2025-04-03(Thu.) : 파이썬 라이브러리와 활용 1일차

- 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
- ⭐ 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)

Pandas는 무엇인가요?

- 데이터 분석 및 가공에 사용되는 파이썬 라이브러리

```
프레임워크 많은것을 주입해준다.
라이브러리는 개발자가 주도적으로 해야

엔지니어링 - 없던것을 만들어낸다
피처엔지니어링(특성공학) - 프레임워크가 아니고 라이브러리라고 하는 이유
데이터 전처리
데이터 조작
```

- pandas documentation

데이터 프레임

- 판다스 == 데이터 프레임
- 행과 열로 구성되어 있는 데이터 구조
- 엑셀
- 장판지
- 행열이 가장 이해하기 좋은 데이터 구조
- 행 - 가로 - row
- 열 - 세로 - column

```
#모듈 패키지(pandas) 가져오기
import pandas as pd  # as pd : 별칭(as: )
```

```
# df - dataFrame
# 대입 연산자 오른쪽에 있는것을 왼쪽 변수에 저장
df = pd.read_csv(dataFilePath, encoding="utf-8")

# 대략적으로 보고싶을 때 앞부분에 있는 출력
df.head()

# 대략적으로 보고싶을 때 뒷부분에 있는 출력
df.tail()
```

🔷 pandas.read_csv

```
read_csv 함수는 txt파일도 읽음

                헤더
csv파일을 다룰때는 타이틀을 포함하고 있는지 없는지 확인
                컬럼

헤더를 포함하고 있다
기본값을 헤더를 있다고 봄  헤더 키워드 인자를 고민하라
```

🔷 pandas.DataFrame.head

```
DataFrame.head(n=5)
🔷 Return the first n rows.

DataFrame.tail(n=5)[source]
🔷 Return the last n rows.
```

- 행(row)는 자동으로 숫자 인덱싱이 들어감.
- 열(column)

csv 파일 : 콤마를 구분자로 하는 단순 텍스트 파일

- 각 행의 수가 동일해야 함
- ','를 기준으로 하기 때문에 공백이 없어야 함
- read_csv 함수는 txt 파일도 읽음

판다스의 데이터 구조는 시리즈, 데이터프레임 두개로 구성

> **데이터프레임은 무엇인가요?**
>
> - 데이터프레임은 행,열이 여러개인 표 모양
> - 가로축과 세로축이 있는 엑셀과 유사한 데이터구조
> - 가로축은 로우(행), 세로축은 컬럼(열)

시리즈는 무엇인가요?

- 판다스가 제공하는 데이터 타입(객체) 중 하나
- 실제로는 시리즈를 이어붙여서 데이터 프레임을 만듬
- 시리즈는 열(column)이 한개인 표
- 데이터프레임의 컬럼(열)은 모두 시리즈임
- 위의 예제는 3개의 시리즈로 구성된 데이터프레임

```
type(data_frame.job)
type(data_frame.["jon"])
type(data_frame.[열인덱스])
#> pandas.core.series.Series
```

```
# 특정 컬럼(시리즈)를 선택 후 문자열을 대문자화 해서 업데이트
data_frame.job = data_frame.job.str.upper()
data_frame.head()
```

- 시리즈는 단순히 파이썬 리스트를 간직한 오브젝트.
- 시리즈는 Ndarray를 간직한 객체
  - 리스트 [1,2,3,4]
  - Ndarray [1 2 3 4]
- 시리즈는 단순히 파이썬 리스트를 간직한 오브젝트입니다.
- 시리스트를 파라미터로 주면 바로 시리즈가 생성됩니다.
- 시리즈는 데이터 가공 및 분석이 파이썬 리스트보다 훨씬 쉽습니다.
- 리스트를 파라미터로 주면 바로 시리즈가 생성됨
- 판다스가 제공하는 메소드로 가공 분석을 할수있는 상태로 바꿔준다.

```
s1 = pd.core.series.Series(['one', 'two', 'three'])
s2 = pd.core.series.Series([1, 2, 3])
```

- 리스트 1개를 입력
- 리스트 타입의 데이터를 pandas가 가지고 있는 데이터 타입인 시리즈로 변환
- s1 시리즈의 데이터 타입은 Object
- 리스트를 주어서 시리즈를 만듬

```
pd.DataFrame(data=dict(word=s1, num=s2))
```

- class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)
- data : ndarray (structured or homogeneous), Iterable, dict, or DataFrame
- index : Index or array-like
- columns : Index or array-like
- dtype : dtype, default None

- copy : bool or None, default None

왜 팬더스를 쓰나요?

