Numpy (Numerical Python)

- 다차원 배열 객체인 `ndarray`를 중심으로, 빠르고 효율적인 수치 연산을 제공 / 선형대수, 푸리에 변환 등 고성능 과학 계산 기능을 포함

적용 가능 상황

- 대규모 다차원 배열의 빠른 산술 연산이 필요할 때.
- 선형대수(행렬 곱, 분해 등), 난수 생성, 푸리에 변환 등 수학적 계산이 필요할 때.
- 이미지 데이터를 픽셀 값의 배열로 다루거나, 수치 시뮬레이션을 수행할 때.

주의사항

- Numpy 배열은 모든 원소가 동일한 데이터 타입을 가져야 함
- `for` 루프를 직접 사용하는 것보다 벡터화 연산(배열 전체에 대한 연산)을 사용하는 것이 훨씬 빠름

코드 예시

- `ndarray` 생성, 벡터화 연산, 브로드캐스팅, 집계 함수 사용법을 보여줌
- `axis` 인자를 통해 연산을 적용할 차원을 지정 가능
    - `axis=0`은 행 방향, `axis=1`은 열 방향을 의미

`ndarray` 생성 방법

```
# 0으로 채워진 배열
a = np.zeros((2, 2))
'''
[[ 0. 0.]
 [ 0. 0.]]
'''
# 1로 채워진 배열
b = np.ones((1, 2))
# [[ 1. 1.]]

# 특정 값으로 채워진 배열
c = np.full((2, 2), 7.)
'''
[[ 7. 7.]
 [ 7. 7.]]
'''
# 2x2 단위 행렬(identity matrix), 정방향행렬
d = np.eye(2)
'''
[[ 1. 0.]
 [ 0. 1.]]
'''
# 랜덤 값으로 채운 배열
e = np.random.random((2, 2))
'''
```

```
[[0.17458815, 0.65392064],
 [0.62288583, 0.25213172]]
'''
# 연속된 값으로 배열을 만들 때 (start, stop, step)
f = np.arange(1, 10)
'''
[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10]
'''
g = np.arange(1, 10, 2)
'''
[1, 3, 5, 7, 9]
'''
```

데이터 구조 확인

- 용어 정리
    - Axis : 배열의 각 축 (axis0 = 행, axis1 = 열)
    - Rank : 축의 개수(차원의 수)
    - Shape: 축의 길이(갯수)
- `.ndim` : rank가 몇인지 보여줌(몇 차원인지)
- `.shape` : shape이 어떻게 되는지
- `.dtype` : 배열 내의 데이터 형식이 어떻게 되는지 (요소들의 데이터 형식)

```
a = np.array([1, 2, 3, 4, 5])
print(a)                        # [1 2 3 4 5]
print(type(a))                  # <class 'numpy.ndarray'>
print(a.ndim)                   # 1     <--차원
print(a.shape)                  # (5, ) <--모양
print(a.dtype)                  # int32 <--데이터 형식
print(a[0], a[1], a[2])         # 1 2 3
```

Pandas

- Numpy를 기반으로 구축된 `Series`와 `DataFrame`이라는 유연하고 강력한 데이터 구조를 제공
- 정형 데이터를 직관적 처리, 시계열 분석, 데이터 정제, 결합 등 고수준의 데이터 조작 작업을 쉽게 수행

데이터 확인이 필요한 상황

- **초기 데이터 탐색**: CSV, Excel 등 외부 소스에서 데이터를 처음 불러왔을 때, 데이터가 올바르게 로드되었는지, 어떤 변수들이 있는지 확인하는 가장 첫 단계에서 사용
- **데이터 클리닝 계획 수립**: `info()`와 `isnull().sum()`을 통해 변수들의 데이터 타입이 적절한지(예: 숫자가 object로 되어있지 않은지), 결측치가 얼마나, 어디에 있는지 파악하여 처리 계획을 세울 때 사용
- **데이터 특징 파악**: `describe()`를 통해 수치형 변수들의 분포(평균, 편차, 범위 등)를 개략적으로 파악하고, 이상치의 존재 가능성을 짐작할 때 사용

- **데이터 조작 후 확인**: 필터링, 결합 등 데이터프레임을 변경하는 작업을 수행한 후, `shape`이나 `head()`를 통해 의도한 대로 데이터가 변경되었는지 확인할 때 사용

