제조데이터 분석 및 시각화 ('22.02.22-'22.02.25) 2일차

Prepared by DaeKyeong Kim
Ph.D.

학습일정 및 내용

일차	시 간	교육 내용	세부 학습 내용	비고
1일차 (○○/○○)		Syllabus Python Basics		
	5교시		Syllabus와 작업 환경 이해	강의, 실습 및 코칭
	6교시		파이썬 기초	강의, 실습 및 코칭
	7교시		파이썬 기초	강의, 실습 및 코칭
	8교시		파일 읽고 쓰기	강의, 실습 및 코칭
2일차 (○○/○○)	1교시	Data Structures Data Cleanup		
	2교시			
	3교시			
	4교시			
	5교시		수치배열 데이터 패키지 소개	강의, 실습 및 코칭
	6교시		과학기술계산패키지 소개	강의, 실습 및 코칭
	7교시		sympy	강의, 실습 및 코칭
	8교시		데이터분석 패키지 소개	강의, 실습 및 코칭

학습일정 및 내용

일차	시 간	교육 내용	세부 학습 내용	비고
3일차 (○○/○○)	1교시	데이터 셋 적재하기와 Cleanup		
	2교시			
	3교시			
	4교시			
	5교시		데이터 셋 적재하기	강의, 실습 및 코칭
	6교시		Data Structures	강의, 실습 및 코칭
	7교시		Data Cleanup: Investigation, Matching, and Formatting	강의, 실습 및 코칭
	8교시		Data Cleanup: Investigation, Matching, and Formatting	강의, 실습 및 코칭
4일차 (○○/○○)	1교시	Data Structures Data Cleanup		
	2교시			
	3교시			
	4교시			
	5교시		시각화 패키지 소개	
	6교시		시각화 패키지 소개	
	7교시		시각화 패키지 소개	
	8교시		Iris를 통한 시각화 연습	

학습일정 및 내용

일차	시 간	교육 내용	세부 학습 내용	비고
5일차 (○○/○○)	1교시	제조데이터 분석 및 시각화 미니 프로젝트		
	2교시			
	3교시			
	4교시			
	5교시		미니 프로젝트	강의, 실습 및 코칭
	6교시		미니 프로젝트	강의, 실습 및 코칭
	7교시		미니 프로젝트	강의, 실습 및 코칭
	8교시		미니 프로젝트	강의, 실습 및 코칭

Contents

1

수치배열 데이터 패키지 소개

1. 프리미티브 타입과 라이브러리
2. 넘파이

학습목표

- 배열과 리스트의 차이점을 알고 배열을 사용하는 이유를 이해한다.
- 배열을 생성하고 다루는 방법을 익힌다.
- 넘파이를 사용하여 기술 통계를 낼 수 있다.
- 난수를 발생시키고 그 결과를 분석하는 방법을 공부한다.

Unit 1

프리미티브 타입과 라이브러리

프리미티브 타입 예

리스트, 딕셔너리

❖

```
L=[88, 90, 100]
L
```

```
[88, 90, 100]
```

```
#88과 100의 평균 구하기

(L[0]+L[2])/2
```

```
94.0
```

```
D={
    "math": 88,
    "english": 90,
    "history" : 100
}

D
```

```
{'english': 90, 'history': 100, 'math': 88}
```

```
#수학점수와 영어점수 평균 구하기

(D["english"]+D["math"])/2
```

```
89.0
```

프리미티브 타입 예

튜플, Set

```
T=(88, 90, 100)
T
```

```
(88, 90, 100)
```

```
S={88, 90, 100}
S
```

```
{88, 90, 100}
```

```
#88과 100의 평균 구하기

(T[0]+T[2])/2
```

```
94.0
```

기본 라이브러리

❖주피터 노트북

- ✓ 프로그램 코드를 브라우저에서 실행해주는 대화식 환경.
- ✓ 탐색적 데이터 분석에 적합.

❖ Numpy 넘파이

- ✓ 파이썬으로 과학 계산을 하려면 꼭 필요한 패키지.
- ✓ 다차원 배열을 위한 기능과 선형 대수 연산과 푸리에 변환 같은 고수준 수학 함수와 유사 난수 생성기를 포함
- ✓ 다음 명령으로 설치한다.

```
pip install numpy
```

- ✓ 임포트할 때는 보통 np라는 별명으로 임포트한다.

```
import numpy as np
```

기본 라이브러리

❖Scipy 싸이파이

- ✓ 과학 계산용 함수를 모아놓은 파이썬 패키지.
- ✓ 고성능 선형 대수, 함수 최적화, 신호 처리, 특수한 수학 함수와 통계 분포등을 포함한 많은 기능 제공
- ✓ 다음 명령으로 설치한다.

```
pip install scipy
```

- ✓ 임포트할 때는 보통 sp라는 별명으로 임포트한다.

```
import scipy as sp
```

❖ matplotlib 맷플랏립

- ✓ 대표적인 과학 계산용 그래프 라이브러리.
- ✓ 선 그래프, 히스토그램, 산점도 등을 지원하며 고품질 그래프를 그려준다.

기본 라이브러리

❖ matplotlib 맷플랏립

✓ 대표적인 과학 계산용 그래프 라이브러리.

✓ 선 그래프, 히스토그램, 산점도 등을 지원하며 고품질 그래프를 그려준다.
✓ 다음 명령으로 설치한다.

```
pip install matplotlib
```

✓ 임포트할 때는 보통 pylab 서브패키지를 plt라는 별명으로 임포트한다.

```
import matplotlib.pylab as plt
```

기본 라이브러리

❖ pandas 판다스

- ✓ 데이터 처리와 분석을 위한 파이썬 라이브러리.
- ✓ R의 data.frame을 본떠서 설계한 DataFrame이라는 데이터 구조를 기반으로 만들어졌다.
- ✓ 엑셀의 스프레드시트와 비슷한 테이블 형태.
- ✓ SQL처럼 테이블에 쿼리나 조인을 수행할 수 있다.
- ✓ SQL, 엑셀 파일, CSV 파일 같은 다양한 파일과 DB에서 읽어들일 수 있다.
- ✓ 다음 명령으로 설치한다.

```
pip install pandas
```

- ✓ 임포트할 때는 보통 pd라는 별명으로 임포트한다.

```
import pandas as pd
```

기본 라이브러리

❖seaborn 패키지

- ✓ seaborn("시본"이라고 읽는다) 패키지는 맷플롯립 패키지에서 지원하지 않는 고급 통계 차트를 그리는 통계용 시각화 기능을 제공한다.

- ✓ 다음 명령으로 설치한다.

```
pip install seaborn
```

- ✓ 임포트할 때는 보통 sns라는 별명으로 임포트한다.

```
import seaborn as sns
```

기본 라이브러리

❖statsmodels 패키지

- ✓ statsmodels("스탯츠모델즈"라고 읽는다) 패키지는 추정 및 검정, 회귀분석, 시계열분석 등의 기능을 제공하는 파이썬 패키지다. 기존에 R에서 가능했던 다양한 회귀분석과 시계열분석 방법론을 그대로 파이썬에서 이용할 수 있다. 다음은 statsmodels 패키지가 제공하는 기능의 일부다.
- ✓ 예제 데이터셋
- ✓ 검정 및 모수추정
- ✓ 회귀분석
  - 선형회귀
  - 강건회귀
  - 일반화 선형모형
  - 혼합효과모형
  - 이산종속변수
- ✓ 시계열 분석
  - SARIMAX 모형
  - 상태공간 모형
  - 벡터 AR 모형
- ✓ 생존분석
- ✓ 요인분석
- ✓ 다음 명령으로 설치한다.

```
pip install statsmodels
```

- ✓ 임포트할 때는 보통 api 서브패키지를 sm이라는 별명으로 임포트한다.

```
import statsmodels.api as sm
```

머신러닝 패키지

❖scikit-learn 패키지

- ✓ scikit-learn("사이킷런"이라고 읽는다) 패키지는 머신러닝 교육을 위한 최고의 파이썬 패키지다. scikit-learn 패키지의 장점은 다양한 머신러닝 모형을 하나의 패키지에서 모두 제공하고 있다는 점이다. 다음은 scikit-learn 패키지에서 제공하는 머신러닝 모형의 목록의 일부다.
- ✓ 데이터셋
  - 회귀분석, 분류, 클러스터링용 가상 데이터셋 생성
  - 각종 벤치마크 데이터셋
- ✓ 전처리
  - 스케일링
  - 누락데이터 처리
  - 텍스트 토큰화
- ✓ 지도학습
  - 회귀분석
  - LDA/QDA
  - 서포트벡터머신
  - 퍼셉트론, SGD
  - KNN
  - 가우스프로세스
  - 나이브베이즈
  - 의사결정나무
  - 랜덤포레스트, 부스팅

