PANDAS 라이브러리

BIND SOFT ㈜바인드소프트

PANDAS

- **관계형** 또는 **레이블이 된** 데이터로 쉽고 직관적 으로 작업할 수 있도록 설계되었고, 빠르고, 유연한 데이터 구조를 제공하는 Python 패키지
- 어떤 언어로도 사용할 수 있는 가장 **강력하고 유연한 오픈 소스 데이터 분석 / 조직 도구**
- Pandas가 다루기 적합한 종류의 데이터
  - SQL 테이블 또는 Excel 스프레드 시트에서와 같은 열과 행으로 이루어진 테이블 형식 데이터
  - 정렬거나 정렬되지 않은 시계열 데이터
  - 다른 형태의 관찰 / 통계 데이터 세트

시리즈(Series)

- Pandas의 Series는 1차원 배열로서의 특징
  - 데이터를 담는 **차원 배열 구조**
  - **인덱스(index)를 사용 가능**
  - **데이터 타입**을 가짐
- 다음과 같이 넘파이 배열로 생성할 수 있고, 데이터 타입을 지정하거나 리스트로 만들 수 있음

```
import numpy as np
import pandas as pd

arr = np.arange(100, 105)
print(arr)
s1 = pd.Series(arr)
print(s1)
s2 = pd.Series(arr, dtype='int32')
print(s2)
s3 = pd.Series(['부장', '차장', '대리', '사원', '인턴'])
print(s3)
```

인덱싱

- 리스트 처럼 인덱싱을 통해 원하는 자료 출력
- 번호를 리스트 형태로 넣어서 출력(fancy indexing)
- 넘파이 어레이로 인덱스를 출력
- 조건문을 주면 boolen 으로 결과 값을 출력

```
import pandas as pd
import numpy as np

s = pd.Series(['부장', '차장', '대리', '사원', '인턴'])
print(s[0])
print(s[[1, 2, 3]]) # fancy indexing
print(s[np.arange(1, 4, 2)])

np.random.seed(0)
s = pd.Series(np.random.randint(10000, 20000, size=(10,)))
print( s > 15000)
```

사용자 정의 인덱스

- 사용자 정의의 index 부여시 **변경된 index**로 조회 가능
- 먼저, Series를 생성 후 index 속성 값에 새로운 index를 할당하여 인덱스를 지정 가능

```
import numpy as np
import pandas as pd

s = pd.Series(['마케팅', '경영', '개발', '기획', '인사'],
index=['a', 'b', 'c', 'd', 'e'])
print(s['c'])
print(s[['a', 'd']])

s2 = pd.Series(['마케팅', '경영', '개발', '기획', '인사'])
print(s2)
s2.index = list('abcde')
print(s2)
```

속성(attribute)

- values() 는 Series 데이터 값(value)만 **numpy array** 형식으로 가져 옴
- ndim() 은 차원을 출력
- Shape 는 데이터의 모양을 튜플로 출력
- NaN 은 결측치 데이터를 의미

```
import numpy as np
import pandas as pd

s = pd.Series(['마케팅', '경영', '개발', '기획', '인사'], index=['a', 'b', 'c',
'd', 'e'])
print(s.values)
print(s.ndim)
print(s.shape)

s2 = pd.Series(['선화', '강호', np.nan, '소정', '우영'])
print(s2)
print(s2.isnull())
print(s2.isna())
print(s2.notnull())
print(s2[s.notnull()])
```

연습문제

- 50개의 랜덤 값(100~200) 으로 pd 데이터를 만든 후에 160 이하인 값 만 출력하시오.

```
import numpy as np
import pandas as pd

np.random.seed(20)
sample2 = pd.Series(np.random.randint(100, 200, size=(50,)))
print(sample2[sample2<160])
```

- 다음 데이터를 가지고 결측치 데이터와 결측치가 아닌 데이터를 분리 하시오.
  ['IT서비스', np.nan, '반도체', np.nan, '바이오', '자율주행']

```
import numpy as np
import pandas as pd

sample = pd.Series(['IT서비스', np.nan, '반도체', np.nan,
'바이오', '자율주행'])

print(sample[sample.isna()])
print(sample[sample.notnull()])
```

