

Python Data Analysis

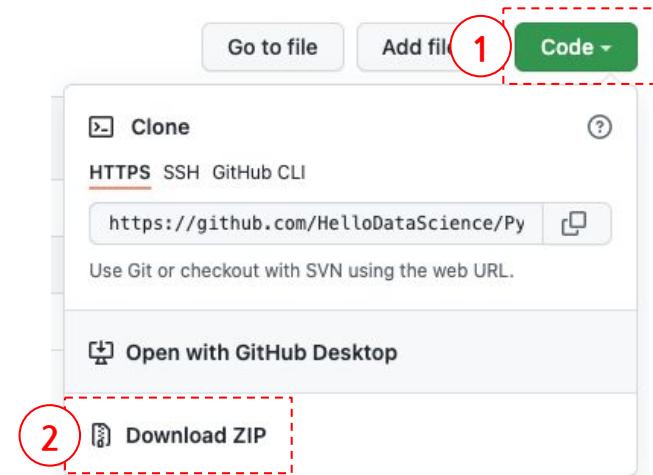
NH농협

강의 내용

- 프로그래밍 시작
- 자료형
- 자료구조
- 제어문
- 사용자 정의 함수
- numpy 라이브러리
- pandas 라이브러리
- 데이터 입출력
- 데이터 전처리
- 데이터 시각화
- 탐색적 데이터 분석
- 기술통계 분석
- 확률분포의 이해
- 가설검정의 이해
- 데이터 분석 개요
- 선형 회귀분석

실습 데이터셋 내려받기

- 크롬에서 깃허브 저장소로 접속합니다. [주의] 크롬 아니면 에러 발생 가능!
 - URL: <https://github.com>HelloDataScience/NH>
- 초록색 Code와 Download ZIP을 차례대로 클릭하면 zip 파일을 다운로드 폴더에 저장합니다.
- zip 파일을 적당한 위치(예: 문서 폴더)에서 압축을 풀고 code와 data 폴더를 확인합니다.
 - code: Jupyter Notebook 파일을 포함하는 폴더입니다.
 - data: 실습 데이터 파일을 포함하는 폴더입니다.



프로그래밍 시작

Python이란?

- Python은 네덜란드 출신의 컴퓨터 프로그래머이자 구글, 드롭박스를 거쳐 현재는 MS 개발자인 **귀도 반 로섬**^{Guido Van Rossum}이 1991년에 발표한 프로그래밍 언어입니다.
 - 1989년 12월, 귀도는 취미가 될만한 프로젝트를 찾고 있었고 마침 크리스마스 휴일이라 사무실 문을 닫으면서 집에 있는 컴퓨터로 Python 개발에 착수하였다고 전해집니다.
- Python이라는 이름은 당시 귀도가 좋아하던 코미디 쇼였던 "**Monty Python's Flying Circus**"에서 따온 것입니다.
 - Python(피تون)은 그리스 신화에서 나오는 큰 뱀입니다.
 - Python 로고는 뱀 모양입니다.
 - Python의 버전은 2와 3이 있습니다.



Python의 특징

- Python은 인간다운 언어이므로 사람이 생각하는 방식으로 프로그래밍합니다.
 - 영어를 모국어로 하는 사람들에게 해당합니다. 최소한 한글은 아니니까요.
- Python은 문법이 쉬워 빠르게 배울 수 있습니다.
 - 프로그래밍에 익숙하다면 일주일이면 충분하겠지만, 초심자에게는 절대로 아닙니다.
- Python은 오픈소스이므로 무료고, 수많은 라이브러리를 사용할 수 있습니다.
 - 라이브러리 설치 에러가 발생하고, 일관성이 부족한 것도 오픈소스의 특징입니다.
- Python은 간결하고, 들여쓰기 규칙으로 코드 가독성을 높일 수 있습니다.

출처: 점프 투 파이썬(<https://wikidocs.net/6>)에서 일부 발췌

Python의 특징

Life is too short, you need Python!

기본 구문: 코드 입력 및 실행

- Python을 실행하면 콘솔이 열리고, 콘솔 왼쪽에 >>> 기호(프롬프트)가 있습니다.
 - 프롬프트 오른쪽에서 커서^{cursor}가 깜빡거립니다.
- Python은 스크립트 언어이므로 콘솔에서 코드를 입력하고 [enter] 키를 누르면 코드 실행 결과를 코드 아래에 출력합니다.[대화식 모드]
- 여러 줄 코드를 입력하고 실행하려면 IDLE 편집기를 이용합니다.[스크립트 모드]
 - IDLE 편집기에서 [enter] 키를 누르면 코드를 실행하지 않고 커서를 아래로 옮깁니다.
 - 실행할 코드를 py 파일로 저장하고 상단 메뉴에서 Run → Run Module 버튼을 차례대로 클릭하면 py 파일을 실행합니다.

이번 강의에서는 Jupyter Notebook을 이용한 라이브 코딩 실습 방식으로 진행합니다.

Jupyter Notebook 메인화면

The screenshot shows the Jupyter Notebook interface. On the left is a file browser with a sidebar containing 'Applications', 'Desktop', 'Documents', 'Downloads', 'Movies', 'Music', 'Pictures', and 'Public' folders. The current path is shown as '/'. A red dashed arrow points from the text '# Jupyter Notebook의 시작 위치는 User 폴더입니다.' to the root directory icon in the file browser. On the right, a 'New' dropdown menu is open, showing options for 'Notebook:', 'Python 3' (which is highlighted with a red box), 'R', 'Other:', 'Text File', 'Folder', and 'Terminal'. A red dashed arrow points from the text '# 새 Python Jupyter Notebook을 열려면 New → Python 3를 차례대로 선택합니다.' to the 'Python 3' option in the dropdown. At the bottom left, there is a note in red: '# [참고] Jupyter Notebook을 열었을 때 500 : Internal Server Error가 발생하면 아래 코드를 실행하세요. % pip install --upgrade jupyter'.

Jupyter Notebook의 시작 위치는 User 폴더입니다.

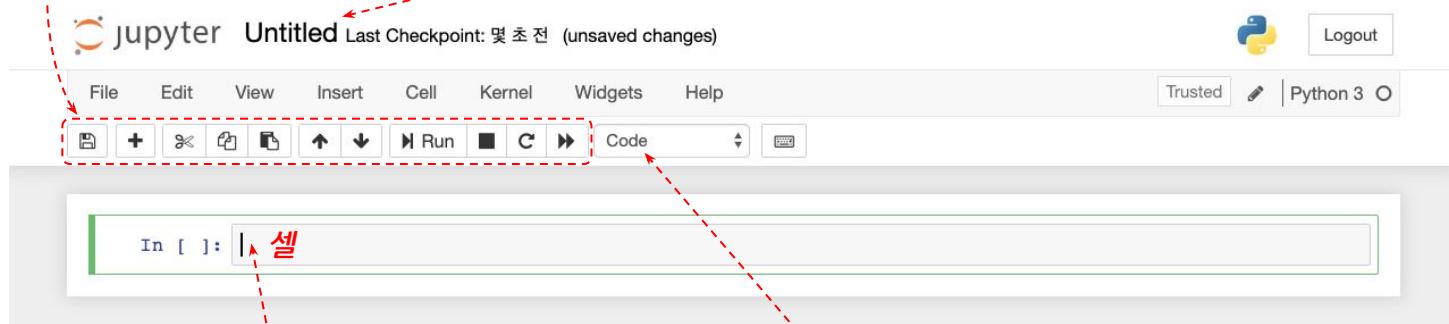
새 Python Jupyter Notebook을 열려면 New → Python 3를 차례대로 선택합니다.

[참고] Jupyter Notebook을 열었을 때 500 : Internal Server Error가 발생하면 아래 코드를 실행하세요.
% pip install --upgrade jupyter

Jupyter Notebook 메뉴

저장, 셀(Cell) 추가, 잘라내기, 복사, 붙여넣기 및 코드 실행 메뉴입니다.

Untitled를 클릭하면 Rename Notebook 팝업이 뜨는데 Jupyter Notebook의 파일명을 변경할 수 있습니다.



코드와 주석을 셀에 입력합니다.
[참고] 주석은 # 기호로 시작합니다.

코드를 실행하려면 ▶ 버튼을 클릭하거나
[shift] + [enter] 키를 누릅니다.

셀은 편집모드와 명령모드가 있습니다.
편집모드일 때 코드를 입력할 수 있습니다.

셀을 세 가지 형태^{type}로 변경할 수 있습니다.
- Code는 주석과 코드를 입력할 수 있습니다.
- Markdown은 마크다운 문서로 작성할 수 있습니다.
- Raw는 코드를 원형으로 보관할 때 사용합니다.

[참고] 마크다운은 텍스트 기반의 마크업 언어이며
HTML 또는 PDF로 쉽게 변환할 수 있습니다.

Jupyter Notebook 셀 모드

- 편집모드는 셀 테두리 색이 초록색으로 활성화된 상태입니다.
 - 편집모드일 때, 셀에 코드와 주석을 입력하고 실행합니다.
- 명령모드는 셀 테두리 색이 파란색으로 비활성화된 상태입니다.
 - 명령모드일 때, 단축키로 셀 관련 작업을 실행합니다.

편집모드

- ① 키보드에서 `esc` 키를 누릅니다.
② 마우스로 셀 바깥쪽을 클릭합니다.

명령모드

- ① 키보드에서 `enter` 키를 누릅니다.
② 마우스로 셀 안쪽을 클릭합니다.

Jupyter Notebook 주요 단축키(명령모드에서 실행)

단축키	동작	단축키	동작
[y]	셀을 Code로 변환	[a], [b]	현재 셀 위, 아래로 셀 추가
[m]	셀을 Markdown으로 변환	[x]	선택한 셀 잘라내기
[r]	셀을 Raw로 변환	[c]	선택한 셀 복사
[1] ~ [6]	셀을 Headings 1~6으로 변환	[v]	선택한 셀 붙여넣기
[shift] + [enter]	코드 실행 후 아래 셀로 이동	[d] + [d]	선택한 셀 삭제
[ctrl] + [enter]	코드 실행 후 현재 셀에 멈춤	[z]	삭제한 셀 되돌리기(undo)
[alt] + [enter]	코드 실행 후 아래에 셀 추가	[f]	패턴 찾기 및 바꾸기
[↑], [↓]	현재 셀을 위, 아래로 이동	[l]	셀마다 행 숫자 보이기/감추기
[shift] + [↑], [↓]	현재 셀부터 위, 아래로 확장	[h]	단축키 목록 보이기

Jupyter Notebook 매직 명령어

- 매직 명령어^{magic commands}는 Jupyter Notebook에서 사용할 수 있는 명령어입니다.

명령어	동작	명령어	동작
%lsmagic	사용 가능한 매직 명령어 출력	%precision	실수를 소수점 최대한 길게 출력
%pwd	현재 작업 경로 출력	%precision 3	실수를 소수점 셋째 자리까지 출력
%cd '경로'	작업 경로 변경	%precision %i	실수에서 정수 부분만 출력
%ls	현재 작업 경로의 파일명 출력	%precision %e	실수를 과학적 표기법으로 출력
%who_ls	전역 변수 목록을 리스트로 출력	%time	코드 실행 소요시간 출력
%whos	전역 변수 목록을 표로 출력	%timeit	코드 실행 평균 소요시간 출력
%reset	전역 변수 일괄 삭제	%matplotlib	matplotlib 라이브러리 관련 설정

[참고] 키보드의 주요 키 key 위치

[~] 틸데 tilde
[`] 백틱 backtick

[#] 샵 sharp

[^] 캐럿 caret
[&] 앤퍼샌드 ampersand

[|] 바 bar
[\] 백슬래시 backslash



출처: <https://m.post.naver.com/viewer/postView.nhn?volumeNo=17962736&memberNo=11534881>

변수와 객체

- Python 프로그래밍을 할 때 변수^{variable}와 객체^{object}의 개념을 알아야 합니다.
 - 변수에 어떤 값을 할당^{assign}하면, 값으로 객체를 생성하고 변수는 객체를 가리킵니다.
 - 변수가 가리키는 객체는 바뀔 수 있습니다.
 - 값을 변수에 할당하지 않으면, 값이 바뀔 때마다 매번 직접 코딩해야 합니다.
 - 하지만 변수에 할당하면 값이 바뀌더라도 기존 코드를 재사용할 수 있습니다.

```
>>> 1 + 2  
>>> 2 + 2  
>>> 3 + 2
```



```
>>> a = 1  
>>> a = 2  
>>> a = 3
```



```
>>> a + 2  
>>> a + 2  
>>> a + 2
```

값을 직접 코딩

값을 변수에 할당

기존 코드 재사용

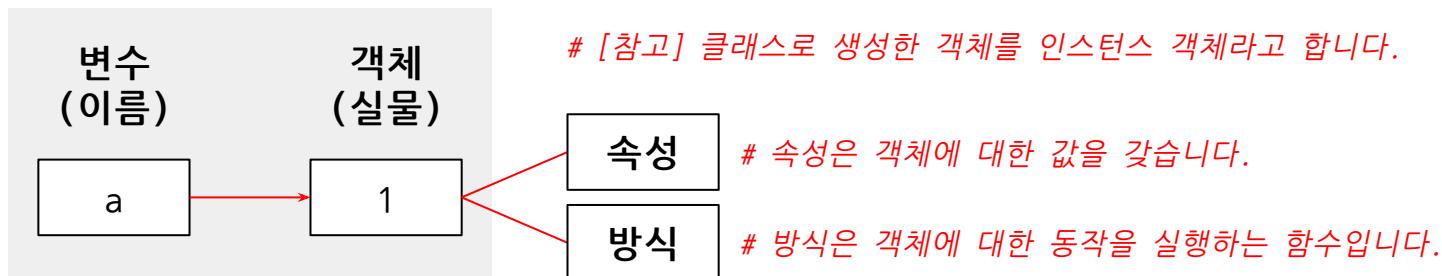
변수명

- 변수명은 알파벳과 숫자, 밑줄 underscore을 조합하여 만듭니다.
 - 변수명은 알파벳 또는 밑줄로 시작해야 하며, 숫자로 시작하면 에러를 반환합니다.
 - 변수명은 알파벳 대소문자를 구분합니다.
 - 변수명에 한글을 사용하는 것은 추천하지 않습니다.
- 변수명으로 예약어 reserved keywords를 사용할 수 없습니다.

```
>>> help('keywords') # Python 예약어를 확인합니다.
```
- 내장 함수명도 피하는 것이 좋습니다. # [참고] `print()`, `type()`, `dir()` 등이 Python 내장 함수이며, 아래 링크에서 71개 내장 함수 목록을 확인할 수 있습니다.
 - 링크: <https://docs.python.org/3/library/functions.html>

객체의 특징

- Python에서 변수에 어떤 값을 할당하면 값의 자료형에 따라 객체를 생성합니다.
객체는 메모리에 할당되며 변수는 객체가 할당된 메모리 주소를 저장합니다.
 - 변수는 객체가 할당된 메모리 주소로 객체를 연결하므로 객체가 바뀌면 주소도 바뀝니다.
- 객체는 클래스^{class}로 생성합니다. # [참고] 많은 책에서 클래스를 봉어빵틀에 비유하여 설명합니다.
[참고] 객체는 봉어빵틀에서 만들어진 봉어빵이라 할 수 있습니다.
 - 클래스는 데이터를 다룰 때 필요한 여러 속성^{attribute}과 방식^{method}을 정의한 것입니다.



객체의 생성과 확인

- 변수에 어떤 값을 할당하여 객체를 생성하고, 변수의 클래스를 확인합니다.

```
>>> a = 10; a # a에 정수 10을 할당하고, 변수명만 실행하면 변수가 가리키는 객체를 출력합니다.  
[참고] 여러 줄 코드 사이에 세미콜론을 추가하면 한 줄로 작성할 수 있습니다.
```

- `print()` 함수는 변수가 가리키는 객체를 표준 출력 위치에 출력합니다.

```
>>> print(a) # [참고] 변수명을 실행한 결과와 다를 수 있습니다. 문자열은 따옴표를 출력하지 않습니다.
```

- `type()` 함수는 변수가 가리키는 객체의 클래스를 반환합니다.

```
>>> type(a) # a의 클래스를 확인합니다. a의 클래스는 int입니다.  
[참고] 같은 클래스 함수로 생성한 객체는 같은 속성과 방식을 갖습니다.
```

- `dir()` 함수는 변수가 갖는 속성과 방식을 리스트로 반환합니다.

```
>>> dir(a) # a에 많은 속성과 방식이 있지만 알파벳으로 시작하는 것만 사용합니다.  
[참고] 밑줄로 시작하는 속성과 방식은 특별한 기능을 갖습니다.
```

객체의 속성과 방식에 접근

- 객체의 속성^{attribute} 또는 방식^{method}에 접근하려면 변수명 뒤에 마침표(온점)를 찍고 접근하려는 속성 또는 방식을 덧붙입니다.

Object.attribute

Object.method()

- 객체의 속성 또는 방식에 접근하는 예시 코드입니다.

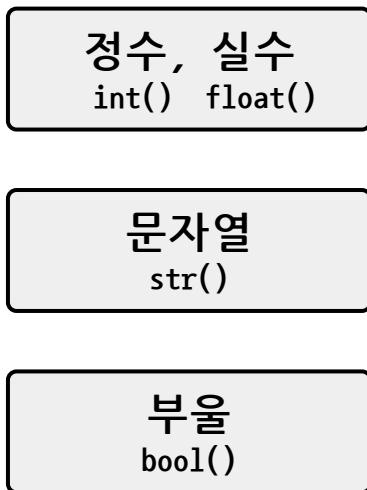
```
>>> a.numerator # numerator는 int의 속성이며 분자를 출력합니다.  
[주의] 객체의 속성 뒤에 괄호를 추가하면 에러를 반환합니다.
```

```
>>> a.denominator # denominator는 int의 속성이며 분모를 출력합니다.
```

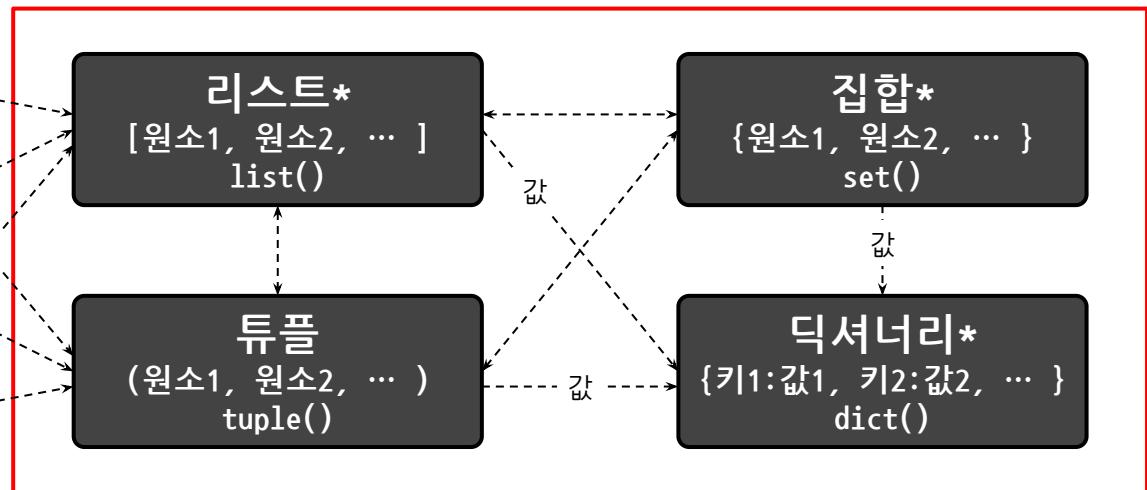
```
>>> a.bit_length() # bit_length는 int의 방식이며 10진수를 2진수로 변환할 때 길이를 반환합니다.  
[주의] 방식 뒤에 괄호를 생략하면 에러를 반환합니다.
```

Python 자료형과 자료구조

자료형: 원소^{Base}



자료구조: 여러 원소를 담는 그릇^{Container}



자료형 변환^{casting}

*인 자료구조는 mutable이므로 원소를 추가, 삭제 또는 변경할 수 있습니다.

자료형

수: 정수, 실수

- 정수^{integer}는 소수점이 없는 자료형입니다.

```
>>> a = 3 # a에 정수 3을 할당합니다.
```

```
>>> a # a를 출력합니다.
```

```
>>> type(a) # a의 클래스를 확인합니다. a의 클래스는 int입니다.
```

- 실수^{float}는 소수점이 있는 자료형입니다.

```
>>> b = 3.0 # b에 실수 3.0을 할당합니다.
```

```
>>> b # b를 출력합니다.
```

```
>>> type(b) # b의 클래스를 확인합니다. b의 클래스는 float입니다.
```

산술 연산자

- 정수 또는 실수로 산술 연산을 실행합니다.

연산자	상세 내용
$a + b$	<ul style="list-style-type: none">a와 b를 더합니다. # [참고] $+a$는 단항 덧셈 연산자로 양의 부호를 의미합니다.
$a - b$	<ul style="list-style-type: none">a에서 b를 뺍니다. # [참고] $-a$는 단항 뺄셈 연산자로 음의 부호를 의미합니다.
$a * b$	<ul style="list-style-type: none">a와 b를 곱합니다.
$a ** b$	<ul style="list-style-type: none">a를 b로 거듭제곱합니다.
a / b	<ul style="list-style-type: none">a를 b로 나눕니다.
$a \% b$	<ul style="list-style-type: none">a를 b로 나눈 나머지를 반환합니다.
$a // b$	<ul style="list-style-type: none">a를 b로 나눈 몫을 반환합니다.

산술 연산자

- 수를 가리키는 두 변수로 산술 연산을 실행합니다.

```
>>> a + b # a와 b를 더합니다. [참고] 둘 다 정수면 정수를 반환하고 실수가 섞여 있으면 실수를 반환합니다.
```

```
>>> a - b # a에서 b를 뺍니다.
```

```
>>> a * b # a와 b를 곱합니다.
```

```
>>> a ** b # a를 b로 거듭제곱합니다.
```

```
>>> a / b # a를 b로 나눕니다. [참고] 정수 간 나눗셈을 포함하여 나눗셈 연산은 항상 실수를 반환합니다.
```

```
>>> a % b # a를 b로 나눈 나머지를 반환합니다.
```

```
>>> a // b # a를 b로 나눈 몫을 반환합니다. [참고] 소수점 이하를 절사한 결과와 같습니다.
```

할당 연산자

- 값 또는 산술 연산 결과를 변수에 할당합니다.

연산자	상세 내용
$a = b$	<ul style="list-style-type: none">a에 b를 할당합니다.
$a += b$	<ul style="list-style-type: none">a에 b를 더하고 a에 재할당합니다.
$a -= b$	<ul style="list-style-type: none">a에서 b를 빼고 a에 재할당합니다.
$a *= b$	<ul style="list-style-type: none">a에 b를 곱하고 a에 재할당합니다.
$a **= b$	<ul style="list-style-type: none">a를 b로 거듭제곱하고 a에 재할당합니다.
$a /= b$	<ul style="list-style-type: none">a를 b로 나누고 a에 재할당합니다.
$a %= b$	<ul style="list-style-type: none">a를 b로 나눈 나머지를 a에 재할당합니다.
$a //= b$	<ul style="list-style-type: none">a를 b로 나눈 몫을 a에 재할당합니다.

할당 연산자

- 수를 가리키는 변수로 할당 연산을 실행합니다.

```
>>> a = 5; a # a에 정수 5를 할당하고, a를 출력합니다.
```

```
>>> a += 1; a # a에 1을 더한 값을 a에 재할당하고, a를 출력합니다.
```

```
>>> a -= 2; a # a에서 2를 뺀 값을 a에 재할당하고, a를 출력합니다.
```

```
>>> b *= 3; b # b에 3을 곱한 값을 b에 재할당하고, b를 출력합니다.
```

```
>>> b /= 4; b # b를 4로 나눈 값을 b에 재할당하고, b를 출력합니다.
```

비교 연산자

- 값의 크기를 비교하여 True 또는 False를 반환합니다.

연산자	상세 내용
$a > b$	<ul style="list-style-type: none">a가 b보다 크면 True, 작거나 같으면 False를 반환합니다.
$a \geq b$	<ul style="list-style-type: none">a가 b보다 크거나 같으면 True, 작으면 False를 반환합니다.
$a < b$	<ul style="list-style-type: none">a가 b보다 작으면 True, 크거나 같으면 False를 반환합니다.
$a \leq b$	<ul style="list-style-type: none">a가 b보다 작거나 같으면 True, 크면 False를 반환합니다.
$a == b$	<ul style="list-style-type: none">a가 b와 같으면 True, 다르면 False를 반환합니다.
$a != b$	<ul style="list-style-type: none">a가 b와 다르면 True, 같으면 False를 반환합니다.

[참고] True 또는 False를 진리값 *truth value*라고 합니다.

비교 연산자

- 변수와 상수로 비교 연산을 실행합니다.

```
>>> a > 4 # a가 4보다 크면 True, 작거나 같으면 False를 반환합니다.
```

```
>>> a >= 4 # a가 4보다 크거나 같으면 True, 작으면 False를 반환합니다.
```

```
>>> a < 4 # a가 4보다 작으면 True, 크거나 같으면 False를 반환합니다.
```

```
>>> a <= 4 # a가 4보다 작거나 같으면 True, 크면 False를 반환합니다.
```

```
>>> a == 4 # a가 4이면 True, 4가 아니면 False를 반환합니다. [주의] 등호(=)를 2번 사용합니다.
```

```
>>> a != 4 # a가 4가 아니면 True, 4이면 False를 반환합니다.
```

논리 연산자

- 진리값 사이에서 논리곱, 논리합 또는 논리부정 연산을 실행합니다.

연산자	상세 내용
and	<ul style="list-style-type: none">[논리곱] 양쪽 진리값이 모두 True면 True, 아니면 False를 반환합니다.
or	<ul style="list-style-type: none">[논리합] 양쪽 진리값 중 하나라도 True면 True, 모두 False이면 False를 반환합니다.
not	<ul style="list-style-type: none">[논리부정] True를 False로, False를 True로 반전합니다.

[논리곱]

True		True		True
True	and	False	→	False
False		True		False
False		False		False

[논리합]

True		True		True
True	or	False	→	True
False		True		True
False		False		False

[논리부정]

not	True	→	False
False			True

논리 연산자

- 비교 연산을 실행하여 True 또는 False를 반환합니다.

```
>>> print(a > 3) # 원쪽 두 줄 코드를 같은 셀에 입력하고 실행합니다.  
[참고] 같은 셀에 두 개 이상의 결과를 출력하려면 print() 함수를 실행해야 합니다.
```

```
>>> print(b > 3)
```

- 비교 연산 결과로 논리 연산을 실행합니다.

```
>>> a > 3 and b > 3 # [논리곱] 두 진리값이 모두 True면 True, 아니면 False를 반환합니다.
```

```
>>> a > 3 or b > 3 # [논리합] 두 진리값 중 하나 이상 True면 True, 아니면 False를 반환합니다.
```

```
>>> not (a > 3 or b > 3) # [논리부정] not 뒤에 오는 진리값을 반전합니다.  
[주의] 괄호로 감싸지 않으면 not 연산자는 바로 뒤 a > 3을 반전합니다.
```

[참고] 연산자 우선순위

- Python 연산자의 우선순위 precedence를 정리한 표입니다.

연산자	상세 내용	연산자	상세 내용
()	• 괄호로 묶기	$^$	• 비트 연산자 배타적 논리합
$**$	• 거듭제곱	$ $	• 비트 연산자 논리합
$\sim x$	• 비트 연산자 논리부정	$in, not in,$ $<, \leq, >, \geq,$ $==, !=$	• 멤버 연산자
$+x, -x$	• 단항 덧셈, 단항 뺄셈		• 비교 연산자
$\ast, /, \%, //$	• 곱셈, 나눗셈, 나머지, 몫	not	• 논리 연산자 논리부정
$+, -$	• 덧셈, 뺄셈	and	• 논리 연산자 논리곱
&	• 비트 연산자 논리곱	or	• 논리 연산자 논리합

[참고] 괄호와 거듭제곱 사이에 자료구조(리스트, 튜플, 딕셔너리, 집합)의 생성, 합수, 슬라이싱, 인덱싱 및 객체 속성 순으로 우선순위를 적용합니다. 아울러 마지막에 람다 표현식에 대한 우선순위를 적용합니다.

