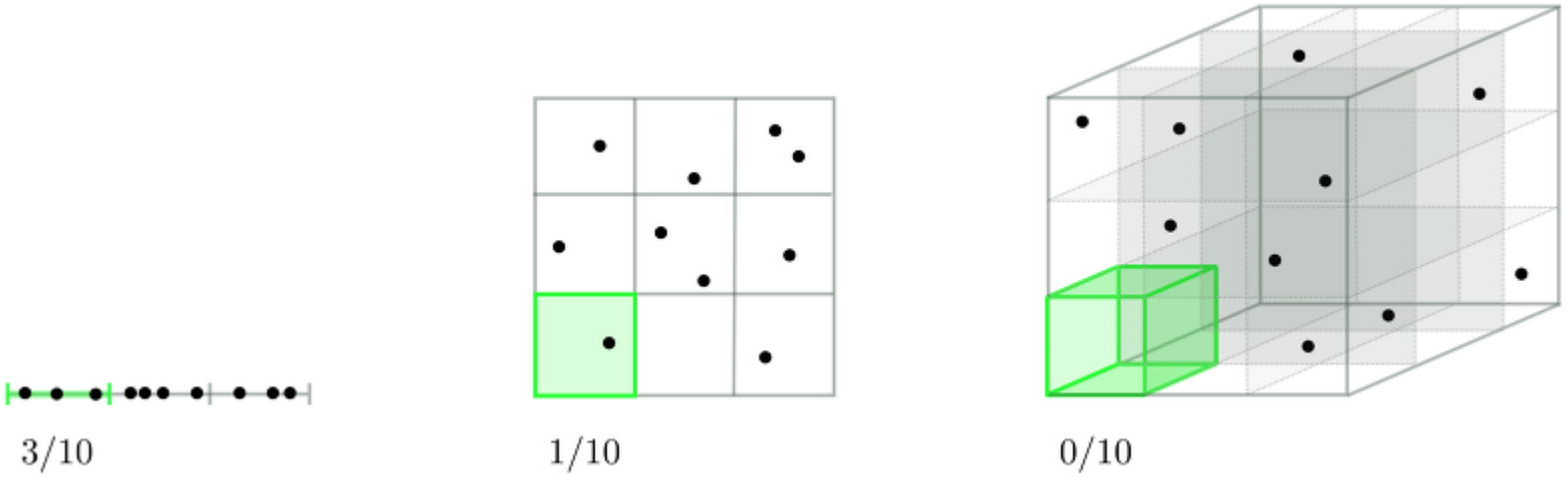
- **차원의 저주 (Curse of dimensionality)**
  - **변수가 늘어나 차원이 커지면서 발생하는 문제**
  - **선형회귀에서 데이터 샘플이 많아 진다면, 성능도 좋아질까 ?**
    - Y = a*X1 + b*X2 + c*X3 + d*X4 + e*X5
    - 다섯 개 변수의 **선형 결합**으로 결정되는 변수 Y
    - **선형 결합의 개수가 증가할수록, 데이터의 설명하지 못함**



데이터 축소 (Data reduction)

차원의 저주 (Curse of dimensionality)