데이터 프레임

- 2차원 데이터 구조 (Excel 데이터 시트)
- 행(row), 열(column)으로 구성
- 열(column)은 각각의 데이터 타입 (dtype)을 가짐

- 2차원 리스트로 생성
- 컬럼명을 지정 가능
- 딕션너리 데이터로 생성

```
import pandas as pd

df = pd.DataFrame([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
print(df)

df2 = pd.DataFrame([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]], columns=['가', '나
', '다'])

print(df2)

data = {
    'name': ['Kim', 'Lee', 'Park'],
    'age': [24, 27, 34],
    'children': [2, 1, 3]
}
df3 = pd.DataFrame(data)
print(df3)
```

파일 입출력 – 엑셀

- 데이터 분석에서 일반적으로 사용되는 파일 형식인 엑셀(Excel)과 CSV (Comma Separated Value)을 로드하고 데이터프레임(DataFrame)을 엑셀(Excel)이나 CSV형식으로 저장
- Path 길이가 길어서 안 될 때는 longpathsenabled 를 레지스트리 편집기에서 체크
- Excel 데이터를 바로 읽음(read_excel()), sheet_name을 지정하면 해당 sheet를 가져옴.

[참고] pd.read_excel()로 엑셀 데이터 로드시 에러 발생한다면 engine='openpyxl'을 추가 (Pip 로 설치 해야됨)

```
import numpy as np
import pandas as pd

excel = pd.read_excel('data/seoul_transportation.xlsx',
sheet_name='철도', engine='openpyxl')

print(excel.head())

excel.to_excel('data/sample1.xlsx', index=False, sheet_name='샘
플')
```

파일 입출력 - CSV (Comma Separated Values)

- 한 줄이 한 개의 행에 해당하며, 열 사이에는 **쉼표(,)를 넣어 구분**합니다. - Excel보다는 훨씬 가볍고 **차지하는 용량이 적기 때문에 대부분의 파일데이터는 csv 형태**로 제공
- 저장하는 방법은 excel과 유사
- csv파일 형식에는 sheet_name 옵션은 없음

```
import pandas as pd

df = pd.read_csv('data/seoul_population.csv')

print(df[['연도', '자치구']])
df[['연도', '자치구']].to_csv('data/sample.csv',
index=False)
```

조회, 정렬, 조건 필터

- Head(), tail() 로 데이터조회
- Info()로 컬럼별 정보 조회
- **value_counts()** 값의 분포 확인
- **타입 변환 (astype)**
- **sort_values: 값에 대한 정렬**
- **loc - 조건 필터**
  - boolean index을 만들어 조건에 맞는 데이터만 추출
- 다중 조건은 먼저 condition을 정의하고 **&** 와 **|** 연산자로 **복합 조건을 생성**

```
import numpy as np
import pandas as pd
import seaborn as sns

df = sns.load_dataset("titanic")
print(df.head())
print(df.info())

print(df['who'].value_counts())
# 타입 변환
df['pclass'].astype('int32').head()
df['pclass'].astype('float32').head()
# 정렬
df.sort_values(by='age').head()
df.sort_values(by='age', ascending=False).head()
# 조건
cond = (df['age'] >= 70)
df.loc[cond]
print(df.loc[cond][['sex', 'who', 'alive']])
cond2 = (df['who'] == 'woman')
cond1 = (df['fare'] > 30)
print(df.loc[cond1 & cond2, ['sex', 'who',
'alive']])
```

통계

- 각 함수에 의미에 맞는 결과가 출력

```
import numpy as np
import pandas as pd
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
print(df.head())

print(df.describe())
print(df.count())
print(df['age'].count())
condition = (df['adult_male'] == True)
print(df.loc[condition, 'age'].mean())
print(df.loc[:, ['age', 'fare']].sum())
print(df['fare'].var())
print(df['fare'].std())
```