데이터 클리닝2

데이터 클리닝 과정

- **데이터 평가**Access Data
  - 누락 데이터 평가
  - 이상치 평가
  - 중복 데이터 평가

- **데이터 정제**Cleaning Data
  - 누락 데이터 처리
  - 이상치 처리
  - 중복 데이터 처리

이상치 평가

- **이상치** Outliers
  - 변수의 분포에서 비정상적으로 **분포**를 벗어난 값
  - 통계적 자료분석의 결과를 왜곡시키거나, 자료분석의 적절성을 위협

- **이상치 평가** Assess outliers
  - Dirty Data 판별법의 Step2. Validity에서와 같이 제한사항constraint을 충족하는지 확인하는 경우와는 구별
  - 변수에 대한 이해가 있는 경우, 최대값과 최소값 등을 확인하여 오류 데이터인지 여부를 판단 가능 예) 나이, 키, 체중 등
  - 통상적으로 단변량 분포에서는 개체의 변수값이 평균으로부터 ±2~3 표준편차 밖에 위치하는 값을 이상치로 평가

다변량 데이터

- **다변량 데이터**
  - 단변량 분포에서만 아니라 다변량 분포에서도 이상치는 존재
  - 통상 둘 이상의 서로 상관되어 있는 변수들을 포함하고 있는 자료
  - 단변량에서는 이상치로 판단되었던 값도 다변량에서는 다르게 평가될 수 있다.

- **이상치 평가**Assess outliers
  - 다변량 분포에서는 산점도scatter plot 를 활용하여 이상치 평가

다변량 예제

- 예제
  - 수학 성적과 과학 성적이 높은 상관을 갖는 것이 정상인 경우

- 이상치 평가Assess outliers
  - 어느 학생의 수학 점수는 매우 높고, 과학 점수는 매우 낮다면 이는 비정상적인 점수 결합으로서 정상적인 이변량 분포를 벗어나고, 자료의 타당성도 결여된다고 볼 수 있음
  - 다만, 이 예에서는 수학과 과학은 수리 능력이라는 공통 개념이라는 전제에서 이상치로 판단한 것임
  - 수학 점수 내에서 이상치를 파악하면 단변량

단변량 이상치 평가

- **평균 및 표준편차 활용법**
  - 통상적으로 단변량 분포에서는 개체의 변수값이 평균으로부터 ±2~3 표준편차 밖에 위치하는 사례를 이상치로 평가
  - 예를 들어, 편차를 2로 두는 경우, (평균 ± 2*표준편차) 밖에 있는 값은 이상치로 평가

- **표준편차**
  - 자료가 나타내는 값들을 x1, x2,…, xn이라 하고 그 (산술)평균을 m이라 할 때, 표준편차의 정의는 다음과 같다.

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - m)^2}$$

표준편차

- **표준편차(Standard deviation)의 의미**
  - 많은 데이터에 대해 데이터를 대표하는 하나의 값 (대푯값) 으로 평균값이 자주 사용된다.
  - 표준편차는 데이터가 평균을 중심으로 얼마나 퍼져 있는지를 나타내는 대표적인 수치이다.
  - 표준편차가 0에 가까우면 데이터가 평균 근처에 모여 있다는 의미이다. 반대로 표준편차가 클수록 데이터가 넓게 퍼져 있다는 의미이다.
  - 즉, 표준편차는 수식과 같이 퍼져있는 평균 거리이므로 이 거리보다 많이 외곽에 위치한 데이터는 이상치일 가능성이 높다고 판단할 수 있다.

이상치 평가 예제

- 예제
  - 1.2, 1.5, 2.4, 2.8, 3.5.4.1, 그리고 9.0이 주어진 변수의 분포

- 이상치 평가Assess outliers
  - 주어진 데이터의 평균은 3.5, 표준편차는 2.63
  - 예를 들어, 편차를 2로 두는 경우, (평균 ± 2*표준편차) 밖에 있는 값은 이상치로 평가
  - 즉, -1.77~8.77 밖에 있으면, 이상치로 평가하므로 주어진 데이터세트에서는 9.0이 유일한 이상치가 됨

실습1.1 이상치 평가 준비

- 표준편차 구하기
  - DataFrame 표준편차 method : std( )

```
import pandas as pd
```

```
sr = pd.Series([1.2, 1.5, 2.4, 2.8, 3.5, 4.1, 9.0])
```

```
print('max=',sr.max(),', min=',sr.min())
```

```
max= 9.0 , min= 1.2
```

```
print('mean=',sr.mean(),', std=',sr.std())
```

```
mean= 3.5 , std= 2.6331223544175333
```