머신러닝 패키지

✓ 비지도학습

- 가우스 혼합모형
- 클러스터링
- PCA

✓ 성능 최적화

- 교차검증
- 특징선택
- 하이퍼파라미터 최적화

✓ 설치와 임포트에 사용하는 이름은 sklearn이다. 다음 명령으로 설치한다.

```
pip install sklearn
```

✓ 임포트할 때는 보통 sk이라는 별명으로 임포트한다.

```
import sklearn as sk
```

머신러닝 패키지

✓ 비지도학습

- 가우스 혼합모형
- 클러스터링
- PCA

✓ 성능 최적화

- 교차검증
- 특징선택
- 하이퍼파라미터 최적화

✓ 설치와 임포트에 사용하는 이름은 sklearn이다. 다음 명령으로 설치한다.

```
pip install sklearn
```

✓ 임포트할 때는 보통 sk이라는 별명으로 임포트한다.

```
import sklearn as sk
```

데이터 전처리용 패키지

❖ missingno 패키지
- ✓ pandas 데이터프레임 데이터에서 누락된 데이터를 찾고 시각화하는 기능을 제공한다.
- ✓ 다음 명령으로 설치한다.
- ✓ pip install missingno

❖ patsy 패키지
- ✓ pandas 데이터프레임 데이터에서 선택, 변형하는 기능을 제공한다.
- ✓ statsmodels가 의존하는 패키지이므로 statsmodels 패키지를 설치하면 별도로 설치할 필요가 없다.

딥러닝

- ❖ Theano
  - ✓ 최초의 딥러닝 라이브러리 중 하나로 파이썬(Python) 기반이며 CPU 및 GPU의 수치계산에 매우 유용한 선형대수 심벌 컴파일러
- ❖ 텐서플로(TensorFlow)
  - ✓ 가장 인기 있는 딥러닝 라이브러리 중 하나인 텐서플로(TensorFlow)는 구글 팀에서 개발했으며 2015년 오픈소스로 공개
- ❖ 케라스(Keras)
  - ✓ Theano 와 텐서플로(TensorFlow)는와 함께 사용하는 케라스(Keras)는 효율적인 신경망 구축을 위한 단순화 된 인터페이스로 개발
- ❖ 토치(Torch)
  - ✓ 토치(Torch)는 루아(Lua)라는 스크립트 언어를 기반으로 제작된 딥러닝 프레임워크로 페이스북, 구글 등과 같은 대기업에서도 토치를 기반으로 하는 자체 버전을 별도로 개발
- ❖ DL4J(DeepLearning4J)
  - ✓ DL4J는 자바(Java)로 개발된 인기 있는 딥러닝 프레임워크

기타 패키지

- ❖ 텍스트 전처리용 패키지
  - ✓ nltk 패키지
  - ✓ spacy 패키지
  - ✓ konlpy 패키지
  - ✓ soynlp 패키지
  - ✓ gensim 패키지
- ❖ 이미지 전처리용 패키지
  - ✓ opencv 패키지
- ❖ 사운드 전처리용 패키지
  - ✓ librosa 패키지
- ❖ 지리정보 전처리용 패키지
  - ✓ geopandas 패키지

Unit 2

넘파이

개요

❖ 넘파이(NumPy)

❖ 넘파이는 수치해석용 파이썬 패키지이다.

❖ 많은 숫자 데이터를 하나의 변수에 넣고 관리 할 때 리스트는 속도가 느리고 메모리를 많이 차지하는 단점이 있다. 배열(array)을 사용하면 적은 메모리로 많은 데이터를 빠르게 처리할 수 있다. 배열은 리스트와 비슷하지만 다음과 같은 점에서 다르다.

❖ 모든 원소가 같은 자료형이어야 한다.
❖ 원소의 갯수를 바꿀 수 없다.
❖ 파이썬은 자체적으로 배열 자료형을 제공하지 않는다. 따라서 배열을 구현한 다른 패키지를 임포트해야 한다. 파이썬에서 배열을 사용하기 위한 표준 패키지는 넘파이(NumPy)다.

1차원 배열 만들기

❖ 넘파이의 array 함수에 리스트를 넣으면 ndarray 클래스 객체 즉, 배열로 변환해 준다. 따라서 1 차원 배열을 만드는 방법은 다음과 같다.

```
A1=np.array(
    [0,1,2,3,4,5,6,7,8,9,10]
)

A1
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
# 첫번째 값과 가장 마지막 값에 대한 평균을 구하시오.

(A1[0]+A1[-1])/2
```

```
5.0
```

2차원 배열 만들기

❖ ndarray 는 N-dimensional Array의 약자이다. 이름 그대로 1차원 배열 이외에도 2차원 배열, 3차원 배열 등의 다차원 배열 자료 구조를 지원한다. 2차원 배열은 행렬(matrix)이라고 하는데 행렬에서는 가로줄을 행(row)이라고 하고 세로줄을 열(column)이라고 부른다. 다음과 같이 리스트의 리스트(list of list)를 이용하면 2차원 배열을 생성할 수 있다. 안쪽 리스트의 길이는 행렬의 열의 수 즉, 가로 크기가 되고 바깥쪽 리스트의 길이는 행렬의 행의 수, 즉 세로 크기가 된다. 예를 들어 2개의 행과 3개의 열을 가지는 2 x 3 배열은 다음과 같이 만든다.

```
A2=np.array(
   [
    [0,1,2],
    [3,4,5]
   ]
)

A2
```

```
array([[0, 1, 2],
       [3, 4, 5]])
```

```
# 첫번째 값과 가장 마지막 값에 대한 평균을 구하시오.

(A2[0,0]+A2[-1,-1])/2
```

```
2.5
```

3차원 배열 만들기

❖ 리스트의 리스트의 리스트를 이용하면 3차원 배열도 생성할 수 있다. 크기를 나타낼 때는 가장 바깥쪽 리스트의 길이부터 가장 안쪽 리스트 길이의 순서로 표시한다. 예를 들어 2 x 3 x 4 배열은 다음과 같이 만든다.

```
A3=np.array(
    [
     [
      [0,1,2,3],
      [4,5,6,7],
      [8,9,10,11]
     ],
     [
      [12,13,14,15],
      [16,17,18,19],
      [20,21,22,23]
     ]
    ]
)

A3
```

```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

```
# 첫번째 값과 가장 마지막 값에 대한 평균을 구하시오.

(A3[0,0,0]+A3[-1,-1,-1])/2
```

```
11.5
```

배열의 차원과 크기 알아내기

❖ 배열의 차원 및 크기를 구하는 더 간단한 방법은 배열의 ndim 속성과 shape 속성을 이용하는 것이다. ndim 속성은 배열의 차원, shape 속성은 배열의 크기를 반환한다.

```
# 1차원
#A1.ndim, A1.shape

# 2차원
#A2.ndim, A2.shape

# 3차원
A3.ndim, A3.shape
```

```
(3, (2, 3, 4))
```

```
a = np.arange(12)
a
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
b = a.reshape(3, 4)
b
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

전치 연산

❖ 2차원 배열의 전치(transpose) 연산은 행과 열을 바꾸는 작업이다.

```
A = np.array([[1, 2, 3], [4, 5, 6]])
A
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
A.T
```

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

전치 연산

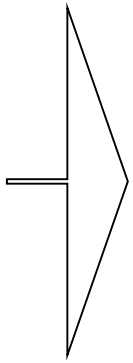
❖ numpy 행렬 연산

```
A=np.array([[1,2,3],[4,5,6]])
B=np.array([[-3,-2,-1],[-6,-5,-4]])
print(A)
print(B)
```

```
[[1 2 3]
 [4 5 6]]
[[-3 -2 -1]
 [-6 -5 -4]]
```

```
print(A+B)
print(A-B)
print(A*B)
print(np.dot(A,B))
```

```
[[-2  0  2]
 [-2  0  2]]
[[ 4  4  4]
 [10 10 10]]
[[ -3  -4  -3]
 [-24 -25 -24]]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-21-50e8d0f8be19> in <module>()
      2 print(A-B)
      3 print(A*B)
----> 4 print(np.dot(A,B))

<__array_function__ internals> in dot(*args, **kwargs)

ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```



```
A
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
A.T
```

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

```
print(np.dot(A.T,B))
```

```
[[-27 -22 -17]
 [-36 -29 -22]
 [-45 -36 -27]]
```

배열 연결

❖ 행의 수나 열의 수가 같은 두 개 이상의 배열을 연결하여(concatenate) 더 큰 배열을 만들 때는 다음과 같은 명령을 사용한다.

- ❖ hstack
- ❖ vstack
- ❖ dstack
- ❖ stack
- ❖ r_
- ❖ c_
- ❖ tile

기타 배열 생성 명령

- zeros, ones
- zeros_like, ones_like
- empty
- arange
- linspace, logspace

- 크기가 정해져 있고 모든 값이 0인 배열을 생성하려면 `zeros` 명령을 사용한다. 인수로는 배열을 크기를 뜻하는 정수를 넣는다.

