Pandas수업안_2 >> DataFrame 데이터가 주어졌을 때 처리하는 방법 실습

<< DataFrame 정의 >>

- Serials가 1차원이라면 DataFrame 은 2차원
- row, column으로 구성됨
- Data Analysis, Machine Learning에서 Data변형을 위해 많이 사용한다.

2차원 array로 만들기

```
df = pd.DataFrame([[1,2,3], [4,5,6],[7,8,9]])
print(type(df))
print(df)

a = np.arange(12).reshape(3,4)
aa = pd.DataFrame(a)
```

배열(array)이나 딕셔너리로도 만들 수 있다. 보통은 파일에서 가져와서 사용한다.

```
student = {'NAME': ['A','B','C','D','E'],
           'KOR': [100,80,85,70,60],
           'ENG' : [88,65,75,80,70],
           'MATH' : [70,80,90,45,75]}
print(student)
type(student)
df = pd.DataFrame(student)
print(type(df))
df
df.shape          //항상확인해야한다.
len(df)

#컬럼확인하기
df.columns #컬럼이름 목록 & type확인
print(df.columns[0])  //첫 번째 컬럼명
print(df.columns[1])
print(df.columns[-1])
print(df[df.columns[0]])  #1개의 컬럼의 요소 접근, 인덱스와 같이 표시
print(df[df.columns[0]][0])  or df['eng'][0]

#value확인하기
df.values # 전체데이터 추출 숫자+ 'object' = type은 object로

df[0]  #keyError  이렇게 접근은 안됨 / 열로 인식한다. 열의이름이 '0'은 없다.
```

** 행의 인덱싱은 아예안되고 컬럼이름으로 인덱싱은 가능.
** 행은 슬라이싱으로 출력한다.

슬라이싱
df[:3] #df.head(), 3줄출력
df[2:4] //index기준으로 출력
df[::3] //처음~끝까지 3줄씩 건너
df[::-1] #행을 역순으로 출력

#원하는 열의 행을 슬라이싱 하기
df[df.columns[0]][2:4]
df[df.columns[0]][:-1]
df[df.columns[0]][-1:] //0열의 마지막하나 가져오기 ':' 빼면 오류
df[df.columns[0]][::3] # 3개의 step으로 슬라이싱 (0컬럼의 0행에서 3행씩추가)
df[df.columns[0]][::-1] #0번열의 순서를 거꾸로 출력

#여러개의 컬럼을 가져오기 :여러개의 컬럼명을 리스트로 만들어서인덱싱에 사용
df[['KOR','MATH']] //컬럼들을 리스트에 넣는다.
이건 시리즈 객체가 아니고 데이터프레임이다. 즉 데이터프레임에서 리스트로 뽑아 다시 데이터프레임으로만들었다.

df[[df.columns[0], df.columns[1], df.columns[2]]] #같은결과
df[[df.columns[0], df.columns[1], df.columns[2]]][0] #데이터프레임이므로 바로접근 안되고 오류난다. 이건 슬라이싱으로 해야한다.
df[[df.columns[0], df.columns[1], df.columns[2]]][:1]
df[[df.columns[0], df.columns[1], df.columns[2]]][2:5:2]

>> dataframe의 요소 접근하기 : 읽기와 수정
df.loc[행번호, 열] : 명시적인 index
df.iloc[행번호, 열] : 함묵적인 index, 파이썬스타일, int형 0부터시작

df.loc[3,2] //오류
df.loc[3,'KOR'] //index 3의 KOR열
df.loc[:3, 'ENG'] //슬라이싱으로 빼오기, 0~3까지 나온다. 다른슬라이싱과 헷갈리수 있다.
df.loc[:3, ['ENG','KOR','MATH']] //여러열 지정
//:3 3인덱스 까지 나온다. 열은 숫자 안되고 열이름 지정해야한다.

