판다스(Pandas)

https://pandas.pydata.org/docs/user_guide/index.html

- 파이썬에서 데이터 분석과 조작을 위한 핵심 라이브러리입니다. 엑셀처럼 표 형태의 데이터를 다루지만 훨씬 강력하고 유연한 기능을 제공합니다.
    - 1차원은 '시리즈' 라고 부릅니다.
    - 2차원은 '데이터프레임'으로 부릅니다.

1. Series

- 1차원 데이터 구조로, 리스트나 배열과 유사하지만 인덱스(index)를 포함하여 저장합니다.
    - 인덱스를 통해 값에 빠르게 접근 가능
    - 엑셀의 한 열(column)과 비슷한 구조

```
import pandas as pd
```

```
# 산업별 AI 도입률 (%)

industry =  ['제조업', '금융업', '의료업', '물류업', '교육업']
rates =  [35.2, 48.7, 42.1, 29.5, 25.3]

type(industry)

```

```
list
```

```
# 리스트 데이터를  pd.Series(데이터) 로 실행하면 시리즈 생성됩니다.
ser1=pd.Series(industry)
ser1
```

	0
0	제조업
1	금융업
2	의료업
3	물류업
4	교육업

dtype: object

```
print(ser1.index)
print(ser1.values)
print(type(ser1.index))
print(type(ser1.values))  # 값은 넘파이의 ndarray 기반으로 저장
```

```
RangeIndex(start=0, stop=5, step=1)
[35.2 48.7 42.1 29.5 25.3]
<class 'pandas.core.indexes.range.RangeIndex'>
<class 'numpy.ndarray'>
```

```
ser1=pd.Series(rates)
ser1[2]
```

```
np.float64(42.1)
```

```
# 기존의 순차적인 인덱스 값 대신에 '레이블'을 사용할 수 있음.
ser1 = pd.Series(rates, index=industry, name='산업별 AI 도입률')
ser1
type(ser1)
```

```
pandas.core.series.Series
def __init__(data=None, index=None, dtype: Dtype | None=None, name=None, copy: bool | None=None, fastpath:
bool | lib.NoDefault=lib.no_default) -> None
```

```
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
```

```
ser1
```

	산업별 AI 도입률
제조업	35.2
금융업	48.7
의료업	42.1
물류업	29.5
교육업	25.3

dtype: float64

```
ser1['제조업']
```

```
np.float64(35.2)
```

```
print(ser1.index)
print(ser1.values)
print(type(ser1.index))
print(type(ser1.values))  # 값은 넘파이의 ndarray 기반으로 저장
```

```
Index(['제조업', '금융업', '의료업', '물류업', '교육업'], dtype='object')
[35.2 48.7 42.1 29.5 25.3]
<class 'pandas.core.indexes.base.Index'>
<class 'numpy.ndarray'>
```

- 딕셔너리로 시리즈 생성

```
data = {
    '제조업': 35.2,
    '금융업': 48.7,
    '의료업': 42.1,
    '물류업': 29.5,
    '교육업': 25.3
}
print(type(data))
```

```
ai_adoption_rate = pd.Series(data,name='산업별 AI 도입률')
ai_adoption_rate
```

```
<class 'dict'>
```

	산업별 AI 도입률
제조업	35.2
금융업	48.7
의료업	42.1
물류업	29.5
교육업	25.3

dtype: float64

2. 데이터 프레임

- 2차원 테이블 형태의 데이터 구조로, 여러 개의 시리즈가 모여 만들어집니다.
  - 행(row)과 열(column)로 구성된 구조
  - 다양한 데이터 타입을 열마다 다르게 가질 수 있음
  - 엑셀 시트와 유사한 형태

- 리스트와 딕셔너리를 데이터 프레임으로 생성하기

```
# 딕셔너리 데이터
dict_data = {
    '산업군': ['제조업', '금융업', '의료업', '물류업', '교육업'],
    '도입 기업 수': [1200, 850, 430, 670, 310],
    '평균 투자액': [3.2, 5.5, 4.1, 2.8, 1.9]
}

df = pd.DataFrame(dict_data)
df
```

	산업군	도입 기업 수	평균 투자액
0	제조업	1200	3.2
1	금융업	850	5.5
2	의료업	430	4.1
3	물류업	670	2.8
4	교육업	310	1.9

```
# company_count= [1200, 850, 430, 670, 310]    # 도입 기업수
# average_investment = [3.2, 5.5, 4.1, 2.8, 1.9]  # 평균 투자액
```

```
# 리스트 데이터
list_data = [['제조업',1200,3.2],
        ['금융업',850,5.5],
        ['의료업',430,4.1],
        ['물료업',670,2.8],
        ['교육업',310,1.9]]
df = pd.DataFrame(list_data, columns=['산업군','도입 기업 수','평균 투자액'])
df
```

