




Day46; 20221110

	날짜
	유형
	태그

GitHub - u8yes/AI

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

 <https://github.com/u8yes/AI>

u8yes/AI



 1 Contributor
 0 Issues
 0 Stars
 0 Forks



dataFrame은 2차원 프레임이라서 행렬구조를 가진다.

row 드롭하기_20221110

- row 인덱스로 row를 drop할 수 있습니다.

```
15]: friend_dict_list = [{'age':20, 'job':'student'},
                        {'age':30, 'job':'developer'},
                        {'age':25, 'job':'teacher'}]

df = pd.DataFrame(friend_dict_list, index=['John', 'Jenny', 'Nate'])
df.head()
```

```
15]:
```

	age	job
John	20	student
Jenny	30	developer
Nate	25	teacher

```
19]: df.drop(['John', 'Nate']) # 수행하고 난 결과만 리턴해줌. # drop된 결과는 데이터프레임에 저장되지 않음
# 저장하고 싶을 경우, 결과를 데이터프레임에 따로 저장해야 됨.
```

```
19]:
```

	age	job
Jenny	30	developer

```
20]: df
```

```
20]:
```

	age	job
John	20	student
Jenny	30	developer
Nate	25	teacher

```
21]: df = df.drop(['John', 'Nate']) # 내 자신에게 업데이트를 하면 반영된다.
df
```

```
21]:
```

	age	job
Jenny	30	developer

drop된 결과를 바로 데이터프레임에 저장하는 방법

- inplace 키워드를 사용하면, 따로 저장할 필요없이 drop된 결과가 데이터프레임에 반영됨.

```
[23]: friend_dict_list = [{'age':20, 'job':'student'},
                        {'age':30, 'job':'developer'},
                        {'age':25, 'job':'teacher'}]

df = pd.DataFrame(friend_dict_list, index=['John', 'Jenny', 'Nate'])
df.head()
```

t[23]:

	age	job
John	20	student
Jenny	30	developer
Nate	25	teacher

```
[24]: df.drop(['John', 'Nate'], inplace=True) # 결과를 내 자신에게 반영해줄 = inplace
df
```

t[24]:

	age	job
Jenny	30	developer

```
[31]: friend_dict_list = [{'name':'Jane', 'age':20, 'job':'student'},
                        {'name':'Jenny', 'age':30, 'job':'developer'},
                        {'name':'Nate', 'age':25, 'job':'teacher'}]

df = pd.DataFrame(friend_dict_list)
df.head()
```

t[31]:

	name	age	job
0	Jane	20	student
1	Jenny	30	developer
2	Nate	25	teacher

```
[32]: # row index로 drop하는 방법
df = df.drop(df.index[[0,2]]) # loc, iloc가 가장 많이 쓰이는 연산자
df
```

[32]:

	name	age	job
1	Jenny	30	developer

컬럼값으로 row drop 하기

```
In [41]: friend_dict_list = [{'name': 'Jane', 'age': 20, 'job': 'student'},  
                             {'name': 'Jenny', 'age': 30, 'job': 'developer'},  
                             {'name': 'Nate', 'age': 25, 'job': 'teacher'}]  
  
df = pd.DataFrame(friend_dict_list)  
df.head()
```

Out [41]:

	name	age	job
0	Jane	20	student
1	Jenny	30	developer
2	Nate	25	teacher

```
In [42]: df[df.age != 30]
```

Out [42]:

	name	age	job
0	Jane	20	student
2	Nate	25	teacher

```
In [43]: friend_dict_list = [{'name': 'Jane', 'age': 20, 'job': 'student'},  
                             {'name': 'Jenny', 'age': 30, 'job': 'developer'},  
                             {'name': 'Nate', 'age': 25, 'job': 'teacher'}]  
  
df = pd.DataFrame(friend_dict_list)  
df.head()
```

Out [43]:

	name	age	job
0	Jane	20	student
1	Jenny	30	developer
2	Nate	25	teacher

```
In [44]: df.drop('age', axis=1) # axis가 default로 0으로 돼있다.
```

Out [44]:

	name	job
0	Jane	student
1	Jenny	developer
2	Nate	teacher

...