>>iloc : index를 int형으로 사용, 자주사용됨
df.iloc[3,0] or df.iloc[3][2] <-> df.loc[3,0]오류
df.iloc[:4,0:3]
df.iloc[:4,[0,2,3]]

// :3 3인덱스 전까지 출력, iiloc은 열도 인덱스로 접근가능

비교 : print(df.loc[:4, ['ENG','KOR','MATH']]) & print(df.iloc[:4, 0:3])
df.iloc[: , :-1] // 마지막 컬럼을 제외, x(독립변수)추출
df.iloc[:, -1] // 마지막 컬럼만 추출, y(종속변수)추출
// iloc이 다른 슬라이싱이과 비슷하여 많이 사용한다.

>>iloc으로 요소값의 변경
df.iloc[3,0]=200
df
df.count()
df.iloc[3,0]=np.NaN
df
df.count() //NaN제외 개수구함

mean, deviation, variance, standard deviation:

평균 :
편차 : 값과 평균과의 차이
분산 : 편차의 제곱의 합의 평균
표준편차 : 분산에 루트(제곱근)을 씌운값

df.max()
df.min()
df.sum()
df.sum(axis=0)
df.sum(axis=1)
df.mean()
df.mean(axis=1)
df.var() //분산
df.std() //표준편차
df.median()

df.mode() //열방행으로 최빈값 찾기 있으면 표시 나머진 NaN, 없으면 모두표시
df.iloc[1,0]=70 //최빈값보기위해 수정
df.iloc[2,2]=80
df.mode() -> 결과, 최빈값표시 나머지NaN표시

ex) 함수활용되는예
df[df['ENG'] > df['ENG'].mean()] //조건에 맞는 열모두
df[df['ENG'] > df['ENG'].mean()]['ENG'] //조건에 맞는 원하는 열만

정리할 시간 주기

>> DataFrame 합치기

ex2)

```
data = {'a':100, 'b':200, 'c':300}
pd.DataFrame(data, index=['x','y','z'])
```

	a	b	c
x	100	200	300
y	100	200	300
z	100	200	300

//각 인덱스에 값을 채움

```
data = {'a':[1,2,3], 'b':[4,5,6], 'c':[10,11,12]}
pd.DataFrame(data, index=['x','y','z'])
```

	a	b	c
x	1	4	10
y	2	5	11
z	3	6	12

// index는 문자 숫자 모두가능하다. index를 0,1,2 로 수정해 본다.
df.index = [1,2,3]

//
```
student1 = {'NAME': ['A','B','C','D','E'],
            'KOR': [100,80,85,70,60],
            'ENG' : [88,70,75,80,70],
            'MATH' : [70,80,90,45,80]}
student2 = {'NAME': ['AA','BB','CC','DD','EE'],
            'KOR': [90,80,95,70,80],
            'ENG' : [92,74,75,88,80],
            'MATH' : [75,85,95,60,88]}
df1 = pd.DataFrame(student1)
df2 = pd.DataFrame(student2)
df1
df2
new_s1= df1.append(df2)  //index중복
new_s1
new_s2 = df1.append(df2,ignore_index = True)  //인덱스중복 안됨
```

#컬럼/행 삭제 : drop() 컬럼을 삭제하는 일이 많다. 필요없은 열을 삭제.
df1.drop(columns=['ENG']) //원본유지
df1.drop(columns=['ENG'],inplace = True) //원본삭제, Fasle 원본유지 결과만 리턴
df1.drop(index=[1,3]) //행삭제 원본유지
df1.drop(index=[1,3], inplace=True) //행삭제 원본삭제

```
# contains() 문자열을 검색한 결과를 Series(1차원) 객체로 받는다.
문자열의 조건 검색, 행의 추출
print(type(df1['KOR']))
name_s = df1['NAME'].str.contains('A')  // Serise객체, 1차원배열과 호환

df1[ name_s]

??????????????????????????????
```

준비data >>
https://www.kaggle.com/hesh97/titanicdataset-traincsv/data
train.csv를 다운받는다.
3. csv데이터로 DataFrame 생성하기

