팬더스를 활용한 데이터 분석

-3장-

2022 동계방학 학부연구생 이소희

넘파이

파이썬으로 수치 해석이나 통계 관련 작업을 구현할 때 가장 기본이 되는 모듈

- 고성능 다차원 배열 객체인 ndarray와 이를 다루는 여러 함수를 제공
- 빠르게 수치 해석과 통계 작업 처리 가능
- 인공지능 관련 개발을 할 때 반드시 필요

배열 생성

array() 함수 이용

```
>>> import numpy as np
>>> A = np.array([[1, 2], [3, 4]])
>>> A
array([[1, 2],
       [3, 4]])
```

배열 정보 보기

- ndim : 배열의 차원을 나타냄
- shape : 배열의 각 차원의 크기를 튜플로 나타냄
- dtype : 배열 원소의 자료형을 나타냄
- max() : 최댓값
- mean() : 평균값
- min() : 최솟값
- sum() : 합계

```
>>> type(A)
<class 'numpy.ndarray'>

>>> A.ndim # 배열의 차원
2

>>> A.shape # 배열 크기
(2, 2)

>>> A.dtype # 원소 자료형
dtype('int32')
```

```
>>> print(A.max(), A.mean(), A.min(), A.sum())
4 2.5 1 10
```

배열 접근

- 대괄호를 사용한다.
- 인덱싱과 슬라이싱을 할 수 있다.
- 조건에 맞는 원소 출력이 가능하다.

배열 이름[행 인덱스][열 인덱스] == 배열 이름[행 인덱스, 열 인덱스]

```
>>> A[A>1]
array([2, 3, 4])
```

```
>>> print(A[0, 0], A[0, 1]); print(A[1, 0], A[1, 1])
1 2
3 4
```

```
>>> print(A[0][0], A[0][1]); print(A[1][0], A[1][1])
1 2
3 4
```

배열 형태 변경

- transpose() : 배열의 전치
- flatten() : 배열 평탄화

```
>>> A
array([[1, 2],
       [3, 4]])

>>> A.T # A.transpose()와 같다.
array([[1, 3],
       [2, 4]])
```

```
>>> A
array([[1, 2],
       [3, 4]])

>>> A.flatten()
array([1, 2, 3, 4])
```

배열 연산

같은 크기의 행렬끼리는 사칙 연산이 가능하다.

```
>>> A
array([[1, 2],
       [3, 4]])

# np.add(A, A)와 같다.
>>> A + A
array([[2, 4],
       [6, 8]])

# np.subtract(A, A)와 같다.
>>> A - A
array([[0, 0],
       [0, 0]])

# np.multiply(A, A)와 같다.
>>> A * A
array([[ 1,  4],
       [ 9, 16]])

# np.divide(A, A)와 같다.
>>> A / A
array([[1., 1.],
       [1., 1.]])
```

브로드캐스팅

행렬 크기가 달라도 연산할 수 있도록 작은 행렬을 확장해준다.

```
>>> A
array([[1, 2],
       [3, 4]])

>>> B = np.array([10, 100])

>>> A * B
array([[ 10, 200],
       [ 30, 400]])
```

그림_ 배열의 브로드캐스팅

내적

dot() 이용

- 스칼라값이다.
- 인공지능 분야에서 신경망을 통해 전달되는 신호값을 계산할 때 주로 쓰인다.
- 1차원 벡터 두 개의 내적의 경우 앞에 오는 것을 행, 뒤에 오는 것을 열 벡터로 간주한다.

```
>>> B.dot(B) # np.dot(B, B)와 같다.
10100
```

$$B \times B = (b_1 \ b_2) \times \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} 10 & 100 \end{pmatrix} \times \begin{pmatrix} 10 \\ 100 \end{pmatrix}$$

$$(10 \times 10 + 100 \times 100) = 10100$$

팬더스

- 금융 데이터 분석을 목적으로 개발
- 구조화된 데이터를 쉽고 빠르게 가공할 수 있는 자료형과 함수 제공
- 대부분의 함수가 넘파이와 유사
- 데이터 과학용 기본 라이브러리로 널리 활용

시리즈

- 인덱스 처리된 1차원 벡터 형태의 자료형
- 시계열 데이터를 다루는 데 적합

- 리스트, 튜플 등의 시퀀스를 생성자의 인수로 받을 수 있다.
- Series() 생성자로 생성한다.
- 인덱스를 지정해주지 않으면 0부터 인덱스가 자동으로 생성된다.
- 시리즈명을 지정해줄 수 있다.

```
>>> import pandas as pd
>>> s = pd.Series([0.0, 3.6, 2.0, 5.8, 4.2, 8.0]) # 리스트로 시리즈 생성
>>> s
0    0.0
1    3.6
2    2.0
3    5.8
4    4.2
5    8.0
dtype: float64
```

```
>>> s.name = 'MY_SERIES' # 시리즈명 설정
>>> s
MY_IDX
0.0    0.0
1.2    3.6
1.8    2.0
3.0    5.8
3.6    4.2
4.8    8.0
Name: MY_SERIES, dtype: float64
```

