2025-04-07(Mon.) : 파이썬 라이브러리와 활용 2일차

- pandas documentation
  - API reference
- The Python Tutorial (v.3.13)
- 파이썬 자습서 (v.3.13)
- yeonsookim-wt/lgtm
- 점프 투 파이썬
- ⭐ 파이썬 독학하기 좋은 위키독스 모음집
- ⭐ Jupyter notebook 단축키
- [Jupyter 주피터] 주요 단축키 모음
- ⭐ 주피터 노트북(Jupyter Notebook) 사용법 - 기본 설치/실행, 단축키, 매직 명령어, Markdown, 테마스킨, nbextensions
- 기초탄탄 파이썬_데이터 시각화
  - 02장 판다스 pandas
- 파이썬 마스터하기 : Pandas
- 파이썬으로 배우는 알고리즘 트레이딩 (개정판-2쇄)
  - 13. Pandas를 이용한 데이터 분석 기초 (revision)
- 파이썬을 이용한 비트코인 자동매매 (개정판)
- Git 설치 및 환경설정(mac os)
- pandas apply 함수와 lambda 설명
- 03-01. 축 기준 (apply)
- Jupyter Notebook에서 Magic Command 사용하기
- 01-IPython 매직커맨드
- pandas.DataFrame.transform
- 판다스 pandas IQR 활용해서 이상점(outlier) 찾고 삭제하기

🔷 [팬더스_명령어_꿀팁.ipynb] #217 부터

```
df_filtered = df[['name', 'age']]
-----
name    age
0   John    20
1   Jenny   30
2   Nate    30
3   Julia   40
4   Brian   45
5   Chris   25
```

```
#217
pandas docu 내가찾고자 하는 메서드 검색해서 옵션 이해하기
메서드를 이용한 필터링
스테틱 메서드
키워드인자 방식으로 필터될 컬럼명을 리스트로 전달
axis 종축, 횡축, 열, 행
```

```
axis = 0 행 0 or 'index'
axis = 1 열 1 or 'columns'
0 or
```

- DataFrame.filter(items=None, like=None, regex=None, axis=None)[source]

🔷 원하는 글자를 가진 로우를 보여줍니다.

```
# db sql에서
# 열을 이름에 a가 들어간 컬럼을 필터해서 데이터프레임으로
# select columns containing 'a'
df.filter(like='a',axis=1)
---
    name    age
0   John    20
1   Jenny   30
2   Nate    30
3   Julia   40
4   Brian   45
5   Chris   25
```

- [SQL] SELECT문과 LIKE절

🔷 정규식으로 필터도 가능합니다. #220

```
정규식
정규식에서 내가 원하는 문자열을 패턴매칭을 통해 가져오기

파이썬에서 정규식 다룰때는 re
완전일치 패턴매칭
위치인덱스 = "안녕하세요. 홍길동님".find("안녕하세요)
"안녕하세요 홍길동님".replace("안녕하세요", "hello")

정규식 문법 공부
sql LLM
정규식 LLM 이용

# b로 끝나는 문장만 검색
# select columns using regex
df.filter(regex='b$',axis=1)
---
job
0   student
1   developer
2   teacher
3   dentist
4   manager
5   intern
```

- 08-2 정규 표현식 시작하기
- YTHON 정규식 문법 및 예제 (전화번호, 이메일, 숫자 출력, 공백 제거, URL 출력, 주민 번호 패턴 찾기)
- 003 정규표현식으로 개인정보를 보호하려면? — re
- https://regexr.com/

🔷 로우 드롭하기

🔷 로우 인덱스로 로우를 드롭할 수 있습니다. #221

```
내가 원하는 것만 남긴다.
드롭된 결과는 데이터프레임에 저장되지 않는다.
원본유지 옵션 주면 원본 바꿈

# 존이나 네이트가 들어있는 행을 드롭 #223
# 행 자체가 사라지니까 정규성이 유지됨
df.drop(['John', 'Nate'])
---
    age job
Jenny   30  developer
```

🔷 드롭된 결과를 바로 데이터프레임에 저장하는 방법

🔷 inplace 키워드를 사용하면, 따로 저장할 필요없이, 드롭된 결과가 데이터프레임에 반영됩니다.

```
friend_dict_list = [{'age': 20, 'job': 'student'},
         {'age': 30, 'job': 'developer'},
         {'age': 30, 'job': 'teacher'}]
df = pd.DataFrame(friend_dict_list, index = ['John', 'Jenny', 'Nate'])
---
    age job
John    20  student
Jenny   30  developer
Nate    30  teacher

df.drop(['John', 'Nate'], inplace = True)
---
    age job
Jenny   30  developer
```

로우 인덱스로 드롭하는 예제입니다.

