DataFrame - Pandas

Pandas

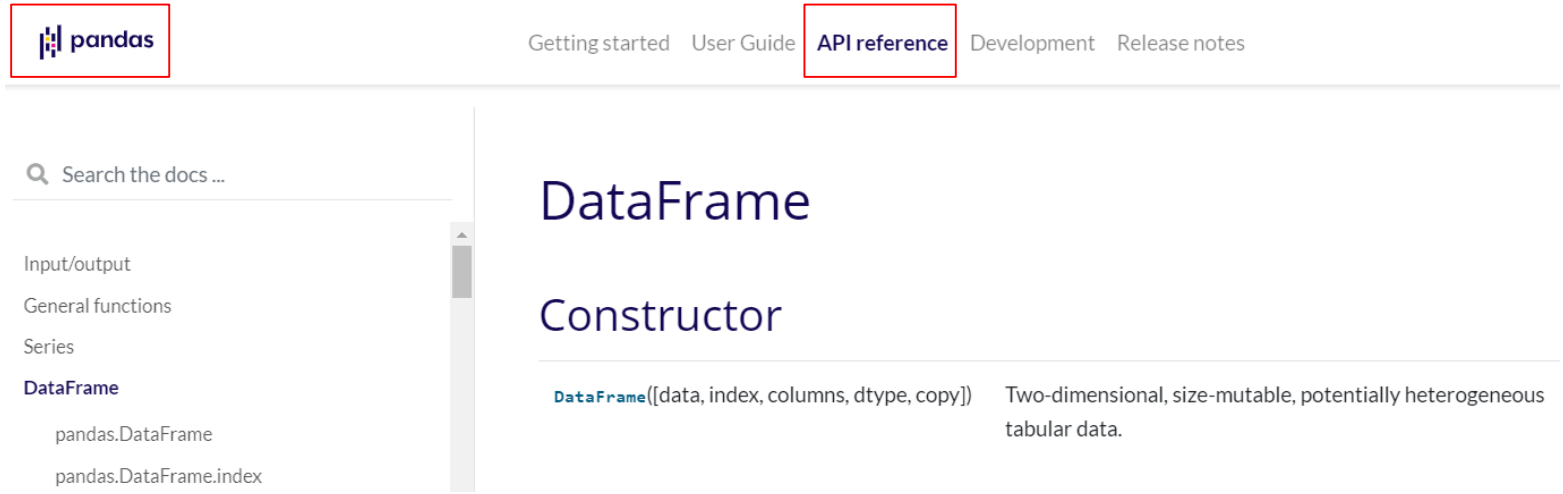
- **판다스** Pandas 라이브러리
  - 데이터를 수집하고 정리하는데 최적화된 도구
  - 오픈소스

- **판다스 자료구조** **vs. 파이썬 기본 자료구조 list, dictionary**
  - 판다스는 시리즈(Series)와 데이터프레임(DataFrame)이라는 구조화된 데이터 형식을 제공

- **API reference (pandas)**
  - https://pandas.pydata.org/docs/reference/index.html

데이터프레임이란

- **데이터프레임Dataframe은 2차원 배열 구조**
  - 2차원 배열 구조는 M/S Excel과 데이터베이스(RDBMS) 등 컴퓨터 관련 다양한 분야에서 사용
  - 대표적 통계 패키지인 R의 데이터프레임에서 유래



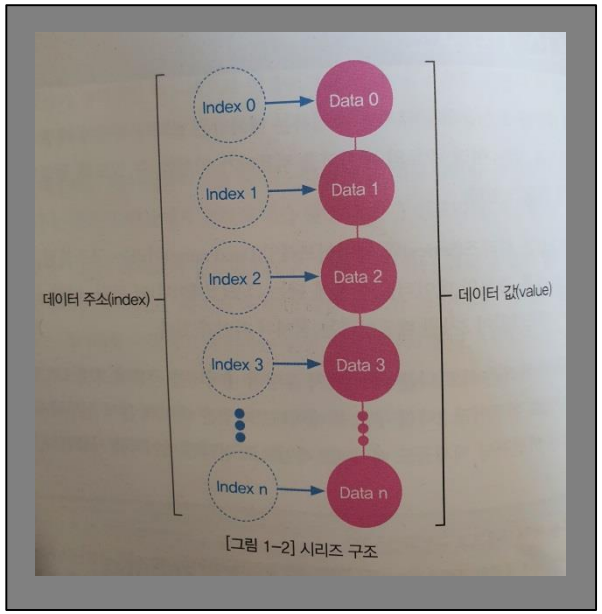
시리즈

- 시리즈Series
  - 시리즈는 데이터가 순차적으로 나열된 1차원 배열의 형태
  - 인덱스Index와 데이터 값Value이 일대일 대응
  - 키 Key와 값 Value이 짝을 이루는 딕셔너리 Dictionary와 비슷한 구조