실습1.2 이상치 평가

- 이상치 평가
  - 예를 들어, 편차를 2로 두는 경우, delta=2
  - (평균 ± 2*표준편차) 밖에 있는 값은 이상치로 평가
  - 즉, -1.77~8.77 밖에 있으면, 이상치로 평가

```
delta = 2
bottom = sr.mean()-delta*sr.std()
print(bottom)
top = sr.mean()+delta*sr.std()
top
```

```
-1.7662447088350666

8.766244708835067
```

```
print('outlier:')
for val in sr:
    if val<bottom:
        print(val)
    elif val>top:
        print(val)
```

```
outlier:
9.0
```

이상치 처리

- **이상치 처리** Handling outliers
  - 데이터 마이닝에서의 일반적인 접근방법은 이상치를 제거 후의 데이터 세트를 포함하여 두 가지 데이터세트에 대해 성능을 확인해 보는 것
  - 때때로 데이터 마이닝의 목적이 이상치를 발견하는 것인 경우도 있어 이상치 제거에 주의를 기울여야 함
  - 예를 들어, 공항의 보안검색에서 이상감지 Anomaly detection 와 같은 경우

- **데이터 정제 방법**
  - 이상치 제거 – 행 제거
  - 이상치 대체 Imputation

중복데이터 평가

- **중복 데이터**duplicated observation
  - 하나의 데이터세트에서 동일한 관측값이 2개 이상 중복되는 경우
  - 동일한 대상이 중복으로 존재하는 것이므로 분석 결과를 왜곡하게 됨

- **중복 데이터평가**Assess duplicated observation
  - 동일한 관측값이 중복되는지 여부, 즉 행의 레코드가 중복되는지 여부를 확인하려면 DataFrame의 duplicated( ) 메소드를 이용
  - duplicated( )는 전에 나온 행들과 비교하여 중복되는 행이면 True를 반환하고, 처음 나오는 행에 대해서는 False를 반환

중복데이터 처리

- **중복 데이터 처리** Handling duplicated observation
  - 하나의 데이터세트에서 동일한 관측값이 2개 이상 중복되는 경우, 중복 데이터를 찾아서 삭제해야 함

- **데이터 정제 방법**
  - 중복 데이터 제거 – 행 제거
  - DataFrame method : drop_duplicates( )

    : 중복되는 행을 제거하고 고유한 관측값을 가진 행들만 남김

실습2.1 중복데이터 평가

- **duplicated( )**
  - DataFrame에 duplicated( ) 메소드를 적용하면 각 행의 중복 여부를 나타내는 boolean Series를 반환
  - 0행의 데이터는 뒤에 나오는 1행의 데이터와 같지만 처음 나오는 값으로 False,
    1행의 데이터는 앞의 0행과 중복되기 때문에 True

```
import pandas as pd
```

```
df = pd.DataFrame({'c1':['a','a','b','a','b'],
                   'c2':[1,1,1,2,2],
                   'c3':[1,1,2,2,2]})
df
```

	c1	c2	c3
0	a	1	1
1	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

```
#assess duplicated data
df_dup = df.duplicated()
df_dup
```

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

실습2.2 중복데이터 처리

- **drop_duplicates( )**
  - 1행의 데이터는 앞에 이웃하고 있는 0행의 데이터와 중복되므로 제거
  - subset옵션에 '열 이름의 리스트' 전달
    : 데이터의 중복 여부를 판별할 때 subset옵션의 열을 기준으로 판단

```
#cleaning duplicated data
df_clean = df.drop_duplicates()
df_clean
```

	c1	c2	c3
0	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

```
df2 = df.drop_duplicates(subset=['c2','c3'])
df2
```

	c1	c2	c3
0	a	1	1
2	b	1	2
3	a	2	2

Dirty String 처리

- **깔끔하지 않은 문자열 데이터 처리**
  - 데이터가 대충 입력되거나 사용자에 의해 직접 입력이 되어 철자 오류나 구문 오류 등이 포함되어 있는 경우
  - 예를 들어, 도시명이 50개로 한정되는 경우, 500개의 입력된 도시명 중 오타나 실수를 정제할 수 있는가?