예제 데이터

- Kaggle의 타이타닉 데이터

```
import pandas as pd
data = pd.read_csv("Titanic-Dataset.csv")
```

head(), tail(), shape, values, columns, dtypes

- `head()`: 상위 데이터 확인 (기본 5개)
- `tail()`: 하위 데이터 확인 (기본 5개)
- `shape`: 데이터프레임 크기
- `values`: 값 정보 확인(저장하면 2차원 numpy 배열이 됨)
- `columns`: 열 정보 확인

```
# 상위 10개 행 데이터
data.head(10)

# 하위 3개 행 데이터
data.tail(3)

# 행 수와 열 수 확인
data.shape

# 데이터프레임 값 확인 (2차원 numpy 배열)
data.values

# 열 확인
print(data.columns)
print(data.columns.values) # np array 형태
list(data) # 데이터프레임을 리스트 함수에 넣으면 열 이름이 리스트로 반환됨.
```

- `dtypes`: 열 자료형 확인
  - `int64`: 정수형 데이터(int)
  - `float64`: 실수형 데이터(float)
  - `object`: 문자열 데이터(string)

```
# 열 자료형 확인
data.dtypes
```

info(), describe()

- `info()`
  - 데이터프레임의 기술적인 요약 정보 출력
  - `Non-Null Count`를 전체 행의 수(`RangeIndex`)와 비교하여 결측치 유무를 빠르게 파악 가능
  - `Dtype`을 통해 데이터 타입이 의도와 다른 경우(e.g., 숫자가 `object` 타입)를 확인 가능
  - 정수값만 있는 열도 결측치가 있을 경우, 실수(float) 타입으로 인식되는 경우 존재

```
# 열 자료형, 값 개수 확인
data.info()
```

- `describe()`
  - 개수(`count`), 평균(`mean`), 표준편차(`std`), 최솟값(`min`),
    사분위값(`25%, 50%, 75%`), 최댓값(`max`)을 표시
  - 수치형 데이터의 분포와 중심 경향성, 산포도를 파악
  - 기본적으로 수치형(int, float) 열에 대해서만 통계를 계산
  - 범주형이나 날짜형 데이터를 포함하려면 `include` 인자를 사용
    - 범주형/날짜형 데이터에 대해 `unique`(고유값 개수),
      `top`(최빈값),`freq`(최빈값의 빈도) 등이 추가로 표시

```
# 기초통계정보
data.describe()
# 모든 데이터 타입에 대한 기술 통계
data.describe(include='all', datetime_is_numeric=True)
```

sort_index()

- `sort_index()` : 인덱스를 기준으로 정렬하는 메소드
  - **ascending** 옵션을 설정해 오름차순, 내림차순을 설정할 수 있습니다.
    - `ascending=True`: 오름차순 정렬(기본값)
    - `ascending=False`: 내림차순 정렬
- `sort_values()` : 특정 열 기준으로 정렬하는 메소드
  - 2가지 방법 → 인덱스를 기준으로 정렬하는 방법과 특정 열을 기준으로 정렬하는 방법
  - **sort_values()** 메소드로 **특정 열**을 기준으로 정렬합니다.
  - **ascending** 옵션을 설정해 오름차순, 내림차순을 설정할 수 있습니다.
    - `ascending=True`: 오름차순 정렬(기본값)
    - `ascending=False`: 내림차순 정렬

```
# 단일 열 정렬
data.sort_values(by='Age', ascending=False)
# 복합 열 정렬
data.sort_values(by=['Fare', 'Age'], ascending=[True, False])
# 복합 열 정렬 : 별도로 저장하고, 인덱스 reset
temp = data.sort_values(by=['Fare', 'Age'], ascending=[True, False])
temp.reset_index(drop = True) # 기존 index는 삭제하고, 현재 데이터 기준으로 인덱스를
재작성
```

unique()

- `unique()` : 고유 값 확인

```
# Embarked 열 고유값 확인
print(data['Embarked'].unique())
```

- `value_counts()` : 고유 값과 그 개수들을 확인, dtype도 확인 가능

```
# Embarked 열 고유값 개수 확인
print(data['Embarked'].value_counts())
```

집계함수

- `sum()`, `max()`, `mean()`, `median()`, `count()` : 기본 집계 메소드, Groupby 기능과 함께 많이 사용

```
# Fare 열 합계 조회
print(data['Fare'].sum())

# Fare 열 최댓값 조회
print(data['Fare'].max())

# 'Age', 'Fare' 열 평균값 확인
print(data[['Age', 'Fare']].mean())

# 'Age', 'Fare' 열 중앙값 확인
print(data[['Age', 'Fare']].median())

# Fare 열 데이터 개수 확인
print(data['Fare'].count())
```