Pandas

Python

저녁이 있는 프로젝트
오상훈
6 Hours, 1 Month

Introduce

- ❖ 금융 데이터 분석을 위해 2008년 설계
- ❖ 계량 경제학 용어인 panel data와 analysis의 합성어
- ❖ 구조화된 데이터를 빠르고 쉽게 다루는 풍부한 자료 구조와 함수 제공
- ❖ **Pandas 기본 구조는 numpy**

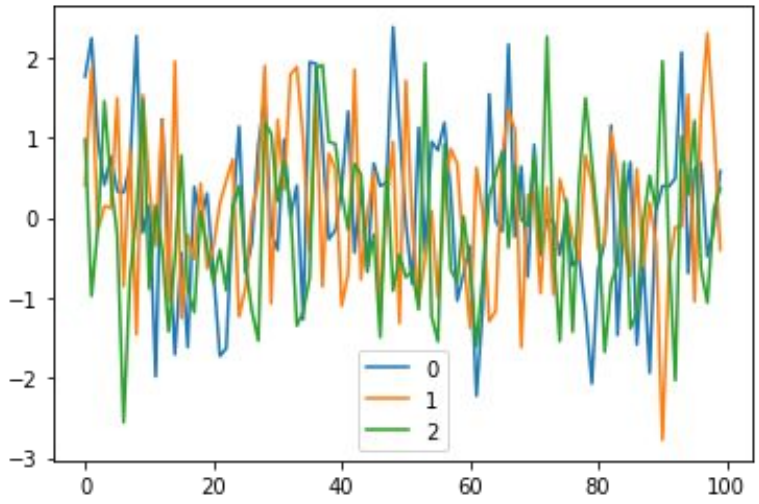
❖ 불러오기

```
import pandas as pd
```

❖ 같이 하기

```
import numpy as np
import matplotlib.pylab as plt
df1 = pd.DataFrame(np.random.randn(100, 3))
df1.tail()
df1.plot()
plt.show()
```

	0	1	2
95	0.608844	-1.045253	1.211145
96	0.689818	1.301846	-0.628088
97	-0.481027	2.303917	-1.060016
98	-0.135950	1.136891	0.097725
99	0.582954	-0.399449	0.370056



Feature

- ❖ 빅데이터 분석에 최적화 된 필수 패키지
- ❖ 데이터는 시계열(series)이나 표(table)의 형태
- ❖ 데이터프레임(dataframe) 클래스 변환 필요.

Series + Series = DataFrame

	apples
0	3
1	2
2	0
3	1

+

	oranges
0	0
1	3
2	7
3	2

=

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Data Structures

❖ pandas.**Series**(**data, index, dtype**, copy): Array, Dict, Scalar value or constant

```
>> obj1 = pd.Series([4, 5, -2], index=["a", "b", "c"])
>> obj1, obj1.values, obj1.index, obj1.dtypes, obj1.shape, type(obj1)
(a    4
 b    5
 c   -2 dtype: int64, array([ 4,  5, -2]), Index(['a', 'b', 'c'], dtype='object'),
 dtype('int64'), (4,), pandas.core.series.Series)
```

❖ pandas.**DataFrame**(**data, index, columns, dtype**, copy) : **Lists, dict, Series, Numpy ndarrays, Another DataFrame**

```
>>> data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
>>> df01 = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
>>> df01.values, df01.index, df01.columns, df01.dtypes
 array([['Tom', 28],
        ['Jack', 34],
        ['Steve', 29],
        ['Ricky', 42]], dtype=object), Index(['Name', 'Age'], dtype='object'),
(Index(['rank1', 'rank2', 'rank3', 'rank4'], dtype='object'),
 Name    object
 Age      int64 dtype: object)
```

❖ pandas.Panel(data, items, major_axis, minor_axis, dtype, copy) : General 3D labeled, size-mutable array.

Data Information

❖ pandas.DataFrame.**info**(self, verbose=None, buf=None, max_cols=None, memory_usage=None, null_counts=None)

```
>>> obj1.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   ID         8 non-null      int64
 1   Name       8 non-null      object
 2   Salary     8 non-null      float64
 3   StartDate  8 non-null      object
 4   Dept       8 non-null      object
dtypes: float64(1), int64(1), object(3)
memory usage: 448.0+ bytes
```

	ID	Name	Salary	StartDate	Dept
count	8.00000	8	8.000000	8	8
unique	NaN	8	NaN	8	4
top	NaN	Nina	NaN	7/30/2013	IT
freq	NaN	1	NaN	1	3
mean	4.50000	NaN	656.881250	NaN	NaN
std	2.44949	NaN	103.059463	NaN	NaN
min	1.00000	NaN	515.200000	NaN	NaN
25%	2.75000	NaN	602.750000	NaN	NaN
50%	4.50000	NaN	628.050000	NaN	NaN
75%	6.25000	NaN	724.125000	NaN	NaN
max	8.00000	NaN	843.250000	NaN	NaN