- **중복된 유사 문자열 처리**
  - 두 개 이상의 데이터세트를 사용하고 있거나 깔끔하지 않고 통일성이 없는 문자열 데이터를 사용하고 있다면, 퍼지 매칭을 활용하여 중복 기록을 찾아 처리할 수 있음

    예) yes / no 답변에 대해 nop … / ya … 등 사용

퍼지 매칭

- **퍼지 매칭** fuzzy matching
  - 퍼지 매칭을 이용하면 고의가 아닌 실수들을 처리 가능
  - 자연어 처리나 머신 러닝처럼 심도 있는 방법은 아니지만 두 문자열이 비슷한 의미인지 연결 지을 수 있음

- **fuzzywuzzy Library**
  - fuzzywuzzy 메소드 : 문자열 배열의 유사도 (1~100)

    : ratio( ), partial_ratio( ), token_sort_ratio( ), token_set_ratio( )
  - process module : 데이터에서 나타나는 선택지가 한정적인 경우, Dirty String을 그 중 하나에 매칭하는 방법

실습3.1 Library 준비

- **fuzzywuzzy** 라이브러리 **import**하기

```
from fuzzywuzzy import fuzz
```

- **fuzzywuzzy** 라이브러리 설치 안 된 경우 (아래 오류 발생)

```
from fuzzywuzzy import fuzz
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-604ebbd17881> in <module>
----> 1 from fuzzywuzzy import fuzz

ModuleNotFoundError: No module named 'fuzzywuzzy'
```

fuzzywuzzy library 설치

- **Anaconda Prompt에서**
  - 가상환경 목록 확인

    : conda env list
  - 가상환경 활성화

    : conda activate data_eng
  - 설치된 library목록 확인

    : conda list

    (fuzzywuzzy library없는 것 확인)
  - fuzzywuzzy library 설치

    : conda install fuzzywuzzy

fuzzywuzzy 설치 문제

- **Anaconda Prompt에서**
  - fuzzywuzzy library 설치

    : conda install fuzzywuzzy

```
(data_eng) C:₩Users₩hwoon>conda install fuzzywuzzy
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.

PackagesNotFoundError: The following packages are not available from current channels:

  - fuzzywuzzy
```

  - fuzzywuzzy library 설치 : Anaconda에서 제공 안 됨

    : pip install fuzzywuzzy

fuzzywuzzy 설치 경고

- **Anaconda Prompt에서**
  - fuzzywuzzy library 설치

    : pip install fuzzywuzzy

```
from fuzzywuzzy import fuzz
```

```
C:₩Users₩hwoon₩anaconda3₩envs₩data_eng₩lib₩site-packages₩fuzzywuzzy₩fuzz.py:11: UserWarning: Using slow pure-python SequenceMatche
r. Install python-Levenshtein to remove this warning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning')
```

  - python-Levenshtein library 설치

    : conda install python-Levenshtein

  - Levenshtein Distance 알고리즘 (편집 거리 알고리즘)

    : 두 문자열 간 유사도 측정하기

Levenshtein Distance

- **Fuzzywuzzy는 두 문자열의 유사도를 Levenshtein 거리로 계산**
  - Levenshtein 거리는 러시아 과학자 블라디미르 레벤슈타인이 창안하였으며 편집 거리 (Edit Distance) 알고리즘이라고도 함.
  - 편집거리란 문자열 A를 문자열 B로 바꾸기 위해 편집 (문자 삽입, 삭제, 바꾸기) 해야 하는 개수를 말함.
  - Levenshtein 거리의 단점은 다른 문장이라도 유사한 의미를 가질 수 있는데 이 경우에는 적용하기 어려움

ratio()

- **ratio() 수식 : 2*(M/T)*100**
  - 두 문자열이 있을 때, T는 두 문자열의 공백을 포함한 총 문자수이고, M은 일치하는 문자수
  - 예를 들어, 'Grapes of Wrath'와 'The Grapes of Wrath ' 의 ratio()를 구하면 각각 문자열 길이가 15와 19이므로 T=34, Grapes부터는 순서대로 같으므로 M=15로 ratio = 2*(15/34)*100 = 88

실습3.2 fuzzywuzzy ratio()