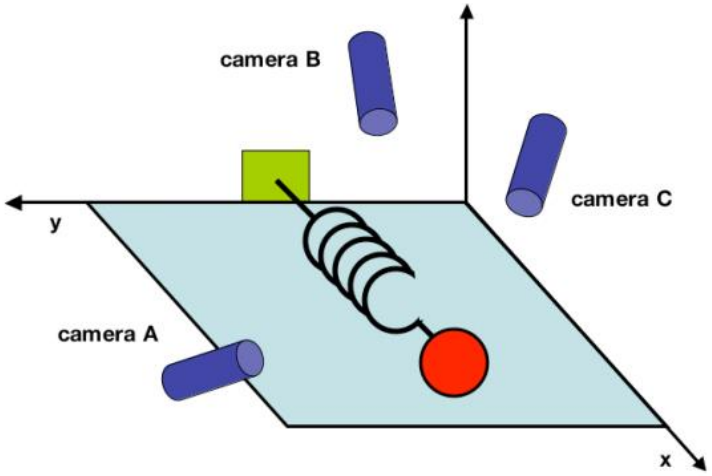
- **변수가 늘어나 차원이 커지면서 발생하는 문제**
- **데이터의 차원이 증가할수록, 부피가 기하 급수적으로 증가하기 때문에, 차원이 커질수록 데이터의 밀도는 희소**
  - 차원이 증가 할수록, 데이터 간의 거리가 증가하므로 모델이 복잡한 형태로 학습을 진행
  - 차원의 저주를 해결하기 위해서는 데이터의 밀도가 높아질 때 까지 데이터 셋의 크기를 늘리는 것 !
  - 하지만, **데이터 셋의 크기에 비해 차원은 기하급수적으로 커지지 때문**에 현실적으로 불가능

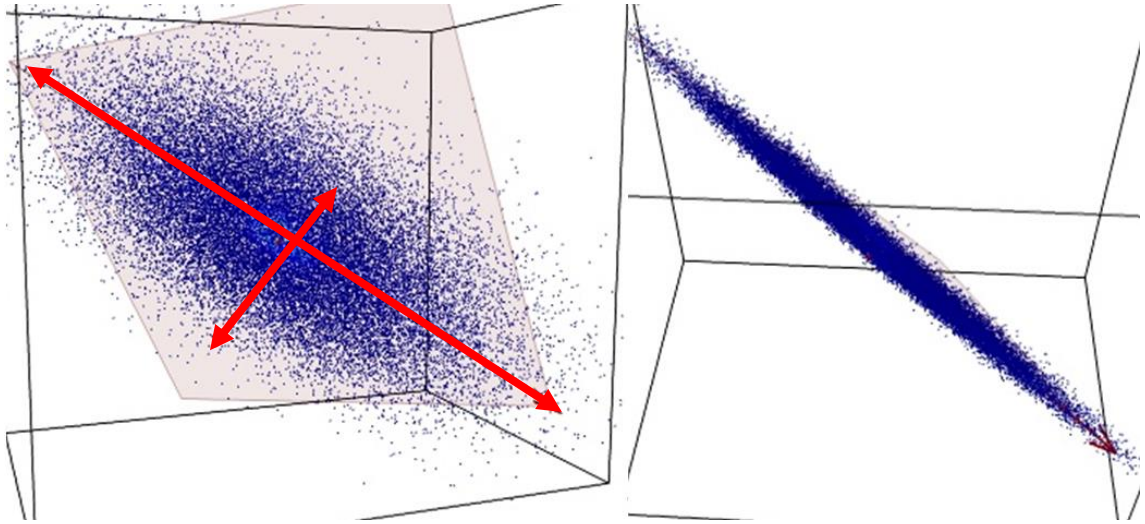


데이터 축소 (Data reduction)

차원의 축소

- 고차원의 데이터를 저 차원으로 축소하는 방법 (Dimension reduction)
  - **10차원의 데이터**는 '어디가 비슷하네', '어디가 이렇게 굽어졌네'과 같은 판단을 하는 것이 불가능함
  - **PCA (Principal component analysis)**는 이러한 것이 가능하도록 하는 방법: 데이터의 특성을 눈으로 파악 할 수 있도록 함
  - 스프링 운동: 3차원 축으로 구성되어 있지만, 모든 축에 대해서 동일한 정도로 움직이는 것은 아님
  - **어떤 축이 가장 운동을 가장 표현하는 축인지 파악하는 방법**

