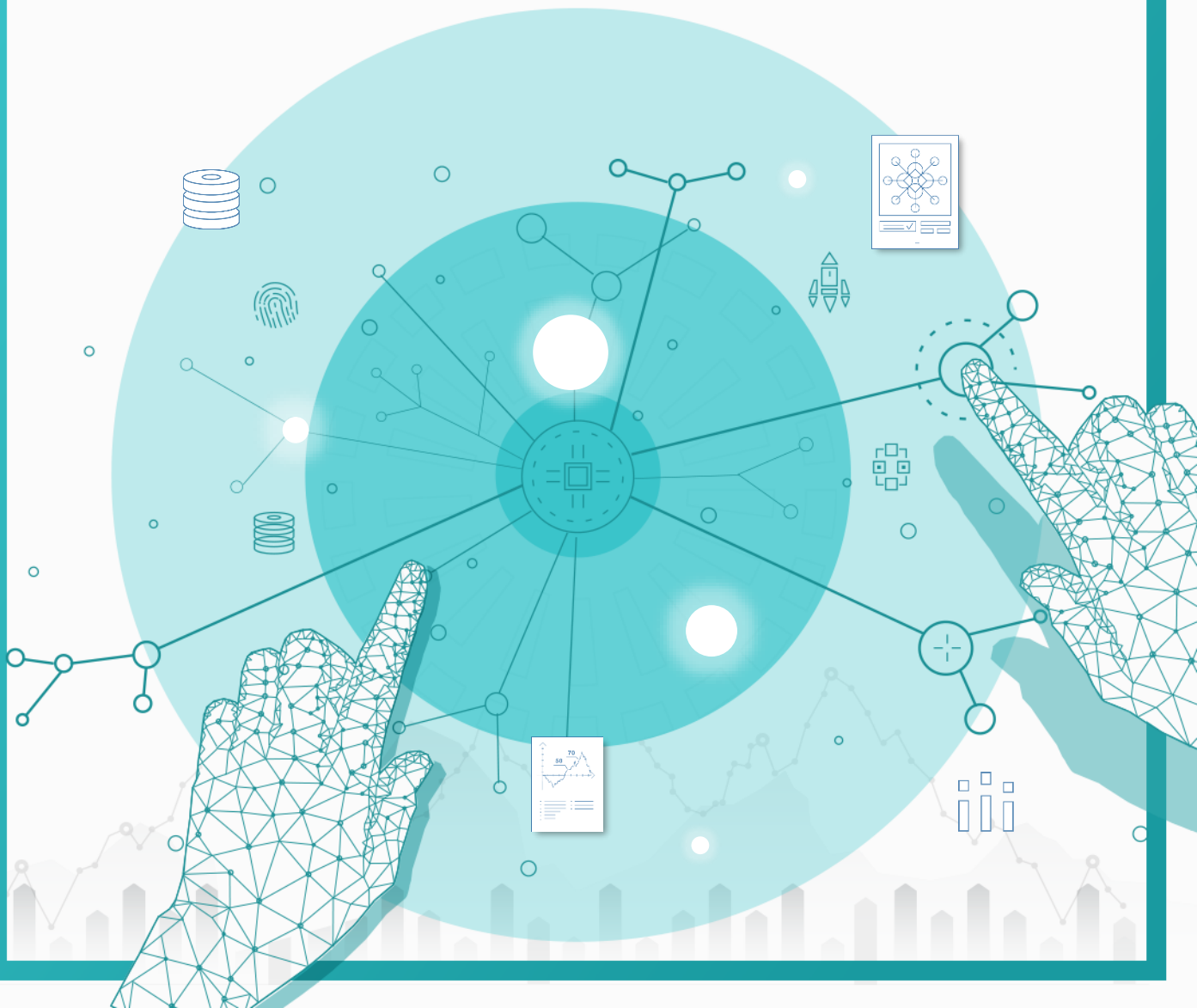
한국기술교육대학교
온라인평생교육원

파이썬 라이브러리로 하는 데이터 분석과 시각화

데이터 분석을 위한 외부 모듈



데이터 분석을 위한 외부 모듈

학습 목표

1. Numpy 모듈을 활용해, 직접 코딩할 수 있다.
2. Pandas 모듈을 활용해, 직접 코딩할 수 있다.
3. Numpy, Pandas 모듈을 활용하여 데이터를 분석할 수 있다.

학습 내용

1. Numpy 모듈 활용
2. Pandas 모듈 활용
3. 모듈을 활용한 데이터 분석 실습

1. Numpy 모듈 활용

1) Numpy 모듈의 개념

(1) Numpy의 정의

- 대규모, 다차원 배열을 쉽게 처리 할 수 있도록 도와주는 파이썬의 외부 모듈
- 기본적으로 array라는 자료형을 사용함
- 행렬의 개념과 비슷함

(2) 설치 방법

- Anaconda 설치 시 함께 설치됨
- pip install numpy로도 설치 가능함

(3) 호출 방법

- import numpy as np로 호출함
- 편의성을 위해 import numpy, np로 줄여서 사용함

```
import numpy as np

a = [1,2,3,4,5]
b = np.array(a)
print(b)
print(type(b))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

1. Numpy 모듈 활용

2) Numpy 모듈의 특징

- 파이썬의 리스트 자료형과 아주 유사함
- 실제로는 리스트와 비슷한 기능들을 사용할 수 있음

(예) 인덱싱과 슬라이싱(a: 리스트, b: Numpy)

```
import numpy as np

a = [1,2,3,4,5]
b = np.array(a)

print(type(a))
print(type(b))
```

```
<class 'list'>
<class 'numpy.ndarray'>
```

```
print(a[0])
print(b[0])
```

```
1
1
```

```
print(a[0:3])
print(b[0:3])
```

```
[1, 2, 3]
[1 2 3]
```

1. Numpy 모듈 활용

2) Numpy 모듈의 특징

(1) 행렬 형태의 행렬 연산 지원

- 리스트: 연결, 반복 연산만 지원
- Numpy: 실제 행렬과 같이 행렬의 *, +, - 등의 연산 제공