문자열

- 문자열 `string`은 여러 문자/숫자를 순서대로 나열하고 따옴표로 감싼 자료형입니다.

```
>>> str1 = 'Life is short, ' # str1에 문자열을 할당합니다. 흘따옴표를 사용했습니다.
```

```
>>> str1 # str1을 출력합니다.
```

```
>>> print(str1) # print() 함수를 실행하면 따옴표 없이 글자만 출력합니다.
```

```
>>> type(str1) # str1의 클래스를 확인합니다. str1의 클래스는 str입니다.
```

```
>>> len(str1) # str1의 글자수를 반환합니다.
```

```
>>> str2 = "you need Python!" # str2에 문자열을 할당합니다. 겹따옴표를 사용했습니다.
```

```
>>> str2 # str2를 출력합니다.
```

[참고] 따옴표 사용법

- 문자열을 생성하는 다양한 따옴표 사용법을 소개합니다.

```
>>> 'Monty Python's Flying Circus' # 훌따옴표로 감싼 문자열 안에 훌따옴표를 추가할 수 없습니다.  
[주의] 왼쪽 코드를 실행하면 에러를 반환합니다.
```

```
>>> "Monty Python's Flying Circus" # 문자열 안에 훌따옴표를 추가하려면 문자열을 훌따옴표 대신에  
겹따옴표로 감싸야 합니다.
```

```
>>> 'Monty Python\'s Flying Circus' # 훌따옴표로 감싼 문자열 안 훌따옴표 앞에 역슬래시(\) 기호를  
추가합니다. [참고] \ 기호는 정규표현식의 이스케이프입니다.
```

```
>>> 'I\'m saying "You need Python!" now.' # 두 가지 경우를 모두 포함하는 예제입니다.
```

```
>>> "I'm saying \"You need Python!\" now." # 문자열을 겹따옴표로 감쌌다면 문자열 안 겹따옴표  
앞에 \ 기호를 추가합니다.
```

```
>>> '''Hello!''' # 여러 줄의 문자열을 생성하려면 훌따옴표 또는 겹따옴표를 세 번 반복하는데 이를  
독스트링docstring이라고 합니다. ex) '''문자열''', """문자열"""
```

```
Good to see you. .... # 왼쪽 코드를 실행하면 '!'와 'G' 사이에 '\n'을 출력합니다.  
[참고] '\n'은 줄바꿈입니다.
```

문자열 연산자

- + 연산자는 두 문자열을 결합합니다.

```
>>> str1 + str2 # [주의] 수와 문자열을 섞으면 에러를 반환합니다.
```

- * 연산자는 왼쪽 문자열을 오른쪽에 지정한 정수만큼 반복합니다.

```
>>> str2 * 3 # [주의] 오른쪽에 실수를 지정하면 에러를 반환합니다.
```

- 멤버 연산자 in은 오른쪽 문자열에 왼쪽 문자열이 있으면 True, 없으면 False를 반환합니다. not in은 반대로 동작합니다.

```
>>> 'a' in str1 # str1에 문자열 'a'가 있으면 True, 없으면 False를 반환합니다.
```

```
>>> 'a' not in str1 # str1에 문자열 'a'가 없으면 True, 있으면 False를 반환합니다.
```

문자열 인덱싱

- 문자열 인덱싱 indexing은 인덱스(위치번호)로 문자를 선택합니다.

인덱스	0 -15	1 -14	2 -13	3 -12	4 -11	5 -10	6 -9	7 -8	8 -7	9 -6	10 -5	11 -4	12 -3	13 -2	14 -1
문자	아	버	지	가		안	방	에		들	어	가	신	다	.

- 문자열에 대괄호를 추가하고 대괄호 안에 선택할 문자의 인덱스를 지정합니다.

```
>>> sen = '아버지가 안방에 들어가신다.' # 실습할 문자열을 생성합니다.
```

```
>>> sen[0] # sen의 0번 인덱스(첫 번째) 문자를 선택합니다. 왼쪽 코드를 실행하면 '아'를 반환합니다.  
[주의] Python 인덱스는 0부터 시작합니다!
```

```
>>> sen[1] # sen의 1번 인덱스(두 번째) 문자를 선택합니다. 왼쪽 코드를 실행하면 '버'를 반환합니다.
```

```
>>> sen[-1] # sen의 -1번 인덱스(마지막) 문자를 선택합니다. 인덱스 앞에 마이너스 부호를 추가하면 마지막에서  
거꾸로 시작합니다. [참고] -0은 0과 같습니다!
```

문자열 슬라이싱

- 문자열의 연속된 문자를 결합합니다.

```
>>> sen[0] + sen[1] + sen[2] # sen의 0~2번 인덱스 문자를 결합합니다.
```

- 문자열 슬라이싱^{Slicing}은 시작과 끝 인덱스를 콜론(:)으로 연결한 슬라이스^{slice}로 연속된 문자열을 선택합니다. (시작:끝+1) # [주의] 끝 인덱스(콜론 오른쪽 정수)를 포함하지 않으므로 끝 인덱스에 1을 더한 값을 지정해야 합니다.

```
>>> sen[0:3] # sen의 0번 인덱스(첫 번째)부터 3-1번 인덱스(세 번째)까지 연속된 문자를 선택합니다.
```

```
>>> sen[:3] # 콜론 앞에 정수를 생략하면 첫 번째 문자부터 선택합니다.
```

```
>>> sen[9:] # 콜론 뒤에 정수를 생략하면 마지막 문자까지 선택합니다.
```

```
>>> sen[:] # 빈 콜론을 입력하면 처음부터 끝까지 연속된 문자를 선택합니다.  
[주의] 대괄호 안에 아무것도 입력하지 않으면 에러를 반환합니다.
```

문자열 공백 제거 및 변경

- 문자열의 속성과 방식으로 문자열을 전처리합니다.

```
>>> sen = '\n 파이썬 \t 문자열 다루기 '; sen # 문자열 양쪽에 공백을 추가합니다.  
[참고] '\n'은 줄바꿈, '\t'는 탭입니다.
```

```
>>> print(sen) # print() 함수는 따옴표와 공백 없이 글자만 출력하므로 문자열의 공백을 확인할 수 없습니다.
```

```
>>> dir(sen) # sen의 속성과 방식을 리스트로 반환합니다.
```

```
>>> sen.strip() # sen에서 양쪽 공백을 제거한 결과를 출력합니다.  
[참고] 코드 실행 결과를 출력할 뿐, sen에 업데이트한 것은 아닙니다.
```

```
>>> sen = sen.strip() # sen에서 양쪽 공백을 제거하고 sen에 재할당합니다.  
[참고] 코드 실행 결과 아무것도 출력하지 않지만, sen을 업데이트합니다.
```

```
>>> sen.replace('\t', '') # sen에서 '\t'를 빈 문자열 ''로 바꾼 결과를 출력합니다.  
[참고] 세 번째 인수에 변경할 횟수를 정수로 지정할 수 있습니다.
```

문자열 분리 및 결합

- 문자열을 분리하거나 결합합니다.

```
>>> sen.split('\t') # sen을 구분자 '\t'으로 분리합니다. [참고] 문자열을 분리하면 리스트로 반환합니다.
```

```
>>> strs = sen.split() # sen을 공백으로 분리하고 strs에 할당합니다.  
[참고] '\n'과 '\t'는 공백으로 처리합니다.
```

```
>>> strs # strs를 출력합니다. 대괄호 안에 여러 문자열이 있습니다.
```

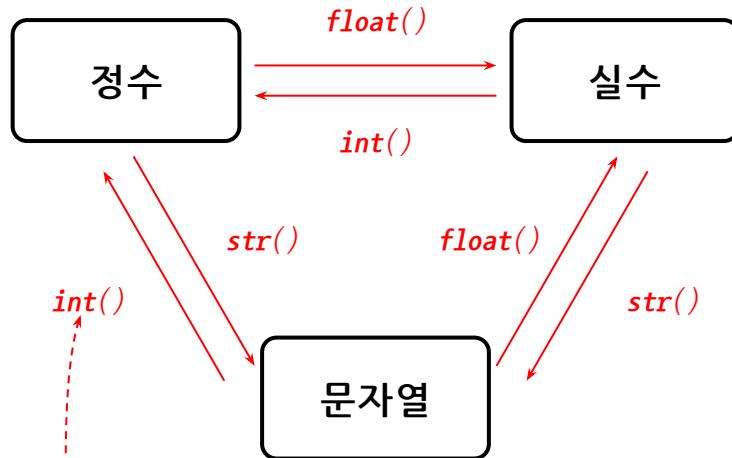
```
>>> type(strs) # strs의 클래스를 확인합니다. strs의 클래스는 list입니다.
```

```
>>> ' '.join(strs) # strs의 원소 사이에 공백을 추가하여 문자열로 결합합니다.
```

```
>>> '-'.join(strs) # 구분자에 따라 실행 결과가 달라집니다.
```

자료형 변환

- 자료형 변환^{casting} 관계도입니다.
 - 클래스 함수로 자료형을 변환합니다.



[주의] 소수점 있는 문자열은 정수로 변환할 수 없으므로 실수로 변환한 다음 정수로 변환합니다.

- 자료형을 변환합니다.

```
>>> float(1) # 정수 1을 실수로 변환합니다.
```

```
>>> str(1) # 정수 1을 문자열로 변환합니다.
```

```
>>> int(1.2) # 실수 1.2를 정수로 변환합니다.
```

```
>>> str(1.2) # 실수 1.2를 문자열로 변환합니다.
```

```
>>> float('1.2') # 문자열 '1.2'를 실수로 변환합니다.
```

```
>>> int('1.2') # 문자열 '1.2'를 정수로 변환합니다. 에러를 반환합니다!
```

문자열 포맷팅: 포맷 코드

- 변수와 문자열을 결합한 결과를 출력합니다.

```
>>> name = '홍길동'; rate = 3.2 # name에 문자열, rate에 실수를 할당합니다.
```

```
>>> name + ' 고객님의 이자율은 ' + str(rate) + ' %입니다.' # 여러 문자열을 결합합니다.
```

- 문자열 포맷팅은 문자열에서 원하는 위치에 포맷 코드를 추가하고, 변수에 할당한 값을 포맷 코드로 전달하여 문자열에 대입합니다.

%s(문자열)	%d(정수)	%f(실수)	%('%' 출력)
---------	--------	--------	-------------

```
>>> '%s 고객님의 이자율은 %f %%입니다.' %(name, rate) # name과 rate에 할당한 값을 포맷 코드(%s와 %f)에 대입합니다.
```

```
>>> '%s 고객님의 이자율은 %.2f %%입니다.' %(name, rate) # 실수는 소수점 자리수를 설정할 수 있습니다.
```

문자열 포맷팅: `format()`, f-문자열

- `format()` 함수는 포맷 코드 대신 중괄호를 사용합니다.

```
>>> '{} 고객님의 이자율은 {} %입니다.'.format(name, rate) # 중괄호에 변수값을 순서대로 대입합니다.
```

```
>>> '{0} 고객님의 이자율은 {1} %입니다.'.format(name, rate) # 중괄호 안에 변수의 정수 인덱스를 지정합니다.
```

```
>>> '{0} 고객님의 이자율은 {1:.2f} %입니다.'.format(name, rate)  
# [주의] 실수의 소수점 자리수를 지정하려면 % 대신 콜론(:)을 추가합니다.
```

- Python 3.6에서 도입한 f-문자열을 사용하면 코드 가독성을 높일 수 있습니다.

```
>>> f'{name} 고객님의 이자율은 {rate:.2f} %입니다.'
```

f-문자열은 따옴표 앞에 `f`를 추가합니다. 만약 `f`를 추가하지 않으면 중괄호 안 내용을 문자 그대로 출력합니다.

부울

- 부울^{bool}은 True 또는 False를 표현하는 자료형입니다.

```
>>> t = True # t에 True를 할당합니다. [주의] 따옴표로 감싸면 문자열이 됩니다.
```

```
>>> type(t) # t의 클래스를 확인합니다. t의 클래스는 bool입니다.
```

```
>>> True + False # True는 정수 1, False는 정수 0이므로 진리값을 더하면 True 개수를 반환합니다.
```

- 어떤 객체에 값 또는 원소가 있으면 True, 없으면 False를 반환합니다.

값 또는 원소가 있으면 True	값 또는 원소가 없으면 False
1, '사과', ['a', 'b'], (1, 2), {'키':165}	0, '', [], (), {}, None

```
>>> bool(1) # 위 표에 있는 내용을 차례대로 실행해보세요.
```

자료구조

리스트

- 리스트 list는 수, 문자열 또는 리스트 등 다양한 원소를 갖는 자료구조입니다.

```
>>> lst1 = []; lst1 # 빈 리스트를 생성합니다. 리스트를 생성할 때 대괄호 안에 원소를 콤마로 나열합니다.  
[참고] lst1 = list()를 실행한 결과와 같습니다.
```

```
>>> type(lst1) # lst1의 클래스를 확인합니다. lst1의 클래스는 list입니다.
```

```
>>> dir(lst1) # lst1의 속성과 방식을 리스트로 반환합니다.
```

```
>>> lst2 = [1, 2.0, '3']; lst2 # 리스트는 원소를 입력한 순서를 유지하며, 다양한 자료형을 원소로  
가질 수 있습니다.
```

```
>>> len(lst2) # 리스트의 길이(원소 개수)를 반환합니다. [참고] 문자열은 글자수를 반환합니다.
```

```
>>> lst3 = [1, 5, 3, 3]; lst3 # 리스트는 원소의 중복을 허용합니다.
```

리스트 연산자

- + 연산자는 두 리스트를 결합합니다.

```
>>> lst2 + lst3 # lst2와 lst3를 결합하고 하나의 리스트로 반환합니다.
```

- * 연산자는 왼쪽 리스트를 오른쪽에 지정한 정수만큼 반복합니다.

```
>>> lst3 * 2 # lst3의 전체 원소를 두 번 반복합니다.
```

- 멤버 연산자 in은 오른쪽 리스트에 왼쪽 원소가 있으면 True, 없으면 False를 반환합니다. not in은 반대로 동작합니다.

```
>>> 1 in lst3 # lst3의 원소 중에 정수 1이 있으면 True, 없으면 False를 반환합니다.
```

```
>>> 1 not in lst3 # lst3의 원소 중에 정수 1이 없으면 True, 있으면 False를 반환합니다.
```

리스트 인덱싱

- 리스트 인덱싱 indexing은 인덱스(위치번호)로 원소를 선택합니다.

인덱스	0	1	2	3	4
원소	1	2	'3'	'4'	['가', '나', '다']

- 리스트에 대괄호를 추가하고 대괄호 안에 선택할 원소의 인덱스를 지정합니다.

```
>>> lst4 = [1, 2, '3', '4', ['가', '나', '다']] # 실습할 리스트를 생성합니다.
```

```
>>> lst4[0] + lst4[1] # lst4의 0번과 1번 인덱스 원소를 더합니다.
```

```
>>> lst4[2] + lst4[3] # lst4의 2번과 3번 인덱스 원소를 결합합니다.
```

```
>>> lst4[4][0] # lst4의 4번 인덱스 원소는 리스트이므로 대괄호를 추가하면 리스트 원소를 반환합니다.
```

리스트 슬라이싱

- 리스트 슬라이싱^{Slicing}은 시작과 끝 인덱스를 콜론(:)으로 연결한 슬라이스^{slice}로 연속된 원소를 선택합니다. (시작:끝+1) *# [주의] 끝 인덱스(콜론 오른쪽 정수)를 포함하지 않으므로 끝 인덱스에 1을 더한 값을 지정해야 합니다.*
- 리스트 슬라이싱은 실행 결과를 항상 리스트로 반환합니다.

```
>>> lst4[0:2] # lst4의 0~1번 인덱스 원소를 선택하고 리스트로 반환합니다.
```

```
>>> lst4[2:4] # lst4의 2~3번 인덱스 원소를 선택하고 리스트로 반환합니다.
```

```
>>> lst4[:4] # lst4의 0~3번 인덱스 원소를 선택하고 리스트로 반환합니다.
```

```
>>> lst4[4:] # lst4의 4번~마지막 원소를 선택하고 리스트로 반환합니다.
```

[참고] 리스트 인덱싱 및 슬라이싱 관련 주의사항

- 리스트 인덱싱 및 슬라이싱과 관련하여 주의해야 할 내용을 정리한 것입니다.

```
>>> lst4[] # 대괄호 안에 아무것도 지정하지 않으면 에러를 반환합니다.
```

```
>>> lst4[:] # 빈 콜론을 지정하면 리스트의 전체 원소를 리스트로 반환합니다.
```

```
>>> lst4[0] # 대괄호 안에 인덱스를 스칼라(길이가 1인 값)로 지정하면 해당 원소를 본래 자료형으로 반환합니다.
```

```
>>> lst4[0:1] # 대괄호 안에 콜론을 사용한 슬라이스를 지정하면 항상 리스트로 반환합니다.
```

```
>>> lst4[[0, 2]] # 대괄호 안에 리스트를 지정하면 에러를 반환합니다.  
[주의] 대괄호 안에 인덱스 스칼라 또는 콜론으로 연결한 슬라이스만 지정할 수 있습니다.
```

```
>>> [lst4[0], lst4[2]] # 위 코드 대신 리스트의 0번과 2번 인덱스 원소를 선택하고 리스트로 반환합니다.
```

range() 함수 사용법

- `range()` 함수는 연속된 정수를 반환합니다. # [주의] `range()` 함수는 정수만 지정할 수 있습니다! 실수를 지정하면 에러를 반환합니다.

```
>>> lst5 = range(6); lst5 # 0부터 5까지 연속된 정수를 lst5에 할당합니다.
```

```
>>> type(lst5) # lst5의 클래스를 확인합니다. lst5의 클래스는 range입니다.
```

```
>>> lst5 = list(lst5); lst5 # lst5를 리스트로 변환하여 lst5에 재할당합니다.  
[참고] list()는 괄호 안 변수를 리스트로 변환하는 클래스 함수입니다.
```

```
>>> type(lst5) # lst5의 클래스를 확인합니다. lst5의 클래스는 list입니다.
```

```
>>> list(range(1, 6)) # 0이 아닌 정수로 시작하려면 range() 함수에 시작과 끝+1 정수를 지정합니다.
```

```
>>> list(range(1, 11, 2)) # 간격을 설정하려면 range() 함수에 시작, 끝+1 및 간격을 지정합니다.
```

이중 콜론 연산자

- 이중 콜론 연산자는 슬라이스에 간격을 설정한 것입니다.(시작:끝+1:간격)

```
>>> lst5[1:4:2] # 1~3번 인덱스 원소에서 두 칸 간격으로 원소를 선택하고 리스트로 반환합니다.
```

```
>>> lst5[:4:2] # 0~3번 인덱스 원소에서 두 칸 간격으로 원소를 선택하고 리스트로 반환합니다.
```

```
>>> lst5[1::2] # 1번 인덱스 원소부터 마지막 원소까지 두 칸 간격으로 원소를 선택하고 리스트로 반환합니다.
```

```
>>> lst5[::-2] # 처음부터 마지막 원소까지 두 칸 간격으로 원소를 선택하고 리스트로 반환합니다.
```

```
>>> lst5[::-1] # 전체 원소를 역순으로 반환합니다. [주의] 내림차순 정렬이 아닙니다.  
[참고] 간격에 -2를 지정하면 역순으로 변경하고 두 칸 간격으로 원소를 선택합니다.
```

리스트 원소 추가 또는 삽입

- **append()** 함수는 괄호 안에 지정한 값을 마지막 원소로 추가합니다.

```
>>> lst5.append(6); lst5 # lst5의 마지막 원소로 정수 6을 추가합니다.
```

```
>>> lst5.append([6, 7]); lst5 # lst5의 마지막 원소로 리스트 [6, 7]을 추가합니다.
```

- **extend()** 함수는 리스트를 입력받아 리스트 원소를 마지막 원소로 추가합니다.

```
>>> lst5.extend([8, 9]); lst5 # lst5의 마지막 원소로 정수 8, 9를 추가합니다.
```

- **insert()** 함수는 지정한 인덱스 앞에 값을 삽입합니다.(인덱스, 값)

```
>>> lst5.insert(1, [6, 7]); lst5 # lst5의 1번 인덱스에 리스트 [6, 7]을 삽입합니다.
```

[참고] **append()**, **extend()**, **insert()** 함수는 실행 결과를 리스트에 반영하므로 리스트에 재할당하지 않습니다.

리스트 원소 삭제

- `remove()` 함수는 지정한 값을 (여러 개 있어도) 한 번만 삭제합니다.

```
>>> lst5.remove([6, 7]); lst5 # lst5에서 원소 [6, 7]을 삭제합니다.
```

```
>>> lst5.remove([6, 7]); lst5 # lst5에서 원소 [6, 7]을 한 번 더 삭제합니다.
```

```
>>> lst5.remove([6, 7]) # lst5에 원소 [6, 7]이 없으므로 에러를 반환합니다.  
[참고] remove() 함수에 없는 원소를 지정하면 에러를 반환합니다.
```

- `pop()` 함수는 지정한 인덱스 원소를 출력하고 해당 원소를 삭제합니다.

```
>>> lst5.pop(6); lst5 # lst5의 6번 인덱스 원소를 출력하고 해당 원소를 삭제합니다.
```

```
>>> lst5.pop(); lst5 # lst5의 마지막 원소를 출력하고 해당 원소를 삭제합니다.  
[참고] pop() 함수에 인덱스를 생략하면 마지막 원소를 제거합니다.
```

[참고] `remove()`, `pop()` 함수는 실행 결과를 리스트에 반영하므로 리스트에 재할당하지 않습니다.

리스트 원소 변경

- 인덱싱으로 특정 원소를 선택하고, 원하는 값으로 변경합니다.

```
>>> lst5[1] = 6; lst5 # lst5의 1번 인덱스 원소를 정수 6으로 변경합니다.
```

```
>>> lst5[2] = [3, 4]; lst5 # lst5의 2번 인덱스 원소를 리스트 [3, 4]로 변경합니다.
```

- 슬라이싱으로 연속된 원소를 선택하고, 원하는 값으로 변경합니다.

```
>>> lst5[3:5] = [7, 9]; lst5 # lst5의 3~4번 인덱스 원소를 리스트 [7, 9]의 원소로 변경합니다.
```

```
>>> lst5[4:5] = [1, 2]; lst5 # lst5의 4번 인덱스 원소를 리스트 [1, 2]의 원소로 변경합니다.  
[참고] 왼쪽과 오른쪽 리스트의 원소 개수가 서로 달라도 됩니다.
```

```
>>> lst5[4:5] = 1 # [주의] 슬라이싱 결과인 리스트에 스칼라를 할당하면 에러를 반환합니다.
```

리스트 원소 정렬

- `sort()` 함수는 리스트 원소를 오름차순/내림차순 정렬합니다.

```
>>> lst5.sort() # 리스트 원소의 자료형이 다르면 원소를 정렬할 수 없으므로 에러를 반환합니다.  
[참고] sort() 함수는 리스트 원소의 클래스가 모두 같을 때 정상 실행됩니다.
```

```
>>> lst5.remove([3, 4]) # lst5에서 원소([3, 4])를 삭제합니다.
```

```
>>> lst5.sort(); lst5 # lst5의 원소를 오름차순 정렬합니다.
```

```
>>> lst5.sort(reverse = True); lst5 # lst5의 원소를 내림차순 정렬합니다.  
[참고] reverse 매개변수의 인수 기본값은 False입니다.
```

- `reverse()` 함수는 리스트 원소를 역순으로 변경합니다.

```
>>> lst3.reverse(); lst3 # lst3의 원소를 역순으로 변경합니다.  
[참고] reverse() 함수는 자료형이 섞여 있어도 에러를 반환하지 않습니다.
```

[참고] `sort()`, `reverse()` 함수는 실행 결과를 리스트에 반영하므로 리스트에 재할당하지 않습니다.

[참고] 함수/방식의 Docstring 사용법

- 함수 괄호에 커서를 놓고 [shift] + [tab] 키를 누르면 Docstring을 출력합니다.

In []: a.sort()

Signature: a.sort(*, key=None, reverse=False)
Docstring:
Sort the list in ascending order and return None.

Signature: a.sort(*, key=None, reverse=False)
Docstring:
Sort the list in ascending order and return None.

The sort is in-place (i.e. the list itself is modified) and stable (i.e. the order of two equal elements is maintained).
If a key function is given, apply it once to each list item and sort them, ascending or descending, according to their function values.
The reverse flag can be set to sort in descending order.
Type: builtin_function_or_method

^ 버튼을 클릭하면 맨 아래에 Docstring을 출력합니다.

[tab] 키를 한 번 더 누르면 제자리에서 팝업을 펼칩니다.

reverse는 플래그 변수이고, 인수 기본값은 False입니다.

아래에 reverse 관련 설명이 있습니다.

[참고] 플래그 변수는 특정 동작의 수행 여부를 불리언 (1비트) 값을 갖습니다.

[참고] 함수 뒤에 괄호 없이 물음표를 추가하고 실행하면 같은 결과를 얻습니다.

튜플

- 튜플 Tuple 은 원소를 변경할 수 없는 리스트입니다.

```
>>> tup = (1, 2.0, '3'); tup # 튜플을 생성합니다. 튜플을 생성할 때 소괄호 안에 원소를 콤마로 나열  
하며, 소괄호로 묶지 않아도 튜플로 자동 패킹 packing 합니다.
```

```
>>> type(tup) # tup의 클래스를 확인합니다. tup의 클래스는 tuple입니다.
```

```
>>> dir(tup) # tup의 속성과 방식을 리스트로 반환합니다.  
[참고] 튜플에 원소를 추가하거나 삭제하는 방식이 없습니다.
```

```
>>> tup[2] = 3 # 튜플은 원소를 할당할 수 없으므로 에러를 반환합니다. 에러 메시지는 아래와 같습니다.  
TypeError: 'tuple' object does not support item assignment
```

```
>>> tuple(lst2) # tuple() 클래스 함수는 리스트를 튜플로 변환합니다.
```

```
>>> list(tup) # list() 클래스 함수는 튜플을 리스트로 변환합니다.
```

딕셔너리

- 딕셔너리 Dictionary는 키 key와 값 value을 콜론으로 결합한 원소를 갖는 자료구조입니다.
 - 딕셔너리를 생성할 때 중괄호 안에 key: value 형태(원소)를 콤마로 연결합니다.
 - 키에 스칼라(수/문자열), 값에 스칼라, 리스트, 튜플 또는 딕셔너리를 지정합니다.
- 딕셔너리는 원소의 순서를 유지하지 않습니다. # [참고] Python 3.6 이후로 원소의 순서를 유지하는 OrderedDict 클래스와 동일하게 동작합니다.
- 딕셔너리는 키로 인덱싱합니다. # [참고] 딕셔너리에는 정수 인덱스가 없으므로 정수 인덱싱이 불가합니다. 만약 키를 정수로 지정하면 비슷하게 코딩할 수 있습니다.
 - 딕셔너리 원소를 추가, 삭제 또는 변경할 때에도 키 인덱싱을 사용합니다.
- 딕셔너리는 키의 중복을 허용하지 않습니다. 따라서 같은 키에 다른 값을 여러 번 할당하면 가장 마지막에 할당한 값으로 업데이트합니다.