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

```
---
    age job
John    20  student
Jenny   30  developer
Nate    30  teacher

df = df.drop(df.index[[0,2]])
---
age job name
1   30  developer   Jenny
---
```

컬럼값으로 로우 드롭하기

```
컬럼 age에 데이터가 30이 아닌것만 필터해서 데이터프레임화
! 아니다.

friend_dict_list = [{'name': 'Jone', 'age': 20, 'job': 'student'},
         {'name': 'Jenny', 'age': 30, 'job': 'developer'},
         {'name': 'Nate', 'age': 30, 'job': 'teacher'}]
df = pd.DataFrame(friend_dict_list)
---
age job name
0   20  student Jone
1   30  developer   Jenny
2   30  teacher Nate

#235
df = df[df.age != 30]
---
    name    age job
0   Jone    20  student
```

컬럼 드롭하기

```
friend_dict_list = [{'name': 'Jone', 'age': 20, 'job': 'student'},
         {'name': 'Jenny', 'age': 30, 'job': 'developer'},
         {'name': 'Nate', 'age': 30, 'job': 'teacher'}]
df = pd.DataFrame(friend_dict_list)
---

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

df = df.drop('age', axis=1)
---
```

```
name    job
0   Jone    student
1   Jenny   developer
2   Nate    teacher
```

🔷 컬럼 추가 또는 변경하기 #240

```
특성공학 - 컬럼을 다루는 것?
피처 엔지니어링
뜬다 프롬프트 엔지니어가...

friend_dict_list = [{'name': 'Jone', 'age': 15, 'job': 'student'},
         {'name': 'Jenny', 'age': 30, 'job': 'developer'},
         {'name': 'Nate', 'age': 30, 'job': 'teacher'}]

df = pd.DataFrame(friend_dict_list, columns = ['name', 'age', 'job'])
---
    name    age job
0   Jone    15  student
1   Jenny   30  developer
2   Nate    30  teacher

아래와 같은 방법으로 새로운 컬럼을 기본값과 함께 추가하실 수 있다.
df['salary'] = 0
```

🔷 기존 컬럼값을 가지고 새로운 컬럼을 생성하는 예제입니다.

```
friend_dict_list = [{'name': 'Jone', 'age': 15, 'job': 'student'},
         {'name': 'Jenny', 'age': 30, 'job': 'developer'},
         {'name': 'Nate', 'age': 30, 'job': 'teacher'}]
df = pd.DataFrame(friend_dict_list, columns = ['name', 'age', 'job'])
---
    name    age job
0   Jone    15  student
1   Jenny   30  developer
2   Nate    30  teacher

#넘파이를 사용하셔서, 한줄에 새로운 컬럼값을 생성하실 수도 있습니다.
import numpy as np

# 넘파이야 조건 데이터프레임 Job 컬럼이 학생이 아니면 yes, 학색이면 no 로 채워라
                                  참/거짓
                                  조건식              참인경우 거짓인경우
df['salary'] = np.where(df['job'] != 'student' , 'yes', 'no')
---
    name    age job salary
0   Jone    15  student no
1   Jenny   30  developer   yes
2   Nate    30  teacher yes
```

```
# 나이를 참조해서 연령대 컬럼 추가
import numpy as np
# df['salary'] = np.where(df['job'] != 'student' , 'yes',
df['연령대'] = np.where(df['age'] > 20 , '성인', '청소년')
---
name    age job 연령대
0   Jone    15  student 청소년
1   Jenny   30  developer   성인
2   Nate    30  teacher 성인
```

🔷 Tip

```
직법 #을 넣으면 안됨
주석처리 단축키 ctrl + '/'

스페이스(4) 로 하면 안됨
들여쓰기 텝을 눌러서 한다.

정렬에 대한 이해
내가 쓰고 있는 문법이 어디에 속해서 작동하는
```