시리즈의 인덱스

인덱스 번호와 인덱스명을 설정할 수 있다.

```
>>> s.index = pd.Index([0.0, 1.2, 1.8, 3.0, 3.6, 4.8]) # 인덱스 변경
>>> s.index.name = 'MY_IDX' # 인덱스명 설정
>>> s
MY_IDX
0.0    0.0
1.2    3.6
1.8    2.0
3.0    5.8
3.6    4.2
4.8    8.0
dtype: float64
```

데이터 추가

1. 인덱스 레이블과 값을 지정해줘서 추가할 수 있다.
2. 시리즈를 인자로 받는 append() 메서드를 이용한다. 시리즈명과 인덱스명은 다시 설정해야 한다.

```
>>> s[5.9] = 5.5
>>> s
MY_IDX
0.0    0.0
1.2    3.6
1.8    2.0
3.0    5.8
3.6    4.2
4.8    8.0
5.9    5.5
Name: MY_SERIES, dtype: float64
```

1

```
>>> ser = pd.Series([6.7, 4.2], index=[6.8, 8.0]) # ser 시리즈를 생성
>>> s = s.append(ser) # 기존 s 시리즈에 신규 ser 시리즈를 추가
>>> s
0.0    0.0
1.2    3.6
1.8    2.0
3.0    5.8
3.6    4.2
4.8    8.0
5.9    5.5
6.8    6.7
8.0    4.2
dtype: float64
```

2

데이터 인덱싱

- index : 인덱스값 구하기
- loc : 인덱스값을 이용해 데이터값 구하기
- values : 인덱스 순서에 해당하는 데이터값 구하기
- iloc : 인덱스 순서에 해당하는 데이터값 구하기

values와 iloc의 차이점

- values : 결과값이 복수 개일 때 배열로 반환
- iloc : 결과값이 복수 개일 때 시리즈로 반환

```
>>> s.index[-1]
8.0
```

```
>>> s.loc[8.0] # 로케이션 인덱서
4.2
```

```
>>> s.values[-1]
4.2
```

```
>>> s.iloc[-1] # 인티저 로케이션 인덱서
4.2
```

```
>>> s.values[:]
array([0. , 3.6, 2. , 5.8, 4.2, 8. , 5.5, 6.7, 4.2])
```

```
>>> s.iloc[:]
0.0    0.0
1.2    3.6
1.8    2.0
3.0    5.8
3.6    4.2
4.8    8.0
5.9    5.5
6.8    6.7
8.0    4.2
dtype: float64
```

데이터 삭제

- drop() 이용
- 인자로 삭제하고자하는 원소의 인덱스값 이용
- 함수 적용 후 대입해줘야 변화값 적용됨

```
>>> s.drop(8.0) # s.drop(s.index[-1])과 같다.
0.0    0.0
1.2    3.6
1.8    2.0
3.0    5.8
3.6    4.2
4.8    8.0
5.9    5.5
6.8    6.7
dtype: float64
```

시리즈 정보 보기

- describe() 이용
- 원소 개수, 평균, 표준편차, 최솟값, 제1 사분위수, 제2 사분위수, 제3 사분위수, 최댓값 확인 가능

```
>>> s.describe()
count    9.000000  # 원소 개수
mean     4.444444  # 평균
std      2.430078  # 표준편차
min      0.000000  # 최솟값
25%      3.600000  # 제1 사분위수
50%      4.200000  # 제2 사분위수
75%      5.800000  # 제3 사분위수
max      8.000000  # 최댓값
dtype: float64
```

