# 기계가 읽을 수 있는 데이터

# 데이터의 종류

- 기계가 읽을 수 있는 데이터 Machine-readable Data

  사람의 개입 없이 의미를 잃지 않으면서 컴퓨터가 쉽게 읽을 수 있는 형식 format의 데이터 (2019, OPEN Government Data Act)

  - CSV
  - JSON
  - XML

- 인간이 읽을 수 있는 데이터 Human-readable Data

  - PDF
  - PPT

# CSV (Comma Separated Values)

- 형식
  - 레코드 내의 각 필드가 콤마(쉼표)로 구분되어 있는 파일
  - 각 레코드는 줄 바꿈 문자로 구분
  - 스프레드시트와 데이터베이스에서 가장 일반적인 가져오기 및 내보내기 형식
  - 콤마 대신 탭Tab도 가능
  - 확장자: .csv
  - 엑셀에서 .csv 형식으로 읽고 저장할 수 있다.

# CSV 파일을 읽을 때 주의사항

- 주의
  - 'csv'모듈은 텍스트 파일의 내용을 구분자에 따라서 잘라서 제공해주는 것 이상의 역할을 하지 않는다.
  - 각 필드의 값은 모두 문자열이며, 정수 및 실수 값으로 변환해서 처리해야 한다면, 이러한 값의 타입 변환의 책임은 항상 프로그래머에게 달려 있다.

# 실험 데이터 세트

- 국가별 기대수명에 대한 데이터Life-expectancy Data

  세계보건기구(World Health Organization, WHO) 데이터

  - WHO : http://bit.ly/life_expectancy_data -> csv, xml, json으로 제공

- 교재 파일 [1] ref. chap3

  - github : https://github.com/jackiekazil/data-wrangling -> data-text.csv

- 실험 데이터Experiment Data

  - data-text.csv (github)
  - seoul.csv (지난 수업 자료)

# ① csv.reader( )

- **csv module로 읽기**

```
import csv
with open('data-text.csv‘, encoding=‘cp949’) as f:      # f : file handler
    reader = csv.reader(f)                              # reader : csv reader object
    for row in reader:                                  #          -> iterate over lines
        print(row)
```

- **Python Standard Library**
  - https://docs.python.org/3/library/index.html File Formats / csv

# Python Standard Library

csv.**reader**(*csvfile*, *dialect='excel'*, ***fmtparams*)

Return a reader object which will iterate over lines in the given *csvfile*. *csvfile* can be any object which supports the iterator protocol and returns a string each time its `__next__()` method is called — file objects and list objects are both suitable. If *csvfile* is a file object, it should be opened with `newline=''`. [1] An optional *dialect* parameter can be given which is used to define a set of parameters specific to a particular CSV dialect. It may be an instance of a subclass of the `Dialect` class or one of the strings returned by the `list_dialects()` function. The other optional *fmtparams* keyword arguments can be given to override individual formatting parameters in the current dialect. For full details about the dialect and formatting parameters, see section Dialects and Formatting Parameters.

Each row read from the csv file is returned as a list of strings. No automatic data type conversion is performed unless the `QUOTE_NONNUMERIC` format option is specified (in which case unquoted fields are transformed into floats).

# Python Standard Library

- **Python Standard Library**
  - Built-in Function (내장함수)

    https://docs.python.org/ko/3/library/functions.html
  - Module : 외장함수, import 필요

    https://docs.python.org/ko/3/py-modindex.html

- **Package**
  - conda, pip 등 패키지매니저로 install 필요

# Module, Package and Library

- **Module**
  - A module is basically a bunch of related code saved in a file with the extension .py
  - 예) csv, random
- **Package**
  - Python packages are basically a directory of a collection of modules
  - 예) Numpy, pandas
- **Library**
  - a package is a collection of modules, a library is a collection of packages
  - 예) Matplotlib, Pytorch, Seaborn
- 참고
  - https://learnpython.com/blog/python-modules-packages-libraries-frameworks/

# import

- **module 전체 import**
  - import random
  - random.randint(10,20)
- **alias (별칭)으로 import**
  - import random as rd
  - rd.randint(20,30)
- **일부 함수만 import**
  - from random import randint
  - randint(1,6)

