

# Data Analysis & Visualization: Part I

📅 Date	@2022년 8월 25일 오전 9:30
🔗 Code: Lecture	
🔗 Code: Practice	
🔗 Code: Practice-Sol	
🔗 Lecture Note	
# No.	4

## Pandas

- Library for data analysis in Python

```
import pandas as pd
```

### pandas

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

<https://pandas.pydata.org/>

## ▼ 1. Series

- One-dimensional ndarray with axis labels (including time series)

### pandas.Series - pandas 1.4.3 documentation

One-dimensional ndarray with axis labels (including time series). Labels need not be unique but must be a hashable type. The object supports both integer- and label-based indexing and provides a host of methods for performing operations involving the index. Statistical methods from ndarray have been overridden to automatically exclude missing data (currently represented as NaN).

<https://pandas.pydata.org/docs/reference/api/pandas.Series.html>

- Series 객체 생성 예제

```
# 1월부터 4월까지 평균 온도 데이터 (-20, -10, 10, 20)
temp = pd.Series([-20, -10, 10, 20])
print(temp)
```

```
print(temp[0]) # 1월 온도
print(temp[2]) # 3월 온도
```

```
# Series 객체 생성: Index 지정
temp = pd.Series([-20, -10, 10, 20], index=['Jan', 'Feb', 'Mar', 'Apr'])
print(temp)
```

```
print(temp['Jan']) #1월 온도: index 'Jan'에 해당하는 데이터 출력
print(temp['Mar']) #3월 온도: index 'Mar'에 해당하는 데이터 출력
```

## ▼ 2. DataFrame

- Two-dimensional, size-mutable, potentially heterogeneous tabular data

#### pandas.DataFrame - pandas 1.4.3 documentation

Two-dimensional, size-mutable, potentially heterogeneous tabular data. Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure. Parameters datarray (structured or homogeneous), Iterable, dict, or DataFrame Dict can contain Series, arrays, constants, dataclass or list-like objects.

 <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html#pandas.DataFrame>

- DataFrame 객체 생성 예제

```
# Data 준비: 사전(dict) 자료구조를 통해 생성
# 원피스 밀집모자 해적단 10명에 대한 데이터

data = {
    "이름" : ["루피", "조로", "나미", "우솜", "상디", "초파", "로빈", "프랑키", "브룩", "징베"],
    "출신" : ["이스트블루", "이스트블루", "이스트블루", "이스트블루", "노스블루", "위대한항로", "웨스트블루", "사우스블루", "웨스트블루", "위대한항로"],
    "공격" : [100, 95, 90, 40, 80, 40, 15, 80, 55, 100],
    "방어" : [70, 90, 85, 35, 75, 60, 20, 100, 65, 85],
    "지력" : [30, 50, 100, 50, 70, 70, 10, 95, 45, 90],
    "민첩" : [95, 90, 95, 55, 80, 75, 35, 85, 40, 95]
}
```

```
# DataFrame 객체 생성
import pandas as pd

df = pd.DataFrame(data)
```

```
# DataFrame 객체 생성: Index 지정 형태

df = pd.DataFrame(data, index=["선장", "전투원", "항해사", "저격수", "요리사", "의사", "고고학자", "조선공", "음악가", "조타수"])
```

```
# DataFrame 객체 생성: Column 지정
# 데이터 중에서 원하는 column 만 선택하거나, 순서 변경 등 가능

df = pd.DataFrame(data, columns=["이름", "공격", "지력"])
# df = pd.DataFrame(data, columns=["이름", "공격", "출신"])
```

```
# DataFrame을 이용한 데이터 접근 예제

df["이름"]
df[["이름", "지력"]]
```