딕셔너리 생성

- 딕셔너리를 생성하고 클래스를 확인합니다. # [주의] 키가 문자열일 때 따옴표로 감싸지 않으면 변수로 인식합니다.

```
>>> dct = {'item': 'pants', 'size': ['S', 'M']}; dct
```

```
>>> type(dct) # dct의 클래스를 확인합니다. dct의 클래스는 dict입니다.
```

```
>>> dir(dct) # dct의 속성과 방식을 리스트로 반환합니다.
```

```
>>> dct[0] # 딕셔너리는 인덱스가 없으므로 에러를 반환합니다.  
[참고] 만약 딕셔너리의 키에 정수 0이 있으면 정상적으로 실행됩니다.
```

```
>>> lst = ['pants', ['S', 'M']]; lst # 같은 값을 원소로 갖는 인덱스  
리스트와 비교합니다.
```

```
>>> lst[0] # 리스트는 인덱스가 있으므로 인덱싱할 수 있습니다.
```

키	'item'	'size'
값	'pants'	['S', 'M']

0	1
'pants'	['S', 'M']

[참고] dict() 클래스 함수 사용법

- dict() 클래스 함수는 키를 따옴표로 감싸지 않고 콜론 대신 등호를 사용합니다.

```
>>> dict(item = 'pants', size = ['S', 'M']) # [참고] 시각화 함수에 그래픽 요소를 딕셔너리를  
    지정할 때 dict() 함수를 자주 사용합니다.
```

- 키를 따옴표로 감싸면 에러를 반환합니다.

```
>>> dict('item' = 'pants', 'size' = ['S', 'M'])
```

- 등호 대신 콜론을 사용하면 에러를 반환합니다.

```
>>> dict(item: 'pants', size: ['S', 'M'])
```

- 빈 딕셔너리를 생성합니다.

```
>>> dict()
```

딕셔너리 연산자와 인덱싱

- 멤버 연산자 `in`은 딕셔너리에 키가 있으면 `True`, 없으면 `False`를 반환합니다.

```
>>> 'item' in dct # dct에 'item'인 키가 있으면 True, 없으면 False를 반환합니다.
```

```
>>> 'shop' in dct # dct에 'shop'인 키가 있으면 True, 없으면 False를 반환합니다.
```

- 딕셔너리는 키로 인덱싱합니다.

```
>>> dct['item'] # dct에서 키가 'item'인 값을 반환합니다.
```

```
>>> dct['shop'] # dct에서 키가 'shop'인 값을 반환합니다.  
[주의] 딕셔너리에 없는 키를 입력하면 에러를 반환합니다.
```

딕셔너리 원소 추가, 삭제 또는 변경

- 딕셔너리에 원소를 추가합니다.

```
>>> dct['shop'] = 'A1'; dct # dct에 키가 'shop'이고 값이 'A1'인 원소를 추가합니다.
```

- 딕셔너리의 원소를 삭제합니다.

```
>>> del dct['item']; dct # dct에서 키가 'item'인 원소를 삭제합니다.  
[참고] del문은 예약어 keyword이고, 변수 또는 변수의 원소를 삭제합니다.
```

- 딕셔너리의 원소를 변경합니다.

```
>>> dct['shop'] = 'A2'; dct # dct에서 키가 'shop'인 값을 'A2'로 변경합니다.
```

[참고] 딕셔너리 키와 값 반환

- 딕셔너리의 키와 값을 반환합니다.

```
>>> dct.items() # items() 함수는 딕셔너리의 키와 값의 쌍을 튜플로 묶어서 dict_items 클래스로 반환합니다.
```

- 반복문으로 딕셔너리의 키와 값을 차례대로 출력합니다.

```
>>> for item in dct.items():
```

```
    print(item) # 변수 item은 dct의 키와 값의 쌍을 원소로 갖는 튜플입니다.
```

집합

- 집합^{Set}은 원소의 중복을 허용하지 않고 순서도 없는 자료구조입니다.

```
>>> set1 = {3, 1, 2, 1}; set1 # 집합을 생성합니다. 집합을 생성할 때 중괄호 안에 원소를 콤마로 나열  
하며, 중복 원소를 제거하고 오름차순 정렬합니다.
```

```
>>> type(set1) # set1의 클래스를 확인합니다. set1의 클래스는 set입니다.
```

```
>>> dir(set1) # set1의 속성과 방식을 리스트로 반환합니다.
```

```
>>> set1[0] # 집합에 인덱스가 없으므로 에러를 반환합니다.
```

```
>>> set2 = set(lst3); set2 # set() 클래스 함수는 리스트를 집합으로 변환합니다.
```

```
>>> list(set1) # list() 클래스 함수는 집합을 리스트로 변환합니다.
```

집합 연산자

- 두 집합 객체로 교집합, 합집합 또는 차집합 연산을 실행합니다.

```
>>> set1 & set2 # 두 집합의 교집합을 반환합니다.
```

```
>>> set1.intersection(set2)
```

```
>>> set1 | set2 # 두 집합의 합집합을 반환합니다.
```

```
>>> set1.union(set2)
```

```
>>> set1 - set2 # 두 집합의 차집합을 반환합니다.  
[참고] 집합 객체의 순서를 바꾸면 다른 결과를 반환합니다.
```

```
>>> set1.difference(set2)
```

[참고] 집합 원소 생성 기준

- 리스트를 집합으로 변환할 때 중복 원소는 아래 기준으로 제거합니다.
 - 정수와 실수는 앞선 자료형을 적용합니다. ex) $\{1, 1.0, 2.0, 2\} \rightarrow \{1, 2.0\}$
 - 문자열은 정수/실수와 별개로 존재합니다. ex) $\{1, '1'\} \rightarrow \{1, '1'\}$
- 집합 원소의 정렬 순서는 다음과 같습니다.
 - 정수, 문자열, 실수 순으로 정렬합니다. ex) $\{1.0, 1, '1'\} \rightarrow \{'1', 1.0\}$
 - 교집합은 오른쪽 집합의 자료형을 따릅니다. ex) $\{1\} \& \{1.0\} \rightarrow \{1.0\}$
 - 합집합은 왼쪽 집합의 자료형을 따릅니다. ex) $\{1\} | \{1.0\} \rightarrow \{1\}$
 - 차집합은 왼쪽 집합의 자료형을 따릅니다. ex) $\{1, 2\} - \{1.0\} \rightarrow \{2\}$

[참고] 다양한 괄호 사용법

- 소괄호, 중괄호, 대괄호 사용법에 대해 표로 정리한 것입니다.

구분	상세내용
소괄호 ()	<ul style="list-style-type: none">- 튜플을 생성할 때 소괄호 안에 원소를 콤마로 연결합니다.- 함수 또는 객체의 방식<small>method</small> 뒤에 추가해야 합니다.
중괄호 {}	<ul style="list-style-type: none">- 딕셔너리 또는 집합을 생성할 때 중괄호 안에 원소를 콤마로 연결합니다.- f-문자열에서 변수를 중괄호로 감싸주어야 합니다.
대괄호 []	<ul style="list-style-type: none">- 리스트를 생성할 때 대괄호 안에 원소를 콤마로 연결합니다.- 객체를 인덱싱할 때 대괄호 안에 정수 스칼라, 슬라이스를 지정합니다.

제어문: 조건문

if 조건문 기본 구조

- if 조건문은 지정한 조건 만족 여부에 따라 실행할 코드를 분기합니다.

>>> if 조건1: # if문에 조건을 추가하고 끝에 콜론을 추가합니다.

 코드A # 조건1을 만족하면 코드A를 실행하고 조건문을 종료합니다.
 탭 또는 공백 4칸으로 들여쓰기합니다. 만약 들여쓰기 하지 않으면 에러를 반환합니다.

 elif 조건2: # 추가 조건이 있으면 elif문을 사용하고, 추가 조건이 없으면 생략합니다.
 elif문만 단독으로 실행할 수 없고, if문 다음에 여러 번 추가할 수 있습니다.

 코드B # 조건2를 만족하면 코드B를 실행하고 조건문을 종료합니다.

 else: # 모든 조건을 만족하지 않을 때 마지막으로 실행할 코드가 있으면 else문을 사용합니다.
 else문도 단독으로 실행할 수 없고, 생략하거나 마지막에 한 번 추가합니다.

 코드C # 모든 조건을 만족하지 않으면 코드C를 실행하고 조건문을 종료합니다.

[중요] if 조건문에 지정하는 조건 코드는 실행 결과로 True 또는 False를 스칼라로 반환해야 합니다.

[참고] 탭 vs 공백

- 조건문, 반복문 또는 사용자 정의 함수의 콜론 다음에 [enter] 키를 누르면 탭 들여쓰기를 지원합니다.
- Python은 탭 크기를 공백 4칸으로 설정합니다.

>>> if 조건1:

코드A # 공백 4칸과 탭 1번을 섞어 쓰지 않는 것이 좋습니다.

elif 조건2:

코드B # [참고] IDE마다 탭 크기가 다르므로 공백 대신 탭을 사용하지 않는 것이 좋습니다.

Jupyter Notebook은 탭을 공백 4칸으로 자동 변경합니다.

When using spacebar instead of tabs



콘솔에서 문자열 입력

- 사용자가 입력한 값을 문자열로 변수에 할당합니다.

```
>>> score = input('점수: ') # input() 함수는 콘솔에 '점수:' 를 출력하고 사용자의 입력을 기다립니다.  
사용자가 값을 입력하면 문자열로 변수에 할당합니다.
```

```
>>> score # score를 출력합니다. score는 따옴표로 감싼 문자열입니다.  
[참고] 전역 변수 목록에 없는 변수를 출력하려고 하면 에러를 반환합니다.
```

- score를 실수로 변환합니다.

```
>>> score = float(score)
```

```
>>> score # score를 출력합니다. score는 따옴표 없이 소수점이 추가된 실수입니다.
```

if 조건문

- 조건문을 실행하고 점수에 따라 합격 여부를 출력합니다.

```
>>> if score >= 90:
```

```
    print('합격') # 첫 번째 조건을 만족하면 왼쪽 코드를 실행하고 조건문을 종료합니다.
```

```
elif score >= 80:
```

```
    print('재검사') # 두 번째 조건을 만족하면 왼쪽 코드를 실행하고 조건문을 종료합니다.
```

```
else:
```

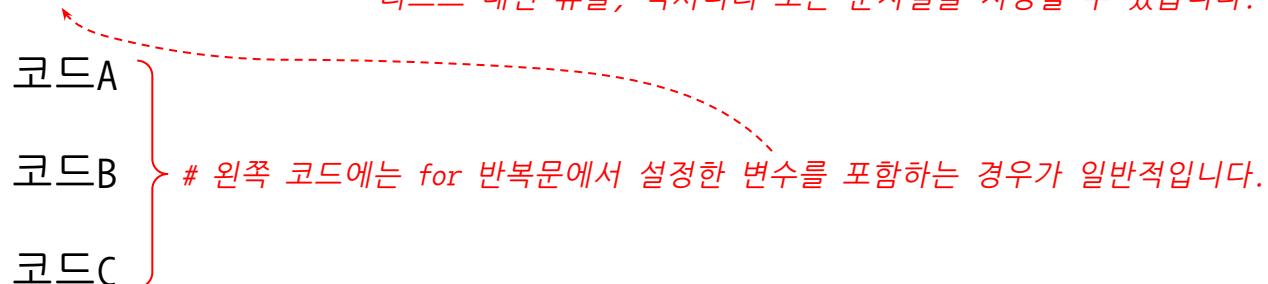
```
    print('불합격') # 모든 조건을 만족하지 않으면 왼쪽 코드를 실행하고 종료합니다.
```

제어문: 반복문

for 반복문 기본 구조

- 반복문은 여러 줄의 코드에서 일부 값을 바꾸면서 반복 실행할 때 사용합니다.
 - for 반복문과 while 반복문이 있으며 상황에 따라 알맞는 반복문을 선택합니다.
- for 반복문은 반복 실행할 범위가 정해져 있을 때 사용합니다.

```
>>> for 변수 in 리스트: # 변수는 리스트 원소를 처음부터 입력받아 콜론 아래 코드를 반복 실행합니다.  
    리스트 대신 튜플, 딕셔너리 또는 문자열을 지정할 수 있습니다.
```



for 반복문

- 중식당 메뉴로 리스트를 생성하고 반복문으로 리스트 원소를 차례대로 출력합니다.

```
>>> menu = ['짜장면', '탕수육', '깐풍기', '짬뽕', '전가복', '샥스핀']
```

```
>>> for item in menu:
```

```
    print(item) # [주의] 콜론 아래 코드에서 변수가 가리키는 값을 출력하려면 반드시 print() 함수를  
               사용해야 합니다!
```

- 여러 객체를 결합한 문자열을 출력합니다.

```
>>> for item in menu:
```

```
    print(item, '시킬까요?', sep = ' ', end = '\n')
```

```
# print() 함수는 여러 객체를 sep 매개변수에 지정한 구분자로 결합한 문자열을 출력합니다.(기본값: ' ')  
[참고] end 매개변수에는 마지막에 추가할 문자열을 지정합니다.(기본값: '\n')
```

반복문 안에 조건문 추가

- 반복문 안에 조건문을 추가하여 코드를 제어합니다.

```
>>> for item in menu:
```

```
    print(item, '시킬까요?')
```

```
    if item in ['짜장면', '짬뽕']: # item이 '짜장면' 또는 '짬뽕'이면 아래 코드를 실행합니다.
```

```
        print('-> 요리부터 주문합시다!')
```

```
    print('-> 다음 메뉴는 뭔가요?\n') # 조건문과 상관없이 왼쪽 코드를 항상 실행합니다.
```

반복문 제어: continue

- 반복문에서 `continue`를 만나면 아래 코드를 실행하지 않고 처음으로 되돌아갑니다.

```
>>> for item in menu:
```

```
    print(item, '시킬까요?')
```

```
    if item in ['짜장면', '짬뽕']:
```

```
        continue # 반복문 실행 도중 continue를 만나면 처음으로 되돌아가므로 아래 코드를 실행하지 않습니다.
```

```
    print('-> 다음 메뉴는 뭔가요?\n') # item이 '짜장면' 또는 '짬뽕'이면 실행하지 않습니다.
```

반복문 제어: break

- 반복문에서 break를 만나면 반복문을 중단합니다.

```
>>> for item in menu:
```

```
    print(item, '시킬까요?')
```

```
    if item in ['전가복', '샥스핀']:
```

```
        break # 반복문 실행 도중 break를 만나면 반복문을 중단하므로 아래 코드를 실행할 수 없습니다.
```

```
    print('-> 다음 메뉴는 뭔가요?\n')
```

[참고] 중첩 for 반복문

- 두 리스트의 원소로 가능한 모든 경우의 수를 반복 실행합니다.(구구단 만들기)

```
>>> for i in range(2, 10): # i는 2를 입력받아 안쪽 반복문을 실행합니다.  
    # 안쪽 반복문을 완료하면 i는 다음 원소를 입력받습니다.  
  
    print(f'*** {i}단 ***') # i에 할당한 값을 대입합니다.
```

```
for j in range(1, 10): # j는 1~9의 값을 차례로 입력받아 안쪽 반복문을 실행합니다.
```

```
    print(f'{i} * {j} = {i*j}')
```

```
print() # 안쪽 반복문이 끝나고 i가 다음 값을 받기 전에 줄바꿈을 실행합니다.
```

```
>>> print('반복문 실행 완료!') # 반복문이 끝나면 왼쪽 코드를 실행합니다.
```

반복문 실행 결과를 리스트로 생성

- 빈 리스트를 미리 생성합니다.

```
>>> sqrs = [] # for 반복문을 실행하기 전에 값을 저장할 빈 리스트를 미리 생성합니다.
```

- 반복문으로 생성한 값을 리스트 원소로 추가합니다.

```
>>> for i in range(1, 11): # 1~10의 정수를 차례대로 i에 할당하고 아래 코드를 반복 실행합니다.
```

```
    sqrs.append(i ** 2) # i를 제곱한 값을 sqrs의 마지막 원소로 추가합니다.  
    [주의] sqrs를 미리 생성하지 않으면 에러를 반환합니다.
```

- sqrs를 출력합니다.

```
>>> sqrs # 1~10의 정수를 제곱한 값을 원소로 갖습니다.
```

반복문 실행 결과를 리스트로 생성(계속)

- sqrs를 다시 빈 리스트로 생성합니다.

```
>>> sqrs = []
```

- 1~10의 정수에서 짝수만 제곱한 값을 원소로 갖는 리스트를 생성합니다.

```
>>> for i in range(1, 11):
```

```
    if i % 2 == 0:
```

```
        sqrs.append(i ** 2) # i가 짝수일 때 i를 제곱한 값을 sqrs의 마지막 원소로 추가합니다.
```

- sqrs를 출력합니다.

```
>>> sqrs # 1~10의 정수에서 짝수를 제곱한 값을 원소로 갖습니다.
```

리스트 컴프리헨션

- 리스트 컴프리헨션 Comprehension은 반복문을 대괄호 안에서 실행하고 코드 실행 결과를 리스트로 반환합니다.
- 한 줄 코드로 for 반복문과 같은 결과를 반환합니다.

```
>>> [i ** 2 for i in range(1, 11)] # [참고] 빈 리스트를 미리 생성할 필요가 없습니다.
```

- 반복문 뒤에 조건문을 추가하면 조건을 만족하는 원소만 리스트에 포함시킵니다.

```
>>> [i ** 2 for i in range(1, 11) if i % 2 == 0]
```

- 중첩 for 반복문을 리스트 컴프리헨션으로 실행합니다.

```
>>> [f'{i} * {j} = {i*j}' for i in range(2, 10) for j in range(1, 10)]
```

zip() 함수 사용법

- zip() 함수는 지정한 여러 객체에서 같은 인덱스 원소 쌍을 튜플로 반환합니다.

```
>>> for i in zip(range(2, 10), range(1, 10)): # [참고] zip() 함수에 리스트, 튜플, 딕셔너리,  
    집합 등 다양한 객체를 지정할 수 있습니다.
```

```
    print(i) # i는 두 객체에서 같은 인덱스 원소 쌍을 튜플로 받습니다.  
    # [참고] zip() 함수에 딕셔너리를 지정하면 키를 출력합니다.
```

두 객체의 길이(원소 개수)가 다르면 원소 개수가 적은 객체를 기준으로 동작합니다.

- 두 객체에서 같은 인덱스 원소를 i와 j로 받아서 반복문을 실행합니다.

```
>>> for i, j in zip(range(2, 10), range(1, 10)): # [참고] 튜플 원소를 i와 j로 받도록 튜플을  
    언패킹(unpacking)합니다.
```

```
    print(f'{i} * {j} = {i*j}')
```

enumerate() 함수 사용법

- `enumerate()` 함수는 지정한 객체의 인덱스와 원소 쌍을 튜플로 반환합니다.

```
>>> for i in enumerate(range(1, 11)): # [참고] enumerate() 함수에 리스트, 튜플, 딕셔너리, 집합 등 다양한 객체를 지정할 수 있습니다.
```

```
    print(i) # i는 range(1, 11)의 인덱스와 원소 쌍을 튜플로 받습니다.  
          # [참고] enumerate() 함수에 딕셔너리를 지정하면 키를 출력합니다.
```

- 객체의 인덱스와 원소를 `i`와 `v`로 받아서 반복문을 실행합니다.

```
>>> for i, v in enumerate(range(1, 11)):
```

```
    print(f'{i}번 인덱스 원소는 {v}입니다.')
```

반복문에서 에러 발생

- 반복문 실행 도중 에러가 발생하면 반복문을 중단합니다.

```
>>> for i in range(1, 11):  
    if i % 3 == 0:  
  
        i = str(i) # i가 3의 배수면 문자열로 변환합니다.  
  
    print(i ** 2) # i가 문자열이면 제곱할 수 없으므로 에러가 발생하고 for 반복문을 중단합니다.
```

예외 처리

- 코드 실행 도중 에러가 발생해도 멈추고 싶지 않을 때 예외 처리를 추가합니다.

>>> try: # try문 아래에 실행 코드를 추가합니다. 실행 코드를 여러 줄 입력할 수 있습니다.

코드A

except 에러 종류: # 코드A 실행 도중 에러가 발생하면 대신 실행할 코드B를 except문에 추가합니다.
[참고] except문에 에러 종류를 명시하면 해당 에러에 대해 아래 코드를 실행합니다.

코드B # [참고] 코드B 대신 pass를 지정하면 아무것도 실행하지 않고 통과합니다.

finally: # 코드 에러와 상관 없이 항상 마지막으로 실행할 코드가 있으면 finally문에 추가합니다.
[참고] finally문은 생략할 수 있습니다.

코드C

반복문에 예외 처리 추가

- 반복문에 예외 처리를 추가하면 에러가 발생해도 반복문을 끝까지 실행합니다.

```
>>> for i in range(1, 11):  
  
    if i % 3 == 0: # [참고] 에러를 발생시키지 않는 코드는 try문에 포함하지 않아도 됩니다.  
  
        i = str(i)  
  
    try:  
  
        print(i ** 2) # i가 문자열일 때 에러가 발생합니다.  
  
    except Exception as e: # 에러가 발생하면 e에 에러 메시지를 할당합니다.  
  
        print(e) # 에러가 발생하면 에러 메시지를 출력합니다.
```

while 반복문 기본 구조

- 사전에 정해진 범위는 없지만 어떤 조건을 만족하는 한 반복문을 계속 실행하려면 while 반복문을 사용합니다.

>>> while 조건: # 지정한 조건이 True면 아래 코드를 반복 실행합니다.

코드A
코드B
코드C } # 왼쪽 코드에 while 반복문 조건을 False로 만드는 증감식 또는 조건문 + break를 추가합니다.
그렇지 않으면 while 반복문을 무한 반복합니다!

while 반복문

- i에 정수 5를 할당합니다.

```
>>> i = 5
```

- while 반복문을 실행하여 i에서 1씩 차감한 값을 출력합니다.

```
>>> while i > 0: # 지정한 조건이 False가 될 때까지 아래 코드를 반복 실행합니다.
```

```
    print(i)
```

i -= 1 # 왼쪽 코드는 반복문을 실행하면서 i에서 1씩 차감하는 증감식입니다.
[주의] 뺄셈 대신 덧셈 연산자를 사용하면 while 반복문을 무한 실행합니다.

- i를 출력합니다.

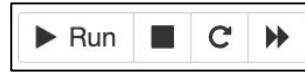
```
>>> print(i) # i는 정수 0을 가리키고 있습니다.
```

while 반복문 제어

- while 반복문에서 break를 만나면 반복문을 중단합니다.

```
>>> while True: # while 반복문에 조건 대신 True를 지정하면 반복문을 무한 반복합니다.  
    코드 실행을 멈추려면 상단 메뉴에서 네모 버튼(interrupt)을 클릭합니다!
```

```
i += 1
```



```
if i >= 10000:
```

```
    break
```

- i를 출력합니다.

```
>>> print(i) # i는 정수 10000을 가리키고 있습니다.
```

사용자 정의 함수

사용자 정의 함수의 필요성

- 당장 실행되는 코딩에 집중하면 같은 코드를 여러 번 중복할 가능성이 있습니다.
 - 예를 들어 체질량지수^{BMI} 계산 코드를 10번 복사하여 값만 바꿔서 코딩했다고 가정합시다.
- 코드를 중복 사용하면 코드가 길어지고 사소한 수정 작업도 어려울 수 있습니다.
 - 중복 사용한 코드만 50줄입니다.
 - hgt를 height로 바꾸려면 30번 수정해야 합니다.
- 코드 중복 대신 함수를 사용하는 것이 좋습니다.
 - 중복 코드를 한 줄 함수로 변경하면 코드가 짧아집니다.
 - 수정사항이 있을 때 함수 코드만 수정하면 됩니다.

```
>>> hgt = 190  
>>> wgt = 95  
>>> hgt /= 100  
>>> bmi = wgt / hgt**2  
>>> bmi
```

[참고] 스파게티 코드

- 프로그래밍 결과 코드가 복잡하게 엉켜 있는 상태를 스파게티 코드라고 합니다.
- 스파게티 코드는 정상적으로 동작하지만 코드를 읽고 동작을 파악하기 어렵다는 특징이 있습니다.
- 만약 코드 실행 도중에 에러가 발생하면 에러를 빠르게 수정하기 어렵습니다.
- 따라서 가독성 높은 코딩을 위해 중복을 피하고 주석을 추가하는 것이 좋습니다.



사용자 정의 함수 기본 구조

- 체질량지수를 반환하는 사용자 정의 함수를 생성합니다.

```
>>> def BMI(hgt, wgt): # 함수명과 매개변수를 정의합니다.
```

```
    bmi = wgt / (hgt/100) ** 2
```

```
    return bmi # return문에 함수가 반환할 값을 지정합니다.
```

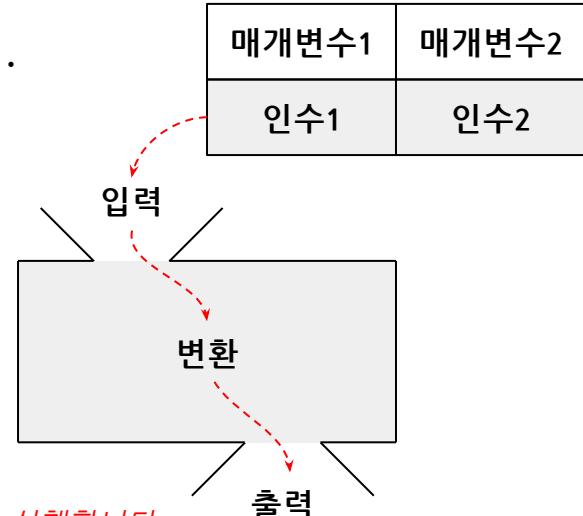
- 사용자 정의 함수를 실행합니다.

```
>>> BMI(175, 65) # 매개변수 없이 인수만 지정하여 사용자 정의 함수를 실행합니다.
```

[참고] 매개변수 없는 인수(위치 인수)의 위치를 바꾸면 함수 실행 결과가 달라집니다.

```
>>> BMI(hgt = 175, wgt = 65) # 매개변수와 인수를 등호로 연결하면 위치를 바꿔도 실행 결과가 같습니다.
```

[참고] hgt는 매개변수 *parameter*, 175는 인수 *argument*입니다.



인수의 기본값 설정 및 독스트링 추가

- 사용자 정의 함수에서 인수의 기본값을 설정하고 독스트링을 추가합니다.

```
>>> def BMI(hgt = 175, wgt = 65): # [주의] 인수의 기본값을 설정하지 않은 매개변수를 뒤쪽에 놓으면  
    # 예를 들어 hgt에 기본값 175를 설정하고 wgt에 기본값을 설정하지  
    ...  
    # 않으면 에러를 반환합니다.
```

This function returns BMI from height(cm) and weight(kg).

```
''' # [참고] 함수 관련 설명을 Docstring으로 추가할 수 있습니다.
```

```
return wgt / (hgt/100) ** 2 # 중간에 변수를 생성하지 않고 반환할 수 있습니다.
```

- 사용자 정의 함수에 인수를 생략하면 인수의 기본값을 적용합니다.

```
>>> BMI() # [주의] 사용자 정의 함수에 인수의 기본값을 설정하지 않았다면 에러를 반환합니다.
```

람다 표현식 사용법

- 람다^{lambda} 표현식은 한 줄 코드로 함수를 만들 때 사용합니다.
 - 아래는 BMI 함수를 람다 표현식으로 바꾼 코드입니다.
 - 람다 표현식에서도 인수의 기본값을 설정할 수 있습니다.

```
>>> BMI2 = lambda hgt, wgt: wgt / (hgt/100) ** 2 # 콜론 오른쪽 코드를 실행하고 반환합니다.
```

```
>>> BMI2(hgt = 190, wgt = 85) # 람다 표현식으로 정의한 함수를 실행합니다.
```

- 람다 표현식을 괄호로 감싸면 함수처럼 실행할 수 있으므로 익명함수라 합니다.

```
>>> (lambda hgt, wgt: wgt / (hgt/100) ** 2)(190, 85)
```

기존 함수명과 중복 문제

- 정수의 합을 반환합니다.

```
>>> sum(range(1, 11)) # sum() 함수는 정수/실수형 원소를 모두 더한 결과를 반환합니다.
```

- 기존 함수와 같은 이름의 사용자 정의 함수를 생성합니다.

```
>>> def sum(x):
```

```
    y = 0
```

```
    for i in x: y == i # for 반복문에서 실행 코드가 한 줄일 때 콜론 오른쪽에 이어서 작성할 수
                        # 있지만 가독성 측면에서 좋지 않으므로 두 줄로 작성하는 것이 좋습니다.
```

```
    return y
```

```
>>> sum(range(1, 11)) # 사용자 정의 함수를 실행하면 수치형 원소를 모두 뺀 결과를 반환합니다.
```

기존 함수명과 중복 문제(계속)

- 사용자 정의 함수에 우선순위를 부여하므로 기존 함수를 사용할 수 없습니다.
- 전역 변수 목록을 확인합니다.

```
>>> %whos # [참고] %whos는 Jupyter Notebook에서 사용할 수 있는 매직 명령어입니다.
```

- 사용자 정의 함수를 삭제하면 기존 함수를 사용할 수 있습니다.

```
>>> del sum # del문 오른쪽에 삭제할 변수명을 지정합니다.  
[참고] 여러 개를 한 번에 삭제하려면 변수명을 콤마로 나열합니다.
```

```
>>> sum(range(1, 11)) # 기존 함수를 실행합니다.
```

- 이와 같이 기존 함수와 같은 이름의 함수를 생성하면 기존 함수를 사용하는 것이 불편하므로 함수를 정의할 때 기존 함수명을 피하는 것이 좋습니다.