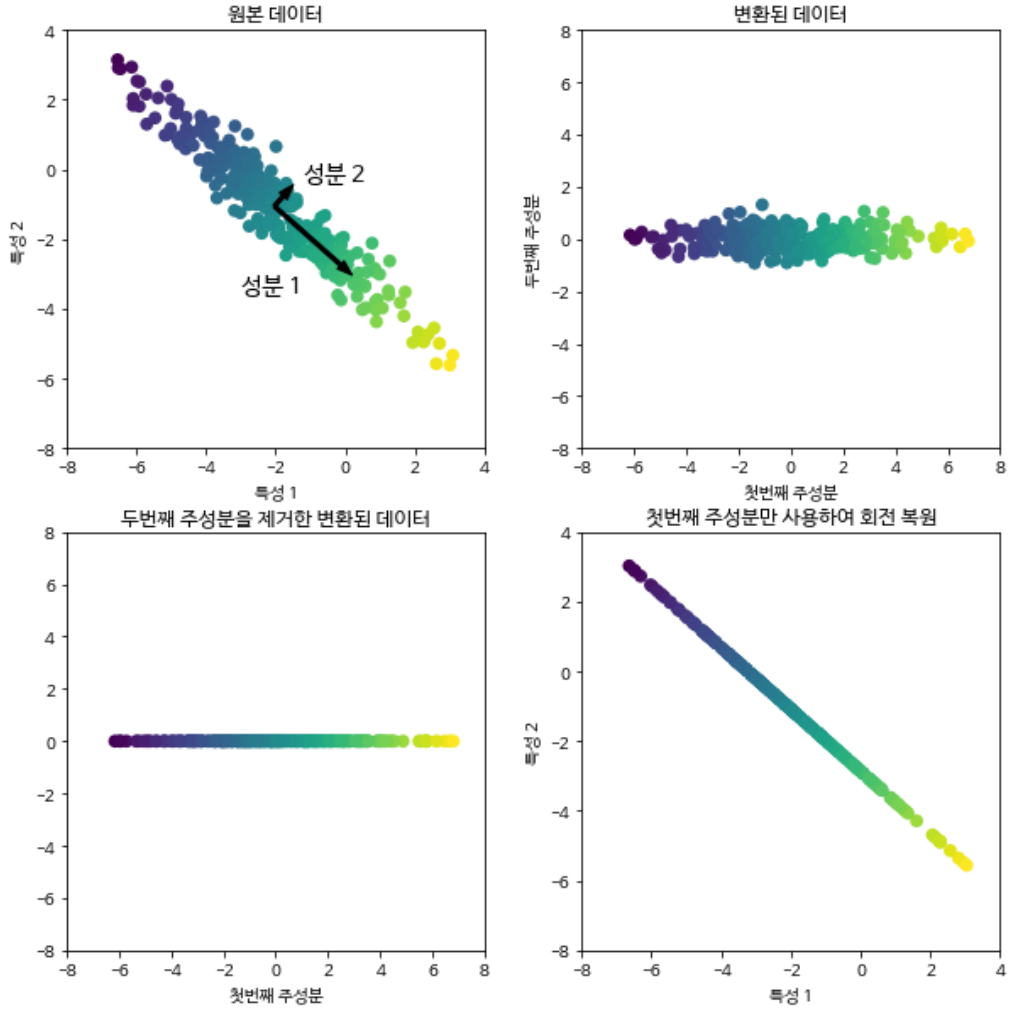




데이터 축소 (Data reduction)

PCA (Principal component analysis)