### ▼ 3. DataFrame: Index

- The index (row labels) of the DataFrame

- DataFrame의 index 사용 예제

```
# Data 준비 및 DataFrame 객체 생성

import pandas as pd

data = {
    "이름" : ["루피", "조로", "나미", "우솜", "상디", "초파", "로빈", "프랑키", "브룩", "징베"],
    "출신" : ["이스트블루", "이스트블루", "이스트블루", "이스트블루", "노스블루", "위대한항로", "웨스트블루", "사우스블루", "웨스트블루", "위대한항로"],
    "공격" : [100, 95, 90, 40, 80, 40, 15, 80, 55, 100],
    "방어" : [70, 90, 85, 35, 75, 60, 20, 100, 65, 85],
    "지력" : [30, 50, 100, 50, 70, 70, 10, 95, 45, 90],
    "민첩" : [95, 90, 95, 55, 80, 75, 35, 85, 40, 95],
    "특기" : ["파왕색", "파왕색", "", "견문색", "견문색", "", "", "무장색", "무장색", ""]
}

df = pd.DataFrame(data, index=["선장", "전투원", "항해사", "저격수", "요리사", "의사", "고고학자", "조선공", "음악가", "조타수"])
```

```
# Index 출력
```

```
df.index
```

```
# Index 이름 설정
```

```
df.index.name = "역할"
```

```
# Index 초기화 방법1
```

```
df.reset_index()
```

```
# Index 초기화 방법2
```

```
df.reset_index(drop=True) # 기존 인덱스 삭제 (하지만 실제 데이터에는 아직 미반영)
```

```
# Index 초기화 방법3
```

```
df.reset_index(drop=True, inplace=True) # 기존 인덱스 삭제 & 삭제 결과를 데이터에 반영
```

```
# Index 설정
```

```
# df.set_index("이름") # "이름" 열이 index로 설정 (실제 데이터에 반영 X)
```

```
df.set_index("이름", inplace=True) # "이름" 열이 index로 설정 (실제 데이터에 반영 O)
```

```
# Index 정렬
```

```
df.sort_index() # index 기준 오름차순 정렬
```


```
df.sort_index(ascending=False) # index 기준 내림차순 정렬
```

#### ▼ 4. DataFrame: 파일 형태로 저장 및 열기

- `pandas.DataFrame.to_csv`: Write object to a comma-separated values (csv) file

`pandas.DataFrame.to_csv` - pandas 1.4.3 documentation


Write object to a comma-separated values (csv) file. Parameters `path_or_bufstr`, path object, file-like object, or None, default None String, path object (implementing `os.PathLike[str]`), or file-like object implementing a `write()` function. If None, the result is returned as a string. If a non-binary file object is passed, it should be opened with `newline=""`, disabling universal newlines.

 [https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to\\_csv.html?highlight=dataframe%20to\\_](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html?highlight=dataframe%20to_)

- `pandas.DataFrame.to_excel`: Write object to an Excel sheet

`pandas.DataFrame.to_excel` - pandas 1.4.3 documentation

Write object to an Excel sheet. To write a single object to an Excel .xlsx file it is only necessary to specify a target file name. To write to multiple sheets it is necessary to create an `ExcelWriter` object with a target file name, and specify a sheet in the file to write to.

 [https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to\\_excel.html](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_excel.html)

- `pandas.DataFrame.read_csv`: Read a comma-separated values (csv) file into DataFrame

`pandas.read_csv` - pandas 1.4.3 documentation

Read a comma-separated values (csv) file into DataFrame. Also supports optionally iterating or breaking of the file into chunks. Additional help can be found in the online docs for IO Tools. Parameters `filepath_or_bufferstr`, path object or file-like object Any valid string path is acceptable. The string could be a URL.

 [https://pandas.pydata.org/docs/reference/api/pandas.read\\_csv.html?highlight=read\\_csv#](https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html?highlight=read_csv#)

- `pandas.DataFrame.read_excel`: Read an Excel file into a pandas DataFrame

#### pandas.read\_excel - pandas 1.4.3 documentation

Read an Excel file into a pandas DataFrame. Supports xls,xlsx,xlsm,xlsb,odf,ods and odt file extensions read from a local filesystem or URL. Supports an option to read a single sheet or a list of sheets. Parameters iostr, bytes, ExcelFile, xlrd.Book, path object, or file-like object Any valid string path is acceptable.