함수에 정의한 인수 누락 에러

- 함수를 정의할 때 입력받을 매개변수를 나열합니다.

```
>>> def printValues1(a, b, c):  
    for i in (a, b, c):  
        print(i)
```

- 함수에서 정의한 매개변수에 인수를 누락하면 에러를 반환합니다.

```
>>> printValues1(a = 1, b = 2, c = 3) # 함수에서 정의한 매개변수 개수만큼 인수를 지정합니다.  
>>> printValues1(a = 1, b = 2) # [주의] c 매개변수(위치 인수)를 누락했으므로 에러를 반환합니다.
```

여러 인수를 튜플로 받는 함수

- 함수를 정의할 때 매개변수 대신 가변 인수 `*args`를 지정합니다.

```
>>> def printValues2(*args):  
  
    for i in args: # args는 인수를 튜플로 전달받습니다.  
  
        print(i)
```

- 함수의 괄호 안에 원하는 개수만큼 인수를 지정할 수 있습니다.

```
>>> printValues2(1, 2, 3) # [주의] a = 1과 같이 키워드 인수로 지정하면 에러를 반환합니다.  
  
>>> printValues2(1, 2) # 인수를 2개 지정해도 에러를 반환하지 않습니다.
```

여러 인수를 딕셔너리로 받는 함수

- 함수를 정의할 때 매개변수 대신 키워드 가변 인수 `**kwargs`를 지정합니다.

```
>>> def printValues3(**kwargs):
```

```
    for k, v in kwargs.items(): # kwargs는 키워드 인수를 딕셔너리로 전달받습니다.  
                                [참고] 튜플을 언패킹하면 키를 k, 값을 v로 받습니다.
```

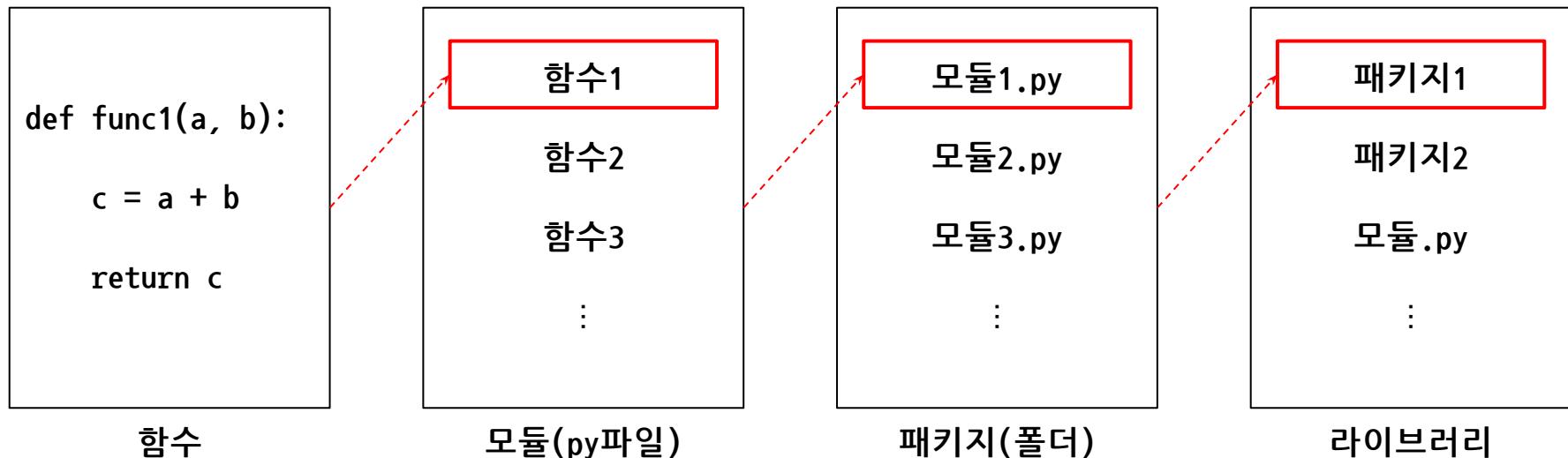
```
        print(f'{k}은(는) {v}입니다.')
```

- 함수의 괄호 안에 `dict()` 클래스 함수처럼 키와 값을 등호로 연결하여 지정합니다.

```
>>> printValues3(메뉴 = '순대국') # 메뉴 = '순대국'처럼 키워드 인수로 지정해야 합니다.  
                                [주의] '순대국'처럼 위치 인수로 지정하면 에러를 반환합니다.
```

```
>>> printValues3(메뉴 = '순대국', 가격 = 9000) # 키워드 인수를 여러 개 지정할 수 있습니다.
```

함수, 모듈, 패키지, 라이브러리의 관계



사용자 정의 함수 모듈 생성

- 사용자 정의 함수 코드를 모듈(py 파일)로 저장하면 필요할 때마다 호출할 수 있으므로 편리합니다.
- Anaconda 메인에서 New Text File을 열고, 모듈(py 파일)로 저장할 사용자 정의 함수 (예를 들어 printValues3) 코드를 붙여넣습니다.
- 상단 메뉴에서 File → Save를 클릭하여 텍스트 파일을 저장합니다.
 - 파일명을 myFuncs.py로 변경합니다. # [주의] py 파일을 현재 사용 중인 Jupyter Notebook 파일과 같은 폴더에 저장하면 쉽게 호출할 수 있습니다.
- Jupyter Notebook으로 돌아와 상단 메뉴에서 Kernel → Restart Kernel and Clear All Outputs를 클릭하고 Jupyter Notebook을 초기화합니다.

사용자 정의 함수 모듈 호출

- 사용자 정의 함수 모듈을 호출하는 첫 번째 방법입니다.

```
>>> import myFuncs # myFuncs 모듈을 호출합니다.
```

```
>>> %whos # 전역 변수 목록을 표로 출력하면 myFuncs 모듈을 확인할 수 있습니다.
```

```
>>> myFuncs.printValues3(메뉴 = '순대국', 가격 = 9000) # 함수명 앞에 myFuncs를 추가해야  
함수를 실행할 수 있습니다.
```

- Jupyter Notebook을 초기화하고, 두 번째 방법으로 모듈을 호출합니다.

```
>>> import myFuncs as mf # myFuncs 모듈을 mf라는 가명으로 호출합니다.
```

```
>>> %whos # 전역 변수 목록을 표로 출력하면 mf 모듈을 확인할 수 있습니다.
```

```
>>> mf.printValues3(메뉴 = '순대국', 가격 = 9000) # 함수명 앞에 mf를 추가해야 함수를 실행  
할 수 있습니다.
```

사용자 정의 함수 모듈 호출(계속)

- Jupyter Notebook을 초기화하고, 모듈에서 함수만 호출합니다.

```
>>> from myFuncs import printValues3 # myFuncs 모듈에서 일부 함수만 호출합니다.  
[참고] 호출할 함수가 여러 개일 때 콤마로 나열합니다.
```

```
>>> %whos # 전역 변수 목록을 표로 출력하면 printValues3 함수를 확인할 수 있습니다.
```

```
>>> printValues3(메뉴 = '순대국', 가격 = 9000) # 함수명 앞에 모듈명을 추가할 수 없습니다.
```

- Python에서 세 가지 형태의 코드를 섞어서 사용하므로 상황에 따라 알맞은 코드를 작성하는 것이 좋습니다.

numpy 라이브러리

numpy 라이브러리

- numpy 라이브러리 함수로 다음과 같은 작업을 수행할 수 있습니다.
 - 1~n차원의 배열`ndarray`을 생성합니다.
 - 배열의 차원`dimension`을 확인하고 형태`shape`를 변경합니다.
 - 1차원 배열(벡터) 또는 2차원 배열(행렬) 연산을 실행합니다.[선형대수]
 - 1차원 배열: 벡터의 덧셈과 뺄셈, 스칼라배, 벡터의 내적 연산 등
 - 2차원 배열: 행렬의 덧셈과 뺄셈, 스칼라배, 행렬의 곱셈, 판별식, 역행렬 등
- numpy 라이브러리를 호출합니다.

```
>>> import numpy as np # numpy를 np라는 가명으로 호출합니다.
```

[참고] 배열의 시각적 예시

[1차원 배열]

axis = 0

81	80	81	59	92	...
----	----	----	----	----	-----

[2차원 배열]

axis = 1

axis = 0

81	80	81	59
92	67	65	63
78	91	73	72
54	79	54	49
71	60	45	60

[3차원 배열]

axis = 2

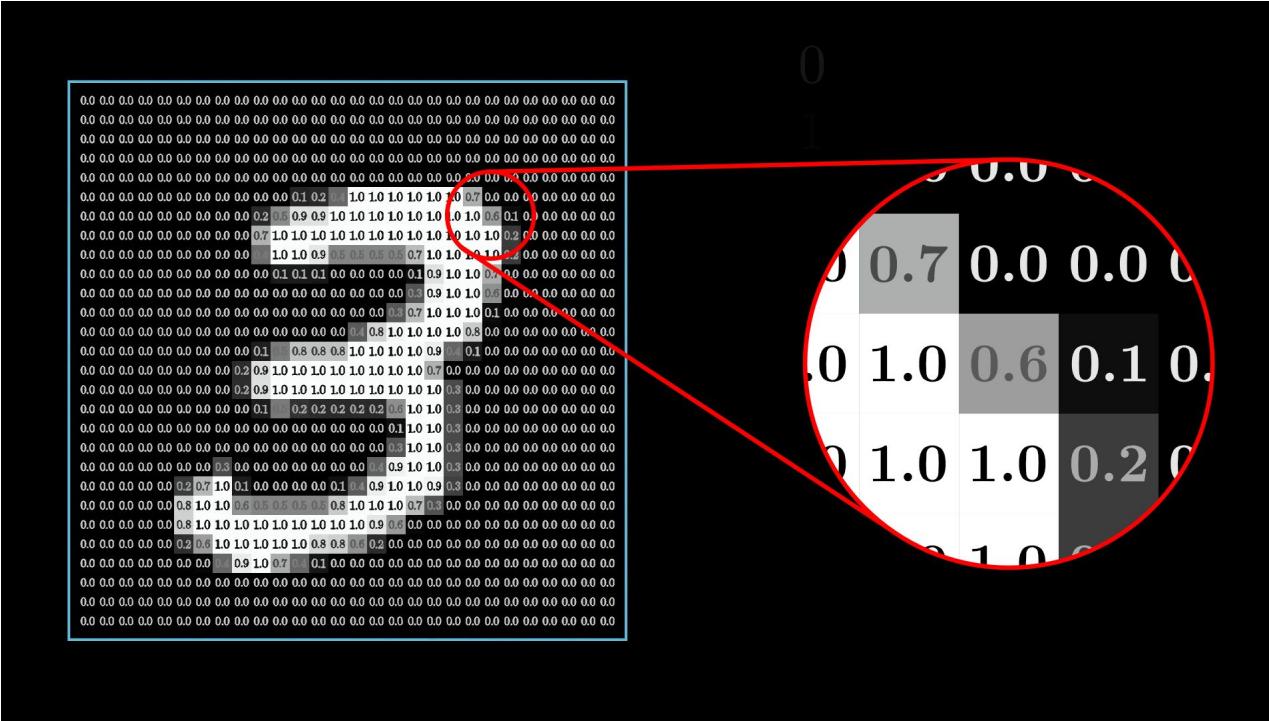
axis = 1

axis = 0

81	80	81	59
92	67	65	63
78	91	73	72
54	79	54	49
71	60	45	60

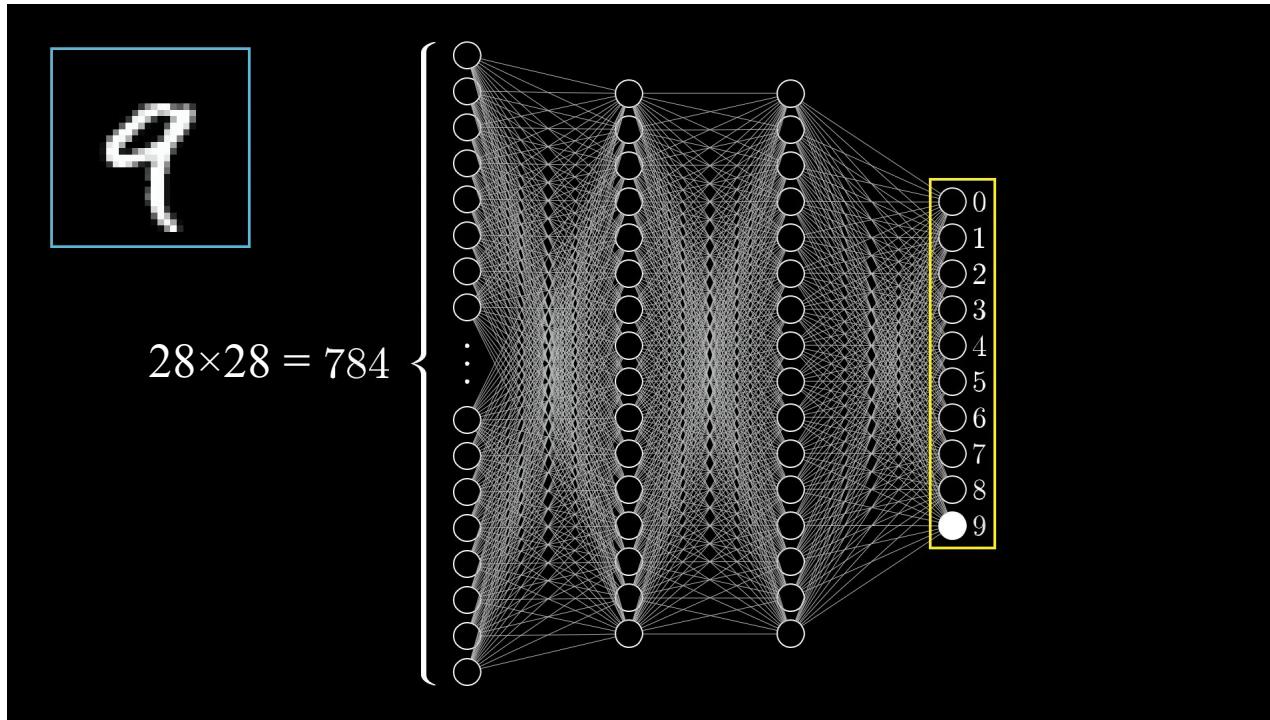
A diagram illustrating a 3D array structure. It shows a stack of five 2D arrays, each with four columns. The bottom-most 2D array has its vertical axis labeled 'axis = 0'. Above it, the stack itself is labeled 'axis = 1'. To the right of the stack, the depth dimension is labeled 'axis = 2'.

[참고] 흑백 이미지를 행렬로 표현



출처: <https://3b1b-posts.us-east-1.amazonaws.com//content/lessons/2017/neural-networks/pixel-values.png>

[참고] 딥러닝을 활용한 손글씨 분류



출처: <https://3b1b-posts.us-east-1.linodeobjects.com//content/lessons/2017/neural-networks/output-layer.png>

1차원 배열 생성

- 1차원 배열을 생성하고 클래스를 확인합니다.

```
>>> ar1 = np.array(object = [1, 2, 3]) # 리스트로 1차원 배열을 생성합니다.
```

```
>>> ar1 # ar1을 출력합니다. 원소를 가로 방향으로 출력합니다.
```

```
>>> type(ar1) # ar1의 클래스를 확인합니다. ar1의 클래스는 numpy.ndarray입니다.
```

- 1차원 배열의 형태, 원소 개수 및 원소 자료형을 확인합니다.

```
>>> ar1.shape # ar1의 형태(행 개수, 열 개수)를 확인합니다.
```

```
>>> ar1.size # ar1의 원소 개수를 확인합니다.
```

```
>>> ar1.dtype # ar1의 원소 자료형을 확인합니다. ar1의 원소 자료형은 numpy.int64입니다.  
[참고] Windows에서는 정수를 numpy.int32로 생성합니다.
```

1차원 배열의 원소 자료형 변환

- 배열은 모든 원소의 자료형이 같아지도록 자료형을 강제 변환합니다.

```
>>> a = [1, 2.0, '3'] # 다양한 자료형을 원소로 갖는 리스트를 생성합니다.
```

```
>>> ar1 = np.array(object = a) # 리스트로 배열을 생성하면 원소 자료형을 강제 변환합니다.  
[참고] 정수 < 실수 < 문자열 방향으로 변환합니다.
```

```
>>> ar1 # ar1을 출력합니다. ar1의 원소 자료형은 '<U32'입니다.  
[참고] '<U32'는 little-endian 32 character string이며, 'U'는 Unicode string입니다.
```

- 배열의 원소 자료형을 변환할 때 `astype()` 함수를 사용합니다.

```
>>> ar1.astype(float) # ar1의 원소 자료형을 실수로 변환합니다.
```

```
>>> ar1.astype(int) # ar1의 원소 자료형을 정수로 변환하려고 하면 '2.0' 때문에 에러를 반환합니다.
```

```
>>> ar1.astype(float).astype(int) # ar1의 원소 자료형을 실수로 변환하면 여전히 배열이므로 이어서  
정수로 변환할 수 있습니다.
```

[참고] 배열을 생성할 때 원소 자료형 지정

- 배열을 생성할 때 원소 자료형을 지정할 수 있습니다.

```
>>> np.array(object = a, dtype = float) # 배열의 원소 자료형을 실수로 지정합니다.
```

```
>>> np.array(object = a, dtype = int) # 배열의 원소 자료형을 정수로 지정합니다.
```

```
>>> np.array(object = a, dtype = str) # 배열의 원소 자료형을 문자열로 지정합니다.
```

```
>>> np.array(object = a, dtype = object) # 배열의 원소 자료형을 객체로 지정합니다.  
object는 리스트의 원소 자료형을 그대로 유지합니다.
```

- [참고] **pandas** 라이브러리에서 1차원 자료구조인 시리즈^{Series}를 생성할 때 원소에 문자열을 포함하고 있으면 시리즈의 원소 자료형을 object로 자동 변환합니다.
 - 따라서 시리즈는 원소 자료형이 섞여 있을 수 있으므로 주의해야 합니다.

간격이 일정한 배열 생성

- 간격이 일정한 실수를 원소로 갖는 1차원 배열을 생성합니다.

```
>>> np.arange(6) # 0부터 5까지 연속된 정수형 배열을 반환합니다.
```

```
>>> np.arange(start = 1, stop = 6) # 1부터 5까지 연속된 정수형 배열을 반환합니다.
```

```
>>> np.arange(start = 1, stop = 11, step = 2) # 1부터 10까지 홀수인 정수형 배열을 반환합니다.
```

```
>>> np.arange(start = 0, stop = 1, step = 0.1) # 0부터 1까지 0.1 간격의 연속된 실수형 배열을 반환합니다. [참고] 1을 포함하지 않습니다.
```

- 전체를 n-1개로 등분하는 (n개의 원소를 갖는) 1차원 배열을 생성합니다.

```
>>> np.linspace(start = 0, stop = 1, num = 1+10) # 0~1을 10등분하는 배열을 생성합니다.
```

```
>>> np.linspace(start = 80, stop = 75, num = 1+30) # 80~75를 30등분하는 배열을 생성합니다.
```

원소를 반복한 배열 생성

- 배열의 전체 원소를 두 번 반복합니다.

```
>>> np.tile(A = ar1, reps = 2)
```

- 배열의 각 원소를 두 번씩 반복합니다.

```
>>> np.repeat(a = ar1, repeats = 2)
```

- 배열 원소별로 반복할 횟수를 지정합니다.

```
>>> np.repeat(a = ar1, repeats = [3, 2, 1])
```

```
>>> np.repeat(a = ar1, repeats = range(3, 0, -1)) # 반복 횟수를 range() 함수로 지정할 수 있습니다.
```

1차원 배열 인덱싱 및 슬라이싱

- 배열 인덱싱은 인덱스(위치번호)로 원소를 선택합니다.

인덱스	0	1	2	3	4	5
원소	1	3	5	7	9	11

- 배열에 대괄호를 추가하고 대괄호 안에 선택할 원소의 인덱스를 지정합니다.

```
>>> ar1 = np.arange(start = 1, stop = 12, step = 2) # 1부터 11까지 홀수인 정수를 원소로 갖는 1차원 배열을 생성합니다.
```

```
>>> ar1[0] # ar1의 0번 인덱스(첫 번째) 원소를 선택합니다.
```

```
>>> ar1[1] # ar1의 1번 인덱스(두 번째) 원소를 선택합니다.
```

```
>>> ar1[-1] # ar1의 -1번 인덱스(마지막) 원소를 선택합니다.
```

1차원 배열 인덱싱 및 슬라이싱(계속)

- 배열 슬라이싱은 슬라이스를 지정하여 연속된 원소를 선택합니다.

```
>>> ar1[:3] # ar1의 0~2번 인덱스 원소를 선택합니다.
```

```
>>> ar1[3:] # ar1의 3번 인덱스 원소부터 마지막 원소까지 선택합니다.
```

```
>>> ar1[:] # ar1의 처음부터 마지막 원소까지 선택합니다.
```

- 대괄호 안에 정수를 원소로 갖는 리스트를 지정하면 해당 원소를 선택합니다.

```
>>> ar1[[3, 2, 1]] # 정수 스칼라 또는 슬라이스 대신 리스트를 지정하는 배열 인덱싱이 가능합니다.  
[주의] 리스트는 배열 인덱싱을 실행할 수 없습니다!
```

```
>>> ar1[[3, 3, 3]] # 리스트 원소를 반복하면 같은 원소를 여러 번 선택합니다.
```

2차원 배열 생성

- 같은 길이의 리스트를 원소로 갖는 리스트로 2차원 배열을 생성합니다.

```
>>> ar2 = np.array(object = [[1, 2, 3], [4, 5, 6]])
```

>>> ar2 # ar2를 출력합니다. 전체 원소를 2행 3열로 배치합니다.

>>> type(ar2) # ar2의 클래스를 확인합니다. ar2의 클래스는 numpy.ndarray입니다.

- 2차원 배열의 형태, 원소 개수 및 원소 자료형을 확인합니다.

>>> ar2.shape # ar2의 형태(행 개수, 열 개수)를 확인합니다.

>>> ar2.size # ar2의 원소 개수를 확인합니다.

>>> ar2.dtype # ar2의 원소 자료형을 확인합니다. ar2의 원소 자료형은 numpy.int64입니다.

배열의 재구조화

- `reshape()` 함수로 1차원 배열을 2차원 배열로 변환합니다.

```
>>> ar1 = np.arange(12) # 0부터 11까지 12개의 정수를 원소로 갖는 1차원 배열을 생성합니다.
```

```
>>> ar1.reshape(4, 3) # ar1을 4행 3열인 2차원 배열로 변환합니다. 원소 입력 방향은 가로입니다.  
[주의] 1차원 배열 원소 개수의 약수만 행 또는 열 개수로 설정할 수 있습니다.
```

```
>>> ar1.reshape(4, 3, order = 'F') # order = 'F'를 추가하면 원소 입력 방향을 세로로 변경합니다.  
[참고] order 매개변수에 전달하는 인수의 기본값은 'C'입니다.
```

- 2차원 배열 인덱싱 및 슬라이싱 실습을 위해 ar2를 생성합니다.

```
>>> ar2 = ar1.reshape(4, -1) # ar1을 4행 3열인 2차원 배열로 변환하고 ar2에 할당합니다.  
[참고] 열 위치에 -1을 지정하면 열 개수를 자동 계산합니다.
```

```
>>> ar2 # ar2는 4행 3열인 2차원 배열입니다.
```

```
>>> ar2.flatten() # 2차원 배열을 1차원 배열로 변환합니다.  
[참고] order 매개변수를 추가할 수 있습니다. (기본값: 'C')
```

[참고] 배열의 결합

- 원소 개수가 같은 1차원 배열을 열(가로) 방향으로 쌓은 2차원 배열을 생성합니다.

```
>>> np.column_stack(tup = (ar1, ar1)) # [주의] 함수의 괄호 안에 1차원 배열을 튜플 또는 리스트로  
    지정해야 합니다!
```

- 원소 개수가 같은 1차원 배열을 행(세로) 방향으로 쌓은 2차원 배열을 생성합니다.

```
>>> np.row_stack(tup = (ar1, ar1))
```

- 두 개 이상의 1차원 배열을 일렬로 결합하여 커다란 1차원 배열을 생성합니다.

```
>>> np.concatenate((ar1, ar1)) # [참고] 리스트를 + 연산자로 결합하는 것과 같은 동작입니다.
```

2차원 배열 인덱싱 및 슬라이싱

- 2차원 배열은 대괄호 안에 콤마를 추가하고 행/열 인덱스를 차례로 지정합니다.

```
>>> ar2[0, 0] # ar2의 1행 1열 원소를 선택합니다.
```

```
>>> ar2[1, 1] # ar2의 2행 2열 원소를 선택합니다.
```

- 2차원 배열의 슬라이싱을 실행합니다.

```
>>> ar2[0:2, 0:2] # ar2의 1~2행 1~2열 원소를 선택합니다.
```

```
>>> ar2[:, 1:3] # ar2의 전체 행 2~3열 원소를 선택합니다.  
[참고] 빈 콜론은 전체 행 또는 열을 선택합니다.
```

- 리스트를 사용한 배열 인덱싱도 가능합니다.

```
>>> ar2[:, [2, 1]] # ar2의 일부 열 순서를 변경합니다.  
[주의] 행과 열 순서를 함께 변경할 수 없습니다.
```

			axis = 1
	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8
3	9	10	11

배열 산술 연산

- 원소가 3개인 1차원 배열을 생성합니다.

```
>>> ar1 = np.array(object = range(3)); ar1
```

- 1차원 배열로 3행 1열의 2차원 배열을 생성합니다.

```
>>> ar2 = ar1.reshape(-1, 1); ar2
```

- 배열의 산술 연산을 실행할 때 차원이 같아지도록 브로드캐스팅broadcasting합니다.