# with ~

- **open() ~ close()**

```
f = open('data-text.csv')
reader = csv.reader(f)
for row in reader :
    print( row )
f.close()
```

- **with로 file open : 자동 close()**

```
with open('data-text.csv') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

실습) list로 저장 관리

# open( file_path )

- 상대경로
  - **f = open(**‘data-text.csv’**)**
- 절대경로
  - **f = open('C:/Temp/datasets/infopub/part2/read_csv_sample.csv')**

# 한글 encoding

- **(기본) OS에 따라**
  - 윈도우의 기본 한글 인코딩 방식 cp949 (MS949)

- **사례**
  - 예를 들어, 윈도우에서 작성된 seoul.csv를 읽을 때 encoding='cp949' 생략 가능 (default)
  - seoul.csv를 리눅스나 macOS에서 읽는 경우, encoding='cp949'을 꼭 입력 (단, 윈도우에서도 IDE등에 따라 글자가 깨지는 경우도 입력이 필요)
  - 반대로 리눅스 등 다른 운영체제에서 작성된 csv파일을 윈도우에서 읽을 때는 encoding='uft8'을 꼭 입력

# encoding tips

- **cp949 또는 utf8 (utf-8) 둘 중에 하나 적용**
  - 일반적으로 영어 문자는 호환이 되지만, 한글이 포함된 경우 인코딩 방식이 달라 문자가 깨짐
  - 기본적으로 이 두 개의 한글 인코딩 방식을 적용해보면 한글 깨지는 문제 해결
  - 이외의 인코딩 방식은 Pandas API Reference의 encoding list 참조

- **메모장 활용 변환 가능**
  - ANSI ⇒ cp949 의미
  - UTF-8 ⇒ utf8

# ② pandas.read_csv( )

- **pandas package로 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

# Pandas API reference

## pandas.read_csv

pandas.**read_csv**(*filepath_or_buffer, sep=',', delimiter=None, header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None, decimal='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None, dialect=None, error_bad_lines=True, warn_bad_lines=True, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None*) [source]

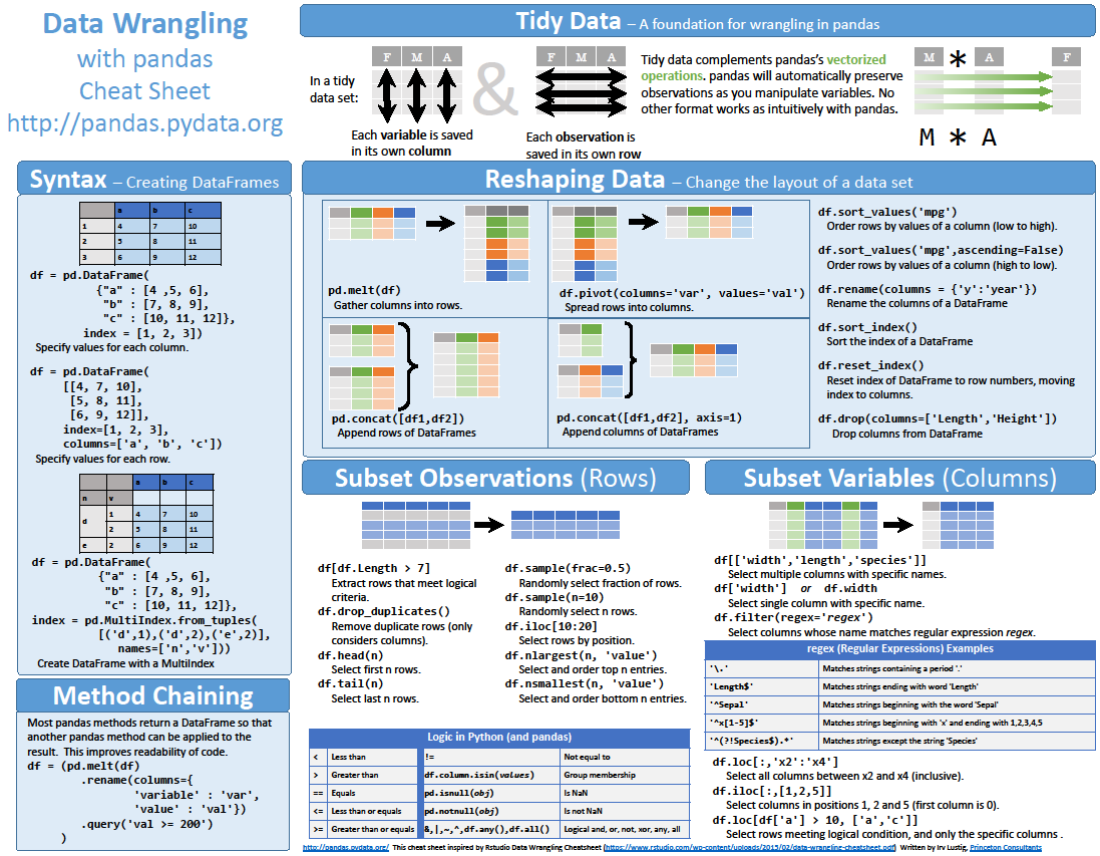
Read a comma-separated values (csv) file into DataFrame.

**encoding** : *str, optional*

Encoding to use for UTF when reading/writing (ex. 'utf-8'). List of Python standard encodings .

# 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:

F M A

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

&

F M A

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 F

M * A

## Syntax – Creating DataFrames

| | a | b | c |
|---|---|---|---|
| 1 | 4 | 7 | 10 |
| 2 | 5 | 8 | 11 |
| 3 | 6 | 9 | 12 |