 [https://pandas.pydata.org/docs/reference/api/pandas.read\\_excel.html?highlight=read\\_excel#](https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html?highlight=read_excel#)

#### • DataFrame 파일로 저장하기 예제

```
# Data 준비 및 DataFrame 객체 생성

import pandas as pd

data = {
    "이름" : ["루피", "조로", "나미", "우솅", "상디", "초파", "로빈", "프랑키", "브룩", "징베"],
    "출신" : ["이스트블루", "이스트블루", "이스트블루", "이스트블루", "노스블루", "위대한항로", "웨스트블루", "사우스블루", "웨스트블루", "위대한항로"],
    "공격" : [100, 95, 90, 40, 80, 40, 15, 80, 55, 100],
    "방어" : [70, 90, 85, 35, 75, 60, 20, 100, 65, 85],
    "지력" : [30, 50, 100, 50, 70, 70, 10, 95, 45, 90],
    "민첩" : [95, 90, 95, 55, 80, 75, 35, 85, 40, 95],
    "특기" : ["패왕색", "패왕색", "", "견문색", "견문색", "", "", "무장색", "무장색", ""]
}

# Index 설정
df = pd.DataFrame(data, index=["선장", "전투원", "항해사", "저격수", "요리사", "의사", "고고학자", "조선공", "음악가", "조타수"])

# Index 이름 설정
df.index.name = "역할"
```

```
# 저장하기: csv 파일

dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'

# df.to_csv(dir + "onepiece.csv")
df.to_csv(dir + "onepiece.csv", encoding='utf-8-sig')
# df.to_csv(dir + "onepiece.csv", encoding='utf-8-sig', index=False)
```

```
# 저장하기: txt 파일

df.to_csv(dir + "onepiece.txt", sep="\t")
```

```
# 저장하기: excel 파일

df.to_excel(dir + "onepiece.xlsx")
```

#### • DataFrame 파일 불러오기 예제

```
# 불러오기: csv 파일

df = pd.read_csv(dir + "onepiece.csv")

# df = pd.read_csv(dir + "onepiece.csv", skiprows=2) # 지정된 개수만큼의 rows 생략
# df = pd.read_csv(dir + "onepiece.csv", skiprows=[1, 3, 5]) # 지정된 rows 생략 (row는 0부터 시작)
# df = pd.read_csv(dir + "onepiece.csv", nrows=5) # 지정된 rows만큼 불러오기
# df = pd.read_csv(dir + "onepiece.csv", skiprows=2, nrows=4) # 처음 2행 생략, 그 아래 4행 불러오기(타이틀행 제외)
```

```
# 불러오기: txt 파일

df = pd.read_csv(dir + "onepiece.txt", sep="\t")

# df = pd.read_csv(dir + "onepiece.txt", sep="\t", index_col="역할")
# df.set_index("역할", inplace=True)
```

```
# 불러오기: excel 파일
```

```
df = pd.read_excel(dir + "onepice.xlsx")

# df = pd.read_excel(dir + "onepice.xlsx", index_col="역할")
```

## ▼ 5. DataFrame: 데이터 확인

DataFrame.describe()	계산 가능한 데이터 형식에 대해 column 별 개수/평균/표준편차 등 정보 표시
DataFrame.info()	각종 정보 표시
DataFrame.head()	처음 n개 row 정보 가져오기 (default: 5개)
DataFrame.tail()	마지막 n 개 row 정보 가져오기 (default: 5개)