- 엑셀과 상당히 유사합니다, 데이터의 수정/가공 및 분석이 용이합니다.
- 공식 API 문서를 확인
- 데이터 가공을 위한 수많은 함수를 지원합니다.
- Numpy 기반으로 데이터 처리가 상당히 빠릅니다. - 선형대수 연산
    - NumPy : NumPy is the fundamental package for scientific computing in Python.

파일을 데이터프레임으로 불러오기

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

> - 데이터프레임 (dataframe)은 2차원 자료구조
>   - 로우와 컬럼으로 엑셀 형식과 유사
> - 기본적으로 csv 포맷을 지원하지만 ',' 구분자로 컬럼이 구분되어 있는 데이터는 모두 지원
>   - 구분자 옵션을 통해 구분자가 있고 각 row(행)마다 길이가 같고 개행이 있고 마지막게 개행이 없으면 호출이 가능
> - read_csv 함수로 파일을 데이터 프레임으로 호출할 수 있음

```
df = pd.read_csv('data/friend_list.txt')
```

> - 파일 내용이 ','로 열(column)이 구분되어 있다면, 확장자 구분없이 dataFrame 생성이 가능
>   - df.head() 로 확인

```
df = pd.read_csv('data/friend_list_tab.txt', delimiter = "\t")
```

> - 열(column)들이 쉼표로 구분되어 있지 않을 경우라도, delimiter 파라미터에 구분자를 지정해줘서 컬럼을 나눠 줄 수 있음

```
df = pd.read_csv('data/friend_list_no_head.csv', header = None)
```

> - 만약 파일에 데이터 헤더가 없을 경우, header = None으로 지정해줘서, 첫번째 데이터가 데이터 헤더로 들어가는 것을 막을 수 있다.

```
df.columns = ['name', 'age', 'job']
```

> - 헤더가 없는 데이터를 데이터프레임으로 호출했을 경우, 데이터프레임 생성 후에, 컬럼 헤더를 지정해주실 수 있습니다.

- 헤더가 없는 데이터를 데이터프레임
    1. 만들면서 옵션으로 지정
        - [Python] pandas csv 읽을 때 칼럼명 지정하기: read_csv()
        - ⭐ read_csv() option정리
            - df = pd.read_csv("data/friend_list_no_head.csv", names = ['이름', '나이', '직업'])
    2. df.columns 프라퍼티(속성)를 리스트 타입으로 지정
        - df.columns = ['이름', '나이', '직업']

```
df = pd.read_csv('data/friend_list_no_head.csv', header = None, names=
['name', 'age', 'job'])
```

- 파일을 읽어서 Dataframe으로 변환할 때, 헤더를 임으로 지정해주실 수도 있음

데이터프레임을 파이썬 코드로 생성하기

딕셔너리로 데이터프레임 생성하기

- 파이썬의 기본 자료구조(List, Dictionary, Tuple)로 데이터프레임 생성이 가능

```
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)
```

- 딕셔너리로 Dataframe을 생성

OrderedDict로 데이터프레임 생성하기

- OrderedDict 자료구조로 데이터프레임을 생성하면, 컬럼의 순서가 뒤바뀌지 않음

```
from collections import OrderedDict
```

```
friend_ordered_dict =
                        OrderedDict(
                            [
                                ('name', ['John', 'Jenny', 'Nate']), # <--
Tuple
                                ('age', [20, 30, 30]),
                                ('job', ['student', 'developer',
'teacher'])
                            ]
                        )
```

```
df = pd.DataFrame.from_dict(friend_ordered_dict)
```