➡ 수학 계산에 용이함

```
import numpy as np

a = [1,2,3,4,5]
b = np.array(a)

print(a + a)
print(b + b)
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
[ 2  4  6  8 10]
```

```
print(a * 2)
print(b * 2)
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
[ 2  4  6  8 10]
```

2) Numpy 모듈의 특징

(1) 행렬 형태의 행렬 연산 지원

- 리스트
  - -, / 연산 불가능
  - + 연결, * 반복 연산만 가능
- Numpy: 사칙연산 모두 가능

```
print(a / 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-6d0c9f5e4280> in <module>
----> 1 print(a / 2)

TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

```
print(a - a)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-9ca2a09805b6> in <module>
----> 1 print(a - a)

TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

```
print(b + 2)
print(b / 5)
print(b - b)
```

```
[3 4 5 6 7]
[0.2 0.4 0.6 0.8 1. ]
[0 0 0 0 0]
```

2) Numpy 모듈의 특징

(2) 다차원 행렬 및 행렬 연산 지원

- shape: 현재 행렬의 크기를 구할 수 있음

```
import numpy as np

a = [[1,1,1],[2,2,2],[3,3,3]]
b = np.array(a)

print(a)
print(b)
```

```
[[1, 1, 1], [2, 2, 2], [3, 3, 3]]
[[1 1 1]
 [2 2 2]
 [3 3 3]]
```

```
print(b.shape)
print(b * 3)
```

```
(3, 3)
[[3 3 3]
 [6 6 6]
 [9 9 9]]
```

1. Numpy 모듈 활용

2) Numpy 모듈의 특징

(3) 크기가 다른 두 행렬의 연산 지원