🔷 아래는 기존에 있는 두 컬럼값을 더해서 새로운 컬럼을 만드는 예제입니다.

```
friend_dict_list = [{'name': 'John', 'midterm': 95, 'final': 85},
         {'name': 'Jenny', 'midterm': 85, 'final': 80},
         {'name': 'Nate', 'midterm': 10, 'final': 30}]
df = pd.DataFrame(friend_dict_list, columns = ['name', 'midterm',
'final'])
---
name    midterm final
0   John    95  85
1   Jenny   85  80
2   Nate    10  30


df['total'] = df['midterm'] + df['final']
---
name    midterm final   total
0   John    95  85  180
1   Jenny   85  80  165
2   Nate    10  30  40
```

🔷 기존의 컬럼을 사용하여 새로운 컬럼을 만드는 예제 #249

```
대표값
평균 mean  산술 기하 조화
중앙값
최빈값

df['average'] = df['total'] / 2
---
    name    midterm final   total   average
0   John    95  85  180 90.0
1   Jenny   85  80  165 82.5
2   Nate    10  30  40  20.0
```

- Average와 Mean의 차이

🔷 아래와 같이, 리스트에 조건별 값을 담아서, 새로운 컬럼으로 추가시킬 수 있다. #251

```
# 리스트 컴르리헨션

grades = []
# 함수화 해서
# list(map())
def 등급확인함수(평균):


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

df['grade'] = grades
```

apply 함수 사용 예제입니다.

🔷 apply를 사용하시면, 깔끔하게 컬럼의 값을 변경하는 코드를 구현하실 수 있다. #253

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

```
df.grade = df.grade.apply(pass_or_fail)
---
A
B
F
```

- pandas apply 함수와 lambda 설명
- 03-01. 축 기준 (apply)

🔷 apply를 사용해서 연월일의 정보에서 연도만 빼보는 예제입니다. #256

```
date_list = [{'yyyy-mm-dd': '2000-06-27'},
         {'yyyy-mm-dd': '2002-09-24'},
         {'yyyy-mm-dd': '2005-12-20'}]
df = pd.DataFrame(date_list, columns = ['yyyy-mm-dd'])
---
yyyy-mm-dd
0   2000-06-27
1   2002-09-24
2   2005-12-20
```

```
# 팬더스_명령어_꿀팁.ipynb #257
# 함수 년도만 추출하는 함수
#     특정 행이 들어오면 split
#     문자열을 기준으로 쪼게서 리스트로 만들어준다.
# 3개의 아이템디 들어었는 0 1 2리스트에서 0번째 항목을 리턴
# 연도만 문자열로 리턴

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

df['year'] = df['yyyy-mm-dd'].apply(extract_year)
---
yyyy-mm-dd  year
0   2000-06-27  2000
1   2002-09-24  2002
2   2005-12-20  2005


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

df['month'] = df['yyyy-mm-dd'].apply(extract_month)
---
    yyyy-mm-dd  year    month
0   2000-06-27  2000    06
1   2002-09-24  2002    09
2   2005-12-20  2005    12
```

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

🔷 apply를 사용해서 연월일의 정보에서 연도만 빼보는 예제입니다. #256

date_list = [{'yyyy-mm-dd': '2000-06-27'},
         {'yyyy-mm-dd': '2002-09-24'},
         {'yyyy-mm-dd': '2005-12-20'}]
df = pd.DataFrame(date_list, columns = ['yyyy-mm-dd'])
df

def extract_YMD(row, idx):
    if idx  == 0:
        ret = row.split('-')[0]
    elif idx == 1:
        ret = row.split('-')[1]
    else:
        ret = row.split('-')[2]
    return ret

df['year'] = df['yyyy-mm-dd'].apply(extract_YMD, idx = 0)
df['month'] = df['yyyy-mm-dd'].apply(extract_YMD, idx = 1)
df['day'] = df['yyyy-mm-dd'].apply(extract_YMD, idx = 2)
df
------------------

    yyyy-mm-dd  year    month   day
0   2000-06-27  2000    06  27
1   2002-09-24  2002    09  24
2   2005-12-20  2005    12  20
```

- python pandas: apply a function with arguments to a series

apply 함수에 파라미터 전달하기(apply: column 각행에 적용)

🔷 키워드 파라미터를 사용하시면, apply가 적용된 함수에 파라미터를 전달할 수 있다. #260

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


df['age'] = df['year'].apply(extract_year, current_year=2018)


def extract_year(year, current_year):
    from datetime import datetime
    now = datetime.now()
    now.year
    current_year = int(now.year)
    return current_year - int(year)
```

```
df['age'] = df['year'].apply(extract_year, current_year=2018)
---

yyyy-mm-dd  year    month   day age
0   2000-06-27  2000    06  27  25
1   2002-09-24  2002    09  24  23
2   2005-12-20  2005    12  20  20
```

apply 함수에 한 개 이상의 파라미터 전달하기 (apply: column 각행에 적용)