```
a = np.zeros(5)
a
```

- 크기를 뜻하는 튜플을 입력하면 다차원 배열도 만들 수 있다.

```
b = np.zeros((2, 3))
b
```

기타 배열 생성 명령

❖ `array` 명령과 마찬가지로 `dtype` 인수를 명시하면 해당 자료형 원소를 가진 배열을 만든다.

```
c = np.zeros((5, 2), dtype="i")
c
```

❖ 문자열 배열도 가능하지면 모든 원소의 문자열 크기가 같아야 한다. 만약 더 큰 크기의 문자열을 할당하면 잘릴 수 있다.

```
d = np.zeros(5, dtype="U4")
d

d[0] = "abc"
d[1] = "abcd"
d[2] = "ABCDE"
d
```

dtype 접두사	설명	사용 예
b	불리언	b (참 혹은 거짓)
i	정수	i8 (64비트)
u	부호 없는 정수	u8 (64비트)
f	부동소수점	f8 (64비트)
c	복소 부동소수점	c16 (128비트)
O	객체	0 (객체에 대한 포인터)
S	바이트 문자열	S24 (24 글자)
U	유니코드 문자열	U24 (24 유니코드 글자)

기타 배열 생성 명령

❖ 0이 아닌 1로 초기화된 배열을 생성하려면 `ones` 명령을 사용한다.

```
e = np.ones((2, 3, 4), dtype="i8")
e
```

❖ 만약 크기를 튜플로 명시하지 않고 다른 배열과 같은 크기의 배열을 생성하고 싶다면 `ones_like`, `zeros_like` 명령을 사용한다.

```
f = np.ones_like(b, dtype="f")
f
```

❖ 배열의 크기가 커지면 배열을 초기화하는데도 시간이 걸린다. 이 시간을 단축하려면 배열을 생성만 하고 특정한 값으로 초기화를 하지 않는 `empty` 명령을 사용할 수 있다. `empty` 명령으로 생성된 배열에는 기존에 메모리에 저장되어 있던 값이 있으므로 배열의 원소의 값을 미리 알 수 없다.

```
g = np.empty((4, 3))
g
```

기타 배열 생성 명령

❖ `arange` 명령은 NumPy 버전의 `range` 명령이라고 볼 수 있다. 특정한 규칙에 따라 증가하는 수열을 만든다.

```
np.arange(10)  # 0 .. n-1

np.arange(3, 21, 2)  # 시작, 끝(포함하지 않음), 단계
```

❖ `linspace` 명령이나 `logspace` 명령은 선형 구간 혹은 로그 구간을 지정한 구간의 수만큼 분할한다.

```
np.linspace(0, 100, 5)  # 시작, 끝(포함), 갯수

np.logspace(0.1, 1, 10)
```

배열 크기 변형

❖ 일단 만들어진 배열의 내부 데이터는 보존한 채로 형태만 바꾸려면 reshape 명령이나 메서드를 사용한다. 예를 들어 12개의 원소를 가진 1차원 행렬은 3x4 형태의 2차원 행렬로 만들 수 있다.

```
a = np.arange(12)
a
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
b = a.reshape(3, 4)
b
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

배열 크기 변형

❖ 사용하는 원소의 갯수가 정해저 있기 때문에 reshape 명령의 형태 튜플의 원소 중 하나는 -1이라는 숫자로 대체할 수 있다. -1을 넣으면 해당 숫자는 다를 값에서 계산되어 사용된다.

```
a.reshape(3, -1)
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
a.reshape(2, 2, -1)
```

```
array([[[ 0,  1,  2],
        [ 3,  4,  5]],

       [[ 6,  7,  8],
        [ 9, 10, 11]]])
```

```
a.reshape(2, -1, 2)
```

```
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5]],

       [[ 6,  7],
        [ 8,  9],
        [10, 11]]])
```

배열 크기 변형

❖ 다차원 배열을 무조건 1차원으로 만들기 위해서는 flatten 나 ravel 메서드를 사용한다.

```
a.flatten()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
a.ravel()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

배열 연결

❖ hstack 명령은 행의 수가 같은 두 개 이상의 배열을 옆으로 연결하여 열의 수가 더 많은 배열을 만든다. 연결할 배열은 하나의 리스트에 담아야 한다. vstack 명령은 열의 수가 같은 두 개 이상의 배열을 위아래로 연결하여 행의 수가 더 많은 배열을 만든다. 연결할 배열은 마찬가지로 하나의 리스트에 담아야 한다.

```
a1 = np.ones((2, 3))
a1
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
a2 = np.zeros((2, 2))
a2
```

```
array([[0., 0.],
       [0., 0.]])
```

```
np.hstack([a1, a2])
```

```
array([[1., 1., 1., 0., 0.],
       [1., 1., 1., 0., 0.]])
```

```
b1 = np.ones((2, 3))
b1
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
b2 = np.zeros((3, 3))
b2
```

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

```
np.vstack([b1, b2])
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

배열 연결

- 다차원 배열일 때는 다음과 같이 콤마(comma ,)를 사용하여 접근할 수 있다. 콤마로 구분된 차원을 축(axis)이라고도 한다. 그래프의 x축과 y축을 떠올리면 될 것이다.
- axis 인수가 1이면 두번째 차원으로 새로운 차원이 삽입된다. 다음 예에서 즉, shape 변화는 2개의 (3 x 4) -> 1개의 (3 x 2 x 4) 이다

```
c = np.stack([c1, c2], axis=1)
c
```

```
array([[[1., 1., 1., 1.],
        [0., 0., 0., 0.]],

       [[1., 1., 1., 1.],
        [0., 0., 0., 0.]],

       [[1., 1., 1., 1.],
        [0., 0., 0., 0.]]])
```

```
c.shape
```

```
(3, 2, 4)
```

배열 슬라이싱

❖ 배열 객체로 구현한 다차원 배열의 원소 중 복수 개를 접근하려면 일반적인 파이썬 슬라이싱(slicing)과 comma(,)를 함께 사용하면 된다.

```
[120] print(a[1:5])
      print(a[5:])
      print(a[:5])
      print(a[1:5:2]) # 3번째 요소는 구할 간격을 뜻함
      print(a[::-1]) #본래의 배열의 순서를 거꾸로 한 배열
```

```
[1 2 3 4]
[5 6 7 8 9]
[0 1 2 3 4]
[1 3]
[9 8 7 6 5 4 3 2 1 0]
```

```
[121] B=np.arange(1,7).reshape(2,3)
      B
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
[125] print(B[:2])
      print("----------")
      print(B[:2,0])
      print("----------")
      print(B[:2, ::-1])
```

```
[[1 2 3]
 [4 5 6]]
----------
[1 4]
----------
[[3 2 1]
 [6 5 4]]
```

시드 설정

- 컴퓨터 프로그램에서 발생하는 무작위 수는 사실 엄격한 의미의 무작위 수가 아니다. 어떤 특정한 시작 숫자를 정해 주면 컴퓨터가 정해진 알고리즘에 의해 마치 난수처럼 보이는 수열을 생성한다. 이런 시작 숫자를 시드(seed)라고 한다. 일단 생성된 난수는 다음번 난수 생성을 위한 시드값이 된다. 따라서 시드값은 한 번만 정해주면 된다. 시드는 보통 현재 시각등을 이용하여 자동으로 정해지지만 사람이 수동으로 설정할 수도 있다. 특정한 시드값이 사용되면 그 다음에 만들어지는 난수들은 모두 예측할 수 있다. 이 책에서는 코드의 결과를 재현하기 위해 항상 시드를 설정한다.
- 파이썬에서 시드를 설정하는 함수는 seed이다. 인수로는 0과 같거나 큰 정수를 넣어준다. 넘파이 random 서브패키지에 있는 rand 함수로 5개의 난수를 생성해 보자.

```
np.random.seed(0)
```

```
np.random.rand(5)
```

```
array([0.5488135 , 0.71518937, 0.60276338, 0.54488318, 0.4236548 ])
```

난수 생성

- 넘파이의 random 서브패키지는 이외에도 난수를 생성하는 다양한 함수를 제공한다. 그 중 가장 간단하고 많이 사용되는 것은 다음 3가지 함수다.
  - rand: 0부터 1사이의 균일 분포
  - randn: 표준 정규 분포
  - randint: 균일 분포의 정수 난수