```
import numpy as np

a1 = [[1,1,1],[2,2,2],[3,3,3]]
a2 = [1,2,3]
b1 = np.array(a1)
b2 = np.array(a2)

print(b1)
print(b2)
```

```
[[1 1 1]
 [2 2 2]
 [3 3 3]]
[1 2 3]
```

```
print(b1 + b2)
```

```
[[2 3 4]
 [3 4 5]
 [4 5 6]]
```

1. Numpy 모듈 활용

2) Numpy 모듈의 특징

(4) 리스트 자료형보다 다양한 방식의 인덱싱 지원

- 리스트: 요소를 기준으로 인덱싱
- 행렬: 위치를 기준으로 인덱싱

```
import numpy as np

a = [[1,1,1],[2,2,2],[3,3,3]]
b = np.array(a)

print(b[:,2])
print(b[1,:2])
```

```
[1 2 3]
[2 2]
```

```
print(b[1][1])
print(b[1,1])
```

```
2
2
```

➡ b[:,2]은 2열에 있는 모든 값,

b[1,:2]는 1행의 2번째까지의 값을 출력할 수 있음

2) Numpy 모듈의 특징

(5) 제공하는 함수 종류

- 수학과 관련된 함수: sqrt, log, max 등
- 행렬을 쉽게 정의할 수 있는 함수: zeros, ones, arange 등

```
# 수학 관련 함수
print(np.sqrt(4))
print(np.log(10))
print(np.max([1,2,3]))
```

```
2.0
2.302585092994046
3
```

```
# random 함수
print(np.random.randint(1,10))
print(np.random.normal())
```

```
8
0.11643799429918289
```

```
print(np.zeros((3,3)))
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

2. Pandas 모듈 활용

1) Pandas 모듈의 개념

(1) Pandas 모듈

- 행과 열로 이루어진 데이터를 쉽게 다룰 수 있도록 도와주는 파이썬의 데이터 분석 전용 외부 모듈

➡ 특히 대용량의 데이터를 처리하는 데 편리함

(2) 설치 방법

- Anaconda 설치 시 함께 설치됨
- pip install pandas로도 설치 가능함

(3) 호출 방법

- 흔히 import pandas as pd로 호출하여 사용
- import pandas로만 호출해도 되지만 편의를 위해 pd로 줄여 사용

2. Pandas 모듈 활용

2) Pandas 모듈의 특징

(1) Pandas의 자료형

- Series 자료형
  - 인덱스와 값을 가지고 있음
  - 별도로 인덱스, 값을 출력할 수 있음
  - 정의할 때 인덱스를 따로 정해줄 수 있음

```
import pandas as pd

a = pd.Series([1,2,3,4],index = ['a','b','c','d'])
print(a)
print(a.index)
```

```
a    1
b    2
c    3
d    4
dtype: int64
Index(['a', 'b', 'c', 'd'], dtype='obj
```

```
import pandas as pd

a = pd.Series([1,2,3,4])
print(a)
print(type(a))
```

```
0    1
1    2
2    3
3    4
dtype: int64
<class 'pandas.core.series.Series'>
```

2) Pandas 모듈의 특징

(1) Pandas의 자료형

- Series 자료형
  - 파이썬의 딕셔너리 자료형
  - Numpy의 Array 자료형도 Series 자료형으로 만들 수 있음
  - 딕셔너리의 키가 Series의 인덱스가 됨

```
import pandas as pd

a = pd.Series({'a':1,'b':2,'c':3})
print(a)
print(a.index)
```

```
a    1
b    2
c    3
dtype: int64
Index(['a', 'b', 'c'], dtype='object')
```

```
import pandas as pd
import numpy as np

a = np.array([1,2,3,4])
b = pd.Series(a)
print(b)
```

```
0    1
1    2
2    3
3    4
dtype: int32
```

2) Pandas 모듈의 특징

(1) Pandas의 자료형

- DataFrame 자료형
  - 행과 열로 이루어진 자료형
  - Series와 마찬가지로 파이썬의 딕셔너리 자료형 또는 Numpy의 array로도 정의할 수 있음