🔷 키워드 파라미터를 추가해주시면, 원하시는만큼의 파라미터를 함수에 전달 가능합니다. #261

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

df['introduce'] = df['age'].apply(get_introduce, prefix="I am ", suffix="
years old")
---
yyyy-mm-dd  year    age introduce
0   2000-06-27  2000    18  I am 18 years old
1   2002-09-24  2002    16  I am 16 years old
2   2005-12-20  2005    13  I am 13 years old
```

apply 함수에 여러개의 컬럼을 동시에 전달하기

🔷 axis=1이라는 키워드 파라미터를 apply 함수에 전달해주면, 모든 컬럼을 지정된 함수에서 사용 가능 #264

```

```

Map 함수로 컬럼 추가 및 변경하기

🔷 파라미터로 함수를 전달하면 apply 함수와 동일하게 컬럼값을 추가 및 변경할 수 있다. #265

```
파이썬 내장객체 map
pandas의 map 메서드 이용한 것

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

date_list = [{'yyyy-mm-dd': '2000-06-27'},
         {'yyyy-mm-dd': '2002-09-24'},
         {'yyyy-mm-dd': '2005-12-20'}]
df = pd.DataFrame(date_list, columns = ['yyyy-mm-dd'])
---
```

```
yyyy-mm-dd
0   2000-06-27
1   2002-09-24
2   2005-12-20

df['year'] = df['yyyy-mm-dd'].map(extract_year)
---
yyyy-mm-dd  year
0   2000-06-27  2000
1   2002-09-24  2002
2   2005-12-20  2005
```

파라미터로 딕셔너리를 전달하면 컬럼값을 원하는 값으로 쉽게 변경이 가능

🔷 기존의 컬럼값은 딕셔너리의 key로 사용되고, 해당되는 value의 값으로 컬럼값이 변경됨 #267

```
job_list = [{'age': 20, 'job': 'student'},
         {'age': 30, 'job': 'developer'},
         {'age': 30, 'job': 'teacher'}]
df = pd.DataFrame(job_list)
---
age job
0   20  student
1   30  developer
2   30  teacher

df.job = df.job.map({"student":1,"developer":2,"teacher":3})
---
age job
0   20  1
1   30  2
2   30  3
```

Applymap

🔷 데이터프레임 전체의 각각의 값을 한번에 변경시키실 때 사용하시면 좋습니다. #270

```
# python float타입
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)

# pandas의 float타입으로 변환
---
x   y
0   5.5 -5.6
1   -5.2    5.5
2   -1.6    -4.5
```

```
df['x'].dtype
---
dtype('float64')

df = df.applymap(np.around)
---
x   y
0   6.0 -6.0
1   -5.0    6.0
2   -2.0    -4.0
```

🔷 데이터프레임에 로우 추가하기

```
# 기본은 행방향으로 합치기
# 열방향으로 합치기
friend_dict_list = [{'name': 'John', 'midterm': 95, 'final': 85},
         {'name': 'Jenny', 'midterm': 85, 'final': 80},
         {'name': 'Nate', 'midterm': 10, 'final': 30}]
df = pd.DataFrame(friend_dict_list, columns = ['name', 'midterm',
'final'])
---
name    midterm final
0   John    95  85
1   Jenny   85  80
2   Nate    10  30


df2 = pd.DataFrame([['Ben', 50,50]], columns = ['name', 'midterm',
'final'])
df2.head()
---
    name    midterm final
0   Ben 50  50


# df 기준으로 df2 데이터를 붙인다.(기본은 하단으로 붙인다.)
# 축생각 기본값이니까 axis = 0 행추가
# ignore_index=true 는 합쳐진 후 인덱스 리셋
df.append(df2, ignore_index=True)

합쳐진 후에도 표준화된 데이터 프레임으로 만들기
```

- pandas.DataFrame.append
- [Python] 데이터프레임 결합 (.concat, .append)

Group by

🔷 데이터에서 정보를 취하기 위해서 그룹별로 묶는 방법 #275