난수 생성

```
np.random.rand(10)
```

```
array([0.64589411, 0.43758721, 0.891773  , 0.96366276, 0.38344152,
       0.79172504, 0.52889492, 0.56804456, 0.92559664, 0.07103606])
```

```
np.random.rand(3, 5)
```

```
array([[0.0871293 , 0.0202184 , 0.83261985, 0.77815675, 0.87001215],
       [0.97861834, 0.79915856, 0.46147936, 0.78052918, 0.11827443],
       [0.63992102, 0.14335329, 0.94466892, 0.52184832, 0.41466194]])
```

```
np.random.randn(10)
```

```
array([ 0.8644362 , -0.74216502,  2.26975462, -1.45436567,  0.04575852,
       -0.18718385,  1.53277921,  1.46935877,  0.15494743,  0.37816252])
```

```
np.random.randn(3, 5)
```

```
array([[-0.88778575, -1.98079647, -0.34791215,  0.15634897,  1.23029068],
       [ 1.20237985, -0.38732682, -0.30230275, -1.04855297, -1.42001794],
       [-1.70627019,  1.9507754 , -0.50965218, -0.4380743 , -1.25279536]])
```

```
np.random.randint(10, size=10)
```

```
array([4, 3, 7, 5, 5, 0, 1, 5, 9, 3])
```

```
np.random.randint(10, 20, size=(3, 5))
```

```
array([[10, 15, 10, 11, 12],
       [14, 12, 10, 13, 12],
       [10, 17, 15, 19, 10]])
```

샘플링

- 데이터가 확률 변수라고 하는 가상의 주사위에 의해 생성된 것이라고 할 때, 이 주사위를 던져서 데이터를 생성하는 과정을 샘플링(sampling)이라고 한다. 또한 샘플링을 통해 얻어진 데이터를 표본(샘플, sample)이라고 한다.
- 통계학에서 샘플링은 다른 의미로도 사용되는데 많은 수의 데이터 집합에서 일부 데이터만 선택하는 과정을 샘플링이라고도 한다.
- 확률실험(random experiments)
  - 어떤 "불확실한 현상"의 관찰결과를 얻는 실험 → 동일한 조건 하에서 다양한 결과가 얻어지는 실험.
- 표본공간(sample space)
  - 한 확률실험에서 관찰 가능한 결과들의 집합.
  - 그리고 표본공간의 각 원소 S_i 를 표본점이라 한다.

$$S = \{s_1, s_2, \cdots\}$$

Unit 4

수열의 의미

❖ 수열(sequence)은 N 개 숫자 또는 변수가 순서대로 나열된 것이다. 다음은 수열의 예다.

1,2,3,4,5

$$x_1, x_2, x_3, x_4, x_5$$

❖ 문자에 붙은 아래 첨자는 순서를 나타내는 숫자로서 인덱스(index)라고 부른다.수열이 아주 길거나 수열의 길이가 숫자가 아닌 문자인 경우에는 … (dots) 기호를 사용하여 다음처럼 가운데 부분을 생략할 수 있다.

$$x_1, x_2, \ldots, x_N$$

Unit 4

수열과 집합의 합과 곱

❖ 그리스 문자의 시그마(Σ)와 파이(Π)를 본따서 만든 기호지만 시그마와 파이로 읽지 않고 영어로 썸(sum), 프로덕트(product)라고 읽는다.

❖ 수열의 합과 곱

$$\sum_{i=1}^{N} x_i = x_1 + x_2 + \cdots + x_N$$

$$\prod_{i=1}^{N} x_i = x_1 \cdot x_2 \cdot \cdots \cdot x_N$$

❖ 집합의 합과 곱

✓ 만약 집합 X 의 원소가 다음과 같다면,

$$X = \{x_1, x_2, x_3\}$$

$$\sum_{X} x = x_1 + x_2 + x_3$$

$$\prod_{X} x = x_1 \cdot x_2 \cdot x_3$$

Unit 4

순열과 조합

정의 13

서로 다른 n개의 원소에서 r개를 선택하여 순서를 고려하여 일렬로 배열하는 것을 n개에서 r개를 택하는 순열이라 하고, 이 순열의 경우의 수를 순열의 수라고 한다.

기호 : 순열의 수 = ${}_nP_r$ 또는 $P(n,r)$

정리 5

서로 다른 n개의 원소에서 r개를 선택하는 순열의 수는

$$ {}_nP_r = n(n-1)(n-2)\cdots(n-(r-1)),\ (0 \le r \le n) $$

이다.

(1) ${}_nP_r = \dfrac{n!}{(n-r)!}$, n, r은 정수, $0 \le r \le n$

(2) $0! = 1$ (3) ${}_nP_n = n!$ (4) ${}_nP_0 = 1$

Unit 4

순열과 조합

정의 14

서로 다른 n개의 원소에서 순서를 고려하지 않고 r개를 선택하는 것을 n개에서 r개를 택하는 조합이라 하고, 이 조합의 경우의 수를 조합의 수라고 한다.

기호 : 순열의 수 = ${}_nC_r$ 또는 $C(n,r)$ 또는 $\binom{n}{r}$

정리 6

서로 다른 n개의 원소에서 r개를 선택하는 조합의 수는

$$ {}_nC_r = \frac{{}_nP_r}{r!} = \frac{n!}{r!(n-r)!}, \quad (0 \le r \le n) $$

이다.

(1) ${}_nC_r = {}_nC_{n-r}$, n, r은 정수, $0 \le r \le n$

(2) ${}_nC_r = {}_{n-1}C_r + {}_{n-1}C_{r-1}$

(3) ${}_nC_n = 1$

(4) ${}_nC_0 = 1$

Unit 4

순열과 조합

정의 15

집합에서 원소들의 반복이 허용된 집합을 중복집합이라 하고, 중복집합에서 원소의 중복된 횟수를 중복수라고 한다.

정의 16

서로 다른 n개의 원소에서 중복을 허락하여 r개를 선택하여 일렬로 배열하는 것을 n개의 원소에서 r개를 택하는 중복순열이라 하고, 이 순열의 경우의 수를 중복순열의 수라고 한다.

기호 : 중복순열의 수 = ${}_n\Pi_r$ 또는 $\Pi(n,r)$

정리 7

서로 다른 n개의 원소에서 중복을 허락하여 r개를 선택하는 중복순열의 수는

$$ {}_n\Pi_r = n^r \qquad (0 \le r \le n) $$

이다.

Unit 4

순열과 조합

정의 17

전체 n개의 원소에 같은 것이 각각 n_1개, n_2개, $\cdots$, n_m개 있을 때, 전체 n개를 모두 일렬로 배열하는 방법을 같은 것을 포함하는 순열이라 하고, 이 순열의 경우의 수를 같은 것을 포함하는 순열의 수라고 한다.

정리 8

각각 n_1개, n_2개, $\cdots$, n_m개씩의 같은 것을 포함하고 있는 전체 n개의 원소를 일렬로 배열하는 같은 것을 포함하는 순열의 수는

$$\frac{n!}{n_1!n_2!\cdots n_m!}, \qquad (n_1+n_2+\cdots+n_m=n)$$

이다.

Unit 4

순열과 조합

정의 18

서로 다른 n개의 원소에서 중복을 허락하여 r개를 선택하는 것을 n개에서 r개를 택하는 중복조합이라 하고, 이 조합의 경우의 수를 중복조합의 수라고 한다.

기호 : 중복조합의 수 = ${}_nH_r$ 또는 $H(n,r)$

정리 9

서로 다른 n개에서 중복을 허락하여 r개를 택하는 중복조합의 수는

$$ {}_nH_r = {}_{n+r-1}C_r = {}_{n+r-1}C_{n-1} $$

Unit 4

순열과 조합

순열 $${}_n\mathrm{P}_r = \frac{n!}{(n-r)!}$$

중복순열 $${}_n\Pi_r = n^r$$

조합 $${}_n\mathrm{C}_r = \frac{n!}{(n-r)!r!}$$

중복조합 $${}_n\mathrm{H}_r = {}_{n+r-1}\mathrm{C}_r$$

	수식	결과		수식	결과
순열	=PERMUT(4,2)	12	중복순열	=PERMUTATIONA(4,2)	16
조합	=COMBIN(4,2)	6	중복조합	=COMBINA(4,2)	10

Unit 4

추출과 경우의 수

◎ n개의 개체 중 r개를 추출할 때의 경우의 수.

- 복원 추출일 때(즉 중복허용) :

  순서표본의 개수 = n^r

  비순서표본의 개수 = $\binom{n-1+r}{n-1} = \binom{n-1+r}{r}$

- 비복원 추출일 때(즉 중복불허):

  순서표본의 개수 = ${}_nP_r$

  비순서표본의 개수 = $\binom{n}{r}$

◎ n개의 개체가 각각 n_1, n_2, …, n_s개로 이루어진 s개의 동일개체 집단으로 되어 있는 경우. 비복원 추출 시 비순서표본의 가지 수는

- n개를 추출할 때 서로 다른 조합의 개수= $\binom{n}{n_1, n_2, \cdots, n_s} = \frac{n!}{n_1!\, n_2! \cdots n_s!}$
- s개의 집단에서 각각 r_1, r_2, …, r_s개를 독립적으로 비복원 추출 할 때 서로 다른 조합의 개수 = $\binom{n_1}{r_1} \times \binom{n_2}{r_2} \times \cdots \times \binom{n_s}{r_s}$

샘플링

❖ 이미 있는 데이터 집합에서 일부를 무작위로 선택하는 것을 표본선택 혹은 샘플링(sampling)이라고 한다. 샘플링에는 choice 함수를 사용한다. choice 함수는 다음과 같은 인수를 가질 수 있다.