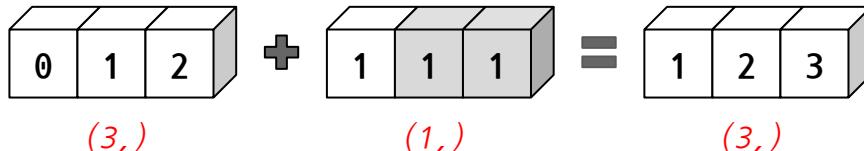
```
>>> ar1 + 1 # ar1의 각 원소에 1을 더합니다.
```

```
>>> ar2 * 2 # ar2의 각 원소에 2를 곱합니다.
```

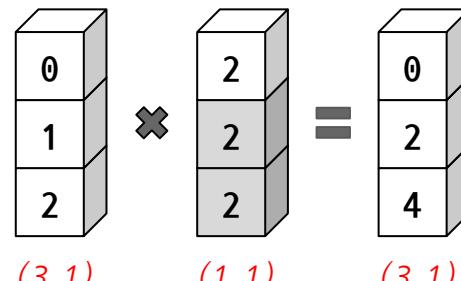
```
>>> ar1 + ar2 # ar1과 ar2를 더합니다.
```

[참고] 배열 브로드캐스팅

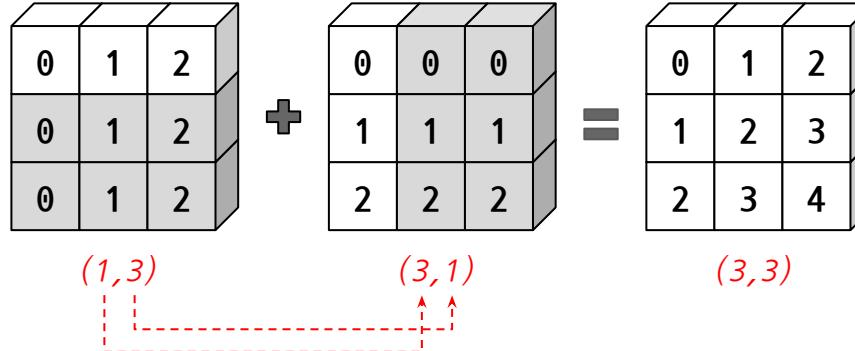
- 두 배열의 원소 개수가 다르면 원소 개수가 작은 배열을 확장합니다.

ar1 
$$\begin{array}{c} \text{ar1} \\ \begin{matrix} 0 & 1 & 2 \end{matrix} \end{array} + \begin{array}{c} \text{ar2} \\ \begin{matrix} 1 & 1 & 1 \end{matrix} \end{array} = \begin{array}{c} \text{Result} \\ \begin{matrix} 1 & 2 & 3 \end{matrix} \end{array}$$

[참고] 원소 개수가 같거나 1이어야 합니다!

ar2 
$$\begin{array}{c} \text{ar1} \\ \begin{matrix} 0 \\ 1 \\ 2 \end{matrix} \end{array} \otimes \begin{array}{c} \text{ar2} \\ \begin{matrix} 2 \\ 2 \\ 2 \end{matrix} \end{array} = \begin{array}{c} \text{Result} \\ \begin{matrix} 0 \\ 2 \\ 4 \end{matrix} \end{array}$$

(3,1) (1,1) (3,1)


$$\begin{array}{c} \text{ar1} \\ \begin{matrix} 0 & 1 & 2 \end{matrix} \end{array} + \begin{array}{c} \text{ar2} \\ \begin{matrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{matrix} \end{array} = \begin{array}{c} \text{Result} \\ \begin{matrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{matrix} \end{array}$$

(1,3) (3,1) (3,3)

[참고] 두 배열의 차원을 열 → 행 순으로 확인합니다.

[참고] 수학 관련 값과 함수

- 데이터 분석 과정에서 자주 사용하는 수학 관련 값과 함수를 정리한 표입니다.

구분	상세 내용	구분	상세 내용	구분	상세 내용
np.e	자연상수(2.7182)	np.sqrt()	제곱근	np.min()	최솟값
np.pi	파이(3.141592)	np.power()	거듭제곱	np.max()	최댓값
np.nan	결측값	np.exp()	자연상수 거듭제곱	np.mean()	평균
np.inf	무한대	np.log()	자연로그	np.median()	중위수
np.floor()	소수점 버림	np.sum()	모든 원소 덧셈	np.var()	분산
np.ceil()	소수점 올림	np.prod()	모든 원소 곱셈	np.std()	표준편차
np.round()	반올림	np.diff()	원소 차이	np.size()	원소 개수

pandas 라이브러리

pandas 라이브러리

- pandas 라이브러리의 함수로 다음과 같은 작업을 실행합니다.
 - 시리즈 ^{Series}: 원소를 세로 방향으로 정렬한 1차원 자료구조입니다.
 - 데이터프레임 ^{DataFrame}: 1차원 시리즈를 원소로 갖는 2차원 자료구조입니다.
 - 패널 ^{Panel}: 시간의 흐름에 따라 여러 데이터프레임을 중첩한 자료구조입니다.
 - 동일 관찰 대상이 시간의 흐름에 따라 어떻게 바뀌는지 확인할 수 있습니다.
- pandas 라이브러리를 호출합니다.

>>> import pandas as pd # pandas를 pd라는 가명으로 호출합니다.

>>> import numpy as np # [참고] numpy 라이브러리도 함께 호출하는 것이 좋습니다.

데이터프레임의 시각적 예시

- 데이터프레임은 행^{row}과 열^{column}을 갖는 2차원 자료구조입니다.
 - 열별 자료형이 다를 수 있으며, 행 또는 열을 하나만 선택하면 시리즈로 반환합니다.
 - 2차원 배열은 전체 원소 자료형이 같다는 점에서 데이터프레임과 다릅니다.

	문자열	문자열	문자열	정수형	실수형	
고객ID	고객명	성별	나이	키	...	
0001	정우성	M	48	186.9	...	
0002	장동건	M	49	182.2	...	
0003	김태희	F	41	163.4	...	
:	:	:	:	:	...	

[참고] 데이터 척도의 종류

범주형
데이터

명목척도 nominal scale

- 이름으로 구분하는 것에 의미 부여
 - 국가명, 학교명, 혈액형 등
- 수치가 아니므로 사칙연산 불가
- 등호, 빈도수와 백분율 계산

서열척도 ordinal scale

- 명목척도에 높낮이(서열) 부여
 - 학력, 등급, 직급 등
- 수치가 아니므로 사칙연산 불가
- 부등호/등호, 빈도수와 백분율 계산

연속형
데이터

등간척도 interval scale

- 높낮이에 등간격으로 수치 부여
 - 온도, 리커트 5점 척도 등
- 절대 영점 없음(ex. 0°C)
- 가감연산(+, -)과 평균 계산

비율척도 ratio scale

- 등간척도에 0과 비율 부여
 - 길이, 무게, 점수 등
- 절대 영점 있음(ex. 0km)
- 사칙연산(+, -, \times , \div)과 평균 계산

[참고] 데이터 척도의 특징

척도	동일 여부	크기 비교	차이 계산	비율 관계
명목척도	○	×	×	×
서열척도	○	○	×	×
등간척도	○	○	○	×
비율척도	○	○	○	○

[참고] 데이터프레임의 명칭

	X_1	X_2	...	X_p
1				
2				
3				
:				
n	셀 Cell			

행 Row 관측값 Observation
레코드 Record
사례 Case, Instance # 행 크기: n

열 Column 변수 Variable
특성 Feature
속성 Attribute # 열 크기: p

셀 값 $value$ 을 컬럼값, 특성값 또는 속성값이라고 합니다.

시리즈 생성

- 시리즈는 1차원 배열, 리스트 또는 딕셔너리로 생성하고 인덱스를 출력합니다.
- 1차원 배열을 생성합니다.

```
>>> ar1 = np.arange(start = 1, stop = 6, step = 2); ar1 # 1차원 배열은 원소를 가로 방향으로 출력합니다.
```

- 1차원 배열로 시리즈를 생성합니다.

```
>>> sr = pd.Series(data = ar1); sr # 시리즈는 인덱스index와 값value을 세로로 출력하고 맨 아래에 원소 자료형dtype을 추가로 출력합니다.
```

- sr의 클래스를 확인합니다.

```
>>> type(sr) # sr의 클래스를 확인합니다. sr의 클래스는 pandas.core.series.Series입니다.
```

[참고] 시리즈를 생성하는 다른 방법

- 시리즈를 생성할 때 인덱스를 지정할 수 있습니다.

```
>>> sr1 = pd.Series(data = [1, 2.0, '3'], index = ['a', 'b', 'c'])
```

```
>>> sr1 # sr1을 출력합니다. sr1의 원소 자료형은 object이며 원소별 자료형이 다를 수 있습니다.  
[참고] 문자열을 따옴표 없이 출력합니다.
```

```
>>> for i in sr1: # 반복문으로 sr1의 원소별 클래스를 출력합니다.
```

```
    print(type(i)) # 원소별 자료형이 제각각입니다.
```

- 딕셔너리로 시리즈를 생성합니다.

```
>>> pd.Series(data = {'a': 1, 'b': 3, 'c': 5}) # [주의] 딕셔너리의 키를 인덱스로 자동 적용  
하므로 별도로 인덱스를 지정할 수 없습니다!
```

시리즈 확인

- 시리즈의 형태, 원소 개수 및 원소 자료형을 확인합니다.

```
>>> sr.shape # sr의 형태(행 개수, 열 개수)를 확인합니다.
```

```
>>> sr.size # sr의 원소 개수를 확인합니다.
```

```
>>> sr.dtype # sr의 원소 자료형을 확인합니다. sr의 원소 자료형은 numpy.int64입니다.  
[참고] Windows에서는 정수를 numpy.int32로 생성합니다.
```

- 시리즈의 값(원소)과 인덱스를 확인합니다.

```
>>> sr.values # sr의 값(원소)을 확인합니다. sr의 값은 numpy.ndarray입니다.
```

```
>>> sr.index # sr의 인덱스를 확인합니다.  
[참고] 시리즈를 생성할 때 인덱스를 지정하지 않으면 정수 0부터 시작합니다.
```

시리즈 인덱스 변경

- 시리즈 인덱스의 클래스를 확인합니다.

```
>>> type(sr.index) # sr 인덱스의 클래스는 pandas.core.indexes.range.RangeIndex입니다.
```

- 시리즈 인덱스를 정수로 변경합니다.

```
>>> sr.index = [1, 2, 3]; sr # [참고] 시리즈 index 속성에 원소 개수와 같은 길이의 리스트를 할당하면  
리스트 원소로 인덱스를 변경합니다. 아울러 중복 인덱스를 허용합니다.
```

```
>>> type(sr.index) # sr 인덱스의 클래스는 pandas.core.indexes.numeric.Int64Index입니다.
```

- 시리즈 인덱스를 문자열로 변경합니다.

```
>>> sr.index = [ 'a', 'b', 'c' ]; sr # [참고] 시리즈 인덱스에 다양한 자료형을 원소로 갖는 리스트를  
할당할 수 있습니다. 예를 들어 [1, 1.0, '1']도 가능합니다.
```

```
>>> type(sr.index) # sr 인덱스의 클래스는 pandas.core.indexes.base.Index입니다.
```

시리즈 인덱스의 이해

- 시리즈 인덱스는 다음과 같은 특징을 갖습니다.
 - 시리즈를 생성할 때 인덱스를 지정하지 않으면 정수 0부터 시작합니다.
 - 시리즈 인덱스를 정수, 실수 또는 문자열로 변경할 수 있습니다.
 - 시리즈 인덱스는 충복을 허용합니다.
 - 시리즈에서 일부 원소를 선택하면 기존 인덱스를 유지합니다.
- 시리즈 인덱스는 위치번호 대신 행이름으로 인식하는 것이 좋습니다.
 - 시리즈를 인덱싱하는 방법은 기존 자료구조(예를 들어 리스트 또는 배열)에서 사용했던 정수 인덱스를 활용하는 방식과 행이름을 지정하는 방식이 있습니다.

시리즈 인덱싱 및 슬라이싱: iloc 인덱서

- `ilocinteger location` 인덱서는 정수 인덱스로 원소를 선택합니다.

```
>>> sr.iloc[0] # 대괄호 안에 정수 인덱스를 스칼라로 지정하면 해당 원소를 반환합니다.
```

- 리스트 또는 배열로 원소를 선택하는 방식을 배열 인덱싱^{Array Indexing}이라고 합니다.

```
>>> sr.iloc[[0]] # 대괄호 안에 정수 스칼라 대신 리스트로 지정하면 해당 원소를 시리즈로 반환합니다.
```

```
>>> sr.iloc[[0, 2]] # 연속하지 않는 인덱스를 함께 선택하려면 반드시 리스트로 지정해야 합니다.  
                    왼쪽 코드를 실행하면 sr의 0, 2번 인덱스 원소를 시리즈로 반환합니다.
```

- 정수 인덱스로 슬라이싱하면 연속된 원소를 시리즈로 반환합니다.

```
>>> sr.iloc[0:2] # sr의 0~1번 인덱스 원소를 시리즈로 반환합니다.  
                    [참고] iloc 인덱서는 끝(콜론 오른쪽) 인덱스를 포함하지 않습니다.
```

시리즈 인덱싱 및 슬라이싱: loc 인덱서

- `loclocation` 인덱서는 행이름으로 원소를 선택합니다.

```
>>> sr.loc['a'] # 대괄호 안에 행이름을 스칼라로 지정하면 해당 원소를 반환합니다.
```

- 행이름을 리스트로 지정하면 해당 원소를 선택하여 시리즈로 반환합니다.

```
>>> sr.loc[['a']] # 대괄호 안에 행이름을 스칼라 대신 리스트로 지정하면 해당 원소를 시리즈로 반환합니다.
```

```
>>> sr.loc[['a', 'c']] # 연속하지 않는 행이름을 함께 선택하려면 반드시 리스트로 지정해야 합니다.  
왼쪽 코드를 실행하면 sr의 행이름이 'a'와 'c'인 원소를 시리즈로 반환합니다.
```

- 행이름으로 슬라이싱하면 연속된 원소를 시리즈로 반환합니다.

```
>>> sr.loc['a':'c'] # sr의 행이름이 'a', 'b' 또는 'c'인 원소를 시리즈로 반환합니다.  
[참고] loc 인덱서는 끝(콜론 오른쪽) 행이름을 포함합니다!
```

[참고] 인덱서를 반드시 사용해야 하나?

- 인덱스가 정수 1부터 시작하는 시리즈를 생성하고 인덱싱합니다.

```
>>> sr2 = pd.Series(data = range(3), index = range(1, 4)); sr2
```

```
>>> sr2[0] # sr2의 행이름에 정수 0이 없으므로 에러를 반환합니다.
```

```
>>> sr2.loc[1] # loc 인덱서로 sr2에서 행이름이 정수 1인 원소를 선택합니다.
```

- 중복 인덱스를 갖는 시리즈를 생성하고 인덱싱합니다.

```
>>> sr3 = pd.Series(data = range(3), index = np.tile(1, 3)); sr3
```

```
>>> sr3.loc[1] # loc 인덱서로 sr3에서 행이름이 정수 1인 원소를 선택합니다.
```

```
>>> sr3.iloc[0] # iloc 인덱서로 sr3에서 0번 인덱스(첫 번째) 원소를 선택합니다.
```

시리즈의 불리언 인덱싱

- 시리즈로 비교 연산을 실행합니다.

```
>>> sr >= 3 # 왼쪽 코드를 실행하면 원소가 True 또는 False인 시리즈를 반환합니다.
```

- 조건을 만족하는 원소를 선택하는 방식을 불리언 인덱싱 Boolean Indexing이라고 합니다.

```
>>> sr.loc[sr >= 3] # 대괄호 안에 비교 연산 코드를 지정하면 True에 해당하는 원소를 선택합니다.  
[주의] iloc 인덱서를 사용하면 에러를 반환합니다.
```

- 두 개 이상의 조건을 고려하려면 비트 연산자를 추가합니다.

```
>>> sr.loc[sr >= 3 & sr < 5] # [주의] 비트 연산자의 우선순위가 비교 연산자보다 높기 때문에 왼쪽  
코드를 실행하면 정수 3과 sr의 원소 간 논리곱 연산을 먼저 실행합니다.
```

```
>>> sr.loc[(sr >= 3) & (sr < 5)] # [주의] 비트 연산자 양 옆 코드를 반드시 소괄호로 감싸야 합니다.
```

[참고] 비트 연산자

- 시리즈에서 불리언 인덱싱을 할 때 논리 연산자 대신 비트 연산자를 사용합니다.
 - [중요] 개별 조건을 반드시 소괄호로 감싸야 합니다.
- 비트 연산자는 정수를 2진수(비트)로 연산한 결과를 반환합니다.

연산자	상세 내용
&	<ul style="list-style-type: none">• [논리곱] 대응하는 비트가 모두 1이면 1, 아니면 0을 반환합니다. (and 연산)
	<ul style="list-style-type: none">• [논리합] 대응하는 비트가 하나라도 1이면 1, 아니면 0을 반환합니다. (or 연산)
^	<ul style="list-style-type: none">• [배타적 논리합] 대응하는 비트가 다르면 1, 같으면 0을 반환합니다. (xor 연산)
~	<ul style="list-style-type: none">• [논리부정] 비트가 1이면 0으로, 0이면 1로 반전합니다. (not 연산)

비트 연산자 사용법

- 부울형 시리즈 사이에 논리 연산자를 사용하면 에러를 반환합니다.

```
>>> sr >= 3 and sr < 5
```

```
>>> sr.iloc[0] >= 3 and sr.iloc[0] < 5 # 논리 연산자는 진리값 사이에서 정상적으로 동작합니다.  
따라서 numpy, pandas에서는 사용할 수 없습니다.
```

- 비트 연산자로 논리곱, 논리합, 배타적 논리합 또는 논리부정 연산을 실행합니다.

```
>>> (sr >= 3) & (sr < 5) # [논리곱] 대응하는 원소가 모두 True면 True, 아니면 False를 반환합니다.
```

```
>>> (sr >= 3) | (sr < 5) # [논리합] 대응하는 원소 중 하나라도 True면 True, 아니면 False를 반환합니다.
```

```
>>> (sr >= 3) ^ (sr < 5) # [배타적 논리합] 대응하는 원소가 다르면 True, 같으면 False를 반환합니다.
```

```
>>> ~ (sr >= 3) # [논리부정] True를 False로, False를 True로 반전합니다.
```

[참고] 얇은 복사

- 반복 가능한 객체를 얇은 복사하면 객체의 메모리 주소만 복사합니다.

```
>>> sr4 = sr1; sr4 # sr1을 얇은 복사한 sr4를 생성합니다.  
[참고] 반복 가능한 객체는 문자열, 리스트, 딕셔너리, 집합입니다.
```

- 두 변수가 가리키는 객체의 메모리 주소를 확인합니다.

```
>>> id(sr1); id(sr4) # sr1과 sr4가 가리키는 객체의 메모리 주소가 같습니다.  
[참고] id() 함수는 변수가 가리키는 객체의 메모리 주소를 반환합니다.
```

- sr4의 일부 원소를 변경하면 sr1의 원소도 바뀝니다.

```
>>> sr4.iloc[0] = 2; sr4 # sr4의 첫 번째 원소를 2로 변경합니다.
```

```
>>> sr1 # sr1의 첫 번째 원소도 바뀌었습니다.
```

[참고] 깊은 복사

- 깊은 복사는 객체 자체를 복사하는 것입니다.

```
>>> sr5 = sr1.copy(); sr5 # sr1을 깊은 복사한 sr5를 생성합니다.  
[참고] copy() 함수의 deep 매개변수에 전달하는 인수 기본값은 True입니다.
```

- 두 변수가 가리키는 객체의 메모리 주소를 확인합니다.

```
>>> id(sr1); id(sr5) # sr1과 sr5가 가리키는 객체의 메모리 주소가 다릅니다.
```

- sr5의 일부 원소를 변경해도 sr1의 원소는 바뀌지 않습니다.

```
>>> sr5.iloc[0] = 1; sr5 # sr5의 첫 번째 원소를 1로 변경합니다.
```

```
>>> sr1 # sr1의 첫 번째 원소는 바뀌지 않았습니다.  
[참고] 따라서 객체의 복사본이 필요하면 반드시 깊은 복사를 해야 합니다.
```

데이터프레임 생성

- 데이터프레임은 2차원 배열, 리스트 또는 딕셔너리로 생성하며 인덱스(행이름)와 컬럼명(열이름)을 갖습니다.
- 2차원 배열로 데이터프레임을 생성하고 클래스를 확인합니다.

```
>>> ar2 = np.arange(start = 1, stop = 7).reshape(3, -1)
```

```
>>> ar2 # ar2를 출력합니다. 원소를 가로 방향으로 입력한 3행 2열의 2차원 배열을 출력합니다.
```

```
>>> df = pd.DataFrame(data = ar2) # 2차원 배열로 데이터프레임을 생성합니다.
```

```
>>> df # df를 출력합니다. 데이터프레임은 왼쪽에 인덱스(행이름)와 위쪽에 컬럼명(열이름)을 출력합니다.  
[참고] 데이터프레임을 생성할 때 인덱스와 컬럼명을 지정하지 않으면 정수 인덱스로 자동 적용합니다.
```

```
>>> type(df) # df의 클래스를 확인합니다. df의 클래스는 pandas.core.frame.DataFrame입니다.
```

[참고] 데이터프레임을 생성하는 다른 방법

- 데이터프레임을 생성할 때 인덱스와 컬럼명을 리스트로 지정할 수 있습니다.

```
>>> df = pd.DataFrame(data = ar2, index = range(1, 4), columns = ['A', 'B'])  
          # 인덱스와 컬럼명을 리스트로 지정합니다.  
>>> df  
          [참고] range() 함수로 지정할 수 있습니다.
```

- 딕셔너리로 데이터프레임을 생성합니다.

```
>>> pd.DataFrame(data = {'A': [1, 2], 'B': [3, 4]}) # 딕셔너리의 키를 열이름으로 적용하고  
          # 값을 세로로 입력합니다.
```

- 딕셔너리를 원소로 갖는 리스트로 데이터프레임을 생성합니다.

```
>>> pd.DataFrame(data = [{ 'A': 1, 'B': 2}, # 딕셔너리의 키를 열이름으로 적용하고  
          # 값을 가로로  
          { 'A': 3, 'B': 4}])
```

데이터프레임 확인

- 데이터프레임의 다양한 정보를 확인합니다.

```
>>> df.shape # df의 형태(행 개수, 열 개수)를 확인합니다.
```

```
>>> df.dtypes # df의 열별 자료형을 확인합니다. [참고] 데이터프레임의 원소는 시리즈입니다.
```

```
>>> df.values # df의 셀 값을 확인합니다.
```

```
>>> df.index # df의 인덱스(행이름)를 확인합니다.
```

```
>>> df.columns # df의 컬럼명(열이름)을 확인합니다.
```

```
>>> df.info() # df의 정보를 확인합니다. (행 개수, 열 개수, 행이름, 열이름, 결측값 아닌 원소 개수 및 자료형)
```

데이터프레임 인덱싱 및 슬라이싱: iloc 인덱서

- iloc 인덱서에 정수 인덱스를 지정하여 행과 열을 선택합니다.

```
>>> df.iloc[0] # 대괄호 안에 정수 인덱스를 스칼라로 지정하면 해당 행을 시리즈로 반환합니다.  
[참고] 데이터프레임은 대괄호 안에 콤마를 생략하면 행을 선택합니다.
```

```
>>> df.iloc[[0]] # 대괄호 안에 스칼라 대신 리스트로 지정하면 해당 행을 데이터프레임으로 반환합니다.
```

```
>>> df.iloc[0:2] # 대괄호 안에 슬라이스를 지정하면 해당 행을 데이터프레임으로 반환합니다.  
[주의] 마지막 인덱스를 포함하지 않습니다.
```

```
>>> df.iloc[0:2, :] # 대괄호 안에 콤마를 추가하고 콤마 뒤에 선택할 열의 정수 인덱스를 지정합니다.  
[참고] 전체 열을 선택하려면 콤마 오른쪽에 빈 콜론을 추가합니다.
```

```
>>> df.iloc[0:2, 0:2] # 행 인덱스가 0~1인 행과 열 인덱스가 0~1인 열을 데이터프레임으로 반환합니다.
```

```
>>> df.iloc[:, [1, 0]] # 행은 전체, 열은 리스트의 원소(인덱스) 순으로 데이터프레임을 반환합니다.
```

데이터프레임 인덱싱 및 슬라이싱: loc 인덱서

- loc 인덱서에 행이름과 열이름을 지정하여 행과 열을 선택합니다.

```
>>> df.loc[1] # 대괄호 안에 행이름을 스칼라로 지정하면 해당 행을 시리즈로 반환합니다.  
[주의] 행이름에 0이 없으므로 0을 지정하면 에러를 반환합니다.
```

```
>>> df.loc[[1]] # 대괄호 안에 스칼라 대신 리스트로 지정하면 해당 행을 데이터프레임으로 반환합니다.
```

```
>>> df.loc[1:2] # 대괄호 안에 슬라이스를 지정하면 해당 행을 데이터프레임으로 반환합니다.  
[주의] 마지막 행이름을 포함합니다.
```

```
>>> df.loc[1:2, :] # 대괄호 안에 콤마를 추가하고 콤마 뒤에 선택할 열이름을 지정합니다.  
[참고] 전체 열을 선택하려면 콤마 오른쪽에 빈 콜론을 추가합니다.
```

```
>>> df.loc[1:2, 'A':'B'] # 행이름이 1~2인 행과 열이름이 A~B인 열을 데이터프레임으로 반환합니다.
```

```
>>> df.loc[:, ['B', 'A']] # 행은 전체, 열은 리스트의 원소(열이름) 순으로 데이터프레임을 반환합니다.
```

[참고] 인덱싱 방법에 따른 결과 비교

- 리스트, 배열, 시리즈 또는 데이터프레임은 대괄호 안에 입력하는 값에 따라 반환되는 클래스가 달라지므로 아래 내용을 반드시 숙지하시기 바랍니다!

구분	[스칼라]	[슬라이스]	[리스트]
리스트	원소	리스트	불가능
배열	원소	배열	배열
시리즈	원소	시리즈	시리즈
데이터프레임	시리즈	데이터프레임	데이터프레임

[참고] 인덱서 없는 인덱싱 결과 비교

- 시리즈와 데이터프레임에서 인덱서 없이 인덱싱할 때 주의해야 할 사항입니다.

시리즈		데이터프레임	
[행이름 스칼라]	원소를 반환	[열이름 스칼라]	열을 시리즈로 반환
[행이름 슬라이스]	원하는 결과를 얻을 수 없음	[열이름 슬라이스]	에러 발생
[행이름 리스트]	원소를 시리즈로 반환	[열이름 리스트]	열을 데이터프레임으로 반환
[부울형 시리즈]	원소를 시리즈로 반환	[부울형 시리즈]	행을 데이터프레임으로 반환

- 데이터프레임에서 조건을 만족하는 행을 필터링한 결과에서 일부 열을 선택하려면 반드시 인덱서를 사용해야 합니다. 슬라이싱할 때에도 인덱서가 필요합니다.

데이터 입출력

작업 경로

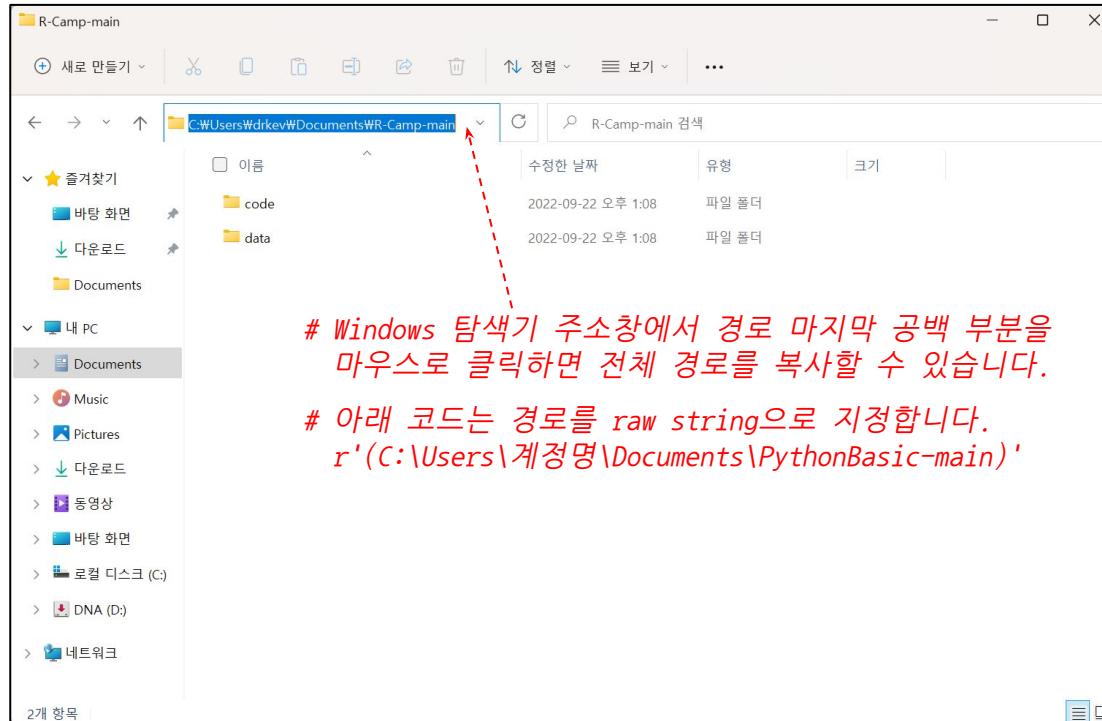
- 외부 파일을 입출력할 때 함수에 '경로/파일명.확장자' 형태의 문자열을 인수로 추가하는데, 만약 파일의 경로가 상당히 길면 코딩이 번거로울 수 있습니다.
 - 만약 현재 작업 경로^{working directory}에 있는 파일이라면 경로를 생략할 수 있습니다.
 - 작업 경로는 현재 작업 중인 Jupyter Notebook 파일에 설정된 파일 탐색 경로입니다.
- 현재 작업 경로는 아래와 같습니다.