- 시리즈 만들기
  - 리스트를 시리즈로 변환하는 경우, 0,1,2…로 index 포함
  - 딕셔너리를 시리즈로 변환하는 경우, 키는 index, 값은 value에 대응

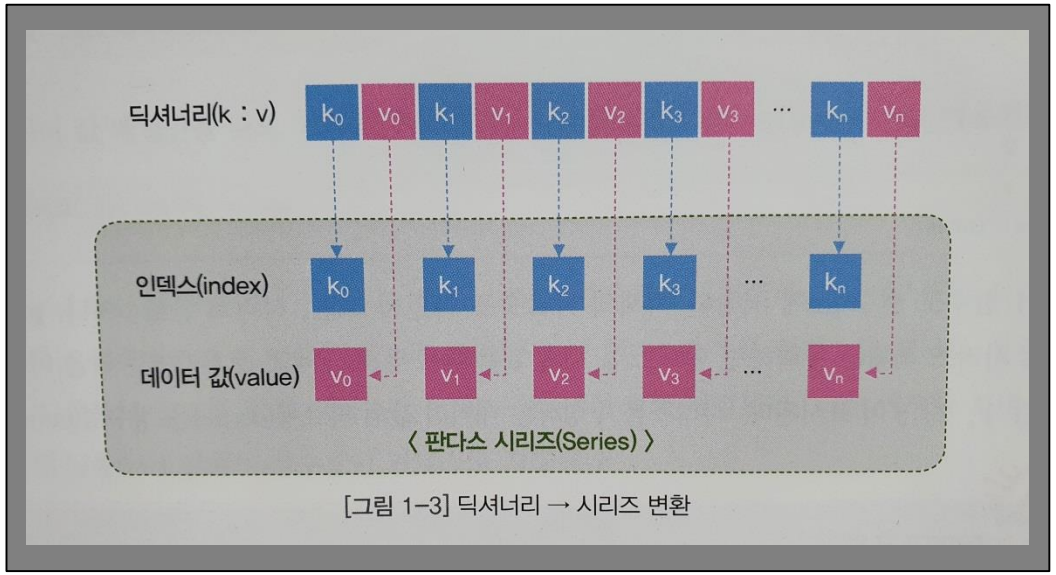
시리즈 구조 및 딕셔너리

- 시리즈 구조



[그림 1-2] 시리즈 구조

- 딕셔너리 -> 시리즈 변환



[그림 1-3] 딕셔너리 → 시리즈 변환

[3] ref.

