Pandas_3 >>

1. DataFrame Boolean Selection으로 데이터 선택하기

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

//데이터 가져오기

//data 분석하기

- 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)

describe() //count에서 개수가 빠진 것을 확인

값만 출력

처음20행중 이름, 성별, 나이 출력

train_d[:20][['Name','Sex', 'Age']]

train_d.loc[:120][['Name','Sex','Age']]

train_d.iloc[:20][['Name','Sex','Age']]

train_d.loc[[200,300],['Name','Sex','Age']]

train_d.iloc[[100,200],[3,4,5]]

>> train.csv에서 30대이면서 1등석에 탑승한 승객만 추출하기

train_data['Pclass']==1 //1등석의 데이터만 추출해 보자

```
c = train_data['Pclass']==1
a = (train_data['Age'] >=30) & (train_data['Age']< 40)
train_data[c & a]
```

train_d[(train_d['Pclass']==1) & (train_d['Age']>=30) & (train_d['Age']<40)]

//Boolean selection도 row를 추출할 때 사용함을 알 수 있다.

2. DataFrame에 새 column(컬럼) 추가 & 삭제하기

>> []에 새 column추가하기

train_data['Age']*2 //전체데이터에 연산이 일어난다.

```
train_data['Age_double'] = train_data['Age']*2
train_data.head()
```

// 마지막에 Age_double이 추가됨을 알 수 있다.

>> 'Age'와 'Age_double'을 더한 'Age_tripple'를 생성해 보자.

train_d['Age_tripple'] = train_d['Age'] + train_d['Age_double']

>> 맨 마지막에 열이 추가되는 것을 볼 수 있다. 중간에 열을 추가하고싶으면 'loc'속성을 이용한다.

```
train_data.insert(3,'Fare10', train_data['Fare']/10)
train_data.head()
```

***row추가는 append로 추가한다.

a= train_d[:1]

train_d.append(a,ignore_index=True)

3. column 삭제하기 : drop >> axis 값 이용 : 2차원이므로 0(행), 1(열)

train_data.drop('Age_double', axis=1)

//결과를 보면 삭제된 것을 볼 수 있다.

axis =0 (기본) : index 삭제, 1 : column삭제

>> 다시확인위해

train_data //그대로 인 것을 볼 수 있다. 왜 그런지 이해하기

// inplace=True 는 원본자체를 지운다.

>>복수 개 지우기

```
train_data.drop(['Age_double', 'Age_tripple'], axis=1, inplace=True)
train_data.head()
```

4. DataFrame NaN 데이터 처리

train_data.head() // NaN값 확인한다.

>> NaN값의 개수확인하기

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Fare10         891 non-null float64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(3), int64(5), object(5)
memory usage: 90.6+ KB
```

//값이 없는 것을 확인해 보라.

```
train_data.isna()
```

	PassengerId	Survived	Pclass	Fare10	Name	Se
0	False	False	False	False	False	Fals
1	False	False	False	False	False	Fals
2	False	False	False	False	False	Fals
3	False	False	False	False	False	Fals

// True인 것은 NaN 인 것이다.

train_d['Age'] //age값확인