컬럼 추가 변경하기

In [46]:

```
df
```

Out [46]:

	name	age	job
0	Jane	20	student
1	Jenny	30	developer
2	Nate	25	teacher

In [48]:

```
df['salary'] = 0  
df
```

Out [48]:

	name	age	job	salary
0	Jane	20	student	0
1	Jenny	30	developer	0
2	Nate	25	teacher	0

In [49]:

```
df = df.drop('salary', axis=1) # salary 컬럼 삭제  
df
```

Out [49]:

	name	age	job
0	Jane	20	student
1	Jenny	30	developer
2	Nate	25	teacher

```

]]: # 넘파이를 이용해서 한줄에 새로운 컬럼값을 생성
import numpy as np

df['salary'] = np.where(df['job'] != 'student', 'yes', 'no') # 파생변수('yes', 'no' 등)를 새로 추가해줌.
df

```

```

]]:

```

	name	age	job	salary
0	Jane	20	student	no
1	Jenny	30	developer	yes
2	Nate	25	teacher	yes

```

]]: friend_dict_list = [{'name': 'John', 'midterm': 95, 'final': 85},
                        {'name': 'Jenny', 'midterm': 85, 'final': 80},
                        {'name': 'Nate', 'midterm': 75, 'final': 95},
                        {'name': 'Brian', 'midterm': 55, 'final': 55}]

score_df = pd.DataFrame(friend_dict_list, columns=['name', 'midterm', 'final'])
score_df

```

```

]]:

```

	name	midterm	final
0	John	95	85
1	Jenny	85	80
2	Nate	75	95
3	Brian	55	55

```

]]: # 파생 변수
score_df['total'] = score_df['midterm'] + score_df['final']
score_df

```

```

]]:

```

	name	midterm	final	total
0	John	95	85	180
1	Jenny	85	80	165
2	Nate	75	95	170
3	Brian	55	55	110

```

: # 평균
score_df['average'] = score_df['total'] / 2
score_df

```

```

:

```

	name	midterm	final	total	average
0	John	95	85	180	90.0
1	Jenny	85	80	165	82.5
2	Nate	75	95	170	85.0
3	Brian	55	55	110	55.0

```

: # 리스트에 조건별 값을 담아서, 새로운 컬럼으로 추가 시킬 수 있다.

```

```

grades = []

for row in score_df['average']:
    if row >= 90:
        grades.append('A')
    elif row >= 80:
        grades.append('B')
    elif row >= 70:
        grades.append('C')
    elif row >= 60:
        grades.append('D')
    else:
        grades.append('F')

```

```

score_df['grade'] = grades
score_df

```

```

:

```

	name	midterm	final	total	average	grade
0	John	95	85	180	90.0	A
1	Jenny	85	80	165	82.5	B
2	Nate	75	95	170	85.0	B
3	Brian	55	55	110	55.0	F

apply() 사용 예제

- 컬럼의 값을 변경하는 코드 구현.

```
In [73]: # 값의 수정

def pass_or_fail(row):
    if row != 'F':
        return 'Pass'
    else:
        return 'Fail'

score_df.grade = score_df.grade.apply(pass_or_fail) # 함수만 넣어주면 됨.
score_df
```

Out [73]:

	name	midterm	final	total	average	grade
0	John	95	85	180	90.0	Pass
1	Jenny	85	80	165	82.5	Pass
2	Nate	75	95	170	85.0	Pass
3	Brian	55	55	110	55.0	Fail

```
]]: # 연월일의 정보에서 연도만 추출하는 예제
```

```
date_list = [{'yyyy-mm-dd': '1984-06-21'},  
             {'yyyy-mm-dd': '1982-03-18'},  
             {'yyyy-mm-dd': '2015-11-16'}]
```

```
date_df = pd.DataFrame(date_list, index=['민용기', '이수향', '결혼기념일'], columns=['yyyy-mm-dd'])  
date_df
```

```
]]:
```

	yyyy-mm-dd
민용기	1984-06-21
이수향	1982-03-18
결혼기념일	2015-11-16

```
]]: def extract_year(row):  
     return row.split('-')[0]
```

```
def extract_month(row):  
     return row.split('-')[1]
```

```
def extract_day(row):  
     return row.split('-')[2]
```

```
]]: date_df['year'] = date_df['yyyy-mm-dd'].apply(extract_year)  
date_df['month'] = date_df['yyyy-mm-dd'].apply(extract_month)  
date_df['day'] = date_df['yyyy-mm-dd'].apply(extract_day)  
date_df
```

```
]]:
```

	yyyy-mm-dd	year	month	day
민용기	1984-06-21	1984	06	21
이수향	1982-03-18	1982	03	18
결혼기념일	2015-11-16	2015	11	16

apply()에 파라미터 전달하기

- 키워드 파라미터를 사용하면, apply가 적용된 함수에 파라미터를 전달할 수 있다.

```
] : def extract_year(year, current_year):  
    return current_year - int(year)
```

```
] : date_df['age'] = date_df['year'].apply(extract_year, current_year = 2022)  
date_df
```

```
] :
```

	yyyy-mm-dd	year	month	day	age
민용기	1984-06-21	1984	06	21	38
이수향	1982-03-18	1982	03	18	40
결혼기념일	2015-11-16	2015	11	16	7

```
] : def get_introduce(age, prefix, suffix):  
    return prefix + str(age) + suffix
```

```
] : date_df['introduce'] = date_df['age'].apply(get_introduce,  
                                              prefix='I am ',  
                                              suffix=' years old')  
date_df
```

```
] :
```