❖ numpy.random.choice(a, size=None, replace=True, p=None)

❖ a : 배열이면 원래의 데이터, 정수이면 arange(a) 명령으로 데이터 생성
❖ size : 정수. 샘플 숫자
❖ replace : 불리언. True이면 한번 선택한 데이터를 다시 선택 가능
❖ p : 배열. 각 데이터가 선택될 수 있는 확률

샘플링

```
np.random.choice(5, 5, replace=False)  # shuffle 명령과 같다.
```

```
array([0, 1, 3, 4, 2])
```

```
np.random.choice(5, 3, replace=False)  # 3개만 선택
```

```
array([4, 0, 1])
```

```
np.random.choice(5, 10)  # 반복해서 10개 선택
```

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

```
np.random.choice(5, 10, p=[0.1, 0, 0.3, 0.6, 0])  # 선택 확률을 다르게 해서 10개 선택
```

```
array([3, 0, 2, 0, 3, 3, 3, 2, 2, 3])
```

샘플링

비복원추출

```
np.random.choice(10, 5, replace=False)
```

```
array([1, 6, 5, 4, 2])
```

```
[73] np.random.choice(10, 5, replace=False)
```

```
array([7, 1, 2, 8, 6])
```

복원추출

```
[71] np.random.choice(10, 5)
```

```
array([8, 8, 2, 3, 2])
```

```
np.random.choice(10, 5, replace=True)
```

```
array([9, 8, 0, 8, 5])
```

2

과학기술계산패키지 소개

1. scipy

Unit 1

scipy

scipy란?

개요

❖ 과학 기술 계산용 함수 및 알고리즘을 제공한다.

❖ scipy subpackages

- ✓ scipy.stats
  - 통계 Statistics
- ✓ scipy.constants
  - 물리/수학 상수 Physical and mathematical constants
- ✓ scipy.special
  - 수학 함수 Any special mathematical functions
- ✓ scipy.linalg
  - 선형 대수 Linear algebra routines
- ✓ scipy.interpolate
  - 보간 Interpolation
- ✓ scipy.optimize
  - 최적화 Optimization
- ✓ scipy.fftpack
  - Fast Fourier transforms → 음성(소리 신호 처리)

scipy.stats 통계

- Random Variable(확률 변수)
  - 확률 밀도 함수, 누적 확률 함수
  - 샘플 생성
  - Parameter Estimation (fitting)

- Common Method

명령어	설명
rvs	샘플 생성
pdf	확률 밀도 함수(연속형)
pmf	확률 질량 함수(이산형)
cdf	누적 확률 분포 함수
stats	평균, 분산, 왜도, 첨도 등 기본 통계
moment	non-central moments of the distribution
fit	모수 추정

scipy.stats 통계

❖ Common Parameters

명령어	설명
random_state	샘플 생성시 사용되는 시드(seed)값
size	생성하려는 샘플의 shape
loc	일반적으로 평균의 값
scale	일반적으로 표준편차의 값

scipy.stats 통계

```
# scipy.stats를 사용하려면 아래 seaborn과 함께 import를 해주어야만 실행이 된다!

import scipy as sp
import seaborn as sns
import numpy as np
import matplotlib as mpl
import matplotlib.pylab as plt
%matplotlib inline
```

```
# random variable 사용 방법

rv = sp.stats.norm(loc=10, scale=10)
rv.rvs(size=(3,10), random_state=1)
```

```
array([[ 26.24345364,   3.88243586,   4.71828248,  -0.72968622,
         18.65407629, -13.01538697,  27.44811764,   2.38793099,
         13.19039096,   7.50629625],
       [ 24.62107937, -10.60140709,   6.77582796,   6.15945645,
         21.33769442,  -0.99891267,   8.27571792,   1.22141582,
         10.42213747,  15.82815214],
       [ -1.00619177,  21.4472371 ,  19.01590721,  15.02494339,
         19.00855949,   3.16272141,   8.77109774,   0.64230566,
          7.3211192 ,  15.30355467]])
```

scipy.constants 상수

❖ 특별 상수
- scipy.pi

❖ 기타 상수
- scipy.constants.XXXX

❖ 단위
- yotta, zetta, exa, peta, tera, giga, mega, kilo, hecto, deka
- deci, centi, milli, micro, nano, pico, femto, atto, zepto
- lb, oz, degree
- inch, foot, yard, mile, au, light_year, parsec
- hectare, acre, gallon
- mph, mach, knot

```
sp.pi
```

```
3.141592653589793
```

```
import scipy.constants
```

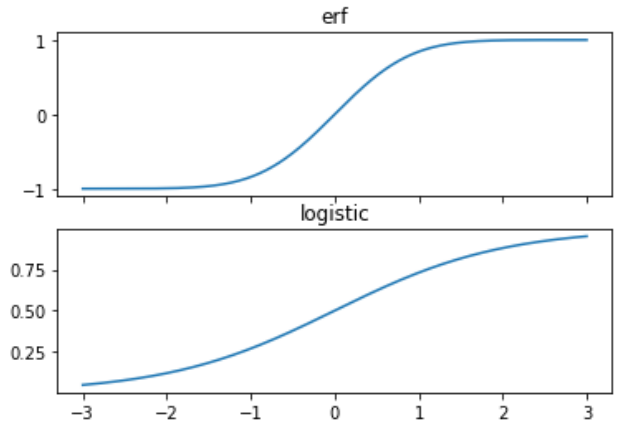
```
# speed of light (광속의 상수)
sp.constants.c
```

```
299792458.0
```

scipy.special 수학 함수

- ❖ Gamma, Beta, Erf, Logit
- ❖ Bessel, Legendre

```
x = np.linspace(-3, 3, 1000)

# erorr function
y1 = sp.special.erf(x)
a = plt.subplot(211)
plt.plot(x, y1)
plt.title("erf")
# x_tick None
a.xaxis.set_ticklabels([])

# logistic
y2 = sp.special.expit(x)
plt.subplot(212)
plt.plot(x, y2)
plt.title("logistic")
plt.show()
```



scipy.linalg 선형대수

- Gamma, Beta, Erf, Logit
- Bessel, Legendre

```
# inverse(역행렬)
A = np.array([[1, 2],
              [3, 4]])
sp.linalg.inv(A)
```

```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

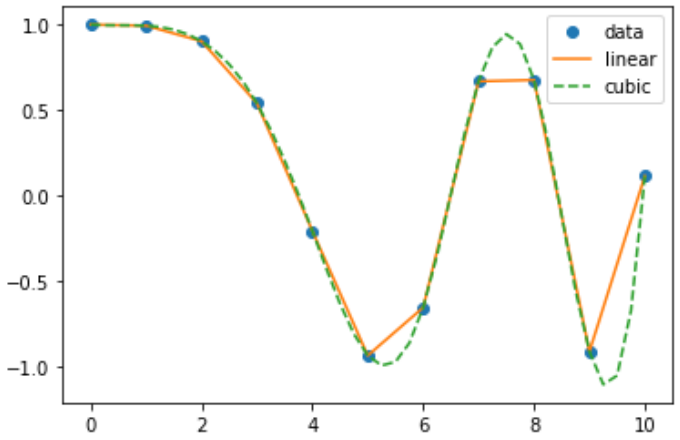
```
# determinant
sp.linalg.det(A)
```

```
-2.0
```

scipy.interpolate 보간

❖ 자료 사이의 빠진 부분을 유추
- 1차원 보간
- 2차원 보간

```
# interp1d → 1차원 보간 / interp2d → 2차원 보간
from scipy.interpolate import interp1d

x = np.linspace(0, 10, num=11, endpoint=True)
y = np.cos(-x**2/9.0)

f = interp1d(x, y)
# kind='cubic' → 선형보다 부드럽게
f2 = interp1d(x, y, kind='cubic')

xnew = np.linspace(0, 10, num=41)
plt.plot(x, y, 'o', xnew, f(xnew), '-', xnew, f2(xnew), '--')
plt.legend(['data', 'linear', 'cubic'])
plt.show()
```



scipy.optimize 최적화

❖ 함수의 최소값 찾기