시리즈 만들기

- 리스트 -> 시리즈 변환 : **(default)** 정수형 위치 인덱스

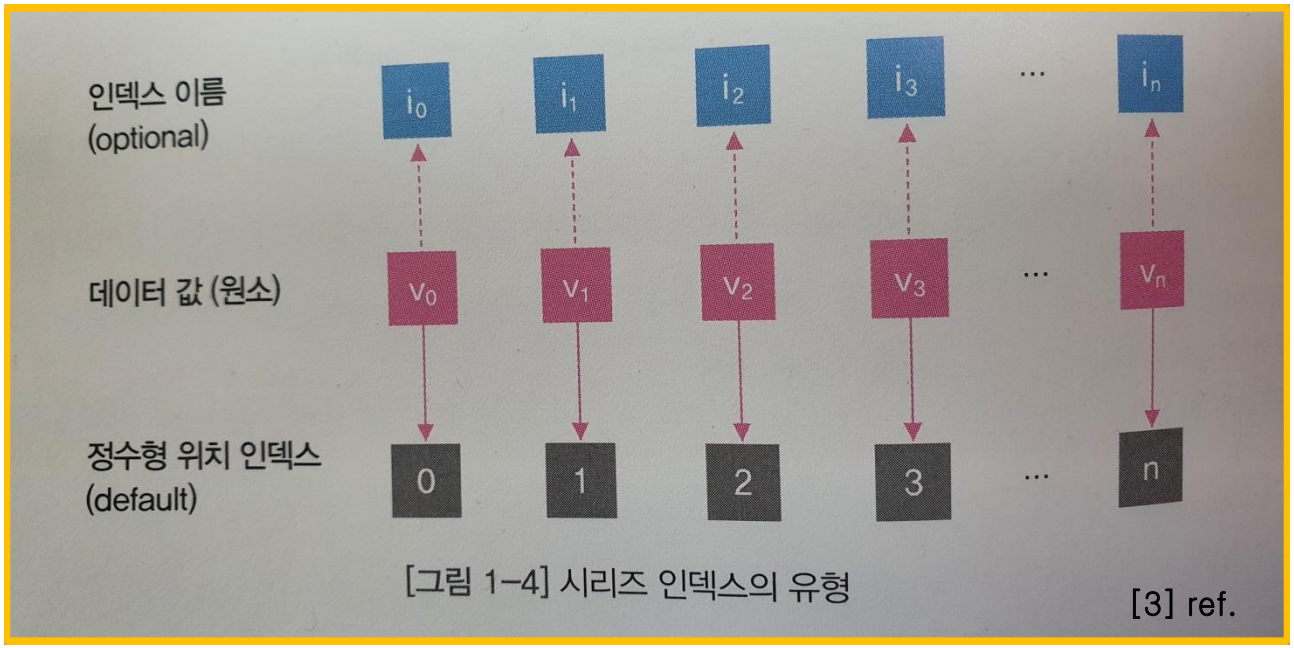
```
import pandas as pd

#list -> series
sr = pd.Series([10,20,30,40])
sr
```

- 딕셔너리 -> 시리즈 변환

```
dict_data = {'a':1, 'b':2, 'c':3}

#dictionary -> series
sr = pd.Series(dict_data)
sr
```

인덱스 구조

- 두 종류의 인덱스Index 표현
  - 정수형 위치 인덱스
  - 인덱스 이름



[그림 1-4] 시리즈 인덱스의 유형

[3] ref.

원소 선택

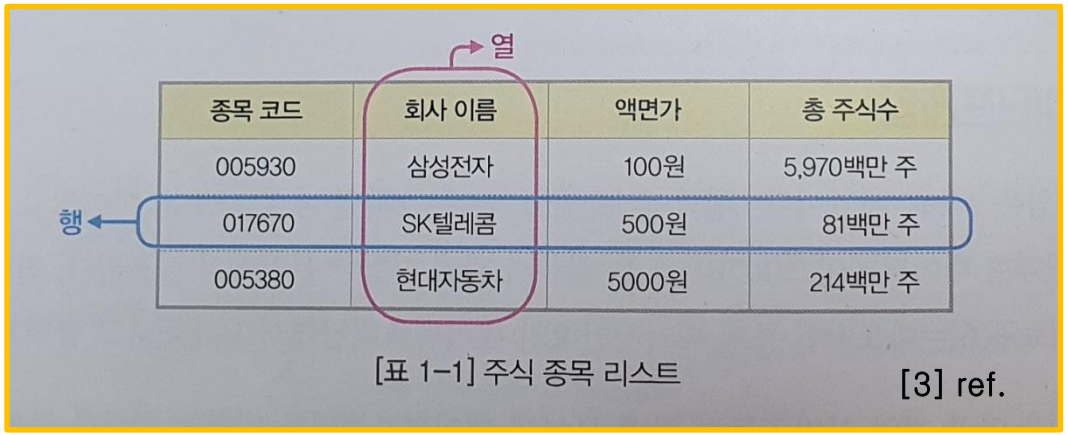
- **vs.** 리스트 인덱싱, 슬라이싱 Slicing
  - 원소의 위치를 나타내는 인덱스를 이용하여 시리즈의 원소를 선택

    vs. 리스트 인덱싱
  - 대괄호 **[ ]** 에 두 가지 인덱스 모두 사용 가능

```
import pandas as pd

dict_data = {'a':1, 'b':2, 'c':3}
sr = pd.Series(dict_data)

sr['b']

sr[1:2]

sr['b':'c']
```