Windows	"C:\Users\계정명\Documents\PythonBasic-main\code"
MacOS	"/Users/계정명/Documents/PythonBasic-main/code"

[주의] 경로 구분자로 Windows는 역슬래쉬(\), MacOS는 슬래쉬(/)를 사용합니다. 만약 작업 경로를 변경하는 함수에 역슬래쉬가 하나인 경로를 지정하면 에러를 반환하므로 역슬래쉬를 추가하거나 슬래쉬로 변경해야 합니다.

[참고] 경로를 문자열로 지정할 때 raw string으로 지정하면 역슬래쉬가 하나인 경로도 에러를 반환하지 않습니다. 문자열을 raw string으로 지정하려면 따옴표 앞에 r을 추가하고 따옴표 안 문자열을 괄호로 묶어주어야 합니다.

[참고] Windows 탐색기에서 경로 복사



절대 경로 vs 상대 경로

- 절대 경로는 경로의 시작과 끝을 모두 표기한 경로입니다.
 - 절대 경로는 현재 사용 중인 컴퓨터에서 유일한 경로입니다.
 - 작업 경로를 절대 경로로 설정하면 현재 작업 경로와 상관 없이 항상 결과가 같습니다.
 - 절대 경로의 길이가 상당히 긴 경우라면 코딩할 때 사용하기 불편합니다.
- 상대 경로는 현재 경로에서의 상대적인 위치를 의미합니다.
 - 상대 경로의 '..'은 현재 폴더, '...'은 상위 폴더를 의미합니다.
 - 상대 경로는 몇 글자만으로도 작업 경로를 설정할 수 있으므로 코딩할 때 편리합니다.
 - 그런데 현재 작업 경로에 따라 실행 결과가 달라지므로 주의를 기울여야 합니다.

일반 우편

서울특별시 강남구 역삼동 0번지
○○기업 △△팀 홍길동 과장님

사내 행낭

△△팀 홍길동 과장님

작업 경로 확인 및 변경

- 관련 라이브러리를 호출합니다.

```
>>> import os # os 라이브러리는 작업 경로 설정과 관련한 함수를 포함하고 있습니다.
```

- 현재 작업 경로를 확인합니다.

```
>>> os.getcwd() # 현재 작업 경로는 PythonBasic-main/code 폴더입니다.  
[참고] Jupyter Notebook 파일이 있는 폴더로 현재 작업 경로를 자동 설정합니다.
```

- 데이터 파일이 있는 폴더로 작업 경로를 변경합니다.(절대 경로)

```
>>> os.chdir(path = 'C:\\\\Users\\\\계정명\\\\Documents\\\\PythonBasic-main\\\\data')
```

- 현재 작업 경로를 다시 확인합니다.

```
>>> os.getcwd() # 현재 작업 경로는 PythonBasic-main/data 폴더입니다.
```

작업 경로 확인 및 변경(계속)

- 상위 폴더로 작업 경로를 변경합니다.(상대 경로)

```
>>> os.chdir(path = '..')
```

- 현재 작업 경로를 다시 확인합니다.

```
>>> os.getcwd() # 현재 작업 경로는 PythonBasic-main 폴더입니다.
```

- 데이터 파일이 있는 폴더로 작업 경로를 변경합니다.(상대 경로)

```
>>> os.chdir(path = './data') # 이번 강의에서는 상대 경로를 사용하여 작업 경로를 변경합니다.
```

- 현재 작업 경로를 다시 확인합니다.

```
>>> os.getcwd() # 현재 작업 경로는 PythonBasic-main/data 폴더입니다.
```

작업 경로에 있는 폴더명과 파일명 확인

- 현재 작업 경로에 있는 폴더명과 파일명을 출력합니다.

```
>>> os.listdir() # 현재 작업 경로에 있는 폴더명과 파일명을 리스트로 반환합니다.
```

```
>>> os.listdir(path = '../code') # [참고] path 매개변수에 경로명을 문자열로 지정하면 해당 경로에 있는 폴더명과 파일명을 리스트로 반환합니다.
```

- [Mac 사용자만!] 현재 작업 경로를 출력했을 때 오름차순 정렬하여 출력합니다.

```
>>> files = os.listdir() # [참고] Windows는 파일명과 폴더명을 오름차순 정렬하여 리스트로 반환하지만 MacOS는 문자열을 정렬하지 않고 반환하는 경우가 있습니다.
```

```
>>> files.sort() # files는 리스트이므로, 리스트의 원소를 오름차순 정렬하려면 sort() 함수를 실행합니다.  
[참고] sort() 함수는 실행 결과를 리스트에 반영하므로 재할당하지 않아도 됩니다.
```

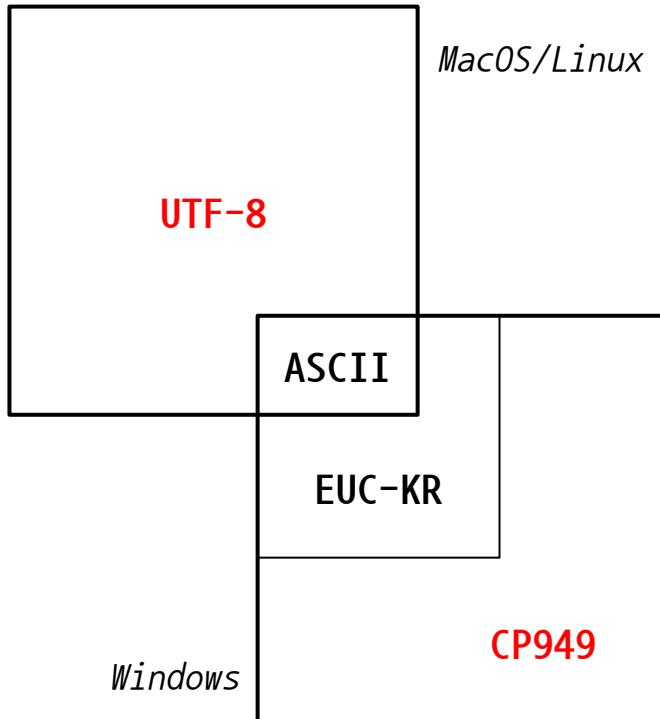
```
>>> files # files를 출력하면 오름차순 정렬되어 있습니다. 외부 파일을 읽을 때 파일명을 직접 타이핑하지 말고 출력된 파일명을 복사하여 붙여넣기 하면 타이핑 에러를 피할 수 있습니다.
```

[참고] 문자 인코딩의 이해

- 컴퓨터는 사람이 사용하는 자연어^{Natural Language} 문자를 이해할 수 없습니다.
 - 자연어 문자를 컴퓨터가 이해할 수 있는 코드로 변환하는 것을 인코딩^{Encoding}이라고 하며, 반대로 컴퓨터가 이해하는 코드를 자연어로 변환한 것을 디코딩^{Decoding}이라고 합니다.
 - 초창기에는 자연어를 2진수 코드로 변환하였지만, 나중에 16진수로 발전했습니다.
- 컴퓨터 운영체제별로 기본 문자 인코딩 방식이 다릅니다.

운영체제	문자 인코딩 방식	16진수 코드	자연어 문자
 	CP949 (EUC-KR) UTF-8	0xB0A1 U+AC00	 <i>Encoding</i> <i>Decoding</i>

[참고] 문자 인코딩 방식 관계도



구분	설명
ASCII	<ul style="list-style-type: none">미국에서 개발한 대표적인 영문 인코딩 방식입니다.알파벳, 숫자, 기호 등 128개 문자를 포함합니다.
UTF-8	<ul style="list-style-type: none">유니코드에 기반한 인코딩 방식입니다.(가변 길이)한글 완성형/조합형 모두 포함합니다.(국제 표준)
Unicode	<ul style="list-style-type: none">전 세계 모든 문자를 포함하는 인코딩 방식입니다.한 글자를 나타내기 위해 4 bytes를 사용합니다.
EUC-KR	<ul style="list-style-type: none">한글 초기 완성형 인코딩 방식입니다.(2350자)조합형은 다른 문자 체계와 호환할 수 없습니다.
CP949	<ul style="list-style-type: none">8822자를 추가한 통합 완성형 인코딩 방식입니다.Windows 점유율 높은 한국에서 사실상 표준입니다.

새 텍스트 파일 생성

- 쓰기모드로 텍스트 파일을 엽니다.

```
>>> file = open(file = 'test.txt', mode = 'w', encoding = 'UTF-8')  
# 현재 작업 경로에 있는 'test.txt' 파일을 쓰기모드로 엽니다. 파일이 없으면 새로 생성합니다.  
[참고] Windows 사용자는 encoding을 추가하지 않으면 CP949를 자동 적용합니다.
```

- 텍스트 파일에 문자열을 입력합니다.

```
>>> for i in range(1, 6): # 반복문을 실행하여 여러 문자열을 텍스트 파일에 입력합니다.  
    file.write(f'{i} 페이지 수집!\n')
```

- 텍스트 파일을 닫습니다.

```
>>> file.close() # 텍스트 파일을 닫으면 텍스트 파일을 저장합니다.
```

기존 텍스트 파일에 문자열 추가

- 추가모드로 텍스트 파일을 엽니다.

```
>>> file = open(file = 'test.txt', mode = 'a', encoding = 'UTF-8')
```

현재 작업 경로에 있는 'test.txt' 파일을 추가모드로 엽니다. 파일이 없으면 새로 생성합니다.

- 텍스트 파일에 문자열을 입력합니다.

```
>>> for i in range(11, 16):
```

```
    file.write(f'{i} 페이지 수집!\n')
```

- 텍스트 파일을 닫습니다.

```
>>> file.close()
```

텍스트 파일을 문자열로 읽기

- 문자열 읽기모드로 텍스트 파일을 엽니다.

```
>>> file = open(file = 'test.txt', mode = 'r', encoding = 'UTF-8')
```

현재 작업 경로에 있는 'test.txt' 파일을 문자열 읽기모드로 엽니다. 파일이 없으면 에러를 반환합니다.

- 텍스트 파일을 읽고 출력합니다.

```
>>> text = file.read(); text # read() 함수는 file을 사람이 읽을 수 있는 문자열str로 반환합니다.  
[참고] readlines() 함수는 리스트로 반환합니다.
```

```
>>> type(text) # text의 클래스를 확인합니다. text의 클래스는 str입니다.  
[참고] str은 사람이 읽을 수 있는 문자열입니다.
```

```
>>> text.encode(encoding = 'UTF-8') # 문자열을 인코딩하면 바이너리 코드로 변환합니다.
```

텍스트 파일을 바이너리로 읽기

- 바이너리 읽기모드로 텍스트 파일을 엽니다. # [참고] 바이너리 모드로 읽을 때 문자 인코딩 방식을 추가할 수 없습니다.

```
>>> file = open(file = 'test.txt', mode = 'rb')
```

현재 작업 경로에 있는 'test.txt' 파일을 바이너리 읽기모드로 엽니다. 파일이 없으면 에러를 반환합니다.

- 텍스트 파일을 읽고 출력합니다.

```
>>> text = file.read(); text
```

```
>>> type(text) # text의 클래스를 확인합니다. text의 클래스는 bytes입니다.  
[참고] bytes는 사람이 읽을 수 없는 코드입니다.
```

```
>>> text.decode(encoding = 'UTF-8') # 바이너리 코드를 디코딩하면 문자열로 변환합니다.
```

[주의] Windows에서 'test.txt' 파일을 생성할 때 encoding = 'UTF-8'를 추가하지 않았다면 반드시 EUC-KR 또는 CP949를 지정해야 합니다.

관련 라이브러리 호출

- 관련 라이브러리를 설치합니다.

```
>>> !pip install chardet # chardet 라이브러리를 설치합니다. 왼쪽 코드는 한 번만 실행합니다.  
[참고] Jupyter Notebook에서 라이브러리를 설치하려면 느낌표를 추가합니다.
```

```
>>> !pip install joblib # joblib 라이브러리를 설치합니다.
```

- 관련 라이브러리를 호출합니다.

```
>>> import chardet # 텍스트 파일의 문자 인코딩 방식을 확인하는 함수를 포함하고 있습니다.
```

```
>>> import numpy as np
```

```
>>> import pandas as pd
```

```
>>> import joblib # Python 객체를 압축 파일로 저장하고 호출하는 함수를 포함하고 있습니다.
```

xlsx 파일 입출력

- xlsx 파일을 읽고 데이터프레임을 생성합니다.

```
>>> df = pd.read_excel(io = 'KBO_Hitters_2021.xlsx', # [참고] 만약 코드 에러가 발생하면  
          sheet_name = 'Hitters', # 시트 인덱스 또는 시트명 하나만 지정합니다.  
          (기본값: 0)  
          skiprows = 4) # 열이름 위로 생략할 행 개수를 지정합니다.
```

- df를 출력합니다.

```
>>> df # [참고] 데이터프레임의 행과 열 개수가 많으면 모든 행과 열을 출력하지 않고 결과 창 크기에 따라 출력할  
      행과 열 개수를 자동 계산하며, 생략하는 행과 열 사이에 … 기호를 대신 출력합니다.
```

- 데이터프레임을 xlsx 파일로 저장합니다.

`index = None`을 추가하면 xlsx 파일을 저장할 때
인덱스를 삭제합니다.

```
>>> df.to_excel(excel_writer = 'test.xlsx', index = None)
```

[참고] 데이터프레임 행/열 최대 출력 옵션 변경

- 최대 행 출력 옵션을 확인하고 변경합니다.

```
>>> pd.get_option('display.max_rows') # 최대 행 출력 개수를 출력합니다. (기본값: 60)  
[참고] 최대 열 출력 개수는 max_columns를 사용합니다.
```

```
>>> pd.set_option('display.max_rows', None) # 최대 행 출력 개수에 None을 할당하면 전체 행을  
    출력합니다.
```

- df를 출력하면 전체 행을 출력합니다.

```
>>> df # [참고] 전체 행을 출력하도록 옵션을 변경하면 Jupyter Notebook이 느려지거나 다운될 수 있습니다.
```

- 최대 행 출력 옵션을 초기화합니다.

```
>>> pd.reset_option('display.max_rows') # [참고] 전체 옵션을 초기화하려면 'all'을 지정합니다.
```

```
>>> pd.options.display.max_rows # 왼쪽 코드를 실행하면 최대 행 출력 개수를 출력합니다.
```

데이터프레임 미리보기

- 데이터프레임의 처음 5행을 출력합니다.

```
>>> df.head() # [참고] n 매개변수에 입력하는 개수만큼 출력합니다. (기본값: 5)
```

```
>>> df.head(n = 10) # 데이터프레임의 처음 10행을 출력합니다.
```

- 데이터프레임의 마지막 5행을 출력합니다.

```
>>> df.tail()
```

- 데이터프레임을 무작위로 1행을 추출합니다.

```
>>> df.sample() # [참고] n 매개변수에 입력하는 개수만큼 출력합니다. (기본값: None)  
random_state 매개변수에 같은 정수를 지정하면 항상 같은 결과를 반환합니다.
```

csv 파일 입출력

- csv 파일명을 재사용할 수 있도록 변수에 할당합니다.

```
>>> file = 'KBO_Hitters_2021.csv'
```

- csv 파일을 읽고 데이터프레임을 생성합니다.

```
>>> df = pd.read_csv(filepath_or_buffer = file) # [주의] csv 파일의 문자 인코딩 방식이  
UTF-8이 아니면 에러를 반환합니다.
```

- csv 파일을 바이너리 모드로 읽습니다.

```
>>> text = open(file = file, mode = 'rb').read() # [참고] text는 bytes 코드이므로 출력하면  
사람이 읽을 수 없습니다.
```

- csv 파일의 문자 인코딩 방식을 확인합니다.

```
>>> chardet.detect(text) # [참고] 바이너리 코드의 문자 인코딩 방식과 신뢰도confidence를 반환합니다.
```

csv 파일 입출력(계속)

- csv 파일의 문자 인코딩 방식을 지정하여 읽습니다.

```
>>> df = pd.read_csv(filepath_or_buffer = file, encoding = 'EUC-KR')
```

- df의 인덱스를 정수 1부터 시작하도록 변경합니다.

```
>>> df.index = range(1, df.shape[0] + 1) # [참고] df.shape은 행과 열 개수를 튜플로 반환합니다.
```

```
>>> df.head() # df의 처음 5행을 출력합니다.
```

- 데이터프레임을 csv 파일로 저장합니다.

```
>>> df.to_csv(path_or_buf = 'test.csv', index = None, encoding = 'UTF-8')  
# [참고] csv 파일의 문자 인코딩  
방식을 지정할 수 있습니다.
```

압축 파일 입출력

- 데이터프레임을 포함한 모든 Python 객체를 외부 파일로 저장하려면 압축 파일을 사용하는 것이 좋습니다.
 - `joblib` 라이브러리는 Python 객체를 다양한 형태의 압축 파일로 저장하고, 압축 파일을 Python으로 호출하는 함수를 포함하고 있습니다.
- `df`를 확장자가 `z`인 압축 파일로 저장합니다.

```
>>> joblib.dump(value = df, filename = 'test.z')
```

- `z` 파일을 호출하고 데이터프레임에 할당합니다.

```
>>> df1 = joblib.load(filename = 'test.z')
```

[참고] 여러 파일 삭제

- 현재 작업 경로에서 'test'를 포함하는 파일명으로 리스트를 생성합니다.

```
>>> files = [file for file in os.listdir() if 'test' in file]
```

>>> files # files는 이번 시간에 생성한 4개 파일명을 원소로 갖습니다.

- 반복문으로 해당 파일을 모두 삭제합니다.

```
>>> for file in files:
```

os.remove(path = file) # [참고] os.remove() 함수는 한 번에 하나의 파일을 삭제합니다.

- 현재 작업 경로에 있는 폴더명과 파일명을 출력합니다.

```
>>> os.listdir() # 'test'를 포함하는 파일을 모두 삭제했습니다.
```

[참고] 프로야구 타자 스탯 데이터 간단 설명



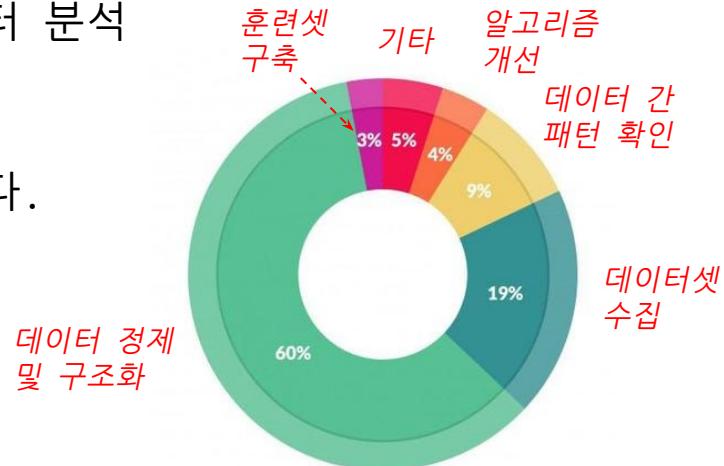
구분	상세 내용	
타석/타수	타수는 타석에서 볼넷, 사구, 희생타 등을 제외한 것	
안타	1루타, 2루타, 3루타, 홈런을 모두 합한 개수	
득점/타점	홈을 밟으면 득점, 안타를 쳐서 다른 선수가 득점하면 타점	
볼넷/삼진	볼 네 개면 1루에 걸어가는 볼넷, 스트라이크 세 개는 삼진 아웃	
BABIP	$(안타 - 홈런) \div (타수 - 삼진 - 홈런 - 희생타)$	# 인플레이 타율
타율	$안타 \div 타수$	# 타율 3할은 우수한 타자의 기준
출루율	$(안타 + 볼넷 + 사구) \div (타수 + 볼넷 + 사구 + 희생타)$	# 출루율 4할이 우수
장타율	$(1루타 + 2루타 \times 2 + 3루타 \times 3 + 홈런 \times 4) \div 타수$	# 장타율 5할이 우수
OPS	출루율 + 장타율	# OPS 0.9 이상이면 리그 수위타자로 인정
WAR	대체 선수 대비 승리 기여	# WAR 4 이상이면 핵심 선수로 인정

출처: http://blog.daum.net/dr_rafael/5012

데이터 전처리

데이터 전처리

- 데이터의 품질이 머신러닝 알고리즘보다 분석 결과에 더 직접적인 영향을 미치기 때문에 데이터 전처리의 중요성을 강조해도 지나치지 않습니다.
- 데이터 과학자는 데이터 수집/전처리에 데이터 분석 전 과정의 80% 시간을 사용한다고 합니다.
- 데이터 전처리는 아래 과정을 위해 수행합니다.
 - 결측값과 이상치를 탐지하고 처리합니다.
 - 다양한 파생변수를 생성합니다.
 - 데이터에 잠재된 패턴과 특징을 파악합니다.



출처: *Forbes*

데이터 전처리 항목

- 필요한 열을 선택합니다.
- 불필요한 열을 삭제합니다.
- 열이름을 변경합니다.
- 열별 자료형을 변환합니다.
- 조건에 맞는 행을 선택합니다.
- 행이름으로 행을 삭제합니다.
- 행이름을 초기화합니다.
- 결측값을 삭제하거나 대체합니다.

데이터 전처리 항목(계속)

- 파생변수를 생성합니다.
- 데이터프레임을 오름차순 또는 내림차순 정렬합니다.
- 그룹을 설정하고 집계함수로 데이터를 요약합니다.
- 데이터프레임의 형태를 변환합니다.(Long type ↔ Wide type)
- 두 데이터프레임을 행 또는 열 방향으로 결합합니다.
- 데이터프레임의 특정 열 기준 중복 여부를 확인하고 필요시 삭제합니다.
- 두 데이터프레임에서 서로 일치하는 행을 병합합니다.

실습 데이터셋 소개

- 공공데이터포털에서 제공하는 국토교통부 아파트매매 실거래자료 중 2021년 동안 서울특별시에서 거래된 아파트 가격 데이터를 2가지 형태로 제공합니다.
 - 'APT_Price_Seoul_2021.csv'
 - 'APT_Price_Seoul_2021.xlsx'
- 국내 대형 포털에서 제공하는 서울특별시 아파트단지별 상세정보를 2가지 형태로 제공합니다.
 - 'Naver_APT_Detail_Seoul.csv'
 - 'Naver_APT_Detail_Seoul.xlsx'

관련 라이브러리 호출

- 관련 라이브러리를 호출합니다.

```
>>> import os
```

```
>>> import chardet
```

```
>>> import numpy as np
```

```
>>> import pandas as pd
```

```
>>> import joblib
```

작업 경로 확인 및 변경

- 현재 작업 경로를 확인합니다.

```
>>> os.getcwd()
```

- data 폴더로 작업 경로를 변경합니다.

```
>>> os.chdir(path = '../data')
```

- 현재 작업 경로에 있는 폴더명과 파일명을 출력합니다.

```
>>> os.listdir()
```

xlsx 파일을 읽고 데이터프레임 생성

- 가격 xlsx 파일을 읽고 데이터프레임을 생성합니다. # 코드 앞에 %time을 추가하면 코드 실행 시간을 반환합니다.

```
>>> price = pd.read_excel(io = 'APT_Price_Seoul_2021.xlsx') # 약 5초 소요됩니다.
```

- price의 정보를 확인합니다.

```
>>> price.info() # 거래일의 자료형은 numpy.datetime64입니다. [참고] xlsx 파일은 셀서식을 유지합니다.
```

- price의 처음 5행을 출력합니다.

```
>>> price.head()
```

csv 파일을 읽고 데이터프레임 생성

- 가격 csv 파일의 문자 인코딩 방식을 확인합니다.

```
>>> file = 'APT_Price_Seoul_2021.csv' # 가격 csv 파일명을 재사용할 수 있도록 변수에 할당합니다.
```

```
>>> text = open(file = file, mode = 'rb').read() # csv 파일을 바이너리 모드로 읽습니다.
```

```
>>> chardet.detect(text[:100]) # csv 파일의 문자 인코딩 방식을 확인합니다.  
[참고] csv 파일 용량이 크면 오래 걸리므로 일부만 확인합니다.
```

- 가격 csv 파일을 읽고 데이터프레임을 생성합니다.

```
>>> price = pd.read_csv(filepath_or_buffer = file, encoding = 'UTF-8')
```

```
>>> price.dtypes # price의 열별 자료형을 확인합니다. 거래일과 거래금액의 자료형은 object입니다.  
[참고] csv 파일은 날짜 데이터와 콤마를 포함하는 숫자를 문자열로 읽습니다.
```

[참고] xlsx 파일 vs csv 파일

- xlsx 파일과 csv 파일을 읽을 때의 다음과 같은 장점과 단점이 있습니다.

구분	xlsx 파일	csv 파일
장점	문자 인코딩 방식 확인 불필요 날짜/시간 셀서식 유지	입출력 속도가 매우 빠름
단점	입출력 속도가 매우 느림	문자 인코딩 방식 확인 필요 날짜/시간을 문자열로 생성

- csv 파일의 셀서식 관련 단점을 보완하는 두 가지 방법입니다.
 - `pd.read_csv()` 함수의 `parse_dates` 매개변수에 날짜형으로 읽을 열이름을 지정합니다.
 - 날짜/시간 열이름을 먼저 문자형으로 읽고 나중에 날짜형으로 변환합니다.