```
train_data['Age'].isna()
```

```
0      False
1      False
2      False
3      False
4      False
```

//특정 column의 값만 확인하기

train_d[train_d['Age'].isna()==True] //177개나 있다.

다른함수> pd.isnull(train_d['Age']) //NaN값 true로 반환

pd.notnull(train_d['Age']) //NaN값 false로 반환

//개수알아보기

★ len(train_data[train_data['Age'].isna()==True]) //NaN개수확인하기

다른함수> len(train_d[pd.isnull(train_d['Age'])==True])

pd.isnull(train_d['Age']).sum() //함수마다 리턴값의 차이에 따라 다름다.

>> NaN값을 삭제하기

train2 = train_d.dropna() //원본유지, 기본적으로 row기반으로 동작한다.

train2 //개수확인
train3=train_data.dropna(subset=['Age']) //특정 column에 NaN 이 있는 것만 삭제
train3 //개수확인
train4=train_data.dropna(axis=1)//column중 하나라도 NaN이 있으면 column자체를 삭제
train4 //열이 사라진 것 확인

★ train_data.dropna(axis=1, inplace=True)
//원본이 바뀐 것을 확인한다.

>>NaN값을 지우는 것이 아니라 다른값으로 대체하기
누락된 데이터(결측지, NaN, NaT:Not a time)
numpy의 NaN과의 연산은 모두 NaN으로
pandas의 NaN은 값이 비어서 나오기 때문에 값을 채워줘서 처리해야 한다.

1. 나이의 NaN(결측치)를 평균으로 채우기 위하여
train_d.describe() //파일정보확인 Age의 평균 :29.699 확인
또는 train_data['Age'].mean() //Age의 평균구하기

2. 나이에 평균중 결측지를 평균으로 채워넣는다.
train_d['Age'].fillna(train_d['Age'].mean(), inplace=True) //NaN을 평균값으로 채운다
train_d[10:20] //결과확인해 보기

>>'Cabin'값도 NaN을 공백으로 바꿔본다.
train_d['Cabin'].fillna('', inplace=True)

2. 'Age'의 NaN(결측치)를 0으로 채움
파일다시 셋팅
train_d['Age'].fillna(0, inplace = True)
train_d.describe() //파일정보확인 Age의 평균 :23.7992 로 내려간 것 확인

>> 나머지 열에 대해서도 NaN값을 숫자는 평균으로 글자는 공백으로 채워본다.

>> 생존자의 나이는 생존자평균으로 / 사망자는 사망자 평균으로 NaN처리하기
train_data[train_data['Survived']==1] //생존자만 보기
train_d[train_d['Survived']==1]['Age'].mean() //생존자 나이 평균구하기 28.34
train_d[train_d['Survived']==0]['Age'].mean() //사망자 나이 평균구하기 30.62

>> 최종
mean1= train_d[train_d['Survived']==1]['Age'].mean()
mean0 = train_d[train_d['Survived']==0]['Age'].mean()
print(mean1, mean0) //확인

train_d[train_d['Survived']==1]['Age'].fillna(mean1)
train_d[train_d['Survived']==0]['Age'].fillna(mean0)

>>실제 데이터에 적용시키기 (값을 바꿔서 넣는다)
train_d.loc[train_d['Survived'] ==1, 'Age'] =
train_d[train_d['Survived']==1]['Age'].fillna(mean1) //28.34로 변함

생존자만 확인하기
train_d.loc[train_d['Survived']==1] //너무많으니

//Age가 Age의 평균과 같은 것만 봐도 생존자만 있는 것을 확인할 수 있다.
train_d.loc[train_d['Age']==mean1]

//사망자의 경우도 NaN 값을 사망자의 평균으로 바꿔넣기
train_d.loc[train_d['Survived'] ==0, 'Age'] =
train_d.loc[train_d['Survived']==0,'Age',].fillna(mean0) //30.62로 변함

//결과학인하기
train_d.loc[train_d['Survived']==0]
train_d.loc[train_d['Age']==mean0]

3. replace 이용 치환
//NaN(결측치)를 다른값으로 치환 (replace(a,b) : a를 b로 치환
train_d.replace(np.nan, 0, inplace=True)
train_d.info()
train_d

//특정열만 치환
data 다시 셋팅
train_d['Cabin'].replace(np.nan, '', inplace = True)

5. 데이터 type 바꾸기
>> ‘train.csv’ 파일의 데이터를 확인하고, info()함수로 type을 확인한다.
>> ‘Pclass'의 type은 무엇인가? int64
>> 'Pclass'의 type을 문자형태로 변환해 본다(stype사용하여 간단히 타입변환하기)

```
train_data['Pclass'] = train_data['Pclass'].astype(object)
train_data.info()                  //type 변환된 것 확인해 본다.
```

>> ‘Pclass'의 type은 무엇인가? object

5.1. 데이터 값 변경하기

>> 세분화된 Age에 대하여 20대는 모두 20으로 30대는 모두 30으로 바꾸기...
  소수자릿수 없애는 함수 floor()
생각=> 29/10-> 2 *10 ->20
★ 특정한 컬럼에 대하여 어떤 함수를 모두 적용시키고자 할 때 apply()
//함수를 먼저 만든다.

```
import math
import numpy as np
```

```
def age_categorize(age):
    return int(age/10)*10
```

```
train_d['Age'].apply(age_categorize)
```

nan값 때문에 오류발생

// Age컬럼의 값들을 age_categorize함수에 모두 적용시킨다. 에러의 이유를 살펴보자
NaN 값이 있어서 그렇다.

//함수를 수정 후 값을 확인해 본다.

```
def age_categorize(age):
    if pd.isnull(age):
        return -1
    return int(age/10)*10
```

NaN은 모두 -1, 그 외는 처리 완료되었다. 파이썬은 floor함수대신 age//10*10 처리가능하다.

```
train_d['Age']= train_d['Age'].apply(age_categorize)
```

```
train_d
```

//nan은 -1, 나머지는 모두 변경됨을 확인한다.

6. 연산이 불가능한 범주형(object형인 성별, 승서클래스, 승선항구명같은)데이터를 전처리하기(one-hot encoding)

- 범주형 데이터는 분석단계에서 계산이 어렵기 때문에 숫자형으로 변경이 필요함
- 범주형 데이터의 각 범주(category)를 column레벨로 변경
- 해당 범주에 해당하면 1, 아니면 0으로 채우는 인코딩기법
- pandas.get_dummies함수 이용

>> pd.get_dummies(train_data)

//범주형데이터를 컬럼으로 하나하나 올린다. 해당 값만 1로 넣는다. 이것이 ono-hot encoding이라 한다.

>> pd.get_dummies(train_data, columns=['Pclass', 'Sex', 'Embarked'])
//원하는 열만 바꾼다.

제외//
>>pd.get_dummies(train_data, columns=['Pclass', 'Sex', 'Embarked'],drop_first="True")
//drop_first -> 값을 유추할 수 있는 변수는 만들지 않는다. 기본은 False이다