	yyyy-mm-dd	year	month	day	age	introduce
민용기	1984-06-21	1984	06	21	38	I am 38 years old
이수향	1982-03-18	1982	03	18	40	I am 40 years old
결혼기념일	2015-11-16	2015	11	16	7	I am 7 years old

```
] : # 여러 개의 컬럼을 동시에 전달하기  
# - axis=1 이라는 키워드 파라미터를 apply()에 전달해 주면, 모든 컬럼을 지정된 함수에서 사용 가능  
  
def get_introduce2(row):  
    return 'I was born in ' + str(row.year) + ' my age is ' + str(row.age)  
  
date_df.introduce = date_df.apply(get_introduce2, axis=1)  
date_df
```

```
] :
```

	yyyy-mm-dd	year	month	day	age	introduce
민용기	1984-06-21	1984	06	21	38	I was born in 1984 my age is 38
이수향	1982-03-18	1982	03	18	40	I was born in 1982 my age is 40
결혼기념일	2015-11-16	2015	11	16	7	I was born in 2015 my age is 7

map()으로 컬럼 추가 및 변경하기

```
1] : date_list = [{'yyyy-mm-dd': '1984-06-21'},
                  {'yyyy-mm-dd': '1982-03-18'},
                  {'yyyy-mm-dd': '2015-11-16'}]

date_df = pd.DataFrame(date_list, index=['민용기', '이수향', '결혼기념일'], columns=['yyyy-mm-dd'])
date_df
```

```
2] :
```

	yyyy-mm-dd
민용기	1984-06-21
이수향	1982-03-18
결혼기념일	2015-11-16

```
3] : date_df['year'] = date_df['yyyy-mm-dd'].map(extract_year) # apply 메서드도같이 컬럼값을 추가해줄 수 있음.
date_df
```


```
4] :
```

	yyyy-mm-dd	year
민용기	1984-06-21	1984
이수향	1982-03-18	1982
결혼기념일	2015-11-16	2015

EduAtoZ)

EduAtoZ

빅분기 5회 학습자 분들은 '구글 클래스 룸'에서 문제를 풀이하실 수 있습니다. 1과목 완료, 2과목 등록 중입니다. 화이팅~~!!

 <https://www.eduatoz.kr/lecture-detail?uuid=1d309976-dd91-41c3-8562-7feeb9d2a057&scope=SALES&campusUuid=33334aa5-b195-4628-a6ad-633effea2857&classUuid=32c7c2fd-0b21-4b34-a193-cf528c67ce87>



파라미터로 딕셔너리를 전달하면 컬럼값을 쉽게 원하는 값으로 변경 가능.

기존의 컬럼값은 딕셔너리의 **Key**로 사용되고, 해당되는 **value**의 값으로 컬럼값이 변경

```
: job_list = [{'age':20, 'job':'student'},  
              {'age':30, 'job':'developer'},  
              {'age':35, 'job':'teacher'}]  
  
df = pd.DataFrame(job_list)  
df
```

```
:  
  
   age  job  
0   20 student  
1   30 developer  
2   35 teacher
```

```
: df.job = df.job.map({'student':1, 'developer':2, 'teacher':3}) # 내용 변경이 가능.  
df
```

```
:  
  
   age  job  
0   20    1  
1   30    2  
2   35    3
```

applymap() 메서드

- 데이터 프레임 전체의 각각의 값을 한 번에 변경시킬 때 사용하면 유용한 메서드
- `np.around(data)` : 소수점 자리 수를 반올림해서 정수로 만들어줌
- `np.round(data, decimals)` : 원하는 소수점 자리수에서 반올림

```
7]: x_y = [{'x':5.5, 'y':-5.6},  
          {'x':-5.2, 'y':5.5},  
          {'x':-1.6, 'y':-4.5}]
```

```
df = pd.DataFrame(x_y)  
df
```

7]:

	x	y
0	5.5	-5.6
1	-5.2	5.5
2	-1.6	-4.5

```
8]: df = df.applymap(np.around) # 전체가 변경됨.  
df
```

8]:

	x	y
0	6.0	-6.0
1	-5.0	6.0
2	-2.0	-4.0

데이터프레임에 row 추가하기

```
2]: friend_dict_list = [{'name': 'John', 'midterm': 95, 'final': 85},
                        {'name': 'Jenny', 'midterm': 85, 'final': 80},
                        {'name': 'Nate', 'midterm': 75, 'final': 95},
                        {'name': 'Brian', 'midterm': 55, 'final': 55}]

score_df = pd.DataFrame(friend_dict_list, columns=['name', 'midterm', 'final'])
score_df
```

```
2]:
```

	name	midterm	final
0	John	95	85
1	Jenny	85	80
2	Nate	75	95
3	Brian	55	55

```
3]: df2 = pd.DataFrame([['Ben', 50, 50]], columns=['name', 'midterm', 'final'])
df2
```

```
3]:
```