- DataFrame 데이터 확인하기 예제

```
# DataFrame 확인

df.describe() # 계산 가능한 데이터 형식 관련 정보 표시
df.info() # 각종 정보 표시
df.head(3) # 처음 3개 rows 표시
df.tail(3) # 마지막 3개 rows 표시
```

```
# DataFrame 기타 정보 확인

df.values # 데이터 전체 표시
df.index # 데이터 index 표시
df.columns # 데이터 column별 index 표시
df.shape # (rows, columns) 표시
```

```
# Series별 데이터 확인

df["공격"].describe()
df["공격"].min()
df["공격"].max()
df["공격"].mean()
df["공격"].sum()

df["공격"].nlargest(3) # 공격력이 가장 썬 사람 순서대로 3명 데이터 값
df["출신"].unique() # 중복 제외한 데이터 값
df["출신"].nunique() # 중복 제외한 데이터의 개수
```

## ▼ 6. DataFrame: 데이터 선택(기본)

- DataFrame: 원하는 데이터 선택(기본) 예제

```
# 파일 불러오기 및 DataFrame 객체 생성

import pandas as pd

dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'

df = pd.read_excel(dir + "onepice.xlsx", index_col="역할")
df
```

```
# Column 선택: label 접근

df['이름']
df[['이름', '출신']]
```

```
# Column 선택: index (정수) 접근

df.columns
df.columns[2]
```

```
df[df.columns[0]] # df['이름']
df[df.columns[-1]] # 맨 끝에 있는 column 값을 가져오기
```

```
# 슬라이싱을 이용한 접근

df['공격'][0:5] # 0~4 까지 공격력 가져오기
df[['이름', '출신']][0:3] # 처음 3명의 이름과 출신 정보 가져오기
df[3:] # 4번째 row부터 마지막 row까지 정보 가져오기
```

## ▼ 7. DataFrame: 데이터 선택(loc)

- Access a group of rows and columns by label(s) or a boolean array

pandas.DataFrame.loc - pandas 1.4.3 documentation

Access a group of rows and columns by label(s) or a boolean array. is primarily label based, but may also be used with a boolean array. Allowed inputs are: A single label, e.g. or , (note that is interpreted as a label of the index, and never as an integer position along the index).

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc.html?highlight=dataframe%20loc#pandas.DataFrame.loc>

- DataFrame: 원하는 데이터 선택(loc) 예제

```
# 파일 불러오기 및 DataFrame 객체 생성

import pandas as pd

dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'

df = pd.read_excel(dir + "onepiece.xlsx", index_col="역할")
df
```

```
# label 을 이용한 데이터 선택1: label에 해당하는 row 데이터 전체 가져오기

df.loc['전투원'] # index 전투원에 해당하는 row 데이터 전체
df.loc['조선공'] # index 조선공에 해당하는 row 데이터 전체
```

```
# label 을 이용한 데이터 선택2: label1 데이터 중 label2에 해당하는 데이터 가져오기

df.loc['전투원', '방어'] # index 전투원의 방어력 데이터

df.loc[['전투원', '요리사'], '출신'] # index 전투원, 요리사의 출신 데이터
df.loc[['전투원', '요리사'], ['이름', '공격']] # index 전투원, 요리사의 이름, 공격 데이터
df.loc['전투원': '의사', '공격': '민첩'] # index 전투원부터 의사까지의 공격부터 민첩까지의 데이터
```

## ▼ 8. DataFrame: 데이터 선택(iloc)

- Purely integer-location based indexing for selection by position

pandas.DataFrame.iloc - pandas 1.4.3 documentation

Purely integer-location based indexing for selection by position. is primarily integer position based (from to of the axis), but may also be used with a boolean array. Allowed inputs are: A boolean array. A function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above).