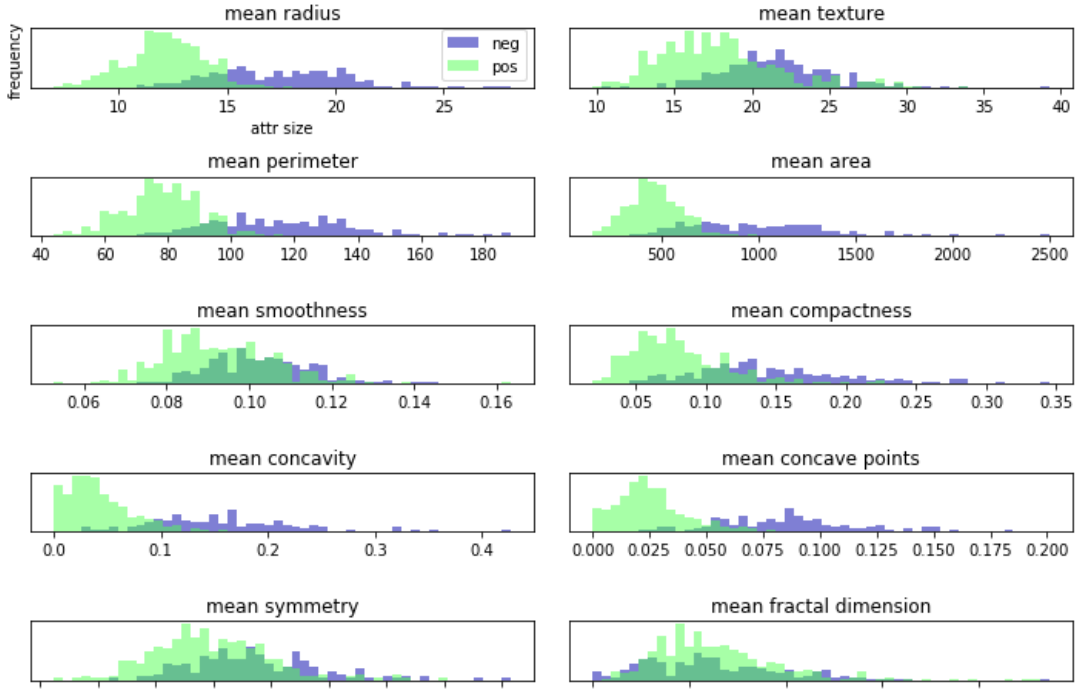
- PCA는 차원을 축소하는 방법으로 정사영 (Projection)을 사용
  - 정사영: 통계적으로 상관관계가 없도록 데이터를 회전시키는 기술
  - 회전을 한 뒤, 데이터를 설명하는데 필요한 특성 중 일부만 선택
- 첫번째 그래프
  - 성분 1의 분산이 **가장 큰 방향**을 갖고 있음
  - 성분 1과 직각 방향 중 **가장 많은 정보를 담은 방향 (성분 2)을 탐색**
  - **주성분 (Principal component): 가장 많은 정보를 담은 방향**
- 두번째 그래프
  - 주성분 1과 2를 각각 x, y축에 나란하도록 회전
  - y축의 값들은 큰 변화가 없기 때문에, **두번째 주성분을 제거 (세번째 그래프)**
- 네번째 그래프 (원래의 형태로 변환)
  - 원래 특성 공간에 있지만 첫번째 주성분의 정보만 포함
  - PCA는 데이터에서 노이즈를 제거하거나, **주성분의 정보를 시작화**