[참고] csv 파일의 단점 해결

- 날짜 데이터는 날짜형, 콤마를 포함하는 숫자는 정수형으로 읽습니다.

```
>>> df = pd.read_csv(filepath_or_buffer = file,
                     encoding = 'UTF-8', # [참고] csv 파일의 문자 인코딩 방식이 UTF-8이면
                     parse_dates = ['거래일'], # 문자열이 날짜 기본형인 열이름을 지정합니다.
                     thousands = ',', # 천 단위 구분자를 문자열로 지정합니다.
                     # [참고] pd.read_excel() 함수에도 적용할 수 있습니다.
```

- df의 열별 자료형을 확인합니다.

```
>>> df.dtypes # df의 열별 자료형을 확인합니다. 거래일은 numpy.datetime64, 거래금액은 numpy.int64입니다.
```

열이름으로 열 선택

- 열을 선택할 때 대괄호 안에 열이름을 나열합니다.

```
>>> price['지역코드'] # 열이름을 문자열 스칼라로 지정하면 시리즈로 반환합니다.  
[참고] 열이름을 리스트로 지정하면 데이터프레임으로 반환합니다.
```

```
>>> price[['아파트', '지역코드']] # 2개 이상의 열을 선택하려면 반드시 리스트로 지정해야 합니다.  
[참고] 열이름 순서를 변경하면 위치를 바꿔서 반환합니다.
```

- 슬라이스로 연속된 열을 선택하려면 loc 인덱서를 추가합니다.(배열 인덱싱)

```
>>> price.loc[:, '거래일':'거래금액'] # [주의] loc 인덱서를 생략하면 에러를 반환합니다.
```

- 조건을 만족하는 열을 선택합니다.(불리언 인덱싱)

```
>>> price.loc[:, price.dtypes == np.int64] # 자료형이 정수인 열만 선택합니다.
```

열이름으로 열 삭제

- 삭제할 열이름을 `drop()` 함수의 `columns` 매개변수에 리스트로 지정합니다.

```
>>> price.drop(columns = ['지역코드']) # [주의] columns 매개변수를 생략하면 반드시 axis = 1을  
추가해야 합니다. 추가하지 않으면 인덱스에서 찾습니다.
```

- `price`의 처음 5행을 출력합니다.

```
>>> price.head() # price는 여전히 지역코드를 포함하고 있습니다.
```

- 열을 삭제하고 재할당하면 데이터프레임에서 해당 열을 삭제합니다.

```
>>> price = price.drop(columns = ['지역코드'])
```

```
>>> price.head() # price에 지역코드가 없습니다.
```

열이름 변경

- 일부 열이름을 변경한 결과를 출력합니다.

```
>>> price.rename(columns = {'시도명': '시도', '시군구': '자치구'})  
# '기존 이름': '새 이름'으로 설정합니다.  
[주의] columns 매개변수를 생략하면 안됩니다!
```

- price의 열이름을 출력합니다.

```
>>> price.columns
```

- 전체 열이름을 변경합니다. # [주의] 데이터프레임의 열이름과 원소 개수가 같은 리스트를 지정해야 합니다.

```
>>> price.columns = ['아파트', '시도', '자치구', '읍면동', '지번',  
'거래일', '전용면적', '층', '거래금액']
```

```
>>> price.head() # 열이름이 변경되었습니다.
```

열별 자료형 변환

- 시리즈의 원소 자료형을 변환합니다.

```
>>> price['거래금액'].astype(dtype = float) # [참고] 거래금액 원소에 콤마가 있으므로 실수로  
변환할 수 없습니다.
```

- 거래금액 원소에 있는 콤마를 삭제하고 재할당합니다. # [참고] 시리즈.str에 문자열을 처리하는
다양한 함수가 있습니다.

```
>>> price['거래금액'] = price['거래금액'].str.replace(pat = ',', repl = '')
```

- 시리즈의 원소 자료형을 실수형으로 변환하고 재할당합니다.

```
>>> price['거래금액'] = price['거래금액'].astype(dtype = float)
```

```
>>> price.dtypes # price의 열별 자료형을 확인합니다. 거래금액을 numpy.float64로 변환했습니다.
```

열별 자료형 변환(계속)

- 데이터프레임의 열별로 자료형 변환 방법을 딕셔너리로 지정합니다.

```
>>> price = price.astype(dtype = {'거래일': np.datetime64, '총': float})
```

```
>>> price.dtypes # 거래일을 numpy.datetime64, 총을 numpy.float64로 변환했습니다.
```

- 데이터프레임의 여러 열을 같은 자료형으로 일괄 변환합니다.

```
>>> cols = ['총', '거래금액'] # 정수형으로 변환할 열이름으로 리스트를 생성합니다.
```

```
>>> price[cols] = price[cols].astype(dtype = int) # 선택한 열의 자료형을 일괄 변환합니다.
```

```
>>> price.dtypes # 총과 거래금액을 numpy.int64로 변환했습니다.
```

[참고] 문자열을 날짜형으로 변환

- 날짜 기본형이 아닌 문자열 리스트로 시리즈를 생성합니다.

```
>>> birth = pd.Series(data = ['2000년 1월 1일']); birth
```

- birth를 날짜형으로 변환하려고 하면 에러를 반환합니다.

```
>>> birth.astype(dtype = np.datetime64)
```

- **to_datetime()** 함수는 문자열을 날짜형으로 변환합니다. # [주의] 반드시 *format* 매개변수에
날짜 포맷을 지정해야 합니다.

```
>>> birth = pd.to_datetime(arg = birth, format = '%Y년 %m월 %d일')
```

```
>>> birth.iloc[0] # birth의 0번 인덱스 원소를 출력합니다.  
[참고] 날짜 시간 데이터는 '년-월-일 시:분:초.마이크로초' 형태로 출력합니다.
```

[참고] 날짜/시간 관련 주요 포맷

- 아래 표는 날짜/시간 포맷을 정리한 것입니다. # [참고] 로 캐일에 따라 출력 결과가 달라집니다.

포맷	상세 내용	예시	포맷	상세 내용	예시
%a	요일(영어 약자)	Sat	%M	분(정수)	01
%A	요일(영어 전체)	Saturday	%p	AM/PM 표기	PM
%b	월(영어 약자)	Jan	%S	초(정수)	01
%B	월(영어 전체)	January	%w	요일(정수)	6
%c	요일 날짜 시간	Sat Jan 1 13:01:01 2000	%x	날짜만 출력	01/01/00
%d	일(정수)	01	%X	시간만 출력	13:01:01
%H	시(정수-24시간)	13	%y	연도(2자리)	00
%I	시(정수-12시간)	01	%Y	연도(4자리)	2000
%m	월(정수)	01	%Z	타임존 출력	KST

[참고] 날짜 시간 데이터 연산

- 현재 날짜 시간 데이터를 생성합니다.

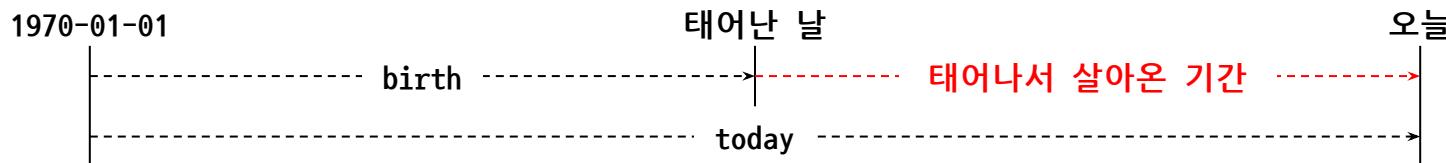
```
>>> today = pd.Timestamp.today()
```

```
>>> today.timestamp() # 날짜 시간 데이터의 timestamp() 함수를 실행하면 Origin(1970-01-01 00:00:00)부터  
누적된 초를 소수점 여섯째 자리 마이크로초(백만 분의 1초) 단위로 반환합니다.
```

- 날짜 시간 데이터는 실수이므로 산술 연산이 가능합니다.

```
>>> dtGap = today - birth.iloc[0] # 태어나서 현재까지 살아온 기간을 dtGap에 할당합니다.
```

```
>>> dtGap.days # dtGap에서 경과일수를 출력합니다.
```



[참고] 날짜 분해 함수

- 날짜 변수에서 년, 월, 일을 정수형 시리즈로 반환합니다.

```
>>> price['거래일'].dt.year # 거래일에서 년year을 정수형 시리즈로 반환합니다.
```

```
>>> price['거래일'].dt.month # 거래일에서 월month을 정수형 시리즈로 반환합니다.
```

```
>>> price['거래일'].dt.day # 거래일에서 일day을 정수형 시리즈로 반환합니다.
```

- 날짜 변수에서 요일을 문자형 시리즈로 반환합니다.

```
>>> price['거래일'].dt.day_name() # 거래일에서 영문 요일을 문자형 시리즈로 반환합니다.
```

```
>>> price['거래일'].dt.day_name(locale = 'ko_KR') # locale 매개변수에 'ko_KR'을 지정하면  
한글 요일을 반환합니다.
```

조건에 맞는 행 선택: 연속형 변수

- 거래금액이 100억 이상인 행을 선택하여 df1에 할당합니다.

```
>>> df1 = price[price['거래금액'] >= 10000000] # [참고] 불리언 인덱싱으로 행을 선택할 때 loc  
인덱서를 생략할 수 있습니다.
```

```
>>> df1.head() # df1의 처음 5행을 출력합니다. [참고] 행이름이 0부터 시작하지 않습니다.
```

- 거래금액이 100억 미만이고 60총 이상인 행을 선택하여 df2에 할당합니다.

```
>>> df2 = price[(price['거래금액'] < 10000000) & (price['총'] >= 60)]  
# 시리즈로 논리곱 연산을 실행할 때 논리 연산자를 사용하면 에러가  
발생하며 비트 연산자를 대신 사용해야 합니다.  
>>> df2.head()  
# [주의] 비트 연산자 앞뒤 코드를 반드시 소괄호로 감싸야 합니다.
```

[참고] 시리즈의 비교 연산 함수

- 시리즈에서 비교 연산을 실행하는 함수는 불리언 시리즈를 반환합니다.

```
>>> price['총'].gt(60).sum() # 총이 60 초과면 True, 아니면 False인 부울형 시리즈를 반환합니다.  
[참고] 마지막에 추가한 sum() 함수는 True의 개수를 반환합니다.
```

```
>>> price['총'].ge(60).sum() # 총이 60 이상이면 True, 아니면 False인 부울형 시리즈를 반환합니다.
```

```
>>> price['총'].lt(60).sum() # 총이 60 미만이면 True, 아니면 False인 부울형 시리즈를 반환합니다.
```

```
>>> price['총'].le(60).sum() # 총이 60 이하이면 True, 아니면 False인 부울형 시리즈를 반환합니다.
```

```
>>> price['총'].eq(60).sum() # 총이 60이면 True, 60이 아니면 False인 부울형 시리즈를 반환합니다.
```

```
>>> price['총'].ne(60).sum() # 총이 60이 아니면 True, 60이면 False인 부울형 시리즈를 반환합니다.
```

```
>>> price['거래금액'].lt(1000000) & price['총'].ge(60) # 비교 연산 함수는 시리즈이므로  
소괄호로 감싸지 않아도 됩니다.
```

조건에 맞는 행 선택: 범주형 변수

- 자치구가 '강남구'인 행을 선택합니다.

```
>>> price[price['자치구'].eq('강남구')]
```

- 자치구가 '강남구' 또는 '서초구'인 행을 선택합니다.

```
>>> price[price['자치구'].eq('강남구') | price['자치구'].eq('서초구')]
```

- `isin()` 함수는 시리즈 원소가 리스트에 있으면 True, 없으면 False를 반환합니다.

```
>>> price[price['자치구'].isin(values = ['강남구', '서초구'])]
```

- `str.contains()` 함수는 원소에 패턴이 있으면 True, 없으면 False를 반환합니다.

```
>>> price[price['자치구'].str.contains(pat = '강남|서초')]
```

[참고] 시리즈를 문자열로 처리하는 주요 함수

- 시리즈를 문자열로 처리할 때 str을 추가합니다. # [주의] str은 시리즈 원소를 문자열로 처리하기 때문에 문자열이 아니면 에러를 반환합니다.

```
>>> addr = pd.Series(data = ['서울특별시 강남구', '경기도 성남시 분당구'])
```

```
>>> addr.str.replace(pat = ' ', repl = '') # 문자열(원소)마다 지정한 패턴을 변경합니다.
```

```
>>> addr.str.split(pat = ' ', expand = True) # 문자열(원소)을 지정한 패턴으로 분리한 결과를 데이터프레임으로 반환합니다.
```

```
>>> addr.str.find(sub = '시') # 문자열(원소)마다 지정한 패턴이 있으면 시작 인덱스를 반환합니다.  
[참고] 지정한 패턴이 없으면 -1을 반환합니다.
```

```
>>> addr.str.slice(start = 0, stop = 2) # 문자열(원소)을 지정한 인덱스로 자릅니다.
```

```
>>> addr.str.extract(pat = '([가-힝]+구)') # 문자열(원소)마다 지정한 패턴에 해당하는 문자열을 추출합니다. [주의] 패턴을 소괄호로 감싸야 합니다.
```

행이름으로 행 삭제

- df1의 처음 5행을 출력합니다.

```
>>> df1.head()
```

- 삭제할 행이름을 `drop()` 함수의 `index` 매개변수에 리스트로 지정합니다.

```
>>> df1.drop(index = [10512, 10513]) # [주의] 행이름에 없는 값을 지정하면 에러를 반환합니다.
```

- df1의 인덱스를 출력합니다.

```
>>> df1.index
```

- 인덱스를 슬라이싱한 코드로 행이름 리스트를 대신할 수 있습니다.

```
>>> df1.drop(index = df1.index[0:2])
```

행이름 초기화

- 행이름을 초기화한 결과를 출력합니다.

```
>>> df1.reset_index(drop = True) # drop = True를 추가하지 않으면 기존 행이름을 열로 추가합니다.
```

- 특정 열을 인덱스로 지정합니다.

```
>>> df1 = df1.set_index(keys = '아파트')
```

```
>>> df1.head() # 아파트를 행이름으로 설정하고 열에서 삭제합니다.
```

- 행이름을 초기화하면서 기존 행이름을 열로 추가합니다.

```
>>> df1 = df1.reset_index()
```

```
>>> df1.head() # 기존 행이름이었던 아파트를 열로 추가합니다.
```

결측값 처리: 단순대체

- 데이터프레임의 셀 값별 결측값 여부를 반환합니다.

```
>>> price.isna() # isna() 함수는 결측값 여부를 True 또는 False로 반환합니다.  
[참고] isnull() 함수는 isna() 함수의 alias이므로 같은 동작을 수행합니다.
```

- 데이터프레임의 열별 결측값 개수를 계산합니다.

```
>>> price.isna().sum() # 지번에 결측값이 14개 있습니다.
```

- 지번이 결측값인 행을 선택합니다.

```
>>> price[price['지번'].isna()] # [참고] 결측값의 원본을 탐색하면 서초포레스타2단지의 지번은 384,  
힐스테이트 서초 젠트리스의 지번은 557입니다.
```

- 지번이 결측값인 행에서 결측값을 빈 문자열로 대체한 결과를 반환합니다.

```
>>> price[price['지번'].isna()].fillna(value = '')
```

[참고] 결측값을 이전 셀 값으로 채우기

- 현재 작업 경로에 있는 폴더명과 파일명을 출력합니다.

```
>>> os.listdir()
```

- 법정동별 평균 거래금액xlsx 파일을 읽고 데이터프레임을 생성합니다.

```
>>> meanPrice = pd.read_excel(io = 'APT_Mean_Price_Dong_2021.xlsx')
```

```
>>> meanPrice # 자치구에 결측값이 많습니다.  
[참고]xlsx에서 병합한 셀은 처음 값만 제대로 읽고 나머지는 결측값으로 대체합니다.
```

- 자치구에 있는 결측값을 이전 셀 값으로 채웁니다.

```
>>> meanPrice.fillna(method = 'ffill') # [참고] method 매개변수에 'bfill'을 지정하면 결측값을  
이후 셀 값으로 채웁니다.
```

[참고] 데이터프레임에서 결측값이 있는 행 삭제

- 결측값이 있는 행을 삭제하기 전에 변수의 중요도를 판단하는 것이 좋습니다.
 - 아래 데이터프레임에서 결측값이 있는 행을 삭제하면 모든 행을 삭제합니다.
 - 만약 지번이 중요한 변수라면 정상값을 갖는 행도 삭제하므로 행 개수가 감소합니다.
 - 지번에서 결측값이 있는 행만 삭제하고 적당한 값으로 결측을 대체하는 것이 좋습니다.

아파트	시도	자치구	읍면동	지번	...
래미안	서울특별시	강남구	압구정동	NaN	...
힐스테이트	서울특별시	강동구	NaN	2	...
아이파크	서울특별시	NaN	성북동	4	...
:	:	:	:	:	...

결측값 처리: 행 삭제

- 데이터프레임에서 결측값이 있는 모든 행을 삭제한 결과를 반환합니다.

```
>>> price.dropna() # [참고] dropna() 함수의 subset 매개변수에 일부 열이름을 지정하면 해당 열에서 결측값이 있는 행을 삭제합니다.
```

- 지번에서 결측값이 없는 행을 선택한 결과를 반환합니다.

```
>>> price[~price['지번'].isna()] # [참고] ~ 연산자는 isna() 함수 실행 결과를 반전합니다.
```

```
>>> price[price['지번'].notna()] # notna() 함수는 결측값이 아니면 True, 결측값이면 False를 반환합니다.
```

- 지번에서 결측값이 없는 행을 선택하고 price에 재할당합니다.

```
>>> price = price[price['지번'].notna()]
```

```
>>> price.shape[0] # price의 행 개수를 확인합니다. price의 행 개수가 감소했습니다. (43447 → 43433)
```

파생변수 생성: 연속형 변수

- 거래금액을 전용면적으로 나누고 3.3을 곱한 평당금액을 생성합니다.

```
>>> price['평당금액'] = price['거래금액'] / price['전용면적'] * 3.3
```

```
>>> price.head() # [참고] 데이터프레임에 없는 열이름으로 시리즈를 생성하면 데이터프레임의 가장 오른쪽에 해당 열이름을 추가합니다.
```

- 거래금액을 10000으로 나누면 단위가 만원에서 억원으로 바뀝니다.

```
>>> price['거래금액'] = price['거래금액'] / 10000
```

- pandas 옵션에서 실수를 출력하는 소수점 자리수를 3으로 설정합니다.

```
>>> pd.options.display.precision = 3
```

```
>>> price.head() # [참고] 평당금액을 소수점 셋째 자리까지 출력합니다.
```

파생변수 생성: 범주형 변수

- 연속형 변수를 기준에 따라 2개 이상의 구간으로 나누어 범주형 변수로 변환하는 것을 구간화^{binning}라고 합니다.
 - 구간화를 통해 이상치와 비선형 문제를 해결하고, 결측값을 쉽게 처리할 수 있습니다.
- 평당금액을 '5천 이상', '5천 미만'으로 구분한 금액구분을 생성합니다.

```
>>> locs = price['평당금액'].ge(5000) # 평당금액이 5000 이상이면 True, 아니면 False인 원소를  
# 갖는 부울형 시리즈를 locs에 할당합니다.  
>>> np.where(locs, '5천 이상', '5천 미만') # np.where() 함수는 조건이 True일 때 두 번째 인수,  
# False일 때 세 번째 인수를 반환합니다.  
>>> price['금액구분'] = np.where(locs, '5천 이상', '5천 미만')  
>>> price.head() # [참고] np.where() 함수에 조건만 지정하면 True인 인덱스를 반환합니다.
```

[참고] pandas에서 추천하는 코딩 방식

- price를 깊은 복사한 imsi를 생성합니다.

```
>>> imsi = price.copy() # [주의] copy()를 생략하면 객체의 메모리 주소를 복사하는 얕은 복사가 됩니다.  
따라서 imsi를 변경하면 price도 함께 변경되므로 주의해야 합니다!
```

- 평당금액이 5000 이상인 행의 새 변수(금액구분2)에 '5천 이상'을 할당합니다.

```
>>> imsi.loc[imsi['평당금액'].ge(5000), '금액구분2'] = '5천 이상'
```

```
>>> imsi.head() # 오른쪽에 금액구분2를 추가하고 평당금액이 5000 이상이면 '5천 이상', 아니면 결측값으로  
채웁니다. 이어서 아래 코드를 실행하면 결측값을 '5천 미만'으로 채웁니다.
```

- 평당금액이 5000 미만인 행의 새 변수(금액구분2)에 '5천 미만'을 할당합니다.

```
>>> imsi.loc[imsi['평당금액'].lt(5000), '금액구분2'] = '5천 미만'
```

```
>>> imsi.head() # 금액구분2에 결측값이 없습니다.
```

[참고] 구간화 함수

- 연속형 변수를 세 개 이상으로 분리할 때 `np.where()` 함수를 중첩합니다.

```
>>> np.where(price['평당금액'].ge(10000),
```

'1억 이상', # 첫 번째 조건을 만족하면 '1억 이상'을 반환하고, 그렇지 않으면 두 번째 조건
만족 여부를 확인합니다.

```
np.where(price['평당금액'].ge(5000),
```

'5천 이상', # 두 번째 조건 만족 여부에 따라 '5천 이상' 또는 '5천 미만'을
반환합니다.

'5천 미만')) # 코드를 구성하는 요소는 단순하지만 가독성이 좋지 않습니다.

[참고] 구간화 함수(계속)

- 연속형 변수를 세 개 이상으로 분리할 때 `np.select()` 함수를 사용합니다.

```
>>> np.select(condlist = [price['평당금액'].ge(10000),  
                           price['평당금액'].ge(5000),  
                           price['평당금액'].lt(5000)],  
             choicelist = ['1억 이상', '5천 이상', '5천 미만'])  
# 조건에 맞는 값을 반환합니다.
```

파생변수 생성: 문자형 변수 결합

- 여러 문자형 변수를 + 연산자로 결합합니다.

```
>>> price['시도'] + ' ' + price['자치구'] + ' ' + \ # [참고] 한 줄로 끝나지 않는 코드에  
          백슬래시를 추가합니다.
```

```
          price['읍면동'] + ' ' + price['지번'] # [주의] 지번에 있던 실수형 결측값을 제거하지 않았  
         다면 에러가 발생합니다.
```

- 여러 문자형 열이름으로 리스트를 생성합니다.

```
>>> cols = ['시도', '자치구', '읍면동', '지번']
```

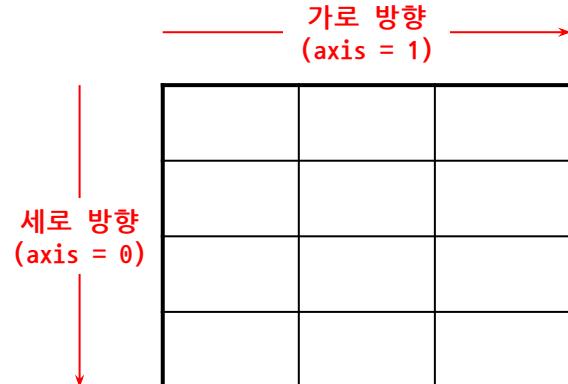
- 데이터프레임의 행(시리즈)별로 문자열을 결합하는 함수를 반복 실행합니다.

```
>>> price['주소'] = price[cols].apply(func = lambda x: ' '.join(x), axis = 1)
```

```
>>> price.head()
```

[참고] 같은 함수 반복 실행 함수

- `map()` 함수는 시리즈의 원소별로 괄호 안에 지정한 함수를 반복 실행합니다.
- `apply()` 함수는 데이터프레임의 행 또는 열 시리즈를 선택하고 시리즈별로 괄호 안에 지정한 함수를 반복 실행합니다.
 - 행을 선택하는 것은 가로 방향이므로 `axis = 1`을 추가합니다. (기본값: 0)
 - 열을 선택하는 것은 세로 방향하므로 `axis = 0`을 추가하거나 생략합니다.
- `applymap()` 함수는 데이터프레임의 셀 값별로 괄호 안에 지정한 함수를 반복 실행합니다.



[참고] 같은 함수 반복 실행

- 아파트(시리즈)의 원소(문자열)별 글자수를 반환합니다.

```
>>> price['아파트'].map(arg = len)
```

- 데이터프레임의 열(시리즈) 또는 행(시리즈)별 원소 개수를 반환합니다.

```
>>> price[cols].apply(func = len, axis = 0)
```

```
>>> price[cols].apply(func = len, axis = 1)
```

- 데이터프레임의 셀 값(문자열)별 글자수를 반환합니다.

```
>>> price[cols].applymap(func = len)
```

데이터프레임 정렬

- 시리즈 또는 데이터프레임을 오름차순 또는 내림차순 정렬합니다.

```
>>> price['총'].sort_values() # 총을 오름차순 정렬합니다.  
[참고] ascending 매개변수에 전달하는 인수의 기본값은 True입니다.
```

```
>>> price['총'].sort_values(ascending = False) # 총을 내림차순 정렬합니다.
```

```
>>> price.sort_values(by = ['총']) # price를 총으로 오름차순 정렬합니다.
```

```
>>> price.sort_values(by = ['총'], ascending = False) # price를 총으로 내림차순 정렬합니다.
```

- 데이터프레임 정렬 기준 열이 2개 이상일 때 열별로 방향을 설정합니다.

```
>>> price.sort_values(by = ['총', '거래금액'], ascending = False)
```

```
>>> price.sort_values(by = ['총', '거래금액'], ascending = [False, True])
```

집계함수로 데이터 요약

- 집계함수로 시리즈의 기술통계량을 확인합니다.

```
>>> price['거래금액'].count() # 거래금액에서 결측값을 제외한 빈도수를 반환합니다.  
[참고] sum() 함수는 합계를 반환합니다.
```

```
>>> price['거래금액'].mean() # 거래금액의 평균을 반환합니다.  
[참고] median() 함수는 중위수를 반환합니다.
```

```
>>> price['거래금액'].std() # 거래금액의 표준편차를 반환합니다.  
[참고] var() 함수는 분산을 반환합니다.
```

```
>>> price['거래금액'].min() # 거래금액의 최솟값을 반환합니다.
```

```
>>> price['거래금액'].max() # 거래금액의 최댓값을 반환합니다.
```

```
>>> price['거래금액'].describe() # 거래금액의 다양한 기술통계량을 반환합니다.
```

범주별 집계함수로 데이터 요약

- 범주형 변수로 그룹을 묶고 집계함수로 요약합니다.

```
>>> price.groupby(by = ['자치구'])['거래금액'].count() # 자치구별 거래금액에서 결측값을  
제외한 빈도수를 반환합니다.
```

```
>>> price.groupby(by = ['자치구'])['거래금액'].mean() # 자치구별 거래금액의 평균을 반환  
합니다.
```

```
>>> price.groupby(by = ['자치구'])['거래금액'].std() # 자치구별 거래금액의 표준편차를  
반환합니다.
```

```
>>> price.groupby(by = ['자치구'])['거래금액'].min() # 자치구별 거래금액의 최솟값을 반환  
합니다.
```

```
>>> price.groupby(by = ['자치구'])['거래금액'].max() # 자치구별 거래금액의 최댓값을 반환  
합니다.
```

```
>>> price.groupby(by = ['자치구'])['거래금액'].describe() # 자치구별 거래금액의 다양한  
기술통계량을 반환합니다.
```

범주형 변수의 도수/상대도수 확인

- 범주형 변수의 중복을 제거한 원소와 원소 개수를 확인합니다.

```
>>> price['자치구'].unique() # 자치구에서 중복을 제거한 원소를 반환합니다.
```

```
>>> price['자치구'].nunique() # 자치구에서 중복을 제거한 원소 개수를 반환합니다.
```

- 범주형 변수의 도수 또는 상대도수를 확인합니다.

```
>>> price['자치구'].value_counts() # 자치구별 빈도수를 내림차순 정렬한 결과를 반환합니다.
```

```
>>> price['자치구'].value_counts(ascending = True) # 자치구별 빈도수를 오름차순 정렬한 결과를 반환합니다.
```

```
>>> price['자치구'].value_counts().sort_index() # 자치구별 빈도수를 시리즈 인덱스로 오름 차순 정렬한 결과를 반환합니다.
```

```
>>> price['자치구'].value_counts(normalize = True) # 자치구별 상대도수를 내림차순 정렬한 결과를 반환합니다.
```

데이터프레임의 2가지 형태

자치구 금액구분 매매건수

강남구	5천 미만	806
강남구	5천 이상	1717
강동구	5천 미만	1103
강동구	5천 이상	935
강북구	5천 미만	921
강북구	5천 이상	9
:	:	:

[Long type]

`df.pivot()`

`df.melt()`

자치구 5천 미만 5천 이상

강남구	806	1717
강동구	1103	935
강북구	921	9
:	:	:

[Wide type]

데이터프레임의 2가지 형태

- 데이터프레임을 형태에 따라 Long type과 Wide type으로 구분합니다.
 - 데이터 분석 과정에서 Wide type을 주로 사용합니다.
 - 하지만 데이터를 요약하거나 준비된 데이터셋이 Long type인 경우가 있습니다.
 - 그래프를 그릴 때 Long type으로 변환해야 할 필요도 있습니다.
 - 따라서 Wide type과 Long type을 상호 변환하는 방법을 알고 있어야 합니다.
- 데이터프레임의 형태를 변환할 때 `pivot()` 또는 `melt()` 함수를 사용합니다.
 - `pivot()` 함수는 Long type을 Wide type으로 변환합니다.
 - `melt()` 함수는 Wide type을 Long type으로 변환합니다.

