[Python] Pandas의 이론과 기초적인 사용법 — 나무늘보의 개발 블로그

노트북:	첫 번째 노트북
만든 날짜:	2020-10-31 오전 10:34
URL:	https://continuous-development.tistory.com/127?category=736681

Python

[Python] Pandas의 이론과 기초적인 사용법

2020. 10. 14. 00:40 수정 삭제 공개

Pandas

컴퓨터 프로그래밍에서 pandas는 데이터 조작 및 분석을 위해 Python 프로그래밍 언어로 작성된 소프트웨어 라이브러리입니다. 특히 숫자 테이블과 시계열을 조작하기 위한 데이터 구조와 연산을 제공합니다.

- 분석할려는 데이터는 대부분 시계열(Series) 이거나 표(table) 형태로 정의해야 한다.
- 1차원의 Series 클래스와 2차원의 DataFrame 클래스를 제공한다.

```
In [3]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

import pandas as pd - 기본 import 하는 방법이다.

#Series 생성

In [10]:
```
s = pd.Series([1,2,3,4],dtype = np.float64)
print(s)
print(s.values)
print(type(s.values))
print(s.index)
print(type(s.index))
```

```
0    1.0
1    2.0
2    3.0
3    4.0
dtype: float64
[1. 2. 3. 4.]
<class 'numpy.ndarray'>
RangeIndex(start=0, stop=4, step=1)
<class 'pandas.core.indexes.range.RangeIndex'>
```

```
value = pd.Series([data])
```

#Series와 numpy array를 비교

In [4]:
```
# Series 와 numpy array 비교
arr = np.array([1,2,3,4,'jslim'],dtype=np.object)
print(arr)
print(arr.dtype)
```

```
[1 2 3 4 'jslim']
object
```

array는 여러 타입의 값들이 들어갈 수 있지만

In [10]:
```
s = pd.Series([1,2,3,4],dtype = np.float64)
print(s)
print(s.values)
print(type(s.values))
print(s.index)
print(type(s.index))
```

```
0    1.0
1    2.0
2    3.0
3    4.0
dtype: float64
[1. 2. 3. 4.]
<class 'numpy.ndarray'>
RangeIndex(start=0, stop=4, step=1)
<class 'pandas.core.indexes.range.RangeIndex'>
```

의 형태고 Series를 만들 수 있다. dtype는 Series의 타입을 지정해준다. Series는 같은 타입의 값들이 들어가야 된다.

In [12]:
```
def serieInfo(s):
    print('value :',s.values)
    print('value type :',type(s.values))
    print('index :',s.index)
    print('index type :',type(s.index))
    print('index + value :',s)
```

간단하게 여러 형태로 값을 보기 위해 간단한 함수를 만들었다.

```
In [19]: # 인덱스의 라벨은 무자열 뿐만 아니라 날짜, 시간, 정수 등 가능
         s = pd.Series([1,2,3,4,5,6],index=['a','b','c','d','e','f'])
         serieInfo(s)
```

```
value : [1 2 3 4 5 6]
value type : <class 'numpy.ndarray'>
index : Index(['a', 'b', 'c', 'd', 'e', 'f'], dtype='object')
index type : <class 'pandas.core.indexes.base.Index'>
index + value : a    1
b    2
c    3
d    4
e    5
f    6
dtype: int64
```

series를 만들고 , **index=[인덱스 값]을** 통해서 해당 series의 인덱스를 부여할 수 있다.

```
In [20]: # 인덱스의 라벨은 무자열 뿐만 아니라 날짜, 시간, 정수 등 가능
         s = pd.Series([1,2,3,4,5,6],index=np.arange(6))
         serieInfo(s)
```

```
value : [1 2 3 4 5 6]
value type : <class 'numpy.ndarray'>
index : Int64Index([0, 1, 2, 3, 4, 5], dtype='int64')
index type : <class 'pandas.core.indexes.numeric.Int64Index'>
index + value : 0    1
1    2
2    3
3    4
4    5
5    6
dtype: int64
```

이런 식으로 arrange를 이용해서 인덱스를 만들 수 있다.

```
In [21]: # 인덱스의 라벨은 무자열 뿐만 아니라 날짜, 시간, 정수 등 가능
         s = pd.Series([1,2,3,4,5,6],index=['서울','대구','부산','울산','인천','경기'])
         serieInfo(s)
```

```
value : [1 2 3 4 5 6]
value type : <class 'numpy.ndarray'>
index : Index(['서울', '대구', '부산', '울산', '인천', '경기'], dtype='object')
index type : <class 'pandas.core.indexes.base.Index'>
index + value : 서울    1
대구    2
부산    3
울산    4
인천    5
경기    6
dtype: int64
```

인덱스를 한글로도 가능하다.