1. head, tail함수 : 데이터 전체가 아닌 일부(처음, 마지막부터)를 간단히 보기 위한 함수

```
import pandas as pd
```

```
train_data = pd.read_csv('data/train.csv')
```

```
train_data
```

>> csv파일은 엑셀과 호환이 되고 ',(쉼표)'로 구분지어 놓은 데이터파일이다.

```
import pandas as pd
train_data = pd.read_csv('./data/train.csv', sep=',')
train_data
```

//sep :구분기호 지정가능, ','는 생략가능

```
train_data = pd.read_csv('./data/train.csv', header=None)
train_data
```

//header가 없는 파일의 경우

```
train_data = pd.read_csv('./data/train.csv', index_col='PassengerId', usecols=['PassengerId','Survived', 'Pclass', 'Name'])
train_data
```

// PassengerId가 index값과 같이 0~891이여서 index로 사용하고, 원하는 column만 출력 가능하다.

- PassengerId : 승객 번호
- Survived : 생존여부(1: 생존, 0 : 사망)
- Pclass : 승선권 클래스(1 : 1st, 2 : 2nd ,3 : 3rd)
- Name : 승객 이름
- Sex : 승객 성별
- Age : 승객 나이
- SibSp : 동반한 형제자매, 배우자 수
- Patch : 동반한 부모, 자식 수
- Ticket : 티켓의 고유 넘버
- Fare 티켓의 요금
- Cabin : 객실 번호
- Embarked : 승선한 항구명(C : Cherbourg, Q : Queenstown, S : Southampton)

>>>실행하기전 문제내고 기다림...

train_data.head() //처음부터 5개의 데이터 보인다.(column들의 내용확인하기)

train_data.head(n=3) //row를 정할 수 있다.
train_data.tail() //마지막부터 5개 표시
train_data.tail(n=3)
train_data.shape //shape으로 row와 column의 개수를 확인
train_data.describe() //숫자데이터인 경우 개수, 평균, 분산... 기본적인 통계의 값을 구한다. 문자데이터는 안나옴
train_data.info() //type, 개수를 확인한다.

2. index : 각 아이템을 특정할 수 있는 고유의 값을 저장, 복잡한 데이터 경우 멀티인덱스 표현 가능하다.
column :각각의 특성을 나타냄, 복잡한 데이터 경우 멀티 컬럼을 표현가능하다.

```
train_data.index
```

```
RangeIndex(start=0, stop=891, step=1)
```

```
train_data.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

4. 원하는 column, row만 선택하기
train_data.head() 와 train_data['Survived'] 의 차이점을 비교해 보자.
//원하는 column을 가져올때는 [ ]안에 컬럼의 이름을 명시해야 한다.

train_data[['Survived','Age','Name']] //복수의 컬럼 지정. 바깥쪽 [ ]는 컬럼, 안쪽 [ ]는 list이다.

>>10행까지 슬라이싱으로 가져오기
train_data[:10] //slicing은 row에 대하여 작동한다. [0]은 컬럼을 의미하므로 오류난다. 즉 컬럼에는 slicing이 되지 않는다.

>> 7~9행까지 슬라이싱으로 가져오기
train_data[7:10] -> train_data[800:] -> train_data[-100:-1]

>> loc : 인덱스 자체를 이용, iloc : 0 based index로 사용
train_data.info() //데이터의 개수확인 (891개이므로)
train_data.index = np.arange(100,991) //index 변경(100+891)
train_data.tail() //끝의 데이터를 보면 990인 것 확인
train_data.loc[986] //해당 row를 출력
train_data.loc[986:990] //list형태로 출력
train_data.loc[[100,110],['Survived', 'Name']]
//row와 column까지도 가져올 수 있다.

```
train_data.iloc[0:5]                    //iloc은 보이는데로가 아닌 내부적으로 기억된 순서이용
train_data.iloc[[0,10,100],[1,4,5]]     //row와 column까지도 가져올 수 있다.
```