	name	midterm	final
0	Ben	50	50

```
4]: score_df.append(df2, ignore_index=True) # ignore_index 기존의 인덱스를 무시하고 새 인덱스 4에 넣어줌.
```

```
4]:
```

	name	midterm	final
0	John	95	85
1	Jenny	85	80
2	Nate	75	95
3	Brian	55	55
4	Ben	50	50

groupby() 함수

- 데이터에서 정보를 취하기 위해서 그룹별로 묶는 방법.

```
48]: student_list = [{'name': 'John', 'major': "Computer Science", 'sex': "male"},
                    {'name': 'Nate', 'major': "Computer Science", 'sex': "male"},
                    {'name': 'Abraham', 'major': "Physics", 'sex': "male"},
                    {'name': 'Brian', 'major': "Psychology", 'sex': "male"},
                    {'name': 'Janny', 'major': "Economics", 'sex': "female"},
                    {'name': 'Yuna', 'major': "Economics", 'sex': "female"},
                    {'name': 'Jeniffer', 'major': "Computer Science", 'sex': "female"},
                    {'name': 'Edward', 'major': "Computer Science", 'sex': "male"},
                    {'name': 'Zara', 'major': "Psychology", 'sex': "female"},
                    {'name': 'Wendy', 'major': "Economics", 'sex': "female"},
                    {'name': 'Sera', 'major': "Psychology", 'sex': "female"}
                    ]

df = pd.DataFrame(student_list, columns=['name', 'major', 'sex'])
df
```

```
48]:
```

	name	major	sex
0	John	Computer Science	male
1	Nate	Computer Science	male
2	Abraham	Physics	male
3	Brian	Psychology	male
4	Janny	Economics	female
5	Yuna	Economics	female
6	Jeniffer	Computer Science	female
7	Edward	Computer Science	male
8	Zara	Psychology	female
9	Wendy	Economics	female
10	Sera	Psychology	female


```

1]: groupby_major = df.groupby('major')
   groupby_major

2]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000025631316E88>

3]: groupby_major.groups # 인덱스를 그룹별로 정보를 제공

4]: {'Computer Science': [0, 1, 6, 7], 'Economics': [4, 5, 9], 'Physics': [2], 'Psychology': [3, 8, 10]}

5]: for name, group in groupby_major:
   print(name + ':' + str(len(group)))
   print(group)
   print()

Computer Science:4
   name      major  sex
0   John  Computer Science  male
1    Nate  Computer Science  male
6  Jeniffer  Computer Science  female
7   Edward  Computer Science  male

Economics:3
   name      major  sex
4  Janny  Economics  female
5   Yuna  Economics  female
9  Wendy  Economics  female

Physics:1
   name      major  sex
2 Abraham  Physics  male

Psychology:3
   name      major  sex
3  Brian  Psychology  male
8   Zara  Psychology  female
10  Sera  Psychology  female

```

그룹 객체를 다시 데이터프레임으로 생성.

```

df_major_cnt = pd.DataFrame({'count':groupby_major.size()}).reset_index()
df_major_cnt

```

그룹 객체를 다시 데이터프레임으로 생성.

```

df_major_cnt = pd.DataFrame({'count':groupby_major.size()})
df_major_cnt

```

	count
major	
Computer Science	4
Economics	3
Physics	1
Psychology	3

그룹 객체를 다시 데이터프레임으로 생성.

```

df_major_cnt = pd.DataFrame({'count':groupby_major.size()}).reset_index()
df_major_cnt

```

	major	count
0	Computer Science	4
1	Economics	3
2	Physics	1
3	Psychology	3

```
34]: groupby_sex = df.groupby('sex')
groupby_sex
```

```
34]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002563117F5C8>
```

```
35]: for name, group in groupby_sex:
      print(name + ':' + str(len(group)))
      print(group)
      print()
```

female:6

	name	major	sex
4	Janny	Economics	female
5	Yuna	Economics	female
6	Jeniffer	Computer Science	female
8	Zara	Psychology	female
9	Wendy	Economics	female
10	Sera	Psychology	female

male:5

	name	major	sex
0	John	Computer Science	male
1	Nate	Computer Science	male
2	Abraham	Physics	male
3	Brian	Psychology	male
7	Edward	Computer Science	male

```
36]: df_major_cnt = pd.DataFrame({'count':groupby_sex.size()}).reset_index()
df_major_cnt
```

```
36]:
```