	산업군	도입 기업 수	평균 투자액
0	제조업	1200	3.2
1	금융업	850	5.5
2	의료업	430	4.1
3	물료업	670	2.8
4	교육업	310	1.9

```
list_data = [[1200,3.2],
        [850,5.5],
        [430,4.1],
        [670,2.8],
        [310,1.9]]
industry =  ['제조업', '금융업', '의료업', '물류업', '교육업']
df = pd.DataFrame(list_data, columns=['도입 기업 수','평균 투자액'],index=industry)
df
```

	도입 기업 수	평균 투자액
제조업	1200	3.2
금융업	850	5.5
의료업	430	4.1
물류업	670	2.8
교육업	310	1.9

직렬화 / 역직렬화

- 직렬화는 파일에 저장할 목적 또는 네트워크으로 보낼 때(쓰기)
- 역직렬화는 파일의 내용 읽기 또는 네트워크로 수신된 내용 확인(읽기)

직렬화 파일 형식

- CSV (.csv) `df.to_csv()` 가장 널리 사용됨. 텍스트 기반, 범용성 높음
- Excel (.xlsx) `df.to_excel()` 엑셀 호환. 시트 이름 지정 가능
- JSON (.json) `df.to_json()` 웹 API, 계층적 데이터에 적합
- Pickle (.pkl) `df.to_pickle()` 파이썬 전용. 객체 그대로 저장, 빠름
- Parquet (.parquet) `df.to_parquet()` 대용량 데이터에 최적. 압축 및 빠른 I/O
- Feather (.feather) `df.to_feather()` 빠른 읽기/쓰기. 머신러닝에 적합
- SQL `df.to_sql()` 데이터베이스에 직접 저장 가능
- HTML (.html) `df.to_html()` 웹 페이지에 테이블로 출력 가능

```
pd.to_pickle(df, 'df.pkl')
```

```
df2 = pd.read_pickle('df.pkl')  # 파일에 저장된 것을 읽어서 df2 로 역직렬화
df2
```

	도입 기업 수	평균 투자액
제조업	1200	3.2
금융업	850	5.5
의료업	430	4.1
물류업	670	2.8
교육업	310	1.9

```
df.to_json('df.json', force_ascii=False)
```

csv 파일로 저장하기

```
df.to_csv('df.csv', encoding='cp949')  # utf-8 한글은 utf-8-sig
 # index=True 기본값 (인덱스로 지정된 레이블을 포함할 것이지.)
```

데이터 프레임 메소드

```
df.sort_values(by='도입 기업 수')
```

	도입 기업 수	평균 투자액
교육업	310	1.9
의료업	430	4.1
물류업	670	2.8
금융업	850	5.5
제조업	1200	3.2

```
df.sort_values(by='평균 투자액', ascending=False)
```

	도입 기업 수	평균 투자액
금융업	850	5.5
의료업	430	4.1
제조업	1200	3.2
물류업	670	2.8
교육업	310	1.9

```
pd.options.display.float_format = '{:.2f}'.format
df.describe()
```

	도입 기업 수	평균 투자액
count	5.00	5.00
mean	692.00	3.50
std	352.87	1.37
min	310.00	1.90
25%	430.00	2.80
50%	670.00	3.20
75%	850.00	4.10
max	1200.00	5.50

```
df.tail() # 데이터가 많을 때 뒤에 5개만 보여줌
# df.head(2)
# df.tail(2)
```

숨겨진 출력 표시

인덱싱

- 특정 요소 또는 열/행 선택 `df['열이름']`, `df.loc[행, 열]`, `df.iloc[행, 열]`
- 인덱스 또는 라벨 기반
- 반환값 : 단일 값 또는 시리즈/데이터프레임

1. 열 인덱싱

- 단일 열 선택: df['열이름']
- 여러 열 선택: df[['열1', '열2']]

```
dict_data = {
    '산업군': ['제조업', '금융업', '의료업', '물류업', '교육업'],
    '기업수': [1200, 850, 430, 670, 310],
    '투자액': [3.2, 5.5, 4.1, 2.8, 1.9]
}

df = pd.DataFrame(dict_data)
df
```

	산업군	기업수	투자액
0	제조업	1200	3.2
1	금융업	850	5.5
2	의료업	430	4.1
3	물류업	670	2.8
4	교육업	310	1.9

```
# 데이터프레임에서 [ ]는 열(column)을 가져오려는 동작
df['산업군']
```

	산업군
0	제조업
1	금융업
2	의료업
3	물류업
4	교육업

dtype: object

```
df[['산업군','투자액']]
```

	산업군	투자액
0	제조업	3.20
1	금융업	5.50
2	의료업	4.10
3	물류업	2.80
4	교육업	1.90

2. 행 인덱싱

- .loc[]: 라벨 기반 인덱싱(행,컬럼의 이름으로 가져오기)
- .iloc[]: 정수 위치 기반 인덱싱(행, 컬럼의 인덱스 로 가져오기)

```
df.loc[0]        # 인덱스 라벨이 0인 행
```