```
from scipy import optimize

# x=4부터 시작해서 최소값을 찾아라
result = optimize.minimize(f, 4)
print(result)
x0 = result['x']
x0
```

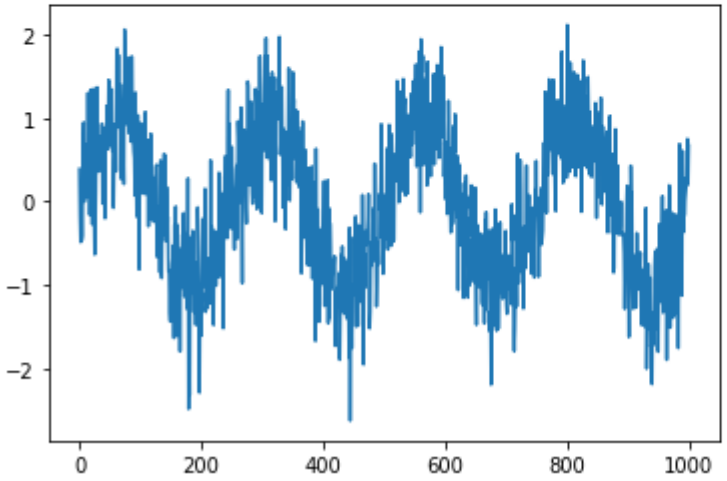
```
      fun: 8.315585579478032
 hess_inv: array([[0.118692]])
      jac: array([3.09944153e-06])
  message: 'Optimization terminated successfully.'
     nfev: 15
      nit: 3
     njev: 5
   status: 0
  success: True
        x: array([3.83746748])
array([3.83746748])
```

scipy.fftpack 고속 퓨리에 변환 Fast Fourier transforms

❖ 신호를 주파수(frequency)영역으로 변환
- 소리 하나를 변환

❖ 스펙트럼(spectrum)

```
time_step = 0.02
period = 5.

time_vec = np.arange(0, 20, time_step)

sig = np.sin(2 * np.pi / period * time_vec) + 0.5 * np.random.randn(time_vec.size)
plt.plot(sig)
plt.show()
```



Unit 3

sympy

심볼릭 연산이란?

개요

- 심볼릭 연산의 장점
- 뉴럴 네크워크와 같이 여러 단계의 계산을 거치는 수치 모형에서 최종단의 필요 계산식을 컴퓨터가 직접 유도
- 수치 계산의 최적화 및 오차 감소
- theano, tensorflow
  - 자체적인 심볼릭 연산 기능

심볼릭 연산 사례

사례1

```
import math
math.sqrt(2)
```

```
1.4142135623730951
```

```
import sympy
from sympy import *

init_printing(use_latex='mathjax')
```

```
sympy.sqrt(2)
```

$$\sqrt{2}$$

```
a = Rational(1,2)
b = Rational(1,4)
a + b
```

$$\frac{3}{4}$$

```
x, y = symbols('x y')
expr = x**2 + 2*x*y + y**2
expr
```

$$x^2 + 2xy + y^2$$

사례2

```
# 인수분해
factor(expr)
```

$$(x + y)^2$$

```
# 식 전개
expand((x + y)**3)
```

$$x^3 + 3x^2y + 3xy^2 + y^3$$

```
x, t, z, nu = symbols('x t z nu')
expr = sin(x)*exp(x)
expr
```

$$e^x \sin(x)$$

```
# 함수 expr를 x에 대해 미분해라.
diff(expr, x)
```

$$e^x \sin(x) + e^x \cos(x)$$

```
# x에 대해 적분해라.
diff(expr, x)
integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
```

$$e^x \sin(x)$$

4

데이터분석 패키지 소개

1. pandas

학습목표

- 판다스(Pandas) 패키지를 임포트할 수 있다.
- 시리즈(Series) 클래스에 대해 알 수 있다.
- 데이터프레임(DataFrame) 클래스를 알 수 있다.

Unit 1

판다스

판다스 패키지 임포트

- 대부분의 데이터는 시계열(series)이나 표(table)의 형태로 나타낼 수 있다. 판다스(Pandas) 패키지는 이러한 데이터를 다루기 위한 시리즈(Series) 클래스와 데이터프레임(DataFrame) 클래스를 제공한다.
- 판다스 패키지를 사용하기 위해 우선 임포트를 해야 한다. 판다스 패키지는 pd라는 별칭으로 임포트하는 것이 관례이므로 여기에서도 해당 관례를 따르도록 한다.

```
import pandas as pd
```

시리즈 생성

❖ 시리즈 Series 클래스는 넘파이에서 제공하는 1차원 배열과 비슷하지만 각 데이터의 의미를 표시하는 인덱스(index)를 붙일 수 있다. 데이터 자체는 값(value)라고 한다.

❖ 시리즈 = 값(value) + 인덱스(index)

```
# 인덱스 없이 시리즈 생성
S1=pd.Series(
    range(11,15)
)

S1
```

```
0    11
1    12
2    13
3    14
dtype: int64
```

```
# 인덱스 있는 상태에서 시리즈 생성
S2=pd.Series(
    [1,2,3,4],
    index=["봄","여름","가을","겨울"]
)

S2
```

```
봄     1
여름    2
가을    3
겨울    4
dtype: int64
```

시리즈에 대한 인덱스와 값 조회

❖ 시리즈 Series 클래스는 인덱스(index)를 통해 조회

```
#S1
print(S1.index)
print(S1.values)
```

```
RangeIndex(start=0, stop=4, step=1)
[11 12 13 14]
```

```
#S2
print(S2.index)
print(S2.values)
```

```
Index(['봄', '여름', '가을', '겨울'], dtype='object')
[1 2 3 4]
```

```
# 봄과 겨울에 대한 평균
# 번호
print((S2[0]+S2[3])/2)
# 이름
print((S2["봄"]+S2["겨울"])/2)
```

```
2.5
2.5
```

시리즈에 대한 명명

❖ 시리즈 Series 클래스는 name을 통해 명명

```
# 시리즈에 명명
S2.name="계절정보"

# 인덱스에 대한 명명
S2.index.name="계절"
# S2.values.name="계절코드" 에러남

S2
```

```
계절
봄     1
여름    2
가을    3
겨울    4
Name: 계절정보, dtype: int64
```

데이터의 갱신, 추가, 삭제

❖ 인덱싱을 이용하면 딕셔너리처럼 데이터를 갱신(update)하거나 추가(add)할 수 있다.

```
S2["봄"] = 11
S2
```

```
계절
봄     11
여름     2
가을     3
겨울     4
Name: 계절정보, dtype: int64
```

❖ 데이터를 삭제할 때도 딕셔너리처럼 ``del`` 명령을 사용한다.

```
del S2["봄"]
S2
```

```
계절
여름    2
가을    3
겨울    4
Name: 계절정보, dtype: int64
```

개요

- 행 인덱스(row index, index)
- 열 인덱스(cplumn index, columns)
- 시리즈가 1차원 벡터 데이터에 행방향 인덱스(row index)를 붙인 것이라면 데이터프레임 DataFrame 클래스는 2차원 행렬 데이터에 인덱스를 붙인 것과 비슷하다. 2차원이므로 각각의 행 데이터의 이름이 되는 행 인덱스(row index) 뿐 아니라 각각의 열 데이터의 이름이 되는 열 인덱스(column index)도 붙일 수 있다.

데이터프레임 생성

❖ 데이터프레임을 만드는 방법은 다양하다. 가장 간단한 방법은 다음과 같다.

- ✓ 하나의 열이 되는 데이터를 리스트나 일차원 배열을 준비한다.
- ✓ 이 각각의 열에 대한 이름(라벨)을 키로 가지는 딕셔너리를 만든다.
- ✓ 이 데이터를 DataFrame 클래스 생성자에 넣는다. 동시에 열방향 인덱스는 columns 인수로, 행방향 인덱스는 index 인수로 지정한다.