데이터프레임

- **데이터프레임DataFrame은 2차원 배열 구조**
  - 2차원은 열과 행으로 만들어지며, 각각의 열은 시리즈 객체
  - 열과 행이 사용하는 주소는 각각 행 인덱스(row index)와 열 이름(column name)
  - 열은 공통의 속성을 갖는 일련의 데이터
  - 행은 개별 관측대상에 대한 다양한 속성 데이터들의 모음인 레코드(record)

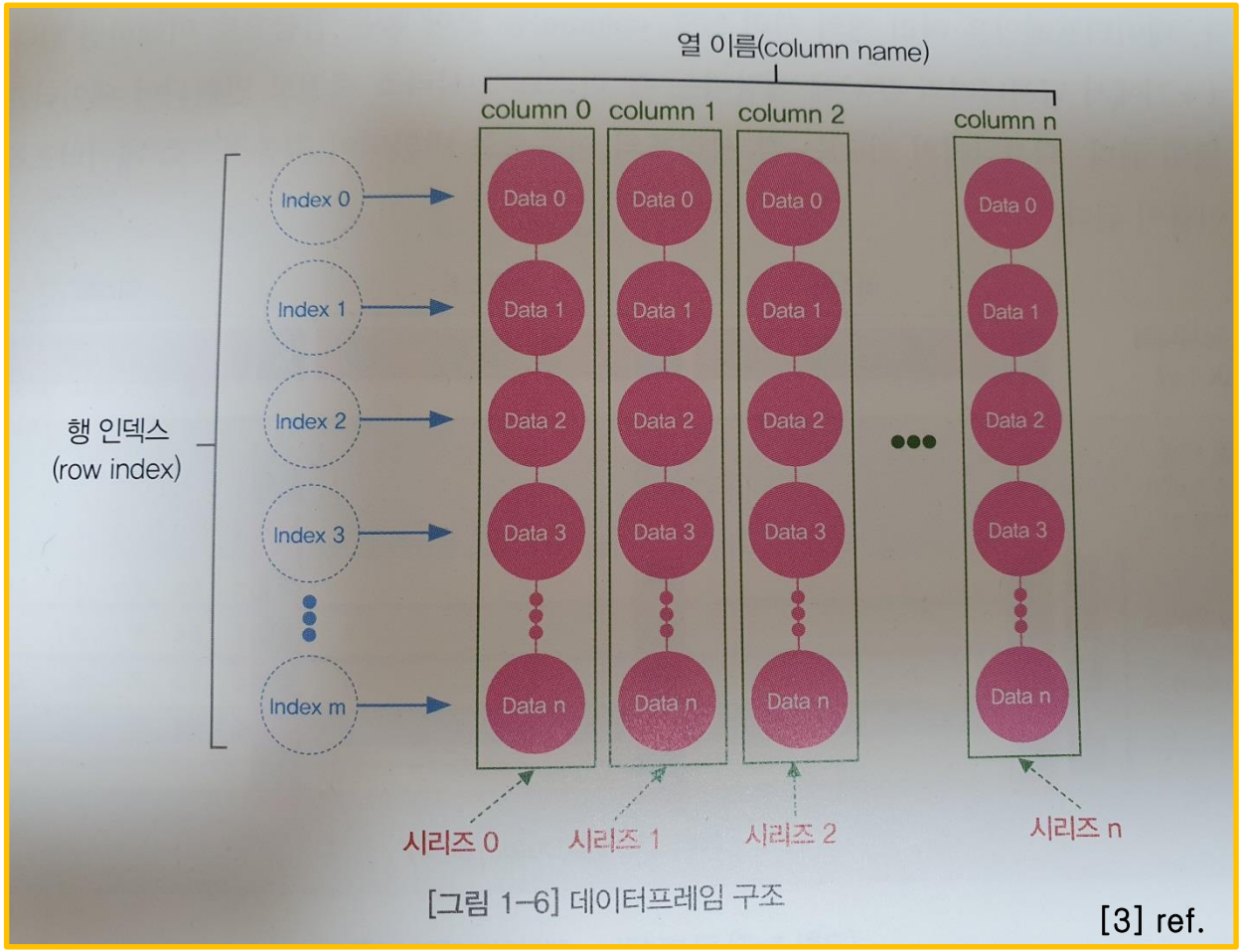


종목 코드	회사 이름	액면가	총 주식수
005930	삼성전자	100원	5,970백만 주
017670	SK텔레콤	500원	81백만 주
005380	현대자동차	5000원	214백만 주

[표 1-1] 주식 종목 리스트

[3] ref.