```
# 학문 이름
#   철학 필라서피
#   필리오 사랑
#   소피아 지혜
#   철학자 지혜를 사랑하는 사람
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'])
----
    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

groupby_major = df.groupby('major')
groupby_major.groups
---
{'Computer Science': Int64Index([0, 1, 6, 7], dtype='int64'),
 'Economics': Int64Index([4, 5, 9], dtype='int64'),
 'Physics': Int64Index([2], dtype='int64'),
 'Psychology': Int64Index([3, 8, 10], dtype='int64')}
```

- Pandas: How to append a dataframe to itself if .append has been deprecated? [duplicate]

```
  import pandas as pd
  df = pd.DataFrame({'col1' : [1,2,3]})
  df = pd.concat([df,df]).reset_index(drop = True)
```

```
print(df)
---
     col1
 0     1
 1     2
 2     3
 3     1
 4     2
 5     3
```

- Jupyter Notebook에서 Magic Command 사용하기
- 01-IPython 매직커맨드

🔷 here we can see, computer science has mostly man, while economic has mostly woman students #279

```
# 내가 원하는데로 보겠다.
# def 함수화

# 이터화하면 2개가 튀어나옴
# __iter__()

for name, group in groupby_major:
    print(name + ": " + str(len(group)))
    print(group)
    print() #줄바꿈

---
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 생성하는 예제입니다. #280

```
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
```

🔷 아래의 출력을 통해, 이 학교의 남녀 성비가 균등하다는 정보를 알 수 있다. #282

```
groupby_sex = df.groupby('sex')

for name, group in groupby_sex:
    print(name + ": " + str(len(group)))
    print(group)
    print()
```

- https://www.kaggle.com/
  - kaggle >> 타이타닉 검색

```
# 내 데이터 읽어오기
dfTitanic = pd.read_csv("titanic.csv", encoding="utf-8")
dfTitanic.head(3)

# P Class로  groupby 하기
# column명은 대소문자 구분한다.
# 대소문자 혼용된 컬럼들 = df.columns
# df.columns = list(map(labmda x : x.lower(), 대소문자 혼용된 컬럼들))

dfTitanic.columns = list(map(lambda x: x.lower(), dfTitanic.columns))
dfTitanic.head(3)

groupby_survived = dfTitanic.groupby('survived')
groupby_survived.head()

groupby_pclass = dfTitanic.groupby('pclass')
groupby_pclass.head()
```

```
groupby_sex = dfTitanic.groupby('sex')
groupby_sex.head()


groupby_pclass = dfTitanic.groupby('pclass')
groupby_pclass.head()

df_survived_cnt = pd.DataFrame({'count' :
groupby_survived.size()}).reset_index()
df_survived_cnt
---
    survived    count
0       0       545
1       1       342

df_sex_cnt = pd.DataFrame({'count' :
groupby_sex.size()}).reset_index()
df_sex_cnt
---
    sex     count
0   female  314
1   male    573

df_pclass_cnt = pd.DataFrame({'count' :
groupby_pclass.size()}).reset_index()
df_pclass_cnt
----
    pclass  count
0   1       216
1   2       184
2   3       487
```

중복 데이터 드롭하기

🔷 중복된 데이터 드롭하는 방법 #284

```
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"},
                {'name': 'John', 'major': "Computer Science", 'sex':
"male"},
        ]
df = pd.DataFrame(student_list, columns = ['name', 'major', 'sex'])
---
    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  John    Computer Science    male
```

🔷 중복된 데이터 확인 하기 #285

```
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     True
dtype: bool
```

drop_duplicates 함수로 중복 데이터를 삭제하는 예제 #286

```
df = df.drop_duplicates()
---
    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
```

name 컬럼이 똑같을 경우, 중복된 데이터라고 표시하라는 예제 #289

```
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': 'Nate', 'major': None, 'sex': "male"},
                {'name': 'John', 'major': "Computer Science", 'sex':
None},
        ]
df = pd.DataFrame(student_list, columns = ['name', 'major', 'sex'])
---
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': 'Nate', 'major': None, 'sex': "male"},
                {'name': 'John', 'major': "Computer Science", 'sex':
None},
        ]
df = pd.DataFrame(student_list, columns = ['name', 'major', 'sex'])
```

```
df.duplicated(['name'])
---
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10     True
11     True
dtype: bool
```

keep 값을 first 또는 last라고 값을 줘서 중복된 값 중, 어느값을 살릴 지 결정하실 수 있다.