	sex	count
0	female	6
1	male	5

중복 데이터 drop 하기

```
: df
```

```
:
```

	name	major	sex
0	John	Computer Science	male
1	Nate	Computer Science	male
2	Abraham	Physics	male
3	Brian	Psychology	male
4	Janny	Economics	female
5	Yuna	Economics	female
6	Jeniffer	Computer Science	female
7	Edward	Computer Science	male
8	Zara	Psychology	female
9	Wendy	Economics	female
10	Sera	Psychology	female

```
: # 중복 데이터 삽입  
df2 = pd.DataFrame(['Zara', 'Psychology', 'female'], columns=['name', 'major', 'sex'])  
df = df.append(df2, ignore_index=True)  
df
```

```
7]: # 중복 데이터 삽입
df2 = pd.DataFrame(['Zara', 'Psychology', 'female'], columns=['name', 'major', 'sex'])
df = df.append(df2, ignore_index=True)
df
```

```
7]:
```

	name	major	sex
0	John	Computer Science	male
1	Nate	Computer Science	male
2	Abraham	Physics	male
3	Brian	Psychology	male
4	Janny	Economics	female
5	Yuna	Economics	female
6	Jeniffer	Computer Science	female
7	Edward	Computer Science	male
8	Zara	Psychology	female
9	Wendy	Economics	female
10	Sera	Psychology	female
11	Zara	Psychology	female

```
3]: # 중복 데이터 확인하기
df.duplicated()
```

```
3]: 0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    False
10   False
11    True
dtype: bool
```

```

: # 중복 데이터를 삭제
df = df.drop_duplicates()
df

```

```

:

```

	name	major	sex
0	John	Computer Science	male
1	Nate	Computer Science	male
2	Abraham	Physics	male
3	Brian	Psychology	male
4	Janny	Economics	female
5	Yuna	Economics	female
6	Jeniffer	Computer Science	female
7	Edward	Computer Science	male
8	Zara	Psychology	female
9	Wendy	Economics	female
10	Sera	Psychology	female

cated (x) cates(0)

```
6]: student_list = [{'name': 'John', 'major': "Computer Science", 'sex': "male"},
                    {'name': 'Nate', 'major': "Computer Science", 'sex': "male"},
                    {'name': 'Abraham', 'major': "Physics", 'sex': "male"},
                    {'name': 'Brian', 'major': "Psychology", 'sex': "male"},
                    {'name': 'Janny', 'major': "Economics", 'sex': "female"},
                    {'name': 'Yuna', 'major': "Economics", 'sex': "female"},
                    {'name': 'Jeniffer', 'major': "Computer Science", 'sex': "female"},
                    {'name': 'Edward', 'major': "Computer Science", 'sex': "male"},
                    {'name': 'Zara', 'major': "Psychology", 'sex': "female"},
                    {'name': 'Wendy', 'major': None, 'sex': "female"},
                    {'name': 'Sera', 'major': "Psychology", 'sex': None},
                    {'name': 'John', 'major': "Computer Science", 'sex': None},
                    {'name': 'Nate', 'major': None, 'sex': "male"}
                    ]

df = pd.DataFrame(student_list, columns=['name', 'major', 'sex'])
df
```

6]:

	name	major	sex
0	John	Computer Science	male
1	Nate	Computer Science	male
2	Abraham	Physics	male
3	Brian	Psychology	male
4	Janny	Economics	female
5	Yuna	Economics	female
6	Jeniffer	Computer Science	female
7	Edward	Computer Science	male
8	Zara	Psychology	female
9	Wendy	None	female
10	Sera	Psychology	None
11	John	Computer Science	None
12	Nate	None	male

```
] : df.duplicated()
```

```
] : 0    False
    1    False
    2    False
    3    False
    4    False
    5    False
    6    False
    7    False
    8    False
    9    False
    10   False
    11   False
    12   False
    dtype: bool
```

```
In [188]: # name 컬럼이 똑같은 경우, 중복된 데이터라고 표시
df.duplicated(['name'])
```

```
Out [188]: 0    False
           1    False
           2    False
           3    False
           4    False
           5    False
           6    False
           7    False
           8    False
           9    False
          10    False
          11     True
          12     True
           dtype: bool
```

```
] : # Keep 값을 first 또는 last라고 값을 써서 중복된 값 중, 어느 값을 살릴지 결정.
df.drop_duplicates(['name'])
```

```
] :
```

	name	major	sex
0	John	Computer Science	male
1	Nate	Computer Science	male
2	Abraham	Physics	male
3	Brian	Psychology	male
4	Janny	Economics	female
5	Yuna	Economics	female
6	Jeniffer	Computer Science	female
7	Edward	Computer Science	male
8	Zara	Psychology	female
9	Wendy	None	female
10	Sera	Psychology	None

