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

```
import numpy as np

# ndarray 생성
arr = np.array([[1, 2, 3], [4, 5, 6]])

# 배열의 크기 및 형태 확인
print(f"Shape: {arr.shape}") # Shape: (2, 3)
print(f"Data type: {arr.dtype}") # Data type: int32

# 벡터화 연산 (Element-wise operation)
arr_plus_10 = arr + 10
# [[11, 12, 13],
#  [14, 15, 16]]

# 브로드캐스팅 (Broadcasting)
vector = np.array([1, 0, 1])
result = arr + vector # (2, 3) 배열과 (3,) 벡터의 연산
# [[2, 2, 4],
#  [5, 5, 7]]

# 집계 함수
print(f"Sum: {np.sum(arr)}") # Sum: 21
print(f"Mean (axis=0): {np.mean(arr, axis=0)}") # Mean (axis=0): [2.5 3.5 4.5]

# 선형대수: 행렬 곱
mat_a = np.array([[1, 2], [3, 4]])
mat_b = np.array([[5, 6], [7, 8]])
mat_mul = np.dot(mat_a, mat_b) # 또는 mat_a @ mat_b
# [[19, 22],
#  [43, 50]]
```

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

Pandas

- Numpy를 기반으로 구축, `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

적용 가능 상황

- CSV, Excel, SQL 등 다양한 소스로부터 데이터를 읽고 쓸 때.
- 데이터 클리닝(결측치, 이상치 처리), 변환, 필터링, 그룹화 등 데이터 전처리가 필요할 때.
- 이종(heterogeneous) 데이터를 테이블 형태로 다루고, 시계열 데이터를 분석할 때.

주의사항

- `SettingWithCopyWarning` 주의
    - DataFrame의 일부를 복사하여 수정하려 할 때 발생하며, 원본이 수정되지 않을 수 있다.
    - `.loc`를 사용하여 명시적으로 데이터를 수정하는 것이 안전

코드 예시

```
import pandas as pd

# DataFrame 생성
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 30, 35, 40],
        'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']}
df = pd.DataFrame(data)

# 데이터 확인
print(df.head(2)) # 상위 2개 행 출력
print(df.info())    # 데이터 타입, 결측치 등 요약 정보
print(df.describe()) # 수치형 데이터의 기술 통계

# 데이터 선택 (.loc, .iloc)
print(df.loc[0]) # 인덱스 레이블 '0'에 해당하는 행
print(df.iloc[0]) # 정수 위치 '0'에 해당하는 행
print(df.loc[df['Age'] > 30]) # 조건 필터링

# 그룹화 및 집계 (Groupby)
age_mean_by_city = df.groupby('City')['Age'].mean()
# City
# Chicago        35.0
# Houston        40.0
# Los Angeles    30.0
# New York       25.0
# Name: Age, dtype: float64
```

```
# 데이터 결합 (Merge)
df2 = pd.DataFrame({'Name': ['Alice', 'Bob'], 'Salary': [70000, 80000]})
merged_df = pd.merge(df, df2, on='Name', how='left')
```

- `DataFrame` 생성, 데이터 탐색, 특정 데이터 선택, 그룹화 연산, 데이터프레임 결합 등 Pandas의 핵심 기능
- `.loc`는 레이블 기반, `.iloc`는 정수 위치 기반 인덱싱으로, 명확히 구분하여 사용 필요

Scipy (Scientific Python)

- Numpy를 기반으로 과학 및 기술 컴퓨팅을 위한 다양한 고급 함수를 제공
- 통계, 최적화, 신호 처리, 선형대수, 이미지 처리 등 폭넓은 분야의 알고리즘을 포함

적용 가능한 상황

- 가설 검정, 확률 분포 등 복잡한 통계 분석이 필요할 때 (`scipy.stats`).
- 함수의 최적해(최소/최대값)를 찾아야 할 때 (`scipy.optimize`).
- 미분 방정식 풀이, 수치 적분, 보간 등 공학적 계산이 필요할 때.

주의사항

- Scipy는 방대한 하위 모듈로 구성
- 필요한 기능이 어떤 모듈에 있는지(e.g., `scipy.stats`, `scipy.optimize`) 공식 문서를 통해 확인하고 사용하는 것이 좋음

코드 예시

```
from scipy import stats
from scipy import optimize

# 1. 통계 (scipy.stats)
# T-검정 예시: 두 집단의 평균이 통계적으로 유의미하게 다른지 검정
group1 = [20, 22, 19, 20, 21, 20, 18, 25]
group2 = [28, 26, 27, 29, 25, 28, 26, 30]

# 등분산성 검정 (Levene's test)
levene_stat, levene_p = stats.levene(group1, group2)
print(f"Levene test p-value: {levene_p:.4f}")
# p-value가 0.05보다 크면 등분산성 가정 만족

# 독립표본 T-검정 (Independent Two-sample t-test)
# equal_var=True (등분산 가정) 또는 False (이분산 가정, Welch's t-test)
t_stat, p_value = stats.ttest_ind(group1, group2, equal_var=True)
print(f"T-statistic: {t_stat:.4f}, P-value: {p_value:.4f}")
# p-value가 유의수준(e.g., 0.05)보다 작으면, 두 집단의 평균은 유의미하게 다르다고 결론

# 2. 최적화 (scipy.optimize)
# 간단한 1차원 함수의 최소값 찾기
def f(x):
```

```
    return x**2 + 10*np.sin(x)

# 함수 f(x)의 최소값을 x=0 근처에서 찾기 시작
result = optimize.minimize(f, x0=0)
print(f"Minimum value found at x = {result.x[0]:.4f}")
print(f"Function minimum value = {result.fun:.4f}")
```

- **T-검정**: P-value가 매우 작게(e.g., 0.0001) 나왔으므로, 두 그룹의 평균 사이에는 통계적으로 유의미한 차이가 있다고 해석할 수 있습니다.
- **최적화**: `minimize` 함수는 주어진 함수 `f(x)`의 값을 최소로 만드는 `x`의 값과 그때의 함수 값을 찾아줍니다. 이 예시에서는 약 -1.3064에서 최소값 -7.9458을 가짐을 보여줍니다.

장단점 및 대안

라이브러리	장점	단점	대안
Numpy	C/Fortran으로 구현되어 매우 빠름, 강력한 N차원 배열 객체, 풍부한 수학 함수	동일한 데이터 타입만 저장 가능, 데이터 레이블링 기능 부재	Pandas (레이블링, 이종 데이터 처리), Dask (병렬 처리로 대용량 데이터 핸들링)
Pandas	직관적인 데이터 구조(DataFrame), 강력한 데이터 조작 및 분석 기능, 결측치 처리 용이	Numpy보다 메모리 사용량이 많고 속도가 느릴 수 있음, 대용량 데이터 처리 시 성능 저하	Dask (Pandas와 유사한 API로 병렬 처리), Polars (Rust 기반의 빠른 DataFrame 라이브러리)
Scipy	통계, 최적화, 신호 처리 등 광범위한 과학 계산 알고리즘 제공, Numpy와 완벽하게 호환	기능이 매우 방대하여 학습 곡선이 존재함	Statsmodels (통계 분석 및 모델링에 더 특화), scikit-learn (머신러닝 알고리즘에 집중)