```
DF1=pd.DataFrame(
    {
        "2015": [9904312, 3448737, 2890451, 2466052],
        "2010": [9631482, 3393191, 2632035, 2431774],
        "2005": [9762546, 3512547, 2517680, 2456016],
        "2000": [9853972, 3655437, 2466338, 2473990],
        "지역": ["수도권", "경상권", "수도권", "경상권"],
        "2010-2015 증가율": [0.0283, 0.0163, 0.0982, 0.0141]
    }
)

DF1
```

	2015	2010	2005	2000	지역	2010-2015 증가율
0	9904312	9631482	9762546	9853972	수도권	0.0283
1	3448737	3393191	3512547	3655437	경상권	0.0163
2	2890451	2632035	2517680	2466338	수도권	0.0982
3	2466052	2431774	2456016	2473990	경상권	0.0141

데이터프레임 생성

```
DF2=pd.DataFrame(
    {
        "2015": [9904312, 3448737, 2890451, 2466052],
        "2010": [9631482, 3393191, 2632035, 2431774],
        "2005": [9762546, 3512547, 2517680, 2456016],
        "2000": [9853972, 3655437, 2466338, 2473990],
        "지역": ["수도권", "경상권", "수도권", "경상권"],
        "2010-2015 증가율": [0.0283, 0.0163, 0.0982, 0.0141]
    },index=["서울","부산","인천","대구"],
    columns=["지역","2000","2005","2010","2015","2010-2015 증가율"]
)

DF2
```

	지역	2000	2005	2010	2015	2010-2015 증가율
서울	수도권	9853972	9762546	9631482	9904312	0.0283
부산	경상권	3655437	3512547	3393191	3448737	0.0163
인천	수도권	2466338	2517680	2632035	2890451	0.0982
대구	경상권	2473990	2456016	2431774	2466052	0.0141

데이터프레임 생성

```
data={
        "2015": [9904312, 3448737, 2890451, 2466052],
        "2010": [9631482, 3393191, 2632035, 2431774],
        "2005": [9762546, 3512547, 2517680, 2456016],
        "2000": [9853972, 3655437, 2466338, 2473990],
        "지역": ["수도권", "경상권", "수도권", "경상권"],
        "2010-2015 증가율": [0.0283, 0.0163, 0.0982, 0.0141]
    }
index=["서울","부산","인천","대구"]
columns=["지역","2000","2005","2010","2015","2010-2015 증가율"]

DF3=pd.DataFrame(data,index=index,columns=columns)

DF3
```

	지역	2000	2005	2010	2015	2010-2015 증가율
서울	수도권	9853972	9762546	9631482	9904312	0.0283
부산	경상권	3655437	3512547	3393191	3448737	0.0163
인천	수도권	2466338	2517680	2632035	2890451	0.0982
대구	경상권	2473990	2456016	2431774	2466052	0.0141

속성확인

❖ *시리즈에서* 처럼 열방향 인덱스와 행방향 인덱스에 이름을 붙이는 것도 가능

```
print(DF3.index)
print(DF3.values)
print(DF3.columns)
```

```
Index(['서울', '부산', '인천', '대구'], dtype='object')
[['수도권' 9853972 9762546 9631482 9904312 0.0283]
 ['경상권' 3655437 3512547 3393191 3448737 0.0163]
 ['수도권' 2466338 2517680 2632035 2890451 0.0982]
 ['경상권' 2473990 2456016 2431774 2466052 0.0141]]
Index(['지역', '2000', '2005', '2010', '2015', '2010-2015 증가율'], dtype='object')
```

```
DF3.index.name="도시"
DF3.columns.name="특성"
DF3
```

특성 / 도시	지역	2000	2005	2010	2015	2010-2015 증가율
서울	수도권	9853972	9762546	9631482	9904312	0.0283
부산	경상권	3655437	3512547	3393191	3448737	0.0163
인천	수도권	2466338	2517680	2632035	2890451	0.0982
대구	경상권	2473990	2456016	2431774	2466052	0.0141

속성확인

❖ 데이터프레임은 전치(transpose)를 포함하여 넘파이 2차원 배열이 가지는 대부분의 속성이나 메서드를 지원한다.

```
DF3.T
```

도시	서울	부산	인천	대구
특성				
지역	수도권	경상권	수도권	경상권
2000	9853972	3655437	2466338	2473990
2005	9762546	3512547	2517680	2456016
2010	9631482	3393191	2632035	2431774
2015	9904312	3448737	2890451	2466052
2010-2015 증가율	0.0283	0.0163	0.0982	0.0141

Unit 3

열 데이터의 갱신, 추가, 삭제

❖ 데이터프레임은 열 시리즈의 딕셔너리으로 볼 수 있으므로 열 단위로 데이터를 갱신하거나 추가, 삭제할 수 있다

```
# "2010-2015 증가율"이라는 이름의 열 추가
DF3["2010-2015 증가율"] = DF3["2010-2015 증가율"] * 100
DF3
```

특성	지역	2000	2005	2010	2015	2010-2015 증가율
도시						
서울	수도권	9853972	9762546	9631482	9904312	2.83
부산	경상권	3655437	3512547	3393191	3448737	1.63
인천	수도권	2466338	2517680	2632035	2890451	9.82
대구	경상권	2473990	2456016	2431774	2466052	1.41

열 데이터의 갱신, 추가, 삭제

❖ 데이터프레임은 열 시리즈의 딕셔너리으로 볼 수 있으므로 열 단위로 데이터를 갱신하거나 추가, 삭제할 수 있다

```
# "2005-2010 증가율"이라는 이름의 열 추가
DF3["2005-2010 증가율"] = ((DF3["2010"] - DF3["2005"]) / DF3["2005"] * 100).round(2)
DF3
```

특성	지역	2000	2005	2010	2015	2010-2015 증가율	2005-2010 증가율
도시							
서울	수도권	9853972	9762546	9631482	9904312	2.83	-1.34
부산	경상권	3655437	3512547	3393191	3448737	1.63	-3.40
인천	수도권	2466338	2517680	2632035	2890451	9.82	4.54
대구	경상권	2473990	2456016	2431774	2466052	1.41	-0.99

열 데이터의 갱신, 추가, 삭제

❖ 데이터프레임은 열 시리즈의 딕셔너리으로 볼 수 있으므로 열 단위로 데이터를 갱신하거나 추가, 삭제할 수 있다

```
# "2010-2015 증가율"이라는 이름의 열 삭제
del DF3["2010-2015 증가율"]
DF3
```

특성	지역	2000	2005	2010	2015	2005-2010 증가율
도시						
서울	수도권	9853972	9762546	9631482	9904312	-1.34
부산	경상권	3655437	3512547	3393191	3448737	-3.40
인천	수도권	2466338	2517680	2632035	2890451	4.54
대구	경상권	2473990	2456016	2431774	2466052	-0.99

열 인덱싱

```
DF3["지역"]
```

```
도시
서울      수도권
부산      경상권
인천      수도권
대구      경상권
Name: 지역, dtype: object
```

```
type(DF3["지역"])
```

```
pandas.core.series.Series
```

```
DF3[["지역","2010-2015 증가율"]]
```

특성	지역	2010-2015 증가율
도시		
서울	수도권	0.0283
부산	경상권	0.0163
인천	수도권	0.0982
대구	경상권	0.0141

```
type(DF3[["지역","2010-2015 증가율"]])
```

```
pandas.core.frame.DataFrame
```

```
# 하나의 열만 인덱싱하면서 데이터 프레임 자료형을 유지하고자 할 때

DF3[["지역"]]
```

특성	지역
도시	
서울	수도권
부산	경상권
인천	수도권
대구	경상권

```
type(DF3[["지역"]])
```

```
pandas.core.frame.DataFrame
```

행 인덱싱

```
DF3[:1]
```

특성	지역	2000	2005	2010	2015	2010-2015 증가율
도시						
서울	수도권	9853972	9762546	9631482	9904312	0.0283

```
DF3[1:2]
```

특성	지역	2000	2005	2010	2015	2010-2015 증가율
도시						
부산	경상권	3655437	3512547	3393191	3448737	0.0163

```
DF3["서울":"부산"]
```

특성	지역	2000	2005	2010	2015	2010-2015 증가율
도시						
서울	수도권	9853972	9762546	9631482	9904312	0.0283
부산	경상권	3655437	3512547	3393191	3448737	0.0163

개별 데이터(도메인 값) 인덱싱

```
DF3[-2:]
```

특성 도시	지역	2000	2005	2010	2015	2010-2015 증가율
인천	수도권	2466338	2517680	2632035	2890451	0.0982
대구	경상권	2473990	2456016	2431774	2466052	0.0141

데이터(도메인 값) 인덱싱

```
# 서울에서 2010년과 2015년 데이터에 대한 평균

(DF3["2010"]["서울"]+DF3["2015"]["서울"])/2
```

```
9767897.0
```

PC에서 파일을 선택하여 업로드 하는 방법

••

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