```
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.

| | | a | b | c |
|---|---|---|---|---|
| n | v | | | |
| d | 1 | 4 | 7 | 10 |
| | 2 | 5 | 8 | 11 |
| e | 2 | 6 | 9 | 12 |

```
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

# JSON (JavaScript Object Notation)

- **형식**
  - 원래 웹 프로그래밍 언어인 자바스크립트의 데이터 객체 표현 방식을 사용하여 데이터 교환의 표준으로 삼은 데이터 표현 언어
  - 파이썬의 딕셔너리와 매우 비슷하여 속성-값attribute-value 쌍 또는 키-값key-value pair 쌍으로 이루어짐
  - XML을 대체하는 주요 데이터 형식
  - 확장자: .json
  - 컴퓨터 프로그램의 변수 값을 표현하는 데 적합

# JSON의 6가지 자료형

- **JSON의 6가지 자료형**
  - 수(Number)
  - 문자열(String)  cf. csv
    - 문자열은 큰 따옴표(")로 구분
    - 역슬래시('\') 이스케이프 문법escaping syntax을 지원한다.
  - 참/거짓(Boolean)
    - true 또는 false 값
  - 배열(Array)
    - 대괄호로 나타내며 요소는 쉼표로 구분한다.
  - 객체(Object)
    - 순서가 없는 이름-값 쌍의 집합으로, 이름(키)은 문자열이다.
  - null
    - 빈 값으로, null을 사용한다.

# JSON 예제

- **The following JSON example defines an employees object, with an array of 3 employees:**

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
  ]
}
```

  - { } 안의 것은 순서가 없는 이름-값 쌍이 있는 객체 cf. dictionary
  - 객체 안에 객체를 넣을 수도 있어서 XML처럼 복잡한 구조 또한 표현이 가능
  - [ ] 안의 것은 순서가 있는 배열

# vs. XML

- **The following XML example defines an employees object, with an array of 3 employees:**

```
<employees>
 <employee>
   <firstName>John</firstName> <lastName>Doe</lastName>
 </employee>
 <employee>
  <firstName>Anna</firstName> <lastName>Smith</lastName>
 </employee>
 <employee>
   <firstName>Peter</firstName> <lastName>Jones</lastName>
 </employee>
</employees>
```

# JSON vs. XML

- **JSON 우세?**
  - Both JSON and XML can be used to receive data from a web server.
  - Why JSON is Better Than XML
    - https://www.w3schools.com/JS/js_json_xml.asp