데이터 축소 (Data reduction) 예시 #1

PCA를 적용해 유방암 데이터셋 시각화

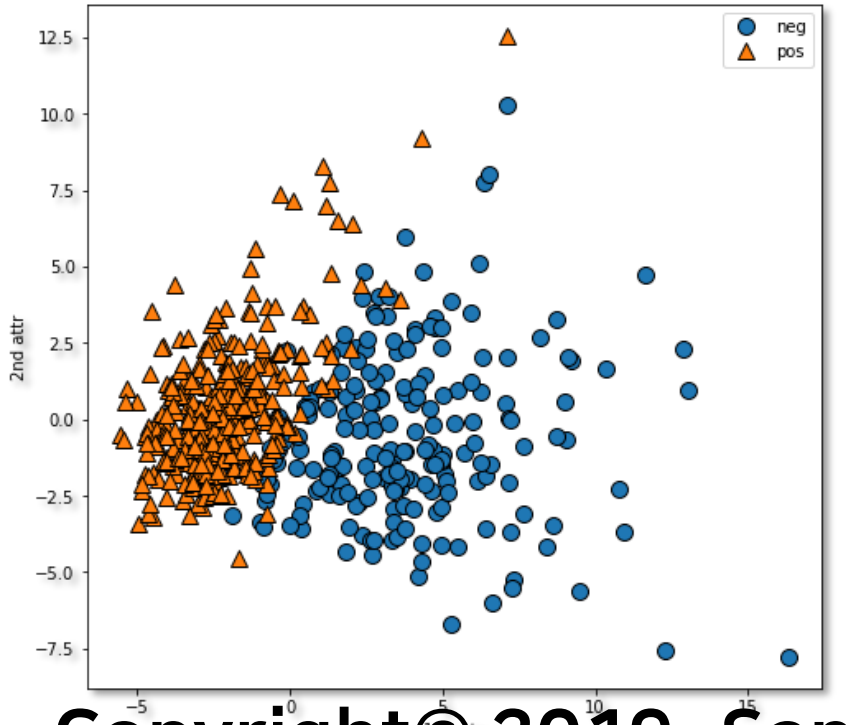
- **세가지 이상의 특성**을 가진 데이터를 산점도로 표현하는 것은 어려움
- 유방암 데이터 셋은 특성을 30개나 가지고 있기 때문에, 산점도로 표현은 불가능
- 따라서, 양성과 악성 두 클래스에 대한 각 특성의 히스토그램을 먼저 분석
  - 각 특성에 대한 히스토그램으로, **특정 간격 (bin)에 얼마나 많은 데이터 포인트가 있는지를 그래프로 표현**
  - **대략적인 경향은 알 수 있으나, 특성 간의 상호작용은 파악 불가능**



데이터 축소 (Data reduction) 예시 #1

PCA를 적용해 유방암 데이터셋 시각화

- StandardScalar() 함수를 사용한 **정규화**
  - 각 특성의 **분산이 1이 되도록 조정**
- **PCA 변환을 학습하고 적용하는 과정**
  - **transform () 함수를 사용하여 데이터를 사용하고 차원 축소, PCA 객체 생성, fit () 함수를 사용하여 주성분 찾기**



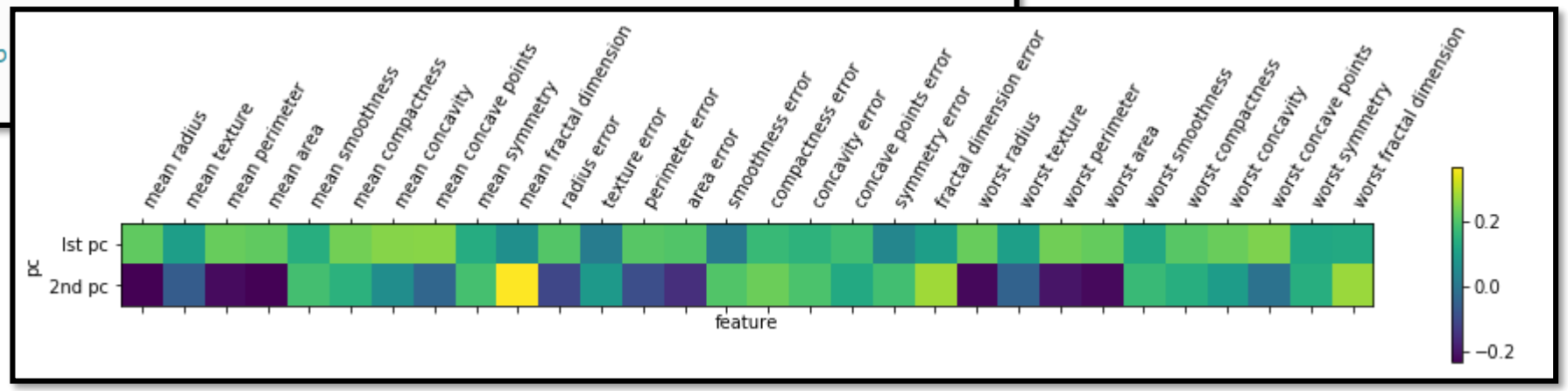

```
standard_scaler =

X_scaled = standard_scaler.transform(cancer.data)

from sklearn.decomposition import PCA
pca =
pca.

X_pca = pca.transform(X_scaled)
print("원본 데이터 형태 : {}".format(str(X_scaled.shape)))
print("축소된 데이터 형태 : {}".format(str(X_pca.shape)))

plt.figure(figsize=(8,8))
mglearn.discrete_scatter(X_pca[:,0],X_pca[:,1],cancer.target)
plt.legend(["neg","pos"],loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("1st attr")
plt.ylabel("2nd attr")
plt.show()
```