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html?highlight=iloc#pandas.DataFrame.iloc>

- DataFrame: 원하는 데이터 선택(기본) 예제

```
# 파일 불러오기 및 DataFrame 객체 생성

import pandas as pd
```

```
dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'

df = pd.read_excel(dir + "onepice.xlsx", index_col="역할")
df
```

```
# 정수형 index 를 이용한 데이터 선택1

df.iloc[0] # 0번째 행 위치의 데이터 전체
df.iloc[4] # 4번째 행 위치의 데이터 전체

df.iloc[0:4] # 0부터 3번째 행 위치의 데이터 전체
```

```
# 정수형 index 를 이용한 데이터 선택2

df.iloc[0, 1] # 0번째 행의 1번째 열 데이터

df.iloc[[0, 1], 2] # 0, 1번째 행(선장, 전투원)의 2번째 열 데이터(공격)
df.iloc[[0, 1], [3, 4]] # 0, 1번째 행(선장, 전투원)의 3, 4번째 열 데이터(방어, 지력)
```

```
# 슬라이싱을 이용한 접근

df.iloc[0:4, 0:3]
```

## ▼ 9. DataFrame: 데이터 선택(조건)1

- DataFrame: 조건에 해당하는 데이터 선택 예제

```
# 파일 불러오기 및 DataFrame 객체 생성

import pandas as pd

dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'

df = pd.read_excel(dir + "onepice.xlsx", index_col="역할")
df
```

```
# 다양한 조건부 데이터 선택1: 부등호

df['공격'] >= 60 # 공격력이 60 이상인지 아닌지에 대해 True / False

filt = (df['공격'] >= 60)
df[filt] # filt가 True인 데이터 출력
df[~filt] # filt가 False인 데이터 출력

df[df['공격'] >= 60]
```

```
# 다양한 조건부 데이터 선택2: loc

df.loc[df['공격'] >= 60, '방어'] # 공격이 60 이상인 캐릭터들의 방어 데이터
df.loc[df['공격'] >= 60, ['이름', '방어', '지력']] # 공격이 60 이상인 캐릭터들의 이름/방어/지력 데이터
```

```
# 다양한 조건부 데이터 선택3: &, |


df.loc[(df['공격'] >= 60) & (df['출신'] == '이스트블루')] # 공격력이 60 이상이고, 출신이 이스트블루인 캐릭터들의 데이터
df.loc[(df['지력'] < 30) | (df['지력'] > 90)] # 지력이 30 미만이거나, 80보다 큰 캐릭터들의 데이터
```

## ▼ 10. DataFrame: 데이터 선택(조건)2

- String 함수를 이용하여 조건부 데이터 선택

#### Working with text data - pandas 1.4.3 documentation

Series and Index are equipped with a set of string processing methods that make it easy to operate on each element of the array. Perhaps most importantly, these methods exclude missing/NA values automatically. These are accessed via the attribute and generally have names matching the equivalent (scalar) built-in string methods: The string methods on Index are especially useful for cleaning up or transforming DataFrame columns.

 [https://pandas.pydata.org/docs/user\\_guide/text.html](https://pandas.pydata.org/docs/user_guide/text.html)

- DataFrame: String 함수를 이용하여 조건에 해당하는 데이터 선택 예제

```
# 파일 불러오기 및 DataFrame 객체 생성

import pandas as pd

dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'

df = pd.read_excel(dir + "onepiece.xlsx", index_col="역할")
df
```

```
# 다양한 조건부 데이터 선택1: str 함수

filt = df['출신'].str.startswith('이스트') # 이스트로 시작하는 출신에 해당하는 캐릭터 데이터
df[filt]

filt = df['이름'].str.contains('로') # 이름에 '로'가 포함되는 캐릭터 데이터
df[filt]
df[~filt] # 이름에 '로'가 포함되는 캐릭터를 제외한 나머지 데이터

filt = df['출신'].str.contains('블루')
df[filt]
```

```
# 다양한 조건부 데이터 선택2: isin 함수

roles = ['노스블루', '위대한항로']
filt = df['출신'].isin(roles)
df[filt]
```