```
In [22]: # 인덱스의 라벨은 무자열 뿐만 아니라 날짜, 시간, 정수 등 가능
         s = pd.Series([1,2,3,4,5,6],
                       dtype=np.int32,
                       index=['서울','대구','부산','울산','인천','경기'])
         serieInfo(s)
```

```
value : [1 2 3 4 5 6]
value type : <class 'numpy.ndarray'>
index : Index(['서울', '대구', '부산', '울산', '인천', '경기'], dtype='object')
index type : <class 'pandas.core.indexes.base.Index'>
index + value : 서울    1
대구    2
부산    3
울산    4
인천    5
경기    6
dtype: int32
```

dtype을 바꿔 줄 수도 있다.

```
In [24]: s.index.name = '지역별'
         print(s)
```

```
지역별
서울    1
대구    2
부산    3
울산    4
인천    5
경기    6
dtype: int32
```

또한 이렇게 **series.index.name = 이름** series. s의 index 자체의 이름을 지어줄 수 있다.

```
In [25]: s / 100
```

```
Out[25]: 지역별
서울    0.01
대구    0.02
부산    0.03
울산    0.04
인천    0.05
경기    0.06
dtype: float64
```

series를 연산자를 통해 연산할 수 있다.

#series indexing

- series indexing

```
In [27]: s['서울']
```

```
Out[27]: 1
```

```
In [35]: s[['서울','대구']] # 2개의 값을 indexing 할 때
```

```
Out[35]: 지역별
서울    1
대구    2
dtype: int32
```

```
In [34]: s[0]
```

```
Out[34]: 1
```

```
In [36]: s[[0,3]]# 2개의 값을 indexing 할 때
```

```
Out[36]: 지역별
서울    1
울산    4
dtype: int32
```

series slicing

- series slicing

In [29]:
```
s[0:2]
```

Out[29]:
```
지역별
서울    1
대구    2
dtype: int32
```

In [33]:
```
s['서울':'부산']
```

Out[33]:
```
지역별
서울    1
대구    2
부산    3
dtype: int32
```

series in

In [37]:
```
'서울' in s
```

Out[37]:
```
True
```

In [38]:
```
'강원' in s
```

Out[38]:
```
False
```

dictonary를 통한 Series

In [39]:
```
for key, value in s.items():
    print('key :{}, value={} '.format(key,value))
```

```
key :서울, value=1
key :대구, value=2
key :부산, value=3
key :울산, value=4
key :인천, value=5
key :경기, value=6
```

In [42]:
```
s2 = pd.Series({'c':1, 'b':5,'a':-8,'k':10})
serieInfo(s2)
```

```
value : [ 1  5 -8 10]
value type : <class 'numpy.ndarray'>
index : Index(['c', 'b', 'a', 'k'], dtype='object')
index type : <class 'pandas.core.indexes.base.Index'>
index + value : c     1
b     5
a    -8
k    10
dtype: int64
```

Fancy indexing , boolean indexing

In [50]:
```
# Fancy Indexing & Boolean Indexing
print('fancy[0,2] indexing: {}'.format(s2[[0,2]]))

# boolean indexing 2의 배수인 것
print('fancy[0,2] indexing: {}'.format(s2[s2 % 2 ==0 ]))
```

```
fancy[0,2] indexing: c    1.0
a   -8.0
dtype: float64
fancy[0,2] indexing: a    -8.0
k    10.0
dtype: float64
```

In [54]:
```
# 인덱스의 라벨은 무자열 뿐만 아니라 날짜, 시간, 정수 등 가능
s = pd.Series({'서울':12324,'부산':345346,'인천':123124,'경기':234234},
              dtype=np.int32,
              index=['광주','서울','부산','인천'])
serieInfo(s)
```

```
value : [    nan  12324. 345346. 123124.]
value type : <class 'numpy.ndarray'>
index : Index(['광주', '서울', '부산', '인천'], dtype='object')
index type : <class 'pandas.core.indexes.base.Index'>
index + value : 광주         NaN
서울     12324.0
부산    345346.0
인천    123124.0
dtype: float64
```

아래에 있는 index가 먼저이다. 이때 이름이 안 맞으면 아래 index에 만 있는 값은 Nan이 되어 나온다.

예제)

In [56]:
```
# A 공장의 2019-01-01 부터 10일간의 생산량을 Series 저장
# 생산량은 평균이 50 이고 편차가 5인 정규 분포 생성(정수)