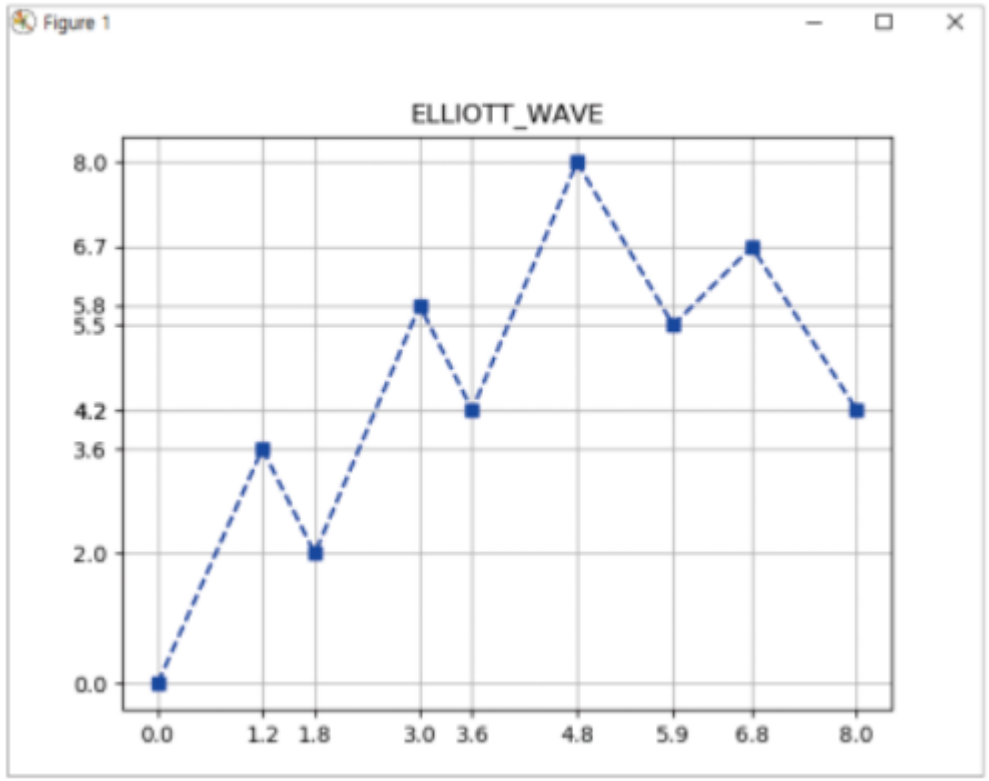
시리즈 시각화

plot() 이용

```
import pandas as pd
s = pd.Series([0.0, 3.6, 2.0, 5.8, 4.2, 8.0, 5.5, 6.7, 4.2]) # 시리즈 생성
s.index = pd.Index([0.0, 1.2, 1.8, 3.0, 3.6, 4.8, 5.9, 6.8, 8.0]) # 시리즈 인덱스
변경
s.index.name = 'MY_IDX' # 시리즈 인덱스명 설정
s.name = 'MY_SERIES' # 시리즈 이름 설정

import matplotlib.pyplot as plt
plt.title("ELLIOTT_WAVE")
plt.plot(s, 'bs--') # 시리즈를 bs--(푸른 사각형과 점선) 형태로 출력
plt.xticks(s.index) # x축의 눈금값을 s 시리즈의 인덱스값으로 설정
plt.yticks(s.values) # y축의 눈금값을 s 시리즈의 데이터값으로 설정
plt.grid(True)
plt.show()
```

그림_ 시리즈로 표시한 엘리어트 파동[3]



데이터프레임

- 시리즈를 모아 표 형태로 만들어준 것
- 여러 변수에 대한 관측값을 함께 기록할 때 사용

- 생성자에 각각의 데이터를 딕셔너리 형식으로 넣어준다.
- 인덱스값을 별도로 지정해줄 수 있다.
- 별도의 인덱스를 지정하지 않으면 0부터 자동으로 매겨진다.

```
>>> import pandas as pd
>>> df = pd.DataFrame({'KOSPI': [1915, 1961, 2026, 2467, 2041],
...     'KOSDAQ': [542, 682, 631, 798, 675]})
>>> df
   KOSPI  KOSDAQ
0   1915     542
1   1961     682
2   2026     631
3   2467     798
4   2041     675
```

```
>>> df = pd.DataFrame({'KOSPI': [1915, 1961, 2026, 2467, 2041],
...     'KOSDAQ': [542, 682, 631, 798, 675]},
...     index=[2014, 2015, 2016, 2017, 2018])
>>> df
      KOSPI  KOSDAQ
2014   1915     542
2015   1961     682
2016   2026     631
2017   2467     798
2018   2041     675
```

데이터프레임 : 시리즈를 이용한 생성

- 데이터프레임화하고자 하는 시리즈를 딕셔너리 형태로 구성하여 생성자에 넘겨준다.
- 각 시리즈는 데이터프레임의 칼럼으로 합쳐진다.

```
>>> kospi = pd.Series([1915, 1961, 2026, 2467, 2041],
...     index=[2014, 2015, 2016, 2017, 2018], name='KOSPI')
>>> kospi
2014    1915
2015    1961
2016    2026
2017    2467
2018    2041
Name: KOSPI, dtype: int64
```

```
>>> kosdaq = pd.Series([542, 682, 631, 798, 675],
...     index=[2014, 2015, 2016, 2017, 2018], name='KOSDAQ')
>>> kosdaq
2014    542
2015    682
2016    631
2017    798
2018    675
Name: KOSDAQ, dtype: int64
```

```
>>> df = pd.DataFrame({kospi.name: kospi, kosdaq.name: kosdaq})
>>> df
      KOSPI  KOSDAQ
2014   1915     542
2015   1961     682
2016   2026     631
2017   2467     798
2018   2041     675
```

데이터프레임 : 리스트를 이용한 생성

리스트를 이용해 한 행씩 추가하여 데이터프레임을 생성할 수 있다.