(예) Series vs. DataFrame

- Series: 인덱스, 값으로만 구성, 1차원 배열 형태의 자료구조
- DataFrame: 행과 열로 구성, 2차원 테이블 형태의 자료구조

```
import pandas as pd

a = pd.Series({'a':[1,1,1],'b':[2,2,2],'c':[3,3,3]})
b = pd.DataFrame({'a':[1,1,1],'b':[3,3,3],'c':[3,3,3]})

print(type(a))
print(type(b))
```

print(a)

```
a    [1, 1, 1]
b    [2, 2, 2]
c    [3, 3, 3]
dtype: object
```

print(b)

```
   a  b  c
0  1  3  3
1  1  3  3
2  1  3  3
```

2. Pandas 모듈 활용

2) Pandas 모듈의 특징

(1) Pandas의 자료형

- DataFrame 자료형
  - 행: index
  - 열: columns

➡ 특정 칼럼의 데이터를 손쉽게 변경할 수 있음

```
import pandas as pd

a = pd.DataFrame({'a':(1,2),'b':1,'c':3})
print(a)
print(a.index)
print(a.columns)
```

```
   a  b  c
0  1  1  3
1  2  1  3
RangeIndex(start=0, stop=2, step=1)
Index(['a', 'b', 'c'], dtype='object')
```

```
a['b'] = [3,4]
print(a)
```

```
   a  b  c
0  1  3  3
1  2  4  3
```

2) Pandas 모듈의 특징

(2) DataFrame의 다양한 함수

- index와 columns를 변경할 수 있음
  - iloc: 행 인덱스로 값을 가져올 수 있음
  - loc: 행 이름으로 값을 가져올 수 있음

```
import pandas as pd

a = pd.DataFrame({'a':(1,2),'b':1,'c':3})

print(a)
a.index = ['x','y']
a.columns = ['i','j','k']
print()
print(a)
```

```
   a  b  c
0  1  1  3
1  2  1  3

   i  j  k
x  1  1  3
y  2  1  3
```

```
print(a['i'])
```

```
x    1
y    2
Name: i, dtype: int64
```

```
print(a.iloc[0])
```

```
i    1
j    1
k    3
Name: x, dtype: int64
```

```
print(a.loc['x'])
```

```
i    1
j    1
k    3
Name: x, dtype: int64
```

2) Pandas 모듈의 특징

(2) DataFrame의 다양한 함수

- describe(): DataFrame에서 계산 가능한 값들에 대한 결과를 간략하게 보여줌

```
import pandas as pd

a = pd.DataFrame({'a':(1,2,3),'b':[4,5,6],'c':[7,8,9]})
print(a)

print(a.describe())
```

```
  a  b  c
0  1  4  7
1  2  5  8
2  3  6  9
       a    b    c
count  3.0  3.0  3.0
mean   2.0  5.0  8.0
std    1.0  1.0  1.0
min    1.0  4.0  7.0
25%    1.5  4.5  7.5
50%    2.0  5.0  8.0
75%    2.5  5.5  8.5
max    3.0  6.0  9.0
```

2) Pandas 모듈의 특징

(2) DataFrame의 다양한 함수

- sum(): 합계를 보여줌
  - axis 옵션으로 행, 열 기준을 변경할 수 있음

```
print(a.sum())
```

```
a    6
b   15
c   24
dtype: int64
```

```
print(a.sum(axis=1))
```

```
0   12
1   15
2   18
dtype: int64
```

- 그 외 다양한 함수들
  - min, max: 최소, 최댓값
  - argmin, argmax: 최소, 최댓값의 인덱스를 반환
  - mean: 평균
  - median: 중간값
  - std, var: 표준편차, 분산
  - unique: 특정 행 또는 열에서 중복 값을 제외한 유니크 값을 반환