> - ⭐ DataFrame 은 클래스, from_dict은 인스턴스 없이 사용가능한 스테틱 메서드(유틸리티성 메서드)
> - OrderedDict 생성자 인자로 리스트 타입 데이터 1개 지정
> - 첫번째 튜플에는 문자열 1개와 문자열 타입 3개로 구성된 리스트 1개
> - 첫번째 튜플에는 문자열 1개와 정수형 타입 3개로 구성된 리스트 1개
> - 첫번째 튜플에는 문자열 1개와 문자열 타입 3개로 구성된 리스트 1개

list로 데이터프레임 생성하기

- 리스트로 데이터프레임을 생성하는 예제입니다.

```
1. 열을 묶어서 Dataframe 생성
# 일반적이 사용예 : 열(column)기반 데이터로 처리
이름들 = []
나이들 = []
직업들 = []
data = zip(이름들, 나이들, 직업들)

2. 행을 묶어서 Dataframe 생성
```

```
friend_list = [ ['John', 20, 'student'],['Jenny', 30, 'developer'],
['Nate', 30, 'teacher'] ]
column_name = ['name', 'age', 'job']
df = pd.DataFrame.from_records(friend_list, columns=column_name)
```

파일로 데이터프레임을 저장하기

```
friend_list = [
                ['name',['John', 'Jenny', 'nate']],
                ['age',[20,30,30]],
                ['job',['student', 'developer', 'teacher']]
              ]
#df = pd.DataFrame.from_items(friend_list) # deprecated
df = pd.DataFrame.from_dict(friend_list)
```

```
df.to_csv('friend_list_from_df.csv')
```

- to_csv 함수를 사용해서 파일로 저장
- 실행할 때마다 덮어쓴다.

```
df.to_csv('friend_list_from_df.txt')
```

- 파일의 확장자명은 임으로 적용 가능

```
df.to_csv('friend_list_from_df.csv', header = True, index = True)
```

- 기본적으로, 헤더와 인덱스값은 지정하지 않아도, 기본적으로 True로 설정되어 있음

```
df.to_csv('friend_list_from_df.csv', header = False, index = False)
```

- **header = False** 는 컬럼 이름을 파일에 저장하지 않겠다라는 의미
  - 0,1,2가 헤더에 저장되지 않습니다.
- **index = False** 는 로우 인덱스를 파일에 저장하지 않겠다라는 의미
  - 0,1,2가 로우 인덱스에 저장되지 않습니다.

```
df.to_csv('friend_list_from_df.csv', header = ['name', 'age', 'job'])
```

- 헤더 정보를 원할 경우, header 키워드로 컬럼 이름을 파일에 저장하실 수 있음

로우 선택하기

1. **인덱스로 로우 선택하기**

   ```
   df[1:3]
   ```

   - 로우 인덱스를 사용하여 로우1부터 2까지 순차적으로 선택

   ```
   df.loc[[0,2]]  #loc : location
   ```

   - 순차적이지 않은 로우를 선택

2. **컬럼값에 따른 로우 선택하기**

   ```
   df_filtered = df[df.age > 25]
   df_filtered = df.query('age>25')
   df_filtered = df[(df.age >25) & (df.name == 'Nate')]
   ```

- 데이터베이스에 쿼리를 전달하듯, 특정한 컬럼값을 충족하는 로우만 선택이 가능
  - RDBMS(관계형 데이터베이스) sql언어 표준 질의 문
  - DDL, DML, DCL
  - DDL 요청
- query 는 static Method

컬럼 필터하기

1. **인덱스로 필터하기**

```
df.iloc[:, 0:2]
```

- 모든 row(행)를 보여주되, column(열)은 0부터 1까지만 출력
- loc - row(행)
- iloc - column(열)

```
df.iloc[:,[0,2]]
```

- 모든 row(행)를 보여주되, column(열)은 0와 2만 출력

⭐ 4월 3일 오후강의 여기까지

2. **컬럼 이름으로 필터하기**

로우 드롭하기

드롭된 결과를 바로 데이터프레임에 저장하는 방법

로우 인덱스로 드롭하기

컬럼값으로 로우 드롭하기

컬럼 드롭하기