데이터 클리닝1

데이터 클리닝

- **필요성**
  - 머신러닝 등 데이터 분석의 정확도는 분석 데이터의 품질에 의해 좌우됨
  - 데이터 품질을 높이기 위해서는 누락 데이터, 중복 데이터 등 오류를 수정하고 분석 목적에 맞게 변형하는 과정이 필요

- **데이터 클리닝 과정**
  - 데이터 평가
  - 데이터 정제

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

누락 데이터 평가

- **누락 데이터** Missing values
  - 분석하고자 하는 데이터 세트에 원소 데이터 값이 종종 누락되는 경우가 있음
  - 데이터를 파일로 입력할 때 빠트리거나 파일 형식을 변환하는 과정에서 데이터가 소실되는 것이 주요 원인
  - 일반적으로 DataFrame에서는 유효한 데이터 값이 존재하지 않는 누락 데이터를 NaN(Not a Number)으로 표시

- **누락 데이터 평가**Assess missing values
  - Dirty Data 판별법의 Step1. Completeness에서와 같이
    모든 데이터가 채워져 있는가? 행, 열에 Null값이 없는지 확인
  - DataFrame의 Library로 판별 가능

실험 데이터세트

- **seaborn 데이터세트**
  - seaborn은 대표적인 시각화 라이브러리 vs. matplotlib
  - seaborn에서는 다양한 시계열 데이터 등의 데이터세트를 가지고 있음
  - 예를 들면, titanic의 생존자 예측을 위한 'titanic' 데이터세트
  - 또한, 붓꽃이라고 해야하나요? 꽃의 width와 length에 따른 종 분류 데이터를 가지고 있는 iris

- **실험 데이터** Experiment Data
  - 'titanic' 데이터세트 사용

실습1.1 데이터 준비

- **seaborn** 데이터세트 가져오기

```
import seaborn as sns

#load titanic dataset
df = sns.load_dataset('titanic')
df
```

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

```
In [1]: #call library
        import seaborn as sns

---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-0dfc58ba636b> in <module>
      1 #call library
----> 2 import seaborn as sns

ModuleNotFoundError: No module named 'seaborn'
```

seaborn library 설치

- **Anaconda Prompt에서**
  - 가상환경 목록 확인

    : conda env list
  - 가상환경 활성화

    : conda activate data_eng
  - 설치된 library목록 확인

    : conda list

    (seaborn library없는 것 확인)
  - seaborn library 설치

    : conda install seaborn

실습1.1 결과

```
import seaborn as sns
```

```
#load titanic dataset
df = sns.load_dataset('titanic')
df
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns

Titanic Dataset Features

pclass	Passenger Class, 승객 등급
survived	생존 여부(생존은 1, 아닌 경우는 0)
name	승객 이름
sex	승객 성별
age	승객 나이
sibsp	동승한 형제 또는 배우자 수
parch	동승한 부모 또는 자녀 수
ticket	티켓 번호
fare	승객 지불 요금
cabin	선실 요금
embarked	승선항(C=쉘 부르크, Q=퀸즈타운, S=사우스햄튼)
body	사망자 확인 번호
home.dest	고향/목적지

실습1.2

- **요약정보 출력 info( )**
  - ‘deck’열은 승객이 몇 번 데크에 승선했는지
  - ‘deck’ 열에 NaN값이 아닌 유효한 데이터가 203개 있는 것을 확인

```
#summary
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.6+ KB
```

실습1.3

- 누락 데이터의 개수 구하기1
  - **value_counts(dropna=False)**
  - ‘deck’ 열의 승선 데크 종류별 개수 확인
  - dropna=False 옵션 (누락데이터 개수 포함)

```
#count NaN in deck column
nan_deck = df['deck'].value_counts(dropna=False)
nan_deck
```

```
NaN    688
C       59
B       47
D       33
E       32
A       15
F       13
G        4
Name: deck, dtype: int64
```

실습1.4

- **isnull( )**
  - 누락 데이터이면 True 반환, 유효한 데이터이면 False반환
  - notnull( )은 반대로

```
#find missing data
df.head().isnull()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False

실습1.5

- 누락 데이터 개수 구하기2
  - isnull()에서 반환되는 값 True는 연산에서는 1, False는 0값으로 취급됨
  - 예를들어,
    isnull().sum()은 참(1)의 합을 구함
    = null값의 개수 구하기
  - sum(axis=0)에서
    axis=0는 각 열의 합

```
#count missing data using isnull()
df.head().isnull().sum(axis=0)
```

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           3
embark_town    0
alive          0
alone          0
dtype: int64
```

누락 데이터 처리

- **누락 데이터 처리** Handling missing values
  - 행 또는 열에서 누락된 데이터는 머신러닝의 결과에 악영향을 끼칠 수 있음
  - 머신러닝 분석 모형에 데이터를 입력하기 전에 반드시 누락 데이터를 제거하거나 다른 적절한 값으로 대체하는 과정이 필요
  - 즉, 누락데이터는 제거하거나 새로운 값(?)을 삽입하여 처리
  - 새로운 값은 코딩으로 평균값 등으로 대체

- **데이터 정제 방법**
  - 누락 데이터 제거 – 행 또는 열 제거
  - 누락 데이터 대체 Imputation

누락 데이터 제거

- 누락 데이터 제거 기준
  - 예를 들어, 분석 대상의 관측값 즉, 행을 제거하는 경우 100개중 5개 정도면 제거해도 무방할 것 임
  - 그러나, 10개의 관측값 중 5개를 제거하기는 어려울 것임 (50%를 제거하는 것이 가능한가?)

- 열 또는 행의 제거
  - 열을 제거하면 분석 대상이 가지는 특성(변수, feature)를 제거 -> 분석에 필요 없는 특성인지 판단
  - 행을 제거하면 분석 대상의 관측값(레코드)을 제거 -> 분석데이터의 양이 부족하지 않다면 가능

실습2.1

- 누락 데이터 개수 파악
  - ‘titanic’ 데이터세트의 각 열에 누락 데이터가 몇 개씩 포함되어 있는지 확인 => 실습1.5
  - isnull()로 구한 참,거짓 테이블에 value_counts()를 적용하여 True, False의 개수를 구하고 그 중 True가 있는 것은 개수를 True가 없는 것은 0을 출력 (True가 누락데이터가 있다는 것)