데이터프레임 구조



[그림 1-6] 데이터프레임 구조

[3] ref.

데이터프레임 만들기1

- **DataFrame 만들기1**
  - **Dictionary** 자료구조를 **DataFrame** 으로 변환하여 생성
  - 딕셔너리의 **key**는 **DataFrame**의 열이름으로
  - 딕셔너리의 **list**로 된 **value**는 **DataFrame**의 열로

```
import pandas as pd

#dictionary -> DataFrame

Dict_data = {'name' : ['Jerry', 'Riah', 'Paul'],
             'algol' : ['A', 'A+', 'B'],
             'basic' : ['C', 'B', 'B+'],
             'C++' : ['B+', 'C', 'C+']
             }
df = pd.DataFrame(dict_data)
```

pandas.DataFrame

- **dictionary -> DataFrame 변환**

[그림 1-7] 딕셔너리 → 데이터프레임 변환

[3] ref.

데이터프레임 만들기2

- **DataFrame 만들기2**
  - 2차원 리스트 자료구조를 **DataFrame** 으로 변환하여 생성
  - 생성자의 인자로 행 인덱스와 열 이름 설정 가능

    index=[ ], columns=[ ]

```
import pandas as pd

#2차원 List -> DataFrame

list_data = [[15,'남','덕영중'],
             [17, '여','수리중'],
             [16, '여','연수중']
            ]

df = pd.DataFrame(list_data)
```