데이터 축소 (Data reduction) 예시 #1

PCA를 적용해 유방암 데이터셋 시각화

- PCA를 통해 얻은 결과는 구분이 잘 되지만, **두 축을 해석하는 것은 쉽지 않음**
- **PCA가 학습 될 때, 여러 특성에 대한 정보는 components_ 속성에 저장**
  - **해당 성분을 사용하여 히트맵으로 시각화 가능**

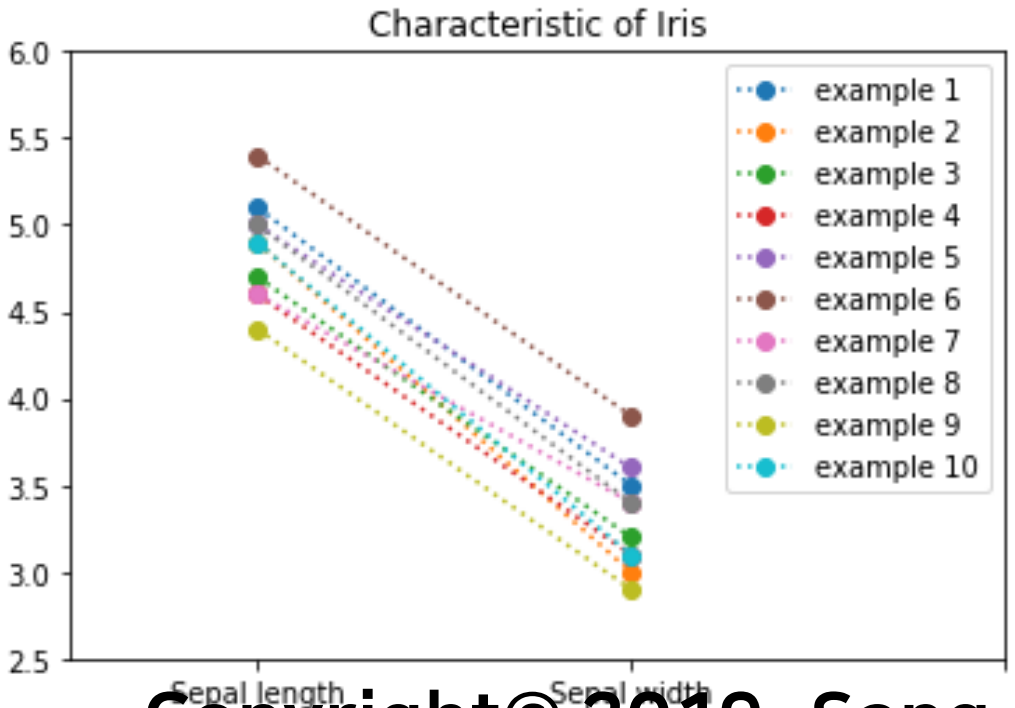
```
22    plt.matshow(pca.components_, cmap="viridis")
23    plt.yticks([0, 1], ["comp 1", "comp 2"])
24    plt.colorbar()
25    plt.xticks(range(len(cancer.feature_names)), cancer.feature_names, rotation=60, ha='left')
26    plt.xlabel("attr")
27    plt.ylabel("principle comp
28    plt.show()
```



데이터 축소 (Data reduction) 예시 #2

PCA를 사용한 붓꽃 데이터 차원축소

- 10송이의 데이터 (10개의 표본)을 선택하여 꽃받침 길이와 폭 데이터는 다음의 그래프와 같음
  - 가로축은 특성의 종류, 세로축은 특성의 값을 나타냄
  - 꽃받침 길이가 크면, 꽃 받침의 폭도 같이 커진다는 규칙이 존재
  - **그래프 그리는 빈칸 만들고 설명하기**