- **fuzzy.ratio( )**
  - 두 문자열 배열의 유사도 (1~100)
  - 모든 문자를 순서대로 비교

```
from fuzzywuzzy import fuzz
```

```
my_records = [{'favorite_book': 'Grapes of Wrath',
               'favorite_movie': 'Free Willie',
               'favorite_show': 'Two Broke Girls',
              },
              {'favorite_book': 'The Grapes of Wrath',
               'favorite_movie': 'Free Willy',
               'favorite_show': '2 Broke Girls',
              }]
```

```
print( fuzz.ratio(my_records[0].get('favorite_book'),
                  my_records[1].get('favorite_book')) )
print( fuzz.ratio(my_records[0].get('favorite_movie'),
                  my_records[1].get('favorite_movie')) )
print( fuzz.ratio(my_records[0].get('favorite_show'),
                  my_records[1].get('favorite_show')) )
```

```
88
86
86
```

실습3.3 partial_ratio()

- **fuzzy.partial_ratio( )**
  - 부분 문자열 배열의 유사도 (1~100)
  - 부분 문자열을 순서대로 (in order)
  - 첫 번째 예는 완전한 포함 관계 (부분 문자열) 이므로 유사도 100

```
print( fuzz.partial_ratio(my_records[0].get('favorite_book'),
                  my_records[1].get('favorite_book')) )
print( fuzz.partial_ratio(my_records[0].get('favorite_movie'),
                   my_records[1].get('favorite_movie')) )
print( fuzz.partial_ratio(my_records[0].get('favorite_show'),
                   my_records[1].get('favorite_show')) )
```

```
100
90
92
```

실습4.1 token_sort_ratio()

- **token_sort_ratio()**
  - 단어 별로 정렬 후 비교 (1~100)
  - 2번째 예제는 같은 단어들이 순서만 다르므로 유사도 100

```
from fuzzywuzzy import fuzz
```

```
my_records = [{'favorite_food': 'cheeseburgers with bacon',
               'favorite_drink': 'wine, beer, and tequila',
               'favorite_dessert': 'cheese or cake',
              },
              {'favorite_food': 'burgers with cheese and bacon',
               'favorite_drink': 'beer, wine, and tequila',
               'favorite_dessert': 'cheese cake',
              }]
```

```
print( fuzz.token_sort_ratio(my_records[0].get('favorite_food'),
                  my_records[1].get('favorite_food')) )
print( fuzz.token_sort_ratio(my_records[0].get('favorite_drink'),
                   my_records[1].get('favorite_drink')) )
print( fuzz.token_sort_ratio(my_records[0].get('favorite_dessert'),
                   my_records[1].get('favorite_dessert')) )
```

```
68
100
88
```

실습4.2 token_set_ratio()

- **fuzzy.token_set_ratio( )**
  - 토큰의 집합을 비교해서 부분집합 확인 (1~100)
  - 세 번째 예는 토큰의 부분집합이 같으므로 유사도 100

```
print( fuzz.token_set_ratio(my_records[0].get('favorite_food'),
                  my_records[1].get('favorite_food')) )
print( fuzz.token_set_ratio(my_records[0].get('favorite_drink'),
                   my_records[1].get('favorite_drink')) )
print( fuzz.token_set_ratio(my_records[0].get('favorite_dessert'),
                   my_records[1].get('favorite_dessert')) )
```

```
68
100
100
```

실습4.3 process module

- **process module**
  - 어떤 질문에 대한 답이 반드시 yes, no, maybe, N/A 중에 하나라는 것을 알 때 (즉, 선택지가 한정적)
  - process의 extract( ) 메소드를 사용하여 비교 기준이 되는 문자열을 선택지 문자열 리스트와 비교해서
  - choices 리스트에서 유사도가 높은 2개를 추출

```
from fuzzywuzzy import process
choices = ['Yes', 'No', 'Maybe', 'N/A']
```

```
process.extract('ya', choices, limit=2)
```

```
[('Yes', 45), ('Maybe', 45)]
```

실습4.4 extractOne()

- **extractOne( )**
  - 선택 가능한 문자열 리스트에서 비교 기준이 되는 문자열과 가장 매칭이 잘 되는 것을 반환

```
process.extract('nope', choices, limit=2)
```

```
[('No', 90), ('Yes', 29)]
```

```
process.extractOne('nope', choices)
```

```
('No', 90)
```

참고도서

- **reference**

[1] 파이썬을 활용한 데이터길들이기

- 프로그래밍인사이트

[2] 모두의 데이터분석

- 길벗

[3] 파이썬머신러닝 판다스데이터분석

- 정보문화사

End