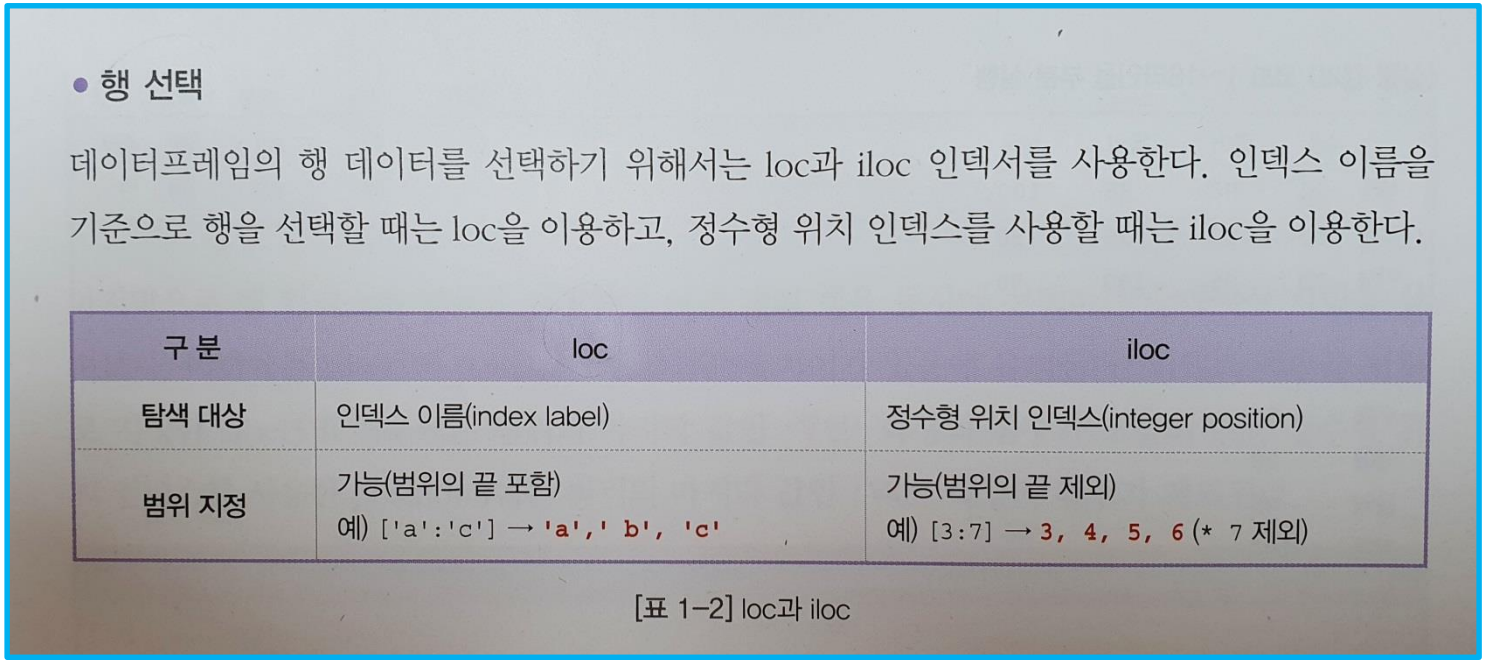
행/열 삭제

- 행/열 삭제
  - **df1 = df.drop('예은‘) : df는 그대로, 삭제된 값 df1에 반환**
  - **인자1 : 행 인덱스, 열 이름 또는 리스트**
  - **인자2 : 행 선택 axis=0 (default) , 열 선택 axis=1**

```
#list_data는 이전 예제 참조
df = pd.DataFrame(list_data, index=['준서', '예은', '길동'],
columns=['나이', '성별', '학교'])

#행 삭제
df1 = df.drop('예은')
df2 = df.drop(['준서','길동'])

#열 삭제
df5 = df.drop('성별', axis=1)
```

행 선택

- **loc 와 iloc 인덱서**
  - loc는 인덱스 명으로, iloc는 정수형 위치 인덱스로 선택



● 행 선택

데이터프레임의 행 데이터를 선택하기 위해서는 loc과 iloc 인덱서를 사용한다. 인덱스 이름을 기준으로 행을 선택할 때는 loc을 이용하고, 정수형 위치 인덱스를 사용할 때는 iloc을 이용한다.

구 분	loc	iloc
탐색 대상	인덱스 이름(index label)	정수형 위치 인덱스(integer position)
범위 지정	가능(범위의 끝 포함) 예) `['a':'c']` → `'a',' b', 'c'`	가능(범위의 끝 제외) 예) `[3:7]` → `3, 4, 5, 6` (`*` 7 제외)

[표 1-2] loc과 iloc

[3] ref.

슬라이싱

- 슬라이싱 : 범위 지정
  - 예) **iloc[3:7]** : 정수형 위치 인덱서로, 3행부터 6행까지 선택 (7은 제외)

```
#df는 이전 예제 참조

#행선택
sr1 = df.loc['길동']
sr2 = df.iloc[0]

#2개이상의 행 list형태로
df1 = df.loc[['준서','길동']]
df2 = df.iloc[[0,2]]

#범위 slicing
df3 = df.loc['예은':'길동']
df4 = df.iloc[0:2]
```

열 선택

- 열 선택
  - 데이터프레임의 열 데이터를 1개만 선택할 때는, 대괄호 [ ] 안에 열 이름을 따옴표와 함께 입력
  - 대괄호 안에 열 이름의 리스트를 입력하면 리스트의 원소인 열 모두 선택하여 데이터프레임으로 반환

```
#df는 이전 예제 참조

#열선택
sr5 = df['나이']

#2개이상의 열 list형태로
df5 = df[['나이', '학교']]
```

원소 선택

- 원소 선택
  - 데이터프레임의 행 인덱스와 열 이름을 [행 , 열] 형식의 2차원 좌표로 입력하여 원소 위치를 지정
  - 원소 선택은 시리즈 및 데이터프레임 단위로 반환도 가능

```
#df는 이전 예제 참조

#원소선택
el1 = df.loc['예은','학교']
el2 = df.iloc[1, 2]

#2개이상의 열 list형태로
sr1 = df.loc['준서', ['나이', '학교']]

#2개이상의 행,열 슬라이싱
df1 = df.loc['준서':'예은','나이':'학교']
```