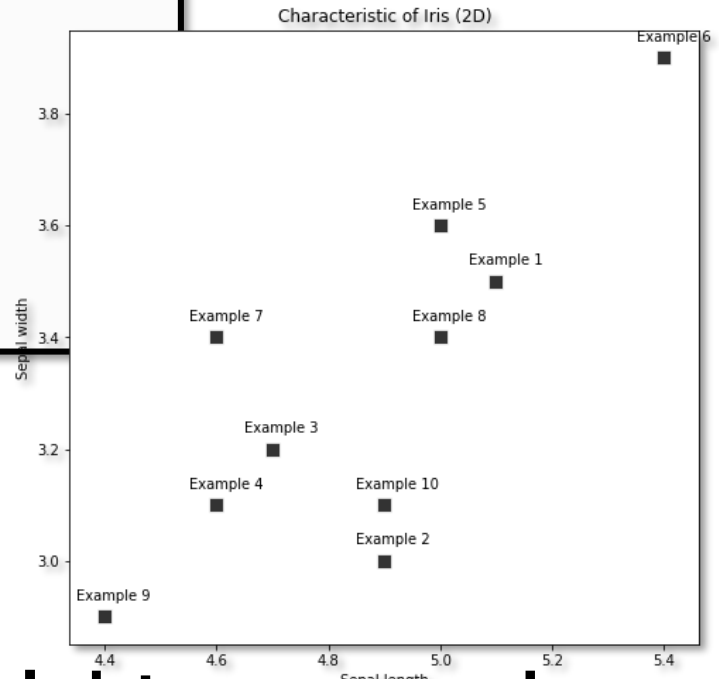

```python
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

iris = load_iris()
N = 10  # 앞의 10송이만 선택
X = iris.data[:N, :2]  # 꽃받침 길이와 꽃받침 폭만 선택

plt.plot(X.T, 'o:')
plt.xticks(range(4), ["Sepal length", "Sepal width"])
plt.xlim(-0.5, 2)
plt.ylim(2.5, 6)
plt.title("Characteristic of Iris")
plt.legend(["example {}".format(i + 1) for i in range(N)])
plt.show()
```

데이터 축소 (Data reduction) 예시 #2

PCA를 사용한 붓꽃 데이터 차원축소

- **가로축 :꽃받침 길이, 세로축: 꽃받침 폭**
  - 가로축은 특성의 종류, 세로축은 특성의 값을 나타냄
  - 데이터를 나타내는 점들의 양의 기울기를 가지기 때문에, 꽃받침의 길이가 크면 꽃받침 폭도 같이 커지는 특성