```
파일 선택 Level1-2_1_data.csv
• Level1-2_1_data.csv(application/vnd.ms-excel) - 48 bytes, last modified: 2017. 10. 25. - 100% done
Saving Level1-2_1_data.csv to Level1-2_1_data.csv
User uploaded file "Level1-2_1_data.csv" with length 48 bytes
```

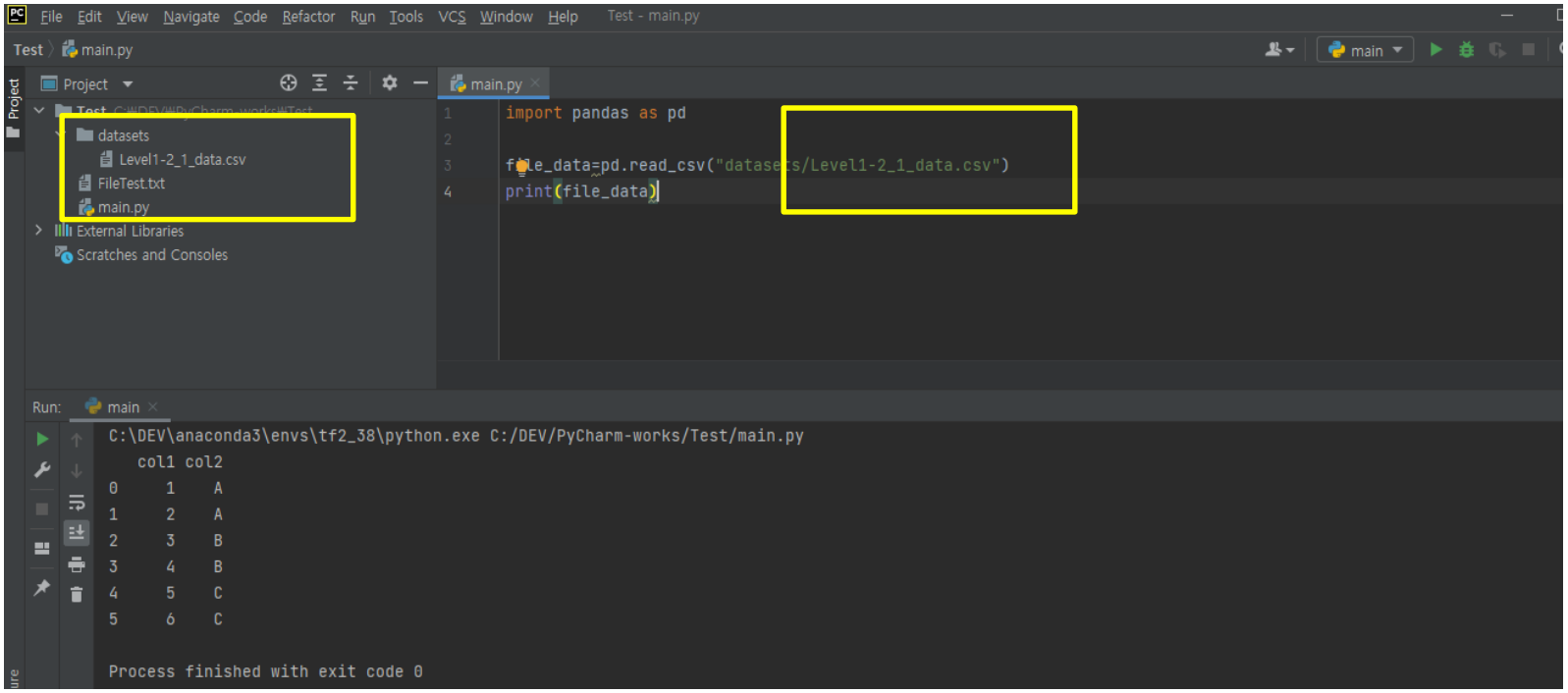
```
import pandas as pd

file_data=pd.read_csv("Level1-2_1_data.csv")
print(file_data)
```

```
   col1 col2
0     1    A
1     2    A
2     3    B
3     4    B
4     5    C
5     6    C
```

Unit 3

PC에서 파일을 선택하여 업로드 하는 방법

파이참 환경에서 작업할 때는, 직접 폴더에 배치한다.



Unit A

참고자료

참고자료

- http://www.ncs.go.kr
- NELLDALE/JOHN LEWIS지음, 조영석/김대경/박찬영/송창근 역, 단계별로 배우는컴퓨터과학, 홍릉과학출판사,2018
- 혼자 공부하는 머신러닝+딥러닝 박해선 지음 | 한빛미디어 | 2020년 12월
- 머신러닝 실무 프로젝트 ,아리가 미치아키, 나카야마 신타, 니시바야시 다카시 지음 | 심효섭 옮김 | 한빛미디어 | 2018년 06월
- 파이썬을 활용한 머신러닝 쿡북 크리스 알본 지음 | 박해선 옮김 | 한빛미디어 | 2019년 09월
- 처음 배우는 머신러닝 김의중 지음 | 위키북스 | 2016년 07월
- 파이썬으로 배우는 머신러닝의 교과서 : 이토 마코토 지음 | 박광수(아크몬드) 옮김 | 한빛미디어 | 2018년 11월
- 기타 서적 및 웹 사이트 자료 다수 참조

나눔고딕 인스톨

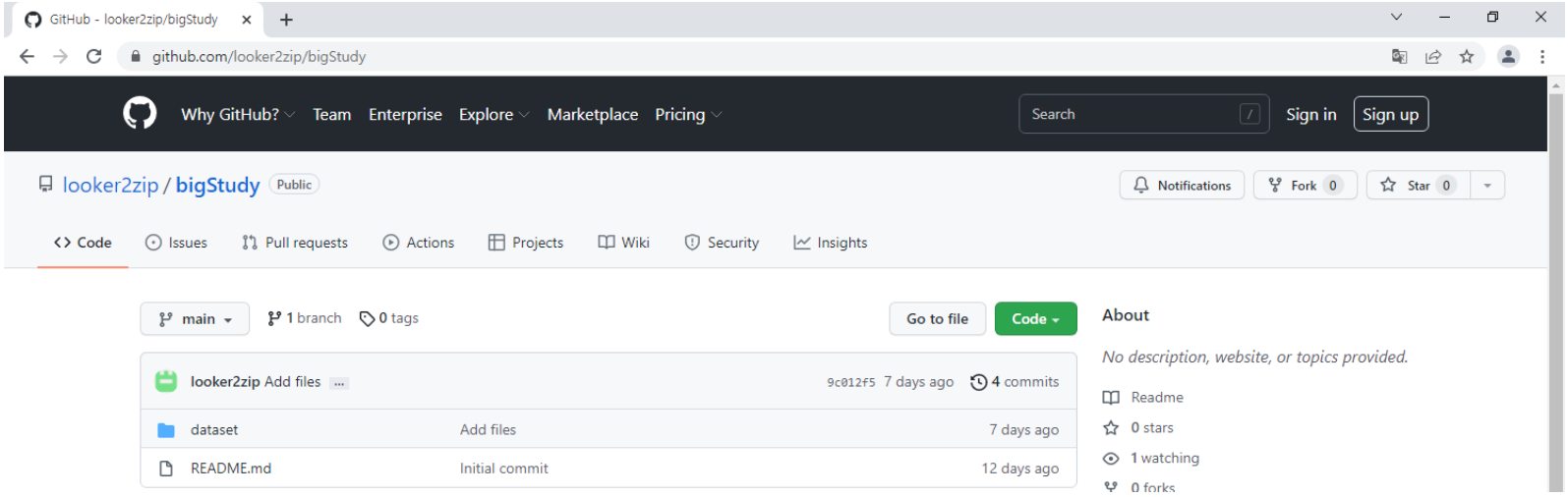
❖ 리눅스 다운로드

!rm -rf bigStudy

!git clone 'https://github.com/looker2zip/bigStudy.git'

❖ 윈도 다운로드

https://github.com/looker2zip/bigStudy



참고자료

나눔고딕 인스톨

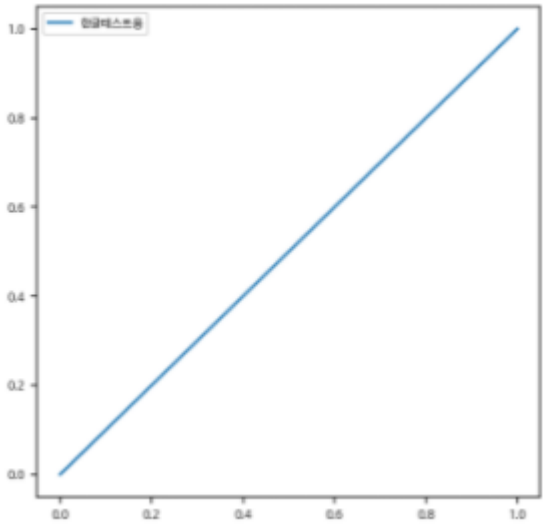
```
%config InlineBackend.figure_format = 'retina'
 !apt -qq -y install fonts-nanum
```

❖ 런타임 다시 시작

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
fontpath = '/usr/share/fonts/truetype/nanum/NanumBarunGothic.ttf'
font = fm.FontProperties(fname=fontpath, size=9)
plt.rc('font', family='NanumBarunGothic')
mpl.font_manager._rebuild()


plt.figure(figsize=(5,5))
plt.plot([0,1], [0,1], label='한글테스트용')
plt.legend()
plt.show()
```

나눔고딕 인스톨

```
plt.figure(figsize=(5,5))
plt.plot([0,1], [0,1], label='한글테스트용')
plt.legend()
plt.show()
```



- Mobile: 010-9591-1401
- E-mail: onlooker2zip@naver.com