❖ pandas.DataFrame.**describe**(self: ~FrameOrSeries, percentiles=None, **include=**None, exclude=None) : column 단위 통계 표시

```
>>> obj1.describe(include='all')                # Series 사용 가능
```

Get Data(15.pandas_lecturetest.ipynb)

❖ Need to upload files(csv, json, xlsx)

```
>>> data = pd.read_csv('./input.csv')              → Try add index_col=''
>>> data.to_csv (r'./output.csv', index = False, header=True)
>>> data = pd.read_json('./input.json')            → Try data.to_json(r'./output.json')
>>> data = pd.read_excel('./input.xlsx')
>>> data.to_excel (r'./output.xlsx', index = True, header=True)
>>> with pd.ExcelFile('./input.xlsx') as xls:
>>>     df2 = pd.read_excel(xls, 'Sheet2')
>>> !pip install sqlalchemy                        → Try without !
>>> from sqlalchemy import create_engine
>>> data = pd.read_csv('./input.csv')
>>> engine = create_engine('sqlite:///:memory:') # Create the db engine
>>> data.to_sql('data_table', engine)                  # Store the dataframe as a table
>>> res1 = pd.read_sql_query('SELECT * FROM data_table', engine)
>>> res2 = pd.read_sql_query(
                'SELECT dept,sum(salary) FROM data_table group by dept', engine)
>>> from collections import Counter
>>> with open(r'./input.txt') as f:                # Unstructured Data
>>>     p_dict = Counter(f.read().split());        print(p_dict)
Counter({'and': 7, 'Python': 5, 'a': 4, 'Line': 3,
```

Try - Call from file any type

❖ 출력결과

➢ All Files

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0
...	...	...	...	...	...	...	...	...	...
16995	-124.26	40.58	52.0	2217.0	394.0	907.0	369.0	2.3571	111400.0
16996	-124.27	40.69	36.0	2349.0	528.0	1194.0	465.0	2.5179	79000.0
16997	-124.30	41.84	17.0	2677.0	531.0	1244.0	456.0	3.0313	103600.0
16998	-124.30	41.80	19.0	2672.0	552.0	1298.0	478.0	1.9797	85800.0
16999	-124.35	40.54	52.0	1820.0	300.0	806.0	270.0	3.0147	94600.0

❖ 해보기

➢ Call All File From Colab Notebook Sample Folder By DataFrame

➢ Check Information

- pandas.DataFrame.**info()**
- pandas.DataFrame.**describe**()

Series - index + values

❖ 인덱스(index, key와 유사) + 값(value, NumPy 1차원 배열 비슷) → Dictionary와 비슷

```
>>> data = {"Kim": 35000, "Park": 67000, "Joon": 12000, "Choi": 4000}
>>> obj2 = pd.Series(data)
>>> obj2, obj2.dtypes, obj2.shape, type(obj2)
(Kim     35000
 Park    67000
 Joon    12000
 Choi     4000
 dtype: int64, dtype('int64'), (4,), pandas.core.series.Series
>>> obj2.values, obj2.values.shape, type(obj2.values)
 array([35000, 67000, 12000,  4000]), (4,), numpy.ndarray
>>> obj2.index, obj2.index.shape, obj2..dtypes, type(obj2.index)
Index(['Kim', 'Park', 'Joon', 'Choi'], dtype='object'),
(4,),pandas.core.indexes.base.Index)
>>> obj2 * 2                    # 스칼라 연산
Kim      70000
Park    134000
Joon     24000
Choi      8000
dtype: int64
```

Series - slicing

```
>>> pd_series01 = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e']) → Try without index
>>> pd_series01[:3], pd_series01[['c','d']]        → pd_series01['a'] 가능, But 비권장.
(a   1
b   2
c   3 dtype: int64,
 c   3
d   4 dtype: int64)
>>> obj1 = pd.Series([4, 5, -2, 8], index=["a", "b", "c", "d"])
>>> obj3 = pd.Series([4, 5, -2, 8, 10], index=["a", "b", "c", "d", 'e'])
>>> obj3, obj3.values, obj3.index, obj3.dtypes
>>> obj1 * obj3                              # 인덱싱 끼리 연산 권장
a   16.0
      ...
d   64.0
e    NaN dtype: float64
>>> obj[obj1 > 4]                            # Mask 시 사용 가능
>>> pd_series01[:3] = [9,8,7]                          # update values by Value Sequence
>>> pd_series01[['a','c','d']] = [20,30,40]            # update values by Index Name
>>> del obj3["b"]
>>> pd_series01.drop(['b','d'])                        # 권장
```