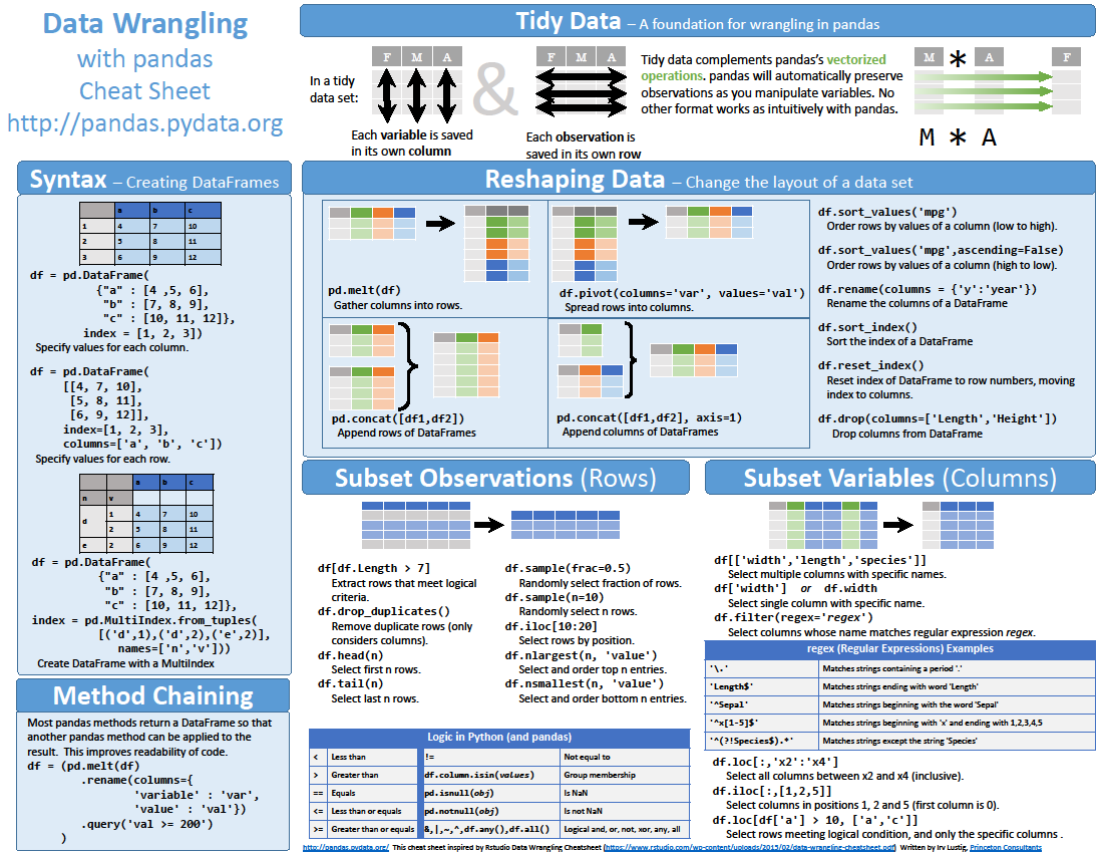
행/열 추가

- 행/열 추가, 원소값 변경

```
#df는 이전 예제 참조
df1 = df.copy()

#열추가
df1['수학'] = 80
df1['과학'] = [90, 80, 70]

#행추가
df1.loc['희수'] = [15, '남', '반포중', 100, 95]

#기존행 복사
df1.loc['수연'] = df1.loc['예은']

#원소값 변경
df1.loc['예은','수학']=75
df1.iloc[1,4]=75

df1.loc['예은',['수학','과학']]=85, 90
```

나만의 cheat sheet 만들기

- 데이터프레임 예제



Data Wrangling
with pandas
Cheat Sheet
http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
        index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v']))
```
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
     )
```

Reshaping Data – Change the layout of a data set

`pd.melt(df)`
Gather columns into rows.

`df.pivot(columns='var', values='val')`
Spread rows into columns.