- **성능 백중세?**
  - Transfer, Parsing, Query 측면에서 봤을 때 성능이 거의 비슷함.
    - 33개의 다른 문서를 1200번 정도 OS, Brower를 바꾸어가며 테스트했다고 함
    - https://www.infoq.com/news/2013/08/xml-json-performance/

# 실험 데이터 세트

- 교재 파일 [1] ref. chap3
  - github : https://github.com/jackiekazil/data-wrangling -> data-text.json

- 실험 데이터 Experiment Data
  - data-text.json (github) : 국가별 기대수명 데이터

# ① json.load( )

- **json module로 읽기**

```
import json
with open('data-text.json') as f:
    py_obj = json.load(f)              # cf. csv
py_obj                                 # python object -> type( )
```

- **cf. json.loads( )** [1] ref. chap3 예제
  - string

# json conversion table

- **json.load()**

json.**load**(*fp*, ***, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize *fp* (a `.read()`-supporting text file or binary file containing a JSON document) to a Python object using this conversion table.

- **conversion table**

| JSON | Python |
|---|---|
| object | dict |
| array | list |
| string | str |
| number (int) | int |
| number (real) | float |
| true | True |
| false | False |
| null | None |

# 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.

# EXCEL

## 개념

- 마이크로소프트사에서 개발한 윈도 환경의 스프레드시트 프로그램
- 스프레드시트(spread sheet)란 여러 가지 도표 형태의 양식에 계산, 표기되는 사무업무를 자동으로 하는 표 계산 프로그램으로 계산기와 계산용지 등이 통합되어 연산 및 표를 작성하고 그래프를 그리는 소프트웨어

## 엑셀 파일을  파싱하기 전에

- 다른 형식의 데이터를 찾아 보았는가?
- 엑셀 또는 사용하는 문서 리더기에서 탭을 CSV 형식으로 내보내기를 해 보았는가? 만일 엑셀 탭의 개수가 적거나 혹은 하나의 탭에만 데이터가 들어 있다면 CSV 형식으로 내보내는 것도 좋은 방법이다.

# EXCEL 패키지

- xlrd – [1] ref. chap4
  - https://pypi.org/project/xlrd/
  - .xls 또는 .xlsx 파일을 읽기, 다른 형식으로 변환하기
  - 버전: 1.2.0 (dec 16, 2018)
  - 지원 파이썬 버전: Python 2.7 & 3.4+
  - 버전 특징: 엑셀 dates 지원, 유니코드 인식
  - API documentation:

    http://xlrd.readthedocs.io/en/latest/
  - 설치

    conda install xlrd

# PDF

- **Portable Document Format (PDF)**
  - 어도비Adobe가 1993년에 개발한 포스트스크립트PostScript 기반의 전자 문서 형식
  - 2008년 ISO 국제 표준으로 인정됨 (ISO 32000-1:2008): **PDF 1.7**
  - 2017년 ISO 32000-2:2017 (PDF 2.0)
  - 엄밀한 정의는 전자 문서의 디지털 인쇄물
    - PDF 파일을 만들 때 'save as'가 아닌 'print'로 만든다.
    - 따라서 출력된 PDF 파일 자체를 편집하는 것은 고려되지 않는다.
  - 어느 환경(소프트웨어, 하드웨어, 운영체제)에서나 동일한 결과물을 보여주기 위해 개발됨

# PDF 파일을 파싱하기 전에

- **PDF 사용을 자제하라!**
  - PDF 파일은 예측하기 힘든 형식으로 존재할 수 있어 엑셀 파일에 비해서 다루기 어렵다. 그럼에도 PDF 외에는 다른 옵션이 없는 경우도 있다.
  - 다른 형식의 데이터를 찾아 보았는가?
  - PDF 문서의 데이터를 복사/붙여넣기 해보았는가?(물론, 항상 가능한 것이 아니고, 규모 확장성이 없다)

- pdfminer
  - [1] ref. chap5

# 참고도서

## ➢ reference


[1] 파이썬을 활용한 데이터길들이기

- 프로그래밍인사이트

[2] 모두의 데이터분석

- 길벗

[3] 파이썬머신러닝 판다스데이터분석

- 정보문화사

End