# B공장의 2019.01-01  부터 10일간의 생산량을 Series 저장
# 생산량은 평균이 70 이고 편차가 8인 정규분포 생성(정수)

# 날짜별로 모든 공장의 생산량 합계를 구한다면?
```

In [60]:
```
import pandas as pd
import numpy as np
from datetime import date,datetime, timedelta
from dateutil.parser import parse
```

In [65]:
```
start_day = datetime(2019,1,1)
print(start_day)

facA = pd.Series([int(x) for x in np.random.normal(50,5,(10,))]) # 이런식으로 기존에 array 형태였던 것을
                                    #  리스트 형태로 만들어 하나씩 뽑아낸다.
print(facA)
```

```
2019-01-01 00:00:00
0    56
1    61
2    49
3    49
4    54
5    53
6    52
7    43
8    49
9    38
dtype: int64
```

In [75]:
```
start_day = datetime(2019,1,1)
print(start_day)

facA = pd.Series([int(x) for x in np.random.normal(50,5,(10,))],
                 index= [start_day+timedelta(days=x) for x in range(10)] ) # 이런식으로 기존에 array 형태였던 것을
                                    #  리스트 형태로 만들어 하나씩 뽑아낸다..
print(facA)

facB = pd.Series([int(x) for x in np.random.normal(70,8,(10,))],
                 index= [start_day+timedelta(days=x) for x in range(10)] ) # 이런식으로 기존에 array 형태였던 것을
                                    #  리스트 형태로 만들어 하나씩 뽑아낸다.
print('*'*50)
print(facB)
```

```
2019-01-01 00:00:00
2019-01-01    54
2019-01-02    54
2019-01-03    53
2019-01-04    52
2019-01-05    45
2019-01-06    54
2019-01-07    54
2019-01-08    56
2019-01-09    47
2019-01-10    53
dtype: int64
**************************************************
2019-01-01    68
2019-01-02    71
2019-01-03    78
2019-01-04    66
2019-01-05    77
2019-01-06    72
2019-01-07    72
2019-01-08    67
2019-01-09    67
2019-01-10    79
dtype: int64
```

In [76]:
```
print(facA + facB)
```

```
2019-01-01    122
2019-01-02    125
2019-01-03    131
2019-01-04    118
2019-01-05    122
2019-01-06    126
2019-01-07    126
2019-01-08    123
2019-01-09    114
2019-01-10    132
dtype: int64
```

이런 식으로 계산할 수도 있다.

In [84]:
```
start_day = datetime(2019,1,1)
print(start_day)

facA = pd.Series([int(x) for x in np.random.normal(50,5,(10,))],
                 index= [start_day+timedelta(days=x) for x in range(10)] ) # 이런식으로 기존에 array 형태였던 것을
                                    #  리스트 형태로 만들어 하나씩 뽑아낸다..
print(facA)
start_day = datetime(2019,1,1)

facB = pd.Series([int(x) for x in np.random.normal(70,8,(10,))],
                 index= [start_day+(timedelta(days=x*2+1)) for x in range(10)] ) # 이런식으로 기존에 array 형태였던 것을
                                    #  리스트 형태로 만들어 하나씩 뽑아낸다.
print('*'*50)
print(facB)
```

```
2019-01-01 00:00:00
2019-01-01    47
2019-01-02    45
2019-01-03    51
2019-01-04    51
2019-01-05    48
2019-01-06    50
2019-01-07    53
2019-01-08    56
2019-01-09    45
2019-01-10    48
dtype: int64
**************************************************
2019-01-02    68
2019-01-04    76
2019-01-06    63
2019-01-08    77
2019-01-10    73
2019-01-12    66
2019-01-14    64
2019-01-16    69
2019-01-18    68
2019-01-20    62
dtype: int64
```

In [85]:
```
print(facA + facB)
```

```
2019-01-01      NaN
2019-01-02    113.0
2019-01-03      NaN
2019-01-04    127.0
2019-01-05      NaN
2019-01-06    113.0
2019-01-07      NaN
2019-01-08    133.0
2019-01-09      NaN
2019-01-10    121.0
2019-01-12      NaN
2019-01-14      NaN
2019-01-16      NaN
2019-01-18      NaN
2019-01-20      NaN
dtype: float64
```


'**Python**' 카테고리의 다른 글□

[Python] Pandas 사용법 - DataFrame 생성, 추가 , 수정, 삭제, indexing□

[Python] Pandas 사용법 - series 에 대한 추가 , 수정, 삭제, 연산, 결측치□

[Python] Pandas의 이론과 기초적인 사용법□

[Python] Numpy를 통한 난수생성, 카운팅, 통계함수 사용법□

[Python] Numpy를 통한 정렬하기□

[Python] Numpy 를 통한 최대값, 최소값 , 통계함수 사용하기□

pandas series　pandas 기초　pandas 기초 사용법　pandas 사용법

나무늘보스

혼자 끄적끄적하는 블로그 입니다.