3. 모듈을 활용한 데이터 분석 실습

1) 영화 장르 데이터를 모듈을 활용하여 분석하기

(1) 영화 장르별 빈도 수 분석

- 데이터 불러오기
  - Pandas 모듈의 read_csv 함수를 활용해 DataFrame으로 불러올 수 있음

```
import pandas as pd

data = pd.read_csv('ml-latest-small/movies.csv')
data
```

	movieId	title
0	1	Toy Story (1995)
1	2	Jumanji (1995)
2	3	Grumpier Old Men (1995)

3. 모듈을 활용한 데이터 분석 실습

1) 영화 장르 데이터를 모듈을 활용하여 분석하기

(1) 영화 장르별 빈도 수 분석

- 장르 분리하기
  - 반복문을 활용하여 genres 칼럼의 장르 값들을 모두 분리하여 리스트에 저장

```
genre = []
for i in data['genres']:
    genre.extend(i.split('|'))
print(len(genre))
print(genre)
```

```
22084
['Adventure', 'Animation', 'Children', 'Comedy', 'Fanta
medy', 'Action', 'Crime', 'Thriller', 'Comedy', 'Romanc
e', 'Comedy', 'Horror', 'Adventure', 'Animation', 'Chil
y', 'Comedy', 'Action', 'Comedy', 'Crime', 'Drama', 'Th
e', 'Thriller', 'Drama', 'Sci-Fi', 'Drama', 'Romance', 'Dra
'Crime', 'Drama', 'Drama', 'Mystery', 'Sci-Fi', 'Thriller',
```

3. 모듈을 활용한 데이터 분석 실습

1) 영화 장르 데이터를 모듈을 활용하여 분석하기

(1) 영화 장르별 빈도 수 분석

- 중복된 장르 제거하기
  - Pandas의 unique 함수를 활용하여 중복을 제거한 장르를 저장

```
unique_genre = pd.unique(genre)
print(len(unique_genre))
print(unique_genre)
```

```
20
['Adventure' 'Animation' 'Children' 'Comedy' 'Fantasy' 'R
 'Action' 'Crime' 'Thriller' 'Horror' 'Mystery' 'Sci-Fi' 'War'
 'Documentary' 'IMAX' 'Western' 'Film-Noir' '(no genres l
```

3. 모듈을 활용한 데이터 분석 실습

1) 영화 장르 데이터를 모듈을 활용하여 분석하기

(1) 영화 장르별 빈도 수 분석

- 빈도 수를 분석하기 위한 DataFrame 생성하기
  - Numpy 모듈의 zeros 함수를 활용하여 장르별 빈도 수를 분석하기 위해 비어 있는 DataFrame 생성

```
import numpy as np
zero_data = np.zeros(len(unique_genre))
result = pd.DataFrame(zero_data,index=unique_genre,columns=['count'])
print(result)
```

	count
Adventure	0.0
Animation	0.0
Children	0.0
Comedy	0.0
Fantasy	0.0
Romance	0.0
Drama	0.0
Action	0.0
Crime	0.0
Thriller	0.0
Horror	0.0
Mystery	0.0
Sci-Fi	0.0
War	0.0
Musical	0.0
Documentary	0.0
IMAX	0.0
Western	0.0
Film-Noir	0.0
(no genres listed)	0.0

3. 모듈을 활용한 데이터 분석 실습

1) 영화 장르 데이터를 모듈을 활용하여 분석하기

(1) 영화 장르별 빈도 수 분석

- 장르 빈도 수 분석하기
  - 반복문을 활용하여 기존 전체 장르가 저장된 리스트를 하나씩 체크하며, 해당 데이터의 값을 +1 해줌

```
for i in genre:
    result.loc[i] +=1
print(result)
```

	count
Adventure	1263.0
Animation	611.0
Children	664.0
Comedy	3756.0
Fantasy	779.0
Romance	1596.0
Drama	4361.0
Action	1828.0
Crime	1199.0
Thriller	1894.0
Horror	978.0
Mystery	573.0
Sci-Fi	980.0
War	382.0
Musical	334.0
Documentary	440.0
IMAX	158.0
Western	167.0
Film-Noir	87.0
(no genres listed)	34.0