## ▼ 11. DataFrame: 결측치

- DataFrame: 비어있는 데이터를 다른 값으로 채우는 예시

```
# 파일 불러오기 및 DataFrame 객체 생성

import pandas as pd

dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'

df = pd.read_excel(dir + "onepiece.xlsx", index_col="역할")
df
```

```
# 비어 있는 데이터(NaN) 채우기: fillna

df.fillna(" ")# NaN 을 빈칸으로 채우기
df.fillna("없음") # NaN 을 "없음"으로 채우기

df["특기"].fillna("없음", inplace = True) # 특기 데이터 중에서 NaN 을 "없음"으로 채우기
```

```
# 비어 있는 데이터(NaN) 채우기: fillna

import numpy as np

df["민첩"] = np.nan # 민첩 데이터 전체를 NaN 으로 채우기
df.fillna("모름") # NaN 을 "모름"으로 채우기
df.fillna("모름", inplace=True) # NaN 을 "모름"으로 채우기 (실제 반영)
```



- DataFrame: 비어있는 데이터를 삭제하는 예시

```
# 파일 불러오기 및 DataFrame 객체 생성

import pandas as pd

dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'

df = pd.read_excel(dir + "onepice.xlsx", index_col="역할")
df
```

```
# 비어 있는 데이터(NaN) 제외하기1: dropna

df.dropna() # NaN 을 포함하는 row 데이터 전체 삭제
# df.dropna(inplace = True)
```

```
# 비어 있는 데이터(NaN) 제외하기2: dropna
# axis: index (=row) or columns
# how: any or all

df.dropna(axis='index', how='any') # NaN 이 하나라도 있는 row 삭제
df.dropna(axis='columns') # NaN 이 하나라도 있는 column 삭제

df["출신"] = np.nan
df.dropna(axis='columns', how='all') # 데이터 전체가 NaN 인 경우에만 column 삭제
```

## ▼ 12. DataFrame: 데이터 정렬

- DataFrame: 데이터 정렬하는 예제

```
# 파일 불러오기 및 DataFrame 객체 생성

import pandas as pd

dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'

df = pd.read_excel(dir + "onepice.xlsx", index_col="역할")
df
```

```
# 오름차순 정렬과 내림차순 정렬1

df.sort_values('공격력') # 공격력 기준으로 오름차순 정렬
df.sort_values('공격력', ascending=False) # 공격력 기준으로 내림차순 정렬
```

```
# 오름차순 정렬과 내림차순 정렬2

df.sort_values(['공격력', '방어력']) # 공격력 기준으로 오름차순, 동일한 공격력 내에서는 방어력 기준으로 오름차순
df.sort_values(['공격력', '방어력'], ascending=False) # 공격력 기준으로 내림차순, 동일한 공격력 내에서는 방어력 기준으로 내림차순
df.sort_values(['공격력', '방어력'], ascending=[True, False]) # 공격력 기준으로 오름차순, 동일한 공격력 내에서는 방어력 기준으로 내림차순
```

```
# 오름차순 정렬과 내림차순 정렬3

df['민첩'].sort_values() # "민첩" 행만을 오름차순 정렬
df.sort_index() # Index 기준으로 오름차순 정렬
```

## ▼ 13. DataFrame: 데이터 수정

- DataFrame: 데이터를 원하는 형태로 수정하는 예제

```
# 파일 불러오기 및 DataFrame 객체 생성

import pandas as pd
```

```
dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'
```

```
df = pd.read_excel(dir + "onepiece.xlsx", index_col="역할")
df
```

```
# Column 수정
```

```
df['출신'].replace({'위대한항로': '신세계'})
# df['출신'].replace({'위대한항로': '신세계'}, inplace=True)
df['출신'].replace({'위대한항로': '신세계', '이스트블루': '동쪽바다'})
# df['출신'].replace({'위대한항로': '신세계', '이스트블루': '동쪽바다'}, inplace=True)

df['출신'] = df['출신'] + ' 바다'
```

```
# Column 삭제
```

```
df.drop(columns=['총합']) # 총합 column 삭제
df.drop(columns=['출신', '공격']) # 총합 column 삭제
```

```
# Column 순서 변경
```

```
cols = list(df.columns)
df = df[[cols[-1]] + cols[0:-1]] # 맨 뒤 column을 맨 앞으로 가져오기
```

```
# Row 추가
```

```
df.loc['연구원'] = ['야마토', '위대한항로', 90, 90, 90, 90, '파랑색'] # 새로운 row 추가
```

```
# Row 삭제
```

```
df.drop(index='음악가') # index 음악에 해당하는 row 데이터 삭제

filt = df['지력'] < 50
df[filt].index
df.drop(index=df[filt].index)
```

```
# Cell 수정
```

```
df.loc['항해사', '특기'] = '무장색'
df.loc['의사', ['출신', '특기']] = ['사우스블루', '건문색']
```