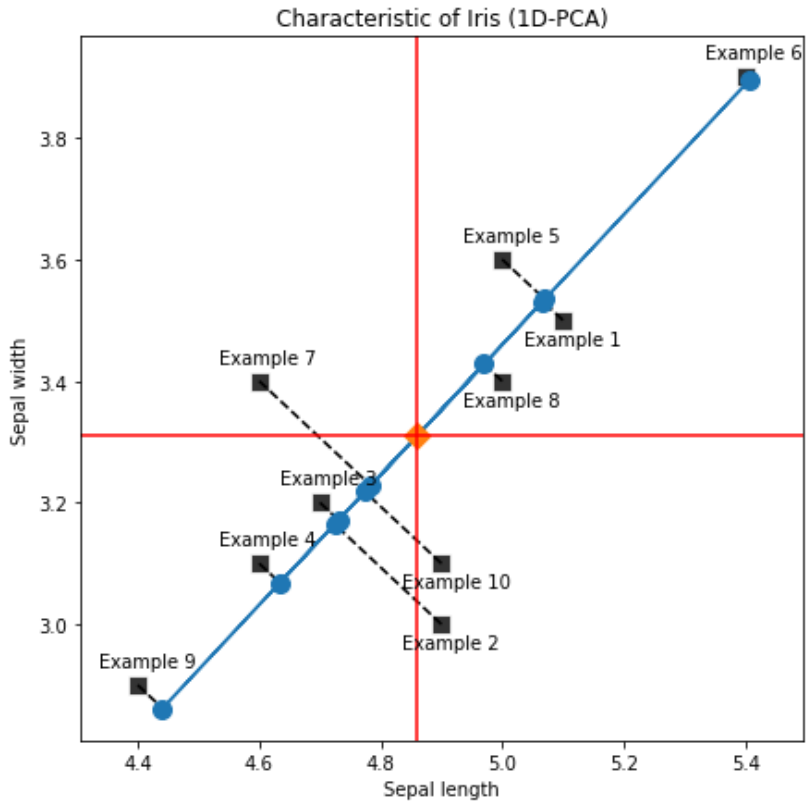
```
19    plt.figure(figsize=(8, 8))
20    ax = sns.scatterplot(0, 1, data=pd.DataFrame(X), s=100, color=".2", marker="s")
21    for i in range(N):
22        ax.text(X[i, 0] - 0.05, X[i, 1] + 0.03, "Example {}".format(i + 1))
23    plt.xlabel("Sepal length")
24    plt.ylabel("Sepal width")
25    plt.title("Characteristic of Iris (2D)")
26    plt.axis("equal")
27    plt.show()
```



데이터 축소 (Data reduction) 예시 #2

- PCA를 사용한 붓꽃 데이터 차원축소
  - fit_transform (): 특징 행렬을 낮은 차원의 근사행렬로 변환
  - inverse_transform (): 변환된 근사행렬을 원래의 차원으로 복귀

```
from sklearn.decomposition import PCA

pca1 = PCA(n_components=1)
X_low = pca1.fit_transform(X)
X2 = pca1.inverse_transform(X_low)

plt.figure(figsize=(7, 7))
ax = sns.scatterplot(0, 1, data=pd.DataFrame(X), s=100, color=".2", marker="s")
for i in range(N):
    d = 0.03 if X[i, 1] > X2[i, 1] else -0.04
    ax.text(X[i, 0] - 0.065, X[i, 1] + d, "Example {}".format(i + 1))
    plt.plot([X[i, 0], X2[i, 0]], [X[i, 1], X2[i, 1]], "k--")
plt.plot(X2[:, 0], X2[:, 1], "o-", markersize=10)
plt.plot(X[:, 0].mean(), X[:, 1].mean(), markersize=10, marker="D")
plt.axvline(X[:, 0].mean(), c='r')
plt.axhline(X[:, 1].mean(), c='r')
plt.grid(False)
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
plt.title("Characteristic of Iris (1D-PCA)")
plt.axis("equal")
plt.show()
```



데이터 변환

알고리즘의 효율성을 극대화 하기 위한 방법 !

데이터 변환 개요

- **데이터 알고리즘의 효율성을 극대화 하기 위한 데이터 조작 과정**
  - 데이터를 [0.0, 1.0]의 작은 범위 내에서 표현하는 **정규화**
  - 속성의 데이터 값을 다른 범위나 상위 레벨의 개념으로 대체하는 **이산화**
  - **데이터는 전처리 단계에서 변환되거나 통합**되어 데이터 마이닝을 효율적으로 수행 가능
  - 변환되거나 통합되어 **발견된 패턴을 보다 쉽게 이해** 할 수 있도록 도움

데이터 변환

데이터 변환의 종류

- **평활화**: 데이터의 잡음을 제거하는 과정
- **응집**: 응집, 요약 연산을 데이터에 적용
  - 예) 일간 매출 지류가 월간, 연간 총계를 계산하기 위해서 결합
- **정규화**: 속성 데이터를 더 작은 범위에 들어가게 하기 위해 데이터를 다듬는 과정
- **이산화**: 수치형 속성의 원시값이 구간 라벨로 대체되거나, 개념적인 라벨로 대체

감사합니다.

오늘하루 수업 듣느라 고생하셨습니다 !