DataFrame - Series + Column Name

- 데이터프레임 = Series{인덱스(index) + 값(value)} + Series
- **공통 인덱스** 가지는 열 시리즈(column Series)로 Dictionary Type 묶음.
  - ➢ Column selection, addition, and deletion : **key로만 접근 가능**.

```
>>> data01 = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c'])
                      , 'two' : pd.Series([1, 2, 3, 4], index=['a', 'c', 'd', 'e'])}
>>> pd_df05 = pd.DataFrame(data01)
>>> pd_df05 ['one'], type(pd_df05 ['one'])
>>> pd_df05 ['one', ‘two’]                    # Error, As Dictionary
>>> pd_df05 [['one', ‘two’]]                  # 권장, Try pd_df05[['one']]

>>> pd_sr06=pd.Series([10,20,30],index=['a','e','f'])
>>> pd_df05['three']=pd_sr06   # No apply 'f' row
>>> pd_df05['four']=pd_df05['two']+pd_df05['three']
>>> pd_df05['five']=np.array([10,20,30,40,50])
>>> del pd_df05['one']                        # or pd_df05.pop('two')
```

Try - DataFrame

❖ 출력 결과

	year	name	points	penalty	zeros	debt	net_points	high_points
one	2013	Choi	1.5	0.1	0	NaN	1.4	False
two	2014	Choi	1.7	0.2	1	-1.2	1.5	False
three	2015	Choi	3.6	0.3	2	NaN	3.3	True
four	2016	Kim	2.4	0.4	3	-1.5	2.0	False
five	2017	Park	2.9	0.5	4	-1.7	2.4	True

❖ 해보기

➢ 아래와 같은 데이터로 출력결과와 같은 DataFrame 만들기

- DataFrame엔 DataFrame을 Parameter 사용 불가.

```
data = {
        'name': ["Choi", "Choi", "Choi", "Kim", "Park"],
        'year': [2013, 2014, 2015, 2016, 2017],
        'points': [1.5, 1.7, 3.6, 2.4, 2.9]
        }
```

DataFrame - Slicing

❖ nan(not a number)으로 나타내는 null

➢ Row selection, addition, and deletion : 숫자 접근 권장.

```
>>> data01 = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
    'two' : pd.Series([1, 2, 3, 4], index=['a', 'c', 'd', 'e'])}
>>> pd_df10 = pd.DataFrame(data01)
>>> pd_df10[0:3]
>>> pd_df10[[‘one’,’two’]][1:4]                    # 권장, Not - pd_df10[[‘one’,’two’],[1:4]]
>>> pd_df10.loc['b'], type(pd_df10.loc['b']) # 행렬 접근 쉽게 함.
(one    2.0
 two    NaN
 Name: b, dtype: float64, pandas.core.series.Series)
>>> pd_df10.iloc[1:4]                              # row slicing
>>> df.iloc[[0,1,3], [1,2]]
>>> pd_df11 = pd.DataFrame([[5, 6], [7, 8]], columns=['one', 'two'])
>>> pd_df12 = pd_df10.append(pd_df11)
>>> pd_df12.drop('c', axis=0)
```

Basic function

- Series
  - axes : Returns a list of the row axis labels

ex) pd_series00 = pd.Series(np.random.randn(10))
pd_series00.**axes**, pd_series00.dtype, pd_series00.empty, pd_series00.ndim, pd_series00.size, pd_series00.values
pd_series00.**head**(3), pd_series00.**tail**(3)

- DataFrame
  - **T** : Transposes rows and columns.
  - **dtypes** : Returns the dtypes in this object.
  - shape : Returns a tuple representing the dimensionality of the DataFrame.
  - **astype()** :
  - description()

pd_df12.T, pd_df12.axes, pd_df12.dtypes, pd_df12.empty, pd_df12.ndim, pd_df12.size, pd_df12.shape, pd_df12.values

Data Cleansing

❖ fillna

```
>>> df = pd.DataFrame(np.random.randn(3, 3), index=['a', 'c', 'e'],columns=['one',
'two', 'three'])
>>> df = df.reindex(['a', 'b', 'c'])
>>> df = df.fillna(0)
```

❖ drop

```
>>> df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
'h'],columns=['one', 'two', 'three'])
>>> df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
>>> df.dropna()
```

❖ replace

```
>>> df = pd.DataFrame({'one':[10,20,30,40,50,2000],
'two':[1000,0,30,40,50,60]})
>>> df.replace({1000:10,2000:60})
```

Data Wrangling

```
ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
        'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
        'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
        'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
        'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Year')
grouped.describe()
grouped.get_group(2014)
ipl_data02 = {'Team': [' 'Royals', 'Riders'],
        'Rank': [4,1,2],
        'Year': [2015,2017],
        'Points':[804,690]}
pd.concat([ipl_data, ipl_data02])
```


수고하셨습니다.