기본적으로 first로 설정되어 있다.

```
df.drop_duplicates(['name'], keep='last')
---
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   Economics   female
10  Nate    None    male
11  John    Computer Science    None
```

None 처리 하기 #292

```
school_id_list = [{'name': 'John', 'job': "teacher", 'age': 40},
                {'name': 'Nate', 'job': "teacher", 'age': 35},
                {'name': 'Yuna', 'job': "teacher", 'age': 37},
                {'name': 'Abraham', 'job': "student", 'age': 10},
                {'name': 'Brian', 'job': "student", 'age': 12},
                {'name': 'Janny', 'job': "student", 'age': 11},
                {'name': 'Nate', 'job': "teacher", 'age': None},
                {'name': 'John', 'job': "student", 'age': None}
         ]
df = pd.DataFrame(school_id_list, columns = ['name', 'job', 'age'])
---
    name    job age
0   John    teacher 40.0
```

```
1   Nate    teacher 35.0
2   Yuna    teacher 37.0
3   Abraham student 10.0
4   Brian   student 12.0
5   Janny   student 11.0
6   Nate    teacher NaN
7   John    student NaN
```

⭐ Null 또는 NaN 확인하기

```
df.info()
---
# 📌 object 타입은 대부분 String 타입
<class 'pandas.core.frame.DataFrame'>  #데이터프레임 의미
RangeIndex: 8 entries, 0 to 7 #8개의 엔트리 존재함을 의미
Data columns (total 3 columns): #3개의column이 있다
name    8 non-null object #object는 string타입?
job     8 non-null object
age     6 non-null float64
dtypes: float64(1), object(2)
memory usage: 272.0+ bytes
```

Null 또는 NaN 값 변경하기 #296

```
# 분석가 마음
# Null을 0으로 설정하는 예제

tmp = df
tmp["age"] = tmp["age"].fillna(0)

#보통은 컬럼 값을 평균
---
    name        job age
0   John    teacher 40.0
1   Nate    teacher 35.0
2   Yuna    teacher 37.0
3   Abraham student 10.0
4   Brian   student 12.0
5   Janny   student 11.0
6   Nate    teacher 0.0
7   John    student 0.0
```

0으로 설정하기 보다는 선생님의 중간 나이, 학생의 중간 나이로, 각각의 직업군에 맞게 Null값을 변경 #297

```
# fill missing age with median age for each group (teacher, student)
#                                       값
```

```
# inplace : 원본을 덮어쓸 것인지 여부
df["age"].fillna(df.groupby("job")["age"].transform("median"),
inplace=True)
---
    name        job age
0   John    teacher 40.0
1   Nate    teacher 35.0
2   Yuna    teacher 37.0
3   Abraham student 10.0
4   Brian   student 12.0
5   Janny   student 11.0
6   Nate    teacher 0.0
7   John    student 0.0
```

- pandas.DataFrame.transform

Unique

🔷 컬럼에 여러 값이 있을 때, 중복 없이 어떤 값들이 있는 지 확인하는 방법 #299

```
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': 'Philip', 'job': "lawyer"},
                {'name': 'Janny', 'job': "basketball player"},
                {'name': 'Gwen', 'job': "teacher"},
                {'name': 'Jessy', 'job': "student"}
          ]
df = pd.DataFrame(job_list, columns = ['name', 'job'])

결측치 비어있는 값
이상치 특이치 아웃라이어 극단치 분포의 끝에 있다.
블랙스완 나심탈레브?
```

- 판다스 pandas IQR 활용해서 이상점(outlier) 찾고 삭제하기

컬럼(시리즈)의 unique() 함수를 사용하여, 중복 없이, 컬럼에 있는 모든 값들을 출력할 수 있다. #300

```
print( df.job.unique() )
---
['teacher' 'student' 'developer' 'dentist' 'lawyer' 'banker'
 'basketball player']
```

각 유니크한 값별로 몇개의 데이터가 속하는 지 value_counts() 함수로 확인할 수 있다 #301

```
df.job.value_counts()
---
teacher              8
student              5
banker               2
lawyer               2
basketball player    1
dentist              1
developer            1
Name: job, dtype: int64
```

두개의 데이터프레임 합치기 #302

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

l2 = [{'name': 'Ed', 'job': "dentist"},
      {'name': 'Jack', 'job': "farmer"},
      {'name': 'Ted', 'job': "designer"}]