1. 데이터프레임의 행에 해당하는 리스트를 각각 생성한 뒤, 이를 rows 리스트에 추가한다.
2. 데이터프레임의 생성자에 rows 리스트를 넘겨준다.

```
>>> columns = ['KOSPI', 'KOSDAQ']
>>> index = [2014, 2015, 2016, 2017, 2018]
>>> rows = []
>>> rows.append([1915, 542]) # ①
>>> rows.append([1961, 682])
>>> rows.append([2026, 631])
>>> rows.append([2467, 798])
>>> rows.append([2041, 675])
>>> df = pd.DataFrame(rows, columns=columns, index=index) # ②
>>> df
      KOSPI  KOSDAQ
2014   1915     542
2015   1961     682
2016   2026     631
2017   2467     798
2018   2041     675
```

데이터프레임 정보 보기

- describe() : 원소의 개수, 평균, 표준편차, 최솟값, 사분위수, 최댓값
- info() : 인덱스 정보, 칼럼 정보, 자료형, 메모리 사용량

```
>>> df.describe()
             KOSPI      KOSDAQ
count     5.000000    5.000000  # 원소의 개수
mean   2082.000000  665.600000  # 평균
std     221.117616   92.683871  # 표준편차
min    1915.000000  542.000000  # 최솟값
25%    1961.000000  631.000000  # 제1 사분위수
50%    2026.000000  675.000000  # 제2 사분위수
75%    2041.000000  682.000000  # 제3 사분위수
max    2467.000000  798.000000  # 최댓값
```

```
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5 entries, 2014 to 2018  # 인덱스 정보
Data columns (total 2 columns):      # 전체 칼럼 정보
KOSPI     5 non-null int64           # 첫 번째 칼럼 정보
KOSDAQ    5 non-null int64           # 두 번째 칼럼 정보
dtypes: int64(2)                     # 자료형
memory usage: 120.0 bytes            # 메모리 사용량
```

데이터프레임 순회 처리

1. 인덱스 이용
2. itertuples() 이용 : 각 행을 이름있는 튜플 형태로 반환
3. iterrows() : 각 행을 인덱스와 시리즈 조합으로 반환

itertuples()의 장점

순회처리가 편하다.
itertuples()가 iterrows()보다 빠르다.

```
>>> for i in df.index:
...     print(i, df['KOSPI'][i], df['KOSDAQ'][i])

2014 1915 542
2015 1961 682
2016 2026 631
2017 2467 798
2018 2041 675
```

```
>>> for row in df.itertuples(name='KRX'):
...     print(row)

KRX(Index=2014, KOSPI=1915, KOSDAQ=542)
KRX(Index=2015, KOSPI=1961, KOSDAQ=682)
KRX(Index=2016, KOSPI=2026, KOSDAQ=631)
KRX(Index=2017, KOSPI=2467, KOSDAQ=798)
KRX(Index=2018, KOSPI=2041, KOSDAQ=675)
```

```
>>> for idx, row in df.iterrows():
...     print(idx, row[0], row[1])

2014 1915 542
2015 1961 682
2016 2026 631
2017 2467 798
2018 2041 675
```

```
>>> for row in df.itertuples():
...     print(row[0], row[1], row[2])

2014 1915 542
2015 1961 682
2016 2026 631
2017 2467 798
2018 2041 675
```

데이터프레임 정보 보기

- head(n) : 데이터프레임의 상위 n행을 출력한다. 인수 생략 시 기본값인 5로 처리된다.
- tail(n) : 데이터프레임의 하위 n행을 출력한다. 인수 생략 시 기본값인 5로 처리된다.
- drop() : 삭제하고자 하는 데이터를 삭제한다. 열 이름을 인자로 주어 열을 삭제할 수도 있다.
- shift(n) : 데이터를 이동시킬 때 사용한다. 전체 데이터가 n행씩 뒤로 이동한다.
- cumsum() : 누적합을 구한다.
- rolling() : 시리즈에서 윈도우 크기에 해당하는 개수만큼 데이터를 추출하여 집계 함수에 해당하는 연산을 실시한다.
- min() : 최솟값을 구한다.
- fillna() : 데이터프레임의 NaN을 없앤다.
- plot() : 데이터를 그래프로 출력한다.
- hist() : 히스토그램을 출력한다.

회귀 분석

데이터의 상관관계를 분석하는 데 쓰이는 통계 방법

회귀 모형을 설정한 후 실제로 관측된 표본을 대상으로 회귀 모형의 계수를 측정