## ▼ 14. DataFrame: 함수 적용 (apply)

- DataFrame: 데이터에 함수를 적용하는 예제

```
# 파일 불러오기 및 DataFrame 객체 생성
```

```
import pandas as pd
```

```
dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'
```

```
df = pd.read_excel(dir + "onepiece.xlsx", index_col="역할")
df
```

```
# 데이터에 함수 적용: apply
```

```
# 함수 정의
```

```
def add_joul(attack):
    return str(attack) + ' Joul'
```

```
df['공격'] = df['공격'].apply(add_joul) # 공격 데이터에 대해서 add_foul 함수를 호출한 결과 데이터 적용
```

## ▼ 15. DataFrame: 그룹화 (groupby)

- 동일한 값을 가진 데이터들끼리 합쳐서 통계 또는 평균 등의 값을 계산하기 위해 사용

pandas.DataFrame.groupby - pandas 1.4.3 documentation

Group DataFrame using a mapper or by a Series of columns. A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

 <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html?highlight=groupby#pandas.DataFrame.groupby>

- DataFrame: 데이터 그룹화 예제

```
# 파일 불러오기 및 DataFrame 객체 생성

import pandas as pd

dir = '/content/drive/MyDrive/Colab Notebooks/C2-004_Python/Lecture_04/'

df = pd.read_excel(dir + "onepice.xlsx", index_col="역할")
df
```

```
# 데이터 그룹화 예제1

df.groupby('출신') # 출신을 기준으로 그룹화

df.groupby('출신').get_group('이스트블루') # "이스트블루" 출신 그룹화 결과
df.groupby('출신').get_group('위대한항로') # "위대한항로" 출신 그룹화 결과
```

```
# 데이터 그룹화 예제2

df.groupby('출신').mean() # 계산 가능한 데이터들의 그룹별 평균값
df.groupby('출신')['공격'].mean() # 공격 데이터에 대한 그룹별 평균값
df.groupby('출신')[['공격', '방어', '지력']].mean() # 공격 데이터에 대한 그룹별 평균값
```

```
# 데이터 그룹화 크기 확인

df.groupby('출신').size() # 각 그룹의 크기
df.groupby('출신').size()['웨스트블루']
```

```
# 추가 데이터 및 그룹화 응용 예제1

df['나이'] = [17, 17, 17, 16, 16, 18, 18, 18, 20, 20] # Column 추가

df.groupby(['출신', '나이']).mean() # 출신별, 나이별 데이터 평균 값 계산
df.groupby('나이').mean()

df.groupby('나이').mean().sort_values('공격', ascending=False)
df.groupby(['나이', '출신']).mean().sort_values('나이')

df.groupby('출신')[['이름', '특기']].count() # 출신으로 그룹화를 한 뒤, 학교별 특기 데이터의 수 확인
```

```
# 추가 데이터 및 그룹화 응용 예제2

home = df.groupby('출신')
home['나이'].value_counts() # 출신으로 그룹화를 한 뒤, 나이별 학생 수 확인

home['나이'].value_counts().loc['위대한항로'] # 출신으로 그룹화 한 뒤, 나이별 인원 수 확인
home['나이'].value_counts(normalize=True).loc['위대한항로'] # 출신으로 그룹화 한 뒤, 나이별 인원 비율 확인
```