	0
산업군	제조업
기업수	1200
투자액	3.20

dtype: object

```
df.iloc[0]  # 첫 번째 행 (정수 위치 0)
```

	0
산업군	제조업
기업수	1200
투자액	3.20

dtype: object

- 행과 열 동시 인덱싱

```
df.loc[0, '투자액']        # 0번 행의 '투자액' 열
```

```
np.float64(3.2)
```

```
df.iloc[0, 1]           # 첫 번째 행의 두 번째 열
```

```
np.int64(1200)
```

```
df.loc[[0, 2], ['산업군', '기업수']]  # 0, 2번 행의 'name', 'age' 열
```

	산업군	기업수
0	제조업	1200
2	의료업	430

슬라이싱

- 연속된 범위의 데이터 `df[start:end], df.loc[:, 'A':'C']`
- 접근 범위 지정 (start 포함, end 제외)
- 반환값 : 연속된 범위의 데이터 선택

```
df.loc[2:4]      # 인덱스 라벨 2~4까지 슬라이싱
```

	산업군	기업수	투자액
2	의료업	430	4.10
3	물류업	670	2.80
4	교육업	310	1.90

```
df.iloc[2:5]     # 위치 기준으로 2~4번째 행
```

	산업군	기업수	투자액
2	의료업	430	4.10
3	물류업	670	2.80
4	교육업	310	1.90

boolean 인덱싱(조건 인덱싱)

```
df
```

	산업군	기업수	투자액
0	제조업	1200	3.2
1	금융업	850	5.5
2	의료업	430	4.1
3	물류업	670	2.8
4	교육업	310	1.9

```
# 행 인덱싱
df[df['기업수'] >= 400]
```

	산업군	기업수	투자액
0	제조업	1200	3.2
1	금융업	850	5.5
2	의료업	430	4.1
3	물류업	670	2.8

```
# 행 인덱싱 후에 결과 데이터프레임으로 열 인덱싱
df[df['기업수'] >= 400]['투자액']
```

	투자액
0	3.2
1	5.5
2	4.1
3	2.8

dtype: float64

```
df[df['기업수'] >= 400][['산업군','투자액']]
```

	산업군	투자액
0	제조업	3.2
1	금융업	5.5
2	의료업	4.1
3	물류업	2.8

```
list_data = [[1200,3.2],
        [850,5.5],
        [430,4.1],
        [670,2.8],
        [310,1.9]]
industry =  ['제조업', '금융업', '의료업', '물류업', '교육업']
df2 = pd.DataFrame(list_data, columns=['도입 기업 수','평균 투자액'],index=industry)
df2
```

	도입 기업 수	평균 투자액
제조업	1200	3.2
금융업	850	5.5
의료업	430	4.1
물류업	670	2.8
교육업	310	1.9

인덱스 변경

- `df.reset_index()`
  - 기존 인덱스를 제거하고 기본 숫자 인덱스 (0, 1, 2, ...) 로 재설정합니다.

- 제거된 인덱스는 새로운 열로 추가됩니다.(기본 동작)

- 속성(옵션)
  - drop : 기존 인덱스를 열로 추가할지 여부 (True면 삭제). 기본값 False
  - name : drop 이 False 일 때, 인덱스 컬럼의 이름
  - inplace : 원본 DataFrame을 직접 수정할지 여부. 기본값 False
  - level : MultiIndex일 경우 제거할 인덱스의 레벨 지정 모든 레벨

```
df2
```

	도입 기업 수	평균 투자액
제조업	1200	3.2
금융업	850	5.5
의료업	430	4.1
물류업	670	2.8
교육업	310	1.9

```
# df.reset_index()
df2.reset_index(drop=False, inplace=False)  #기본값
```

	index	도입 기업 수	평균 투자액
0	제조업	1200	3.2
1	금융업	850	5.5
2	의료업	430	4.1
3	물류업	670	2.8
4	교육업	310	1.9

```
df2  # 원본 변경 없음 : inplace=False
```

	도입 기업 수	평균 투자액
제조업	1200	3.2
금융업	850	5.5
의료업	430	4.1
물류업	670	2.8
교육업	310	1.9

```
df2.reset_index(names='산업군')  # index 컬럼명을 지정함.
```

```
# 원본 df 는 바뀌지 않음 : inplace=False 기본값
# drop=True : 인덱스 컬럼 제거
df2.reset_index(drop=True)
```

	도입	기업 수	평균 투자액	
2	의료업		430	4.1
0		1200	3.2	2.8
1				
4	교육업	850	310	5.5
2		430		4.1
3		670		2.8
4		310		1.9

- 데이터 프레임 복사

```
# inplace=True 를 위한 복사
df_copy=df2.copy()
df_copy
```

	도입	기업 수	평균 투자액
제조업		1200	3.2
금융업		850	5.5
의료업		430	4.1
물류업		670	2.8
교육업		310	1.9

```
# inplace=True 는 원본 데이터프레임을 변경
df_copy.reset_index(names='산업군', inplace=True)
df_copy
```

	산업군	도입	기업 수	평균 투자액
0	제조업		1200	3.2
1	금융업		850	5.5
2	의료업		430	4.1
3	물류업		670	2.8
4	교육업		310	1.9

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```