`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`
Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(columns=['Length','Height'])`
Drop columns from DataFrame

Subset Observations (Rows)

`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Subset Variables (Columns)

`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`'^(?!Species$).*'`	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	`&,|,~,^,df.any(),df.all()`	Logical and, or, not, xor, any, all

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (http://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

기계가 읽을 수 있는 데이터 [chap03]

pandas module로 읽기

- **CSV** Comma Separated Values
  - pandas.read_csv( )
- **JSON** JavaScript Object Notation
  - pandas.read_json( )

② pandas.read_csv( )

- **pandas module로 읽기**

```
import pandas as pd
df = pd.read_csv('data-text.csv', encoding='cp949')   # encoding
df                                                    # Dataframe
```

  - pandas package 설치 : conda install pandas

- **API reference (pandas)**

  - https://pandas.pydata.org/docs/reference/index.html Flat file

② pandas.read_json( )

- **pandas module로 읽기**

```
import pandas as pd
df = pd.read_json('data-text.json')        # encoding : default is ‘utf-8’
df                                         # pandas object  -> type( )
```

- **API reference (pandas)**

pandas.read_json(*args, **kwargs) [source]

Convert a JSON string to pandas object.

Parameters: **path_or_buf** : *a valid JSON str, path object or file-like object*

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, and file. For file URLs, a host is expected. A local file could be: file://localhost/path/to/table.json.

데이터 저장 방법 선택 [chap05]

pandas module로 읽기

- 파일
  - csv 형식으로 파일 저장 실습
- 데이터베이스
  - SQLite로 DB저장 실습

CSV (Comma Separated Values)

- 형식
  - 레코드 내의 각 필드가 콤마(쉼표)로 구분되어 있는 파일
  - 각 레코드는 줄 바꿈 문자로 구분
  - 확장자: .csv

- 실험 데이터Experiment Data [1] ref. chap3
  - github : https://github.com/jackiekazil/data-wrangling
  - data-text.csv (github) : 국가별 기대수명 데이터

② pandas to_csv( )

- **pandas module**로 일부 행들만 쓰기

```
import pandas as pd

file_path = 'data-text.csv'
df = pd.read_csv(file_path, encoding='utf-8')


#iloc인덱서 & slicing
df100 = df.iloc[:100]

#csv write
df100.to_csv('data-text100.csv')
```

SQLite

- **SQLite란**
  - SQLite는 내장 가능한 오픈소스 데이터베이스로, C로 작성됐으며 일반적인 SQL로 쿼리가 가능
  - Anaconda를 설치하면 파이썬 내부에 기본적으로 설치가 되는 sqlite3모듈을 사용해서 데이터베이스를 생성, 조회, 갱신, 삭제 (CRUD) 작업 가능

    즉, 모듈 install이 필요 없음

- **파이썬으로 sqlite3 활용하는 2가지 방법**
  - 하나는 DB에 연결하고 커서를 만든 다음에 SQL문을 보내서 DB에 저장
  - 또 하나는 판다스 데이터프레임을 이용하면 함수를 통해 쉽게 저장

② pandas to_sql()

- **pandas로 SQL활용**

```
#1. DB 연결
con = sqlite3.connect("C:/Temp/sqlite3/test.db")

# DataFrame 생성 예
data = {'name' : ['Jerry', 'Riah', 'Paul'],
        'algol' : ['A', 'A+', 'B'],
        'basic' : ['C', 'B', 'B+'],
        'C++' : ['B+', 'C', 'C+']
       }
df = pd.DataFrame(data)

#2. DB 저장
df.to_sql('score',con, if_exists='append', index=False)


#3. DB 연결 종료 (이전 예제 참고)
```

pandas.DataFrame.to_sql

- **API reference (pandas)**
  - https://pandas.pydata.org/docs/reference/index.html DataFrame

pandas.DataFrame.to_sql

`DataFrame.to_sql(name, con, schema=None, if_exists='fail', index=True, index_label=None, chunksize=None, dtype=None, method=None)` [source]

Write records stored in a DataFrame to a SQL database.

Databases supported by SQLAlchemy [1] are supported. Tables can be newly created, appended to, or overwritten.

Parameters: **name** : *str*

Name of SQL table.

con : *sqlalchemy.engine.(Engine or Connection) or sqlite3.Connection*

Using SQLAlchemy makes it possible to use any DB supported by that library. Legacy support is provided for sqlite3.Connection objects. The user is responsible for engine disposal and connection closure for the SQLAlchemy connectable See here.

참고도서

- **reference**

[1] 파이썬을 활용한 데이터길들이기

- 프로그래밍인사이트

[2] 모두의 데이터분석

- 길벗

[3] 파이썬머신러닝 판다스데이터분석

- 정보문화사

End