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

pd.concat

두번째 데이터프레임을 첫번째 데이터프레임의 새로운 로우(행)로 합침

```
frames = [df1, df2]
result = pd.concat(frames, ignore_index=True)
---
    name    job
0   John    teacher
1   Nate    student
2   Fred    developer
3   Ed      dentist
```

```
4   Jack    farmer
5   Ted     designer
```

df.append - deprecated 되어 concat 을 사용

두번째 데이터프레임을 첫번째 데이터프레임의 새로운 로우(행)로 합침 #305

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

l2 = [{'name': 'Ed', 'job': "dentist"},
      {'name': 'Jack', 'job': "farmer"},
      {'name': 'Ted', 'job': "designer"}]

df1 = pd.DataFrame(l1, columns = ['name', 'job'])
df2 = pd.DataFrame(l2, columns = ['name', 'job'])
result = pd.concat([df1, df2], ignore_index=True)
---

name    job
0   John    teacher
1   Nate    student
2   Fred    developer
3   Ed      dentist
4   Jack    farmer
5   Ted     designer
```

pd.concat

두번째 데이터프레임을 첫번째 데이터프레임의 새로운 컬럼(열)으로 합칩니다.

📌 정리

```
가져오기 가져오는 원천 read read_json(웹개발 과정 중) ⭐ read_sql(데이터베이스 과정 중)

피처엔지니어링
이상치 특이치 처리
탐색적 데이터분석(EDA)
df.info()
df.describe()
count_value()

가공 내가 원하는데로
표준화된 데이터프레임 만들기
시각화는 다른 라이브러리로 하기
```

```
내보내기 내보내는 방업 to_csv, to_excel, to_json(웹개발 과정 중) ⭐ to_sql(데이터베이스 과정 중)
```

나만의 데이터 셋 만들기(개인별 미션)

NumPy

```
판다스
넘파이 숫자 선형대수 행렬연산 핵심
#📌 python 의 array와 다름
#import array

import numpy as np

리스트 = [1,2,3,4,5]

넘파이객체 = np.array(리스트)
넘파이객체
print(f"넘파이객체 - {넘파이객체}", f"\n", f"타입 - {type(넘파이객체)}")
---
넘파이객체 - [1 2 3 4 5]
 타입 - <class 'numpy.ndarray'>
```

- ⭐ NumPy fundamentals
  - Broadcasting
- ⭐ [Numpy] Broadcasting이란? (브로트 캐스트 규칙)
  - 브로드 캐스팅이란 산술연산이 되는 배열(array)에서 모양(shape)이 다른 경우에도, 연산이 가능하도록 배열들의 모양을 처리하는 방법
- [Numpy] 브로드캐스팅(Broadcasting)이란?
- [Python] Numpy Broadcasting 개념
- [넘파이(Numpy)]브로드캐스팅(Broadcasting) - 브로드캐스팅 조건, 예제

```
넘파이객체  = np.array(리스트)
print(type(넘파이객체))

넘파이객체.shape
4x1 - 4행 1열
28x28 - 28행 28열
```

파이썬 1차원 배열(list)로 NumPy 배열 생성

```
import numpy as np

리스트 = [1,2,3,4,5]
```

```
넘파이객체 = np.array(리스트)
넘파이객체
print(f"넘파이객체 - {넘파이객체}", f"\n", f"타입 - {type(넘파이객체)}")
```

파이썬이 제공하는 pprint와는 다른 사용자 함수 정의

```
# 사용자함수 정의
def pprint(arr: np.ndarray):
    print(f"type: {type(arr)}")
    print(f"shape: {arr.shape}")
    print(f"ndim: {arr.ndim}")
    print(f"dtype: {arr.dtype}")

pprint(넘파이객체)
---
type: <class 'numpy.ndarray'>
shape: (5,)
ndim: 1
dtype: int64

print(넘파이객체)
---
[1 2 3 4 5]

리스트 = [1,2,3,4,5]
넘파이객체 = np.array(리스트, dtype='float64')
pprint(넘파이객체)
---
type: <class 'numpy.ndarray'>
shape: (5,)
ndim: 1
dtype: float64
```

파이썬 2차원 배열로 NumPy 배열 생성, 원소 데이터 타입 지정