중복된 것을 전체다 부르는 keep=False

keep = 'last' 지웠던 중복값을 다시 살리느냐

```
1]: # name 컬럼이 똑같은 경우, 중복된 데이터라고 표시
df.duplicated(['name'], keep=False)
```

```
1]: 0      True
     1      True
     2     False
     3     False
     4     False
     5     False
     6     False
     7     False
     8     False
     9     False
    10     False
    11      True
    12      True
     dtype: bool
```

```
0]: # Keep 값을 first 또는 last라고 값을 써서 중복된 값 중, 어느 값을 살릴지 결정.
```

```
df.drop_duplicates(['name'], keep = 'last') # keep = 'last' # 나중에 중복되서 버린 데이터를 반환시켜줄
# keep = 디폴트값은 'first'로 돼있음.
```

```
0]:
```

	name	major	sex
2	Abraham	Physics	male
3	Brian	Psychology	male
4	Janny	Economics	female
5	Yuna	Economics	female
6	Jeniffer	Computer Science	female
7	Edward	Computer Science	male
8	Zara	Psychology	female
9	Wendy	None	female
10	Sera	Psychology	None
11	John	Computer Science	None
12	Nate	None	male

None 처리하기_20221110

[193]: # Null 또는 NaN 확인하기

```
friend_dict_list = [{'name': 'Jane', 'age': 20, 'job': 'student'},  
                    {'name': 'Jenny', 'age': 30, 'job': 'developer'},  
                    {'name': 'Nate', 'age': None, 'job': 'teacher'},  
                    {'name': 'Brian', 'age': 12, 'job': 'teacher'}]  
  
df = pd.DataFrame(friend_dict_list)  
df.head()
```

t[193]:

	name	age	job
0	Jane	20.0	student
1	Jenny	30.0	developer
2	Nate	NaN	teacher
3	Brian	12.0	teacher

[194]: df.info()

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 4 entries, 0 to 3  
Data columns (total 3 columns):  
#   Column  Non-Null Count  Dtype  
---  ---  
0   name    4 non-null      object  
1   age     3 non-null      float64  
2   job     4 non-null      object  
dtypes: float64(1), object(2)  
memory usage: 224.0+ bytes
```

```
5]: df.describe()
```

```
5]:
```

	age
count	3.000000
mean	20.666667
std	9.018500
min	12.000000
25%	16.000000
50%	20.000000
75%	25.000000
max	30.000000

```
5]: df.isna()
```

```
5]:
```

	name	age	job
0	False	False	False
1	False	False	False
2	False	True	False
3	False	False	False

```
1.
```

```
]: df.isnull()
```

```
]:
```

	name	age	job
0	False	False	False
1	False	False	False
2	False	True	False
3	False	False	False

Null 또는 NaN 값 변경.

[199]: # Null을 0으로 설정 예제.

```
tmp = df
tmp['age'] = tmp['age'].fillna(0)
tmp
```

[199]:

	name	age	job
0	Jane	20.0	student
1	Jenny	30.0	developer
2	Nate	0.0	teacher
3	Brian	12.0	teacher

[205]: # 평균(중위수)을 계산해서 Null 값을 대체.

```
df['age'] = df['age'].fillna(df.groupby('job')['age'].transform('median'), inplace=True) # inplace 자식메다가 있어서 업데이트해줌
# 선생님 직업을 가지고 있는 것만 뽑아서 중위수 값을 뽑아줌, # 직군별로
df
```

[205]:

	name	age	job
0	Jane	20.0	student
1	Jenny	30.0	developer
2	Yuna	20.0	teacher
3	Nate	16.0	teacher
4	Brian	12.0	teacher

Unique

- 컬럼에 여러 값이 있을 때, 중복없이 어떤 값들이 있는지 확인하는 방법.

```
08]: job_list = [{'name': 'John', 'job': 'teacher'},
                {'name': 'Nate', 'job': 'teacher'},
                {'name': 'Fred', 'job': 'teacher'},
                {'name': 'Abraham', 'job': 'student'},
                {'name': 'Brian', 'job': 'student'},
                {'name': 'Janny', 'job': 'developer'},
                {'name': 'Nate', 'job': 'teacher'},
                {'name': 'Obrian', 'job': 'dentist'},
                {'name': 'Yuna', 'job': 'teacher'},
                {'name': 'Rob', 'job': 'lawyer'},
                {'name': 'Brian', 'job': 'student'},
                {'name': 'Matt', 'job': 'student'},
                {'name': 'Wendy', 'job': 'banker'},
                {'name': 'Edward', 'job': 'teacher'},
                {'name': 'Ian', 'job': 'teacher'},
                {'name': 'Chris', 'job': 'banker'},
                {'name': 'Phillip', 'job': 'lawyer'},
                {'name': 'Janny', 'job': 'basketball player'},
                {'name': 'Gwen', 'job': 'teacher'},
                {'name': 'Jessy', 'job': 'student'}
            ]
df = pd.DataFrame(job_list, columns = ['name', 'job'])
df
```

not

3] :

	name	job
0	John	teacher
1	Nate	teacher
2	Fred	teacher
3	Abraham	student
4	Brian	student
5	Janny	developer
6	Nate	teacher
7	Obrian	dentist
8	Yuna	teacher
9	Rob	lawyer
10	Brian	student
11	Matt	student
12	Wendy	banker
13	Edward	teacher
14	Ian	teacher
15	Chris	banker
16	Philip	lawyer
17	Janny	basketball player
18	Gwen	teacher
19	Jessy	student