```
#count NaN in column
missing_df = df.isnull()
for col in missing_df.columns:
    missing_count = missing_df[col].value_counts()
    #print(missing_count)
    try:
        print(col,': ',missing_count[True])
    except:
        print(col,': ',0)
```

```
survived :  0
pclass :  0
sex :  0
age :  177
sibsp :  0
parch :  0
fare :  0
embarked :  2
class :  0
who :  0
adult_male :  0
deck :  688
embark_town :  2
alive :  0
alone :  0
```

실습2.2

- 누락 데이터가 많은 열 제거
  - **dropna(axis=1, thresh=500)**
  - axis=1 옵션 : 열 제거
  - thresh=500 옵션 : null이 아닌 유효데이터가 500개 미만이면 삭제
  - ‘deck’열의 유효데이터가 203 (891-688)개로 500개 미만이라 ‘deck’열은 제거 됨
  - 전체 15개 열 중 하나가 삭제되어 14개 남음

```
df_thresh = df.dropna(axis=1, thresh=500)
df_thresh.columns
```

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'embark_town', 'alive',
       'alone'],
      dtype='object')
```

실습2.3

- 누락 데이터를 포함한 행 제거
  - **dropna(subset=[‘age’], how=‘any’)**
  - subset=[‘age’] 옵션 : ‘age’열에 대해
  - how=‘any’ 옵션 : NaN이 하나라도 존재하면 cf. 모두 how=‘all’
  - default (axis=0) 행 삭제
  - 총 891데이터에서 누락 데이터를 포함한 177개 행을 삭제하면 714행
  - 177개의 관측값을 삭제해도 분석에 문제가 없는가?

```
#age열에 나이 데이터가 없는 모든 행 삭제
df_age = df.dropna(subset=['age'],how='any')
len(df_age)
```

```
714
```

누락 데이터 대체

- **누락 데이터 대체** Imputation
  - 데이터세트의 품질을 높일 목적으로 누락 데이터를 무작정 삭제해 버린다면 어렵게 수집한 데이터를 사용하지 못하게 된다
  - 데이터 분석의 정확도는 데이터의 품질 외에도 제공되는 데이터의 양에 의해 상당한 영향을 받는다
  - 따라서 데이터 중에서 일부가 누락되어 있더라도 나머지 데이터를 최대한 살려서 데이터 분석에 활용하는 것이 좋은 결과를 얻는 경우가 많다

- **대체 방법**
  - 데이터의 분포와 특성을 잘 나타낼 수 있는 평균값, 최빈값 등을 활용
  - 최빈값 : 가장 자주 나온 값

실습3.1

- **‘age’열의 누락 데이터 확인**
  - ‘titanic’ 데이터세트의 ‘age’ 열에서 앞 20개 행에 대 NaN이 몇 개 인지 눈으로 확인 : head(20)
  - 5, 17, 19번 행이 누락 데이터인 것 확인

```
df['age'].head(20)
```

```
0     22.0
1     38.0
2     26.0
3     35.0
4     35.0
5      NaN
6     54.0
7      2.0
8     27.0
9     14.0
10     4.0
11    58.0
12    20.0
13    39.0
14    14.0
15    55.0
16     2.0
17     NaN
18    31.0
19     NaN
Name: age, dtype: float64
```

실습3.2

- 평균값 계산
  - mean() cf. median() : 중간값

```
mean_age = df['age'].mean()
mean_age
```

- 평균값으로 대체
  - **fillna(평균값)**
  - ‘age’열의 NaN을 찾아서 모두 평균값으로 대체

```
df['age'].fillna(mean_age, inplace=True)
df['age'].head(20)
```

실습3.2 결과

```
df['age'].fillna(mean_age, inplace=True)
df['age'].head(20)
```

```
0     22.000000
1     38.000000
2     26.000000
3     35.000000
4     35.000000
5     29.699118
6     54.000000
7      2.000000
8     27.000000
9     14.000000
10     4.000000
11    58.000000
12    20.000000
13    39.000000
14    14.000000
15    55.000000
16     2.000000
17    29.699118
18    31.000000
19    29.699118
```

통계 Library

- **Pandas DataFrame 유용한 method**
  - sum 전체 성분의 합을 계산

    sum1 = df.sum(axis = 1) # 행 단위

    sum2 = df.sum(axis = 0) # 열 단위
  - min, max 전체 성분의 최소값, 최대값을 계산
  - idxmin, idxmax 전체 인덱스 중 최소값, 최대값의 key값을반환
  - mean 전체 성분의 평균을 계산
  - median 전체 성분의 중간값을 반환
  - std, var 전체 성분의 표준편차, 분산을 계산

참고도서

- **reference**

[1] 파이썬을 활용한 데이터길들이기

- 프로그래밍인사이트

[2] 모두의 데이터분석

- 길벗

[3] 파이썬머신러닝 판다스데이터분석

- 정보문화사

End