```
arr2 = [(1,2,3), (4,5,6)]
a2 = np.array(arr2, dtype=float)
pprint(a2)
---
type: <class 'numpy.ndarray'>
shape: (2, 3)
ndim: 2
dtype: float64
```

파이썬 3차원 배열로 NumPy 배열 생성, 원소 데이터 타입 지정

```
arr = np.array([[[1,2,3], [4,5,6]], [[3,2,1], [4,5,6]]], dtype = float)
a = np.array(arr, dtype = float)
pprint(a)
a
---
type: <class 'numpy.ndarray'>
shape: (2, 2, 3)
ndim: 3
dtype: float64
array([[[1., 2., 3.],
        [4., 5., 6.]],

       [[3., 2., 1.],
        [4., 5., 6.]]])
```

2.2 배열 생성 및 초기화

```
Numpy는 원하는 shape으로 배열을 설정하고, 각 요소를 특정 값으로 초기화하는
zeros, ones, full, eye 함수를 제공합니다. 또한, 파라미터로
입력한 배열과 같은 shape의 배열을 만드는 zeros_like, ones_like, full_like 함수도
제공합니다. 이 함수를 이용하여 배열 생성하고 초기화할 수 있습니다.

zeros 영행렬
ones 일행렬
full 같은 값이 들어있는 행렬
eye 단위행렬
```

- ⭐ [넘파이(Numpy)]N차원 배열 생성(2) - zeros, ones, full, eye, arange, linspace, logspace
  - np.zeros() 함수는 모든 요소들이 0으로 초기화 된 N차원 배열을 생성
  - np.ones() 함수는 모든 요소들이 1로 초기화 된 N차원 배열을 생성
  - np.full()함수를 이용하면 어떤 요소로 배열을 채울 것인지 정할 수 있다.
    - np.full()함수의 인자로는 배열의 shape과 배열을 채울 요소가 들어간다.
  - np.eye()함수는 입력한 shape을 가진, 대각 원소가 1인 행렬을 생성
  - np.zeros_like()
    - 인자로 배열을 받아서 모든 요소들을 0으로 초기화한다.
  - np.ones_like()
    - 인자로 배열을 받아서 모든 요소들을 1로 초기화한다.
  - np.full_like()
    - 인자로는 배열과, 지정된 값이 들어가고, 입력받은 배열의 모든 요소들을 입력된 값으로 초기화

```
a = np.ones((2,3,4), dtype = np.int16)
pprint(a)
a
---
type: <class 'numpy.ndarray'>
shape: (2, 3, 4)
ndim: 3
```

```
dtype: int16
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]], dtype=int16)

a = np.zeros((3,4), dtype = np.int16)
pprint(a)
a
---
type: <class 'numpy.ndarray'>
shape: (3, 4)
ndim: 2
dtype: int16
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]], dtype=int16)


a = np.eye(4, dtype = np.float64)
pprint(a)
a
---
type: <class 'numpy.ndarray'>
shape: (4, 4)
ndim: 2
dtype: float64
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

2.3 데이터 생성 함수

```
# NumPy는 주어진 조건으로 데이터를 생성한 후, 배열을 만드는 데이터 생성 함수를 제공합니다.
# numpy.linspace
# numpy.arange
# numpy.logspace
# np.linspace 함수


aa = np.linspace(1, 100, num=50, endpoint=True, retstep=False, dtype=None)
pprint(aa)
aa
---
type: <class 'numpy.ndarray'>
shape: (50,)
```

```
ndim: 1
dtype: float64
array([  1.        ,   3.02040816,   5.04081633,   7.06122449,
         9.08163265,  11.10204082,  13.12244898,  15.14285714,
        17.16326531,  19.18367347,  21.20408163,  23.2244898 ,
        25.24489796,  27.26530612,  29.28571429,  31.30612245,
        33.32653061,  35.34693878,  37.36734694,  39.3877551 ,
        41.40816327,  43.42857143,  45.44897959,  47.46938776,
        49.48979592,  51.51020408,  53.53061224,  55.55102041,
        57.57142857,  59.59183673,  61.6122449 ,  63.63265306,
        65.65306122,  67.67346939,  69.69387755,  71.71428571,
        73.73469388,  75.75510204,  77.7755102 ,  79.79591837,
        81.81632653,  83.83673469,  85.85714286,  87.87755102,
        89.89795918,  91.91836735,  93.93877551,  95.95918367,
        97.97959184, 100.        ])
```