```
] : # 컬럼(시리즈)의 unique() 메서드를 사용하여, 중복없이 컬럼에 있는 모든 값들을 출력,  
print(df.job.unique())
```

```
['teacher' 'student' 'developer' 'dentist' 'lawyer' 'banker'  
 'basketball player']
```

```
] : # 각 unique한 값 별로 몇 개의 데이터가 속하는지 value_counts() 함수로 확인  
df.job.value_counts() # 각 열의 빈도수를 보여주게 됨.
```

```
] : teacher      8  
student      5  
lawyer       2  
banker       2  
developer    1  
dentist      1  
basketball player  1  
Name: job, dtype: int64
```

행으로 추가

두 개의 데이터프레임 합치기

```
3]: l1 = [{'name': 'John', 'job': "teacher"},
        {'name': 'Nate', 'job': "teacher"},
        {'name': 'Fred', 'job': "teacher"}]

l2 = [{'name': 'Abraham', 'job': "student"},
        {'name': 'Brian', 'job': "student"},
        {'name': 'Janny', 'job': "developer"}]

df1 = pd.DataFrame(l1, columns=['name', 'job'])
df2 = pd.DataFrame(l2, columns=['name', 'job'])
```

```
4]: # pd.concat() : 두번째 데이터프레임을 첫번째 데이터프레임의 새로운 row(행)으로 합침

frames = [df1, df2]
result = pd.concat(frames, ignore_index=True)
result
```

```
4]:
```

	name	job
0	John	teacher
1	Nate	teacher
2	Fred	teacher
3	Abraham	student
4	Brian	student
5	Janny	developer

```
]: # df.append() : 두번째 데이터프레임을 첫번째 데이터프레임의 새로운 row(행)으로 합침

result = df1.append(df2, ignore_index=True)
result
```

```
]:
```

	name	job
0	John	teacher
1	Nate	teacher
2	Fred	teacher
3	Abraham	student
4	Brian	student
5	Janny	developer

컬럼으로 추가

```
3]: l1 = [{'name': 'John', 'job': "teacher"},
        {'name': 'Nate', 'job': "student"},
        {'name': 'Jack', 'job': "developer"}]

l2 = [{'age': 25, 'country': "U.S"},
        {'age': 30, 'country': "U.K"},
        {'age': 45, 'country': "Korea"}]

df1 = pd.DataFrame(l1, columns=['name', 'job'])
df2 = pd.DataFrame(l2, columns=['age', 'country'])
```

```
7]: result = pd.concat([df1, df2], ignore_index=True, axis=1)
result
```

```
7]:
```

	0	1	2	3
0	John	teacher	25	U.S
1	Nate	student	30	U.K
2	Jack	developer	45	Korea

두 개의 리스트를 묶어서 데이터프레임으로 생성

```
21]: label = [1,2,3,4,5]
prediction = [1,2,2,5,5]

comparision = pd.DataFrame({'label':label,
                            'prediction':prediction
                            })

comparision
```

```
21]:
```

	label	prediction
0	1	1
1	2	2
2	3	2
3	4	5
4	5	5

날짜

0은 첫번째 주를 기준으로 '월요일' 기준만을 출력

- 특정 요일에 해당하는 데이터 조회하기

```
[2]: import pandas as pd

df = pd.read_csv('data/temperatures.csv')
df.head()
```

```
[2]:
```

	Date	Temp
0	1981-01-01	20.7
1	1981-01-02	17.9
2	1981-01-03	18.8
3	1981-01-04	14.6
4	1981-01-05	15.8

```
[3]: ## 문자열을 Datetime으로 변환하기 # Date는 String으로 남아져있음,
df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d') # XXXX년도 표시는 %Y, XX년도 표시는 %y
```

```
[3]: ## 월요일인 데이터 조회
df[df['Date'].dt.dayofweek == 0]
## 또는 df.query('Date.dt.dayofweek == 0')
```

```
[3]:
```

	Date	Temp
4	1981-01-05	15.8
11	1981-01-12	13.3
18	1981-01-19	17.7
25	1981-01-26	18.7
32	1981-02-02	18.8
...
3621	1990-12-03	16.2
3628	1990-12-10	11.0
3635	1990-12-17	13.9
3642	1990-12-24	10.0
3649	1990-12-31	13.0

521 rows × 2 columns


```
In [4]: ## 월, 수, 금에 해당하는 데이터 조회
dayofweek = [0,2,4]
df[df['Date'].dt.dayofweek.isin(dayofweek)]
## 또는 df.query('Date.dt.dayofweek in @dayofweek')
```

Out [4]:

	Date	Temp
1	1981-01-02	17.9
4	1981-01-05	15.8
6	1981-01-07	15.8
8	1981-01-09	21.8
11	1981-01-12	13.3
...
3639	1990-12-21	13.1
3642	1990-12-24	10.0
3644	1990-12-26	14.6
3646	1990-12-28	13.6
3649	1990-12-31	13.0

1564 rows × 2 columns

- 특정 날짜에 해당하는 데이터 조회하기

```
In [5]: ## 특정 날짜
target_date = '1981-01-02'
df[df['Date'] == target_date]
## 또는 df.query('Date == @target_date')
```

Out [5]:

	Date	Temp
1	1981-01-02	17.9

- 특정 연도에 해당하는 데이터 조회하기

```
n [6]: target_year = 1990 # 1990년  
df[df['Date'].dt.year==target_year]  
## 또는 df.query('Date.dt.year == @target_year')
```

ut [6]:

	Date	Temp
3285	1990-01-01	14.8
3286	1990-01-02	13.3
3287	1990-01-03	15.6
3288	1990-01-04	14.5
3289	1990-01-05	14.3
...
3645	1990-12-27	14.0
3646	1990-12-28	13.6
3647	1990-12-29	13.5
3648	1990-12-30	15.7
3649	1990-12-31	13.0

365 rows × 2 columns

```
In [7]: # 특정 월, 일에 해당하는 데이터 조회
target_month = 9 # 9월
df[df['Date'].dt.month==target_month]
## 또는 df.query('Date.dt.month == @target_month')

target_day = 13 # 13일
df[df['Date'].dt.day==target_day]
## 또는 df.query('Date.dt.day == @target_day')
```

Out [7]:

	Date	Temp
12	1981-01-13	16.7
43	1981-02-13	18.3
71	1981-03-13	11.8
102	1981-04-13	13.5
132	1981-05-13	5.3
...
3509	1990-08-13	7.8
3540	1990-09-13	9.8
3570	1990-10-13	7.5
3601	1990-11-13	13.8
3631	1990-12-13	11.4

120 rows × 2 columns

- 특정 날짜 범위에 해당하는 데이터 조회하기

```
8] : # 1981년 1월 1일부터 1981년 1월 4일까지 데이터 조회
df[(df['Date'] >= '1981-01-01') & (df['Date'] <= '1981-01-04')]
## 또는 df.query('1981-01-01 <= Date <= "1981-01-04"')
```

8] :

	Date	Temp
0	1981-01-01	20.7
1	1981-01-02	17.9
2	1981-01-03	18.8
3	1981-01-04	14.6

```
9] : # 1987년부터 1989년까지의 데이터 조회
df[(df['Date'].dt.year >= 1987) & (df['Date'].dt.year <= 1989)]
## 또는 df.query('1987 <= Date.dt.year <= 1989')
```

9] :

	Date	Temp
2190	1987-01-01	12.3
2191	1987-01-02	13.8
2192	1987-01-03	15.3
2193	1987-01-04	15.6
2194	1987-01-05	16.2
...
3280	1989-12-27	13.3
3281	1989-12-28	11.7
3282	1989-12-29	10.4
3283	1989-12-30	14.4
3284	1989-12-31	12.7

1095 rows × 2 columns

- 그룹화 하기

```
] : ## 연도별로 그룹화하기
df['Year'] = df['Date'].dt.year
df.groupby('Year')['Temp'].aggregate(['max', 'min', 'mean']).reset_index()
```

```
] :
```

	Year	max	min	mean
0	1981	25.0	2.1	11.517260
1	1982	26.3	0.0	10.783562
2	1983	22.5	0.0	11.187397
3	1984	24.3	0.1	10.591781
4	1985	22.4	0.3	11.137534
5	1986	21.4	0.8	10.803288
6	1987	24.1	1.5	10.853151
7	1988	23.9	2.8	11.972055
8	1989	22.0	0.5	11.261918
9	1990	22.1	2.1	11.669589

```
[11]: ## 요일별로 그룹화하기
num_to_day = {
    0 : '월요일',
    1 : '화요일',
    2 : '수요일',
    3 : '목요일',
    4 : '금요일',
    5 : '토요일',
    6 : '일요일',
}

df['Day'] = [num_to_day[k] for k in df['Date'].dt.dayofweek] ## 요일 칼럼
df.groupby('Day')['Temp'].aggregate(['max', 'min', 'mean']).reset_index()
```

[11]:

	Day	max	min	mean
0	금요일	23.9	0.7	11.161877
1	목요일	25.0	0.9	11.192720
2	수요일	25.2	0.3	11.337620
3	월요일	26.3	0.7	11.156238
4	일요일	24.8	0.0	11.028544
5	토요일	23.4	0.0	11.114587
6	화요일	22.5	0.2	11.252975