Long type 데이터프레임 생성

- 두 범주형 변수의 빈도수로 Long type 데이터프레임을 생성합니다.

```
>>> cols = ['자치구', '금액구분'] # 두 범주형 변수명으로 리스트를 생성합니다.
```

```
>>> elong = price.groupby(by = cols)[['평당금액']].count()
```

```
>>> elong.head() # [참고] elong은 자치구와 금액구분을 인덱스(행이름)로 갖는 데이터프레임입니다.  
25개 자치구마다 2종의 금액구분이 있으므로 elong의 행 개수는 50입니다.
```

- elong의 행이름을 초기화하고 기존 행이름을 열로 추가합니다.

```
>>> elong = elong.reset_index()
```

```
>>> elong.head() # [참고] elong의 행이름을 초기화하면 50행, 3열인 데이터프레임으로 변환합니다.  
마지막 열이름은 빈도수이므로 아래 코드를 실행하여 '매매건수'로 변경하는 것이 좋습니다.
```

```
>>> elong = elong.rename(columns = {'평당금액': '매매건수'})
```

Long type을 Wide type으로 변환

- Long type을 Wide type으로 변환합니다.

```
>>> widen = elong.pivot(index = '자치구', # index로 적용할 열이름을 지정합니다.  
                      columns = '금액구분', # Wide type의 열이름으로 적용할 Long type의  
                                     열이름을 지정합니다.  
                      values = '매매건수') # Wide type의 값으로 채울 Long type의 열이름을  
                                     지정합니다.
```

```
>>> widen.head() # [참고] widen은 25개 자치구별로 '5천 미만'과 '5천 이상'의 열을 갖습니다.(25행)
```

- widen의 행이름을 출력합니다.

```
>>> widen.index # [참고] 인덱스(행이름)에 name 속성이 있고, 속성값은 '자치구'입니다.
```

```
>>> widen.index.name # 인덱스(행이름)의 name을 출력합니다.  
                      # [참고] widen의 행이름을 초기화하면 인덱스 name을 열이름으로 설정합니다.
```

widen 행이름 초기화

- widen의 행이름을 초기화하고 기존 행이름을 열로 추가합니다.

```
>>> widen = widen.reset_index() # widen의 행이름을 열에 추가하도록 drop = True를 생략합니다.  
[참고] 행이름의 name 속성값인 '자치구'를 열이름으로 설정합니다.
```

```
>>> widen.head() # widen의 처음 5행을 출력합니다.  
[참고] 행이름 위에 '금액구분'을 출력하는데, 이것은 열이름의 name입니다.
```

- widen의 열이름을 출력하고, name 속성값을 삭제합니다.

```
>>> widen.columns # widen의 열이름을 출력합니다.  
[참고] 컬럼명(열이름)에 name 속성이 있고, 속성값은 '금액구분'입니다.
```

```
>>> widen.columns.name # 컬럼명(열이름)의 name을 출력합니다.
```

```
>>> widen.columns.name = '' # 컬럼명(열이름)의 name에 빈 문자열을 할당합니다.
```

```
>>> widen.head() # widen의 처음 5행을 출력합니다.  
[참고] 행이름 위에 아무것도 출력하지 않습니다.
```

Wide type을 Long type으로 변환

- Wide type을 Long type으로 변환하고 오름차순 정렬 및 인덱스를 초기화합니다.

```
>>> widen.melt(id_vars = '자치구', # id_vars 매개변수에 맨 왼쪽에 놓을 열이름을 지정하고,  
           value_vars 매개변수에는 세로로 늘일 열이름을 지정합니다.  
           value_vars = ['5천 미만', '5천 이상'],  
           var_name = '금액종류', # value_vars 매개변수에 지정한 열이름을 원소로 갖는 새  
                           # 열이름을 지정합니다.  
           value_name = '거래건수') \\ # value_vars 매개변수에 지정한 열의 값을 원소로  
                           # 갖는 새 열이름을 지정합니다.  
           .sort_values(by = ['자치구', '금액종류']) \\  
           .reset_index(drop = True) \\ # [참고] 여러 줄로 작성하려면 마지막에 \ 기호를 추가합니다.  
                           # [주의] \ 기호 뒤에 공백 포함 아무것도 입력하면 안됩니다!  
           .head()
```

피벗 테이블 생성

- 피벗 테이블은 두 범주형 변수로 연속형 변수를 요약합니다.

```
>>> pd.pivot_table(data = price, # 데이터프레임을 지정합니다.  
                   [주의] data 매개변수를 생략하면 에러를 반환합니다.  
                   values = '평당금액', # 집계함수에 적용할 연속형 변수의 열이름을 지정합니다.  
                           [참고] 변수가 두 개 이상이면 리스트로 지정합니다.  
                   index = '자치구', # 행이름에 적용할 열이름을 지정합니다.  
                           [참고] 변수가 두 개 이상이면 리스트로 지정합니다.  
                   columns = '금액구분', # 열이름에 적용할 열이름을 지정합니다.  
                           [참고] 변수가 두 개 이상이면 리스트로 지정합니다.  
                   aggfunc = np.mean) # 집계함수를 np.통계함수 또는 문자열로 지정합니다.  
                           [참고] 함수가 두 개 이상이면 리스트로 지정합니다.
```

[참고] `pivot_table()` 함수는 `groupby()`.집계함수의 확장판이라고 볼 수 있습니다.

교차 테이블 생성

- 교차 테이블은 두 범주형 변수의 빈도수/상대도수를 원소로 갖습니다.

```
>>> pd.crosstab(index = price['자치구'], # 행이름에 적용할 시리즈를 지정합니다.  
                  [참고] 입력변수(원인)를 지정합니다.  
                  columns = price['금액구분'], # 열이름에 적용할 시리즈를 지정합니다.  
                           [참고] 목표변수(결과)를 지정합니다.  
                  normalize = 'index', # normalize 매개변수를 추가하면 상대도수를 반환합니다.  
                           [참고] 'index'는 행별, 'columns'은 열별 상대도수입니다.  
                  margins = True, # 행과 열 합계를 추가합니다.  
                  margins_name = '합계') # 행과 열 합계의 이름을 지정합니다. (기본값: 'All')
```

[참고] `crosstab()` 함수는 `value_count()` 함수의 확장판이라고 볼 수 있습니다.

데이터프레임 결합

- 두 데이터프레임의 열이름이 순서까지 정확하게 같은지 확인합니다.

```
>>> df1.columns.equals(other = df2.columns) # 두 데이터프레임의 열이름이 같으므로 True입니다.  
[참고] 순서만 달라도 False를 반환합니다.
```

- 열이름이 같은 두 데이터프레임을 행(세로) 방향으로 결합합니다.

```
>>> pd.concat(objs = [df1, df2]) # [참고] 두 데이터프레임의 기존 행이름을 유지합니다.
```

```
>>> pd.concat(objs = [df1, df2], ignore_index = True) # 두 데이터프레임을 세로로 결합하고  
행이름을 초기화합니다.
```

- df2의 일부 열이름을 변경합니다.

```
>>> df2 = df2.rename(columns = {'아파트': '아파트명'})
```

```
>>> df1.columns.equals(other = df2.columns) # 두 데이터프레임 열이름이 다르므로 False를 반환  
합니다.
```

데이터프레임 결합(계속)

- 열이름이 다른 두 데이터프레임을 행(세로) 방향으로 결합합니다.

```
>>> pd.concat(objs = [df1, df2], ignore_index = True) # [참고] 열이름이 다른 셀 값을 NaN  
(결측값)으로 채웁니다.
```

- 두 데이터프레임을 열(가로) 방향으로 결합합니다.

```
>>> pd.concat(objs = [df1, df2], axis = 1) # [참고] 행이름이 같은 행을 결합하고 행이름이 서로  
다른 셀 값은 결측값으로 채웁니다.
```

- df2의 행이름을 초기화합니다.

```
>>> df2 = df2.reset_index(drop = True)
```

- 행이름을 초기화한 두 데이터프레임을 열(가로) 방향으로 결합합니다.

```
>>> pd.concat(objs = [df1, df2], axis = 1) # 행이름이 같은 행에는 결측값이 없고 행이름이 서로  
다른 행에는 결측값이 있습니다.
```

데이터프레임 병합의 시각적 예시

A	ID	V1	V2
	a	a1	a2
	b	b1	b2
	c	c1	c2



B	ID	V3	V4
	b	b3	b4
	c	c3	c4
	d	d3	d4

데이터프레임을 병합할 때 기준 열을 **외래키**^{foreign key}라고 합니다.
외래키의 값이 일치하는 행을 열(가로) 방향으로 병합합니다.

데이터프레임을 병합하기 전에 두 가지를 확인해야 합니다.
- 두 외래키의 값이 서로 일치하는 원소가 있는가?
- 오른쪽 외래키에 중복 원소가 있는가? (1:1, M:1, M:N 관계)

내부 병합 Inner Join

ID	V1	V2	V3	V4
b	b1	b2	b3	b4
c	c1	c2	c3	c4

외부 병합 Full Outer Join

ID	V1	V2	V3	V4
a	a1	a2	NA	NA
b	b1	b2	b3	b4
c	c1	c2	c3	c4
d	NA	NA	d3	d4

왼쪽 외부 병합 Left Outer Join

ID	V1	V2	V3	V4
a	a1	a2	NA	NA
b	b1	b2	b3	b4
c	c1	c2	c3	c4

병합 데이터셋 준비

- 현재 작업 경로에 있는 폴더명과 파일명을 출력합니다.

```
>>> os.listdir()
```

- 상세정보xlsx 파일을 읽고 데이터프레임을 생성합니다.

```
>>> detail = pd.read_excel(io = 'Naver_APT_Detail_Seoul.xlsx')
```

- detail의 정보를 확인합니다.

```
>>> detail.info()
```

- detail의 처음 5행을 출력합니다.

```
>>> detail.head()
```

외래키 확인 및 전처리

- 두 데이터프레임의 외래키에서 일치하는 원소 개수를 확인합니다.

```
>>> len(set(price['주소']) & set(detail['지번주소']))
```

- 두 데이터프레임의 외래키를 각각 출력합니다.

```
>>> price['주소'].head() # price의 주소를 출력합니다. '서울특별시'로 시작합니다.
```

```
>>> detail['지번주소'].head() # detail의 지번주소를 출력합니다. '서울시'로 시작합니다.
```

- price의 주소에서 '특별'을 삭제하고, 일치하는 원소 개수를 다시 확인합니다.

```
>>> price['주소'] = price['주소'].str.replace(pat = '특별', repl = '')
```

```
>>> len(set(price['주소']) & set(detail['지번주소']))
```

[참고] 표본 추출 및 시드 고정

- 이번 주로또 번호를 출력합니다. 마음에 들 때까지 여러 번 반복해보세요.

```
>>> np.random.choice(a = range(1, 46), size = 6, replace = False)
```

- 시드를 고정하면 항상 재현 가능한^{reproducible} 결과를 얻습니다.

- 시드는 무작위 값을 추출하기 전에 설정하는 초기값입니다.

- 실제로는 무작위 값 추출 알고리즘을 통해 무작위로 보이는 값을 추출합니다.

```
>>> np.random.seed(seed = 1)
```

```
>>> np.random.choice(a = range(1, 46), size = 6, replace = False)
```

[참고] 비복원추출 반복 실행

- 모집단(1~45의 정수)에서 6개 표본을 비복원추출하는 코드를 10번 반복합니다.

```
>>> for i in range(10):  
  
    np.random.seed(seed = 1)  
  
    lotto = np.random.choice(a = range(1, 46), size = 6, replace = False)  
  
    lotto.sort() # lotto의 원소를 오름차순 정렬합니다.  
  
    print(lotto)
```

[참고] 복원추출

- 시드를 고정합니다.

```
>>> np.random.seed(seed = 2)
```

- 1~5의 정수에서 3개를 복원추출합니다.

```
>>> nums = np.random.choice(a = range(1, 6), size = 3)
```

- nums를 시리즈로 변환합니다.

```
>>> nums = pd.Series(data = nums)
```

>>> nums # [참고] 시리즈에는 원소의 중복 여부를 확인하는 *duplicated()* 함수가 있습니다.

[참고] 중복 원소 확인 함수

- `duplicated()` 함수는 시리즈 원소의 중복 여부를 True 또는 False로 반환합니다.

```
>>> nums.duplicated() # 순방향으로 원소 중복 여부를 True/False로 반환합니다.  
[참고] keep 매개변수에 중복일 때 선택할 순서를 지정합니다. (기본값: 'first')
```

```
>>> nums.duplicated(keep = 'last') # 역방향으로 원소 중복 여부를 True/False로 반환합니다.
```

```
>>> nums.duplicated(keep = False) # 모든 중복 원소를 True로 반환합니다. (탐색 방향과 상관 없음)
```

- 위 코드 실행 결과로 불리언 인덱싱을 하면 결과가 매번 달라집니다.

```
>>> nums[nums.duplicated()] # 중복 원소 중 맨 처음 원소 제외만 선택합니다.
```

```
>>> nums[nums.duplicated(keep = 'last')] # 중복 원소 중 맨 마지막 원소 제외만 선택합니다.
```

```
>>> nums[nums.duplicated(keep = False)] # 중복 원소의 전체를 선택합니다.
```

데이터프레임 중복 원소 확인 및 제거

- detail의 지번주소가 중복이면 True, 아니면 False인 시리즈를 생성합니다.

```
>>> dup = detail['지번주소'].duplicated(keep = False)
```

- detail에서 dup이 True인 행을 선택하고 지번주소로 오름차순 정렬합니다.

```
>>> detail[dup].sort_values(by = ['지번주소']) # 실제 업무에서는 중복 발생 원인을 확인하고  
데이터를 전처리해야 합니다.
```

- detail의 지번주소에서 순방향으로 중복인 행을 제거하고 detail에 재할당합니다.

```
>>> detail = detail[~detail['지번주소'].duplicated()] # [참고] ~ 연산자는 진리값을 반전  
합니다.
```

- detail의 행 개수를 확인합니다.

```
>>> detail.shape[0] # detail의 행 개수가 감소했습니다. (9668 → 9640)
```

데이터프레임 병합

- price와 detail에서 일치하는 열이름을 확인합니다.

```
>>> set(price.columns) & set(detail.columns) # 두 데이터프레임에서 일치하는 열이름이 없으면  
# 병합할 때 외래키 이름을 각각 지정해야 합니다.
```

- 두 데이터프레임으로 내부 병합을 실행합니다.

```
>>> pd.merge(left = price, # 왼쪽 데이터프레임을 지정합니다.
```

```
right = detail, # 오른쪽 데이터프레임을 지정합니다.
```

```
how = 'inner', # how 매개변수에 병합 방법을 지정합니다. (기본값: 'inner')  
[참고] 외부 병합은 'outer', 왼쪽 외부 병합은 'left'를 지정합니다.
```

```
left_on = '주소', # 왼쪽 데이터프레임의 외래키 이름을 지정합니다.
```

```
right_on = '지번주소') # 오른쪽 데이터프레임의 외래키 이름을 지정합니다.
```

데이터프레임 병합(계속)

- detail의 외래키 이름을 '주소'로 변경합니다.

```
>>> detail = detail.rename(columns = {'지번주소': '주소'})
```

- price와 detail에서 일치하는 열이름을 확인합니다.

```
>>> set(price.columns) & set(detail.columns) # 두 데이터프레임에서 '주소'만 일치합니다.
```

- 외래키 이름이 같으면 on 매개변수를 사용합니다. # [참고] 두 데이터프레임에서 외래키 이름만 같으면 on 매개변수를 생략할 수 있습니다.

```
>>> apt = pd.merge(left = price, right = detail, how = 'inner', on = '주소')
```

- apt의 정보를 확인합니다.

```
>>> apt.info() # apt는 41264행 23열인 데이터프레임입니다.
```

[참고] 외래키 설정

- 두 데이터프레임의 외래키가 2개 이상이고 서로 다르면(예를 들어 df1과 df2에서 외래키 관계가 x1은 y1과 같고, x2는 y2와 같은 경우) 매칭 순서에 맞게 리스트로 지정합니다.

```
>>> pd.merge(left = df1, right = df2, how = 'inner',  
            left_on = ['x1', 'x2'], right_on = ['y1', 'y2']) # [주의] 외래키 순서를  
                                                매칭해서 지정합니다.
```

- 두 데이터프레임의 외래키가 2개 이상이고 모두 같으면(예를 들어 df1과 df2에서 외래키가 x1과 x2인 경우) 외래키를 리스트로 지정하거나 생략합니다.

```
>>> pd.merge(left = df1, right = df2, how = 'inner', on = ['x1', 'x2'])
```

외부 파일로 저장

- apt를xlsx파일로저장합니다.*#xlsx파일을저장할때가장오래걸립니다.*

```
>>> apt.to_excel(excel_writer = 'APT_List_Seoul_2021.xlsx', index = None)
```

- apt를csv파일로저장합니다.

```
>>> apt.to_csv(path_or_buf = 'APT_List_Seoul_2021.csv', index = None)
```

- apt를z파일로저장합니다.

```
>>> joblib.dump(value = apt, filename = 'APT_List_Seoul_2021.z')
```

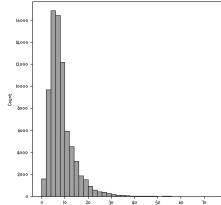
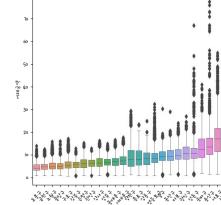
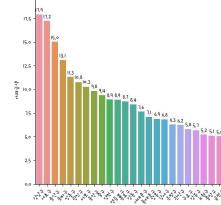
데이터 시각화

데이터 시각화

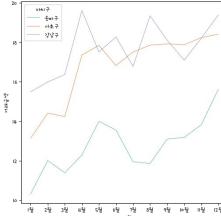
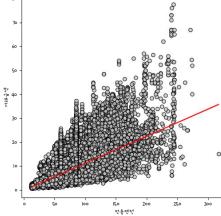
- 일변량 데이터의 분포 또는 이변량 데이터의 관계를 그래프로 확인함으로써 분석 데이터셋에 대한 이해의 폭을 넓힐 수 있습니다.
- 데이터 분석 결과를 시각적으로 표현함으로써 데이터 분석 과정에 참여하지 않은 사람들에게 분석 결과를 쉽고 빠르게 전달할 수 있습니다.
 - 2차원의 정적인 이미지로 표현하는 것이 일반적이지만 3차원 입체 또는 동적인 이미지로 전환하면 더욱 효과적입니다.
- 데이터 시각화는 그래프로 데이터에 잠재된 패턴을 발굴하고 분석 결과를 쉽고 빠르게 전달하는 것을 목적으로 수행하는 분석 방법입니다.



데이터 시각화 종류

구분	예시	특징
히스토그램		<ul style="list-style-type: none">히스토그램은 일변량 연속형 변수의 도수분포표를 시각화한 그래프입니다.세로축은 빈도수이며 밀도로 변경할 수 있습니다.히스토그램의 막대는 서로 붙어 있으며, 세로축을 밀도로 변경하면 막대의 총면적은 확률 1을 의미합니다.
상자 수염 그림		<ul style="list-style-type: none">일변량 연속형 변수의 분포에 사분위수와 이상치를 추가한 그래프입니다.가로축에 범주형 변수를 지정하면 집단 간 연속형 변수의 분포를 비교할 수 있습니다.
막대 그래프		<ul style="list-style-type: none">일변량 막대 그래프는 범주형 변수의 빈도수를 막대로 그린 그래프입니다.이변량 막대 그래프는 범주형 변수에 따라 연속형 변수의 크기를 비교할 수 있습니다.

데이터 시각화 종류(계속)

구분	예시	특징
선 그래프	 A line graph with four distinct lines representing different time series. The x-axis is labeled with dates from 1월 to 12월. The y-axis ranges from 10 to 20. The lines show various trends such as growth, decline, and seasonal fluctuations.	<ul style="list-style-type: none">선 그래프는 시간에 따라 연속형 변수의 변화를 표현한 그래프입니다.주가 데이터와 같이 시계열 변수를 시각화할 때 사용합니다.
산점도	 A scatter plot showing a large number of data points forming a dense cloud. A red regression line is drawn through the points, showing a positive linear trend. The x-axis is labeled '인구밀도' and the y-axis ranges from 0 to 30.	<ul style="list-style-type: none">산점도는 이변량 연속형 변수의 선형관계를 2차원 평면에 점으로 표현한 그래프입니다.선형 회귀분석의 입력변수와 목표변수에 직선의 관계가 존재하는지 확인할 수 있습니다.

관련 라이브러리 호출

- 관련 라이브러리를 호출합니다.

```
>>> import os
```

```
>>> import chardet
```

```
>>> import numpy as np
```

```
>>> import pandas as pd
```

```
>>> import joblib
```

작업 경로 확인 및 변경

- 현재 작업 경로를 확인합니다.

```
>>> os.getcwd()
```

- data 폴더로 작업 경로를 변경합니다.

```
>>> os.chdir(path = '../data')
```

- 현재 작업 경로에 있는 폴더명과 파일명을 출력합니다.

```
>>> os.listdir()
```

실습 데이터셋 준비

- z 파일을 호출하고 데이터프레임 apt에 할당합니다.

```
>>> apt = joblib.load(filename = 'APT_List_Seoul_2021.z')
```

- apt의 정보를 확인합니다.

```
>>> apt.info()
```

- apt의 처음 5행을 출력합니다.

```
>>> apt.head()
```

- apt의 열이름을 출력합니다.

```
>>> apt.columns
```

실습 데이터셋 전처리

- apt에서 필요 없는 열을 삭제합니다.

```
>>> apt = apt.drop(columns = ['주소', '아파트ID', '아파트명'])
```

- apt에 거래월을 추가합니다.

```
>>> apt['거래월'] = apt['거래일'].dt.strftime(date_format = '%m월')
```

거래일 원소(날짜 데이터)에서 '01월', '02월' 형태의 문자열을 생성합니다.

- apt를 거래월로 오름차순 정렬합니다.

```
>>> apt = apt.sort_values(by = ['거래월'])
```

- apt의 처음 5행을 출력합니다.

```
>>> apt.head()
```

시각화 라이브러리 호출

- 관련 라이브러리를 호출합니다.

```
>>> import seaborn as sns # 고급 시각화 함수를 포함하는 라이브러리입니다.
```

```
>>> import matplotlib.pyplot as plt # 그래프 크기, 제목, 축이름 등을 지정할 때 사용합니다.
```

```
>>> import matplotlib.font_manager as fm # 한글폰트를 지정할 때 사용합니다.
```

- 테스트용 그래프를 그립니다.

```
>>> sns.histplot(data = apt, x = '거래금액')
```

```
>>> plt.title(label = '아파트 거래금액 분포'); # [참고] 코드 마지막에 추가한 세미콜론(;)은  
plt.show() 함수와 같은 기능을 실행합니다.
```

위 코드를 실행하면 한글을 네모로 출력하므로 한글폰트를 설정해야 합니다.

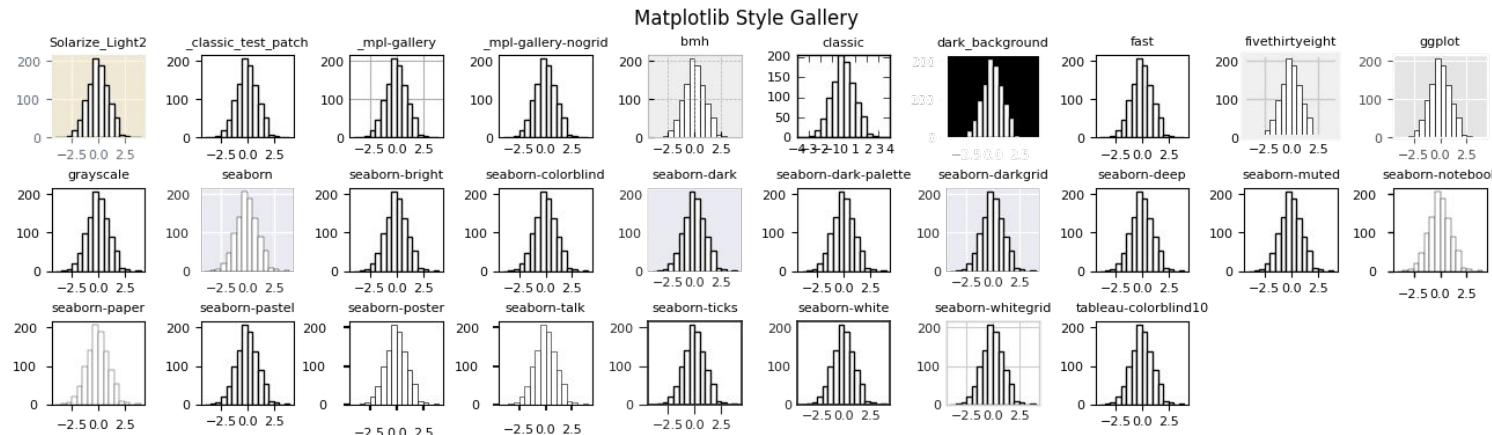
[참고] 한글폰트 외 그래프 크기와 해상도 등 다양한 그래픽 파라미터를 설정할 수 있습니다.

[참고] 스타일시트 설정

- matplotlib 스타일시트 목록을 확인하고 원하는 스타일을 설정합니다.

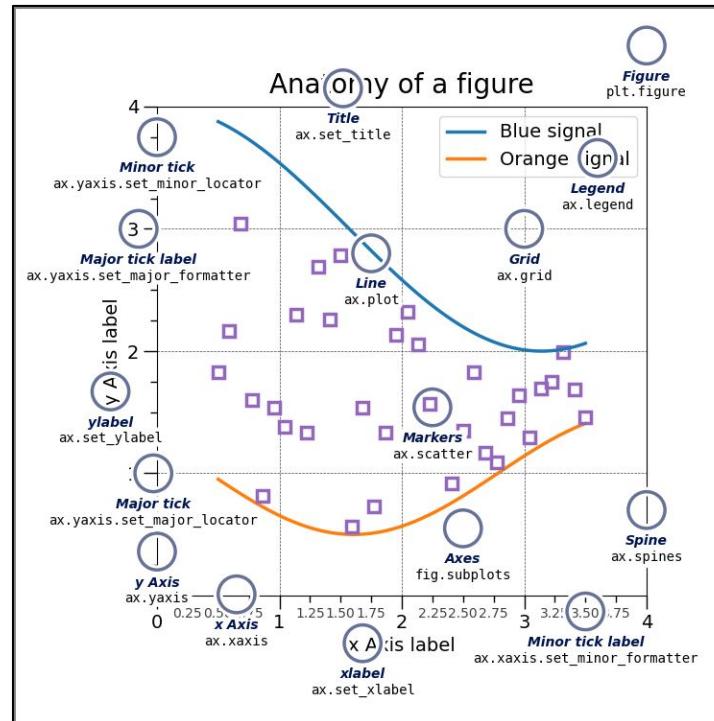
```
>>> plt.style.available # matplotlib 라이브러리에서 사용할 수 있는 스타일시트 목록을 확인합니다.  
[참고] 스타일 갤러리 웹페이지를 참고하세요.
```

```
>>> plt.style.use(style = 'seaborn-white') # 그래프에 적용할 스타일시트를 지정합니다.
```



참고: http://tonysyu.github.io/raw_content/matplotlib-style-gallery/gallery.html

[참고] Anatomy of a figure



출처: <https://matplotlib.org/stable/gallery/showcase/anatomy.html>

[참고] matplotlib.pyplot.rc Property Alias

- 그래픽 파라미터에서 일부 속성^{property}은 가명^{alias}으로 대체할 수 있습니다.

속성	가명	상세 내용
linewidth	lw	선의 두께를 설정합니다.
linestyle	ls	선의 종류를 설정합니다.
color	c	막대 또는 점의 채우기 색을 설정합니다.(산점도)
facecolor	fc	점의 채우기 색을 설정합니다.(상자 수염 그림 이상치)
edgecolor	ec	점의 테두리 색을 설정합니다.(산점도)
markeredgewidth	mew	점의 테두리 색을 설정합니다.(상자 수염 그림 이상치)
antialiased	aa	픽셀 이미지의 각을 부드럽게 보완합니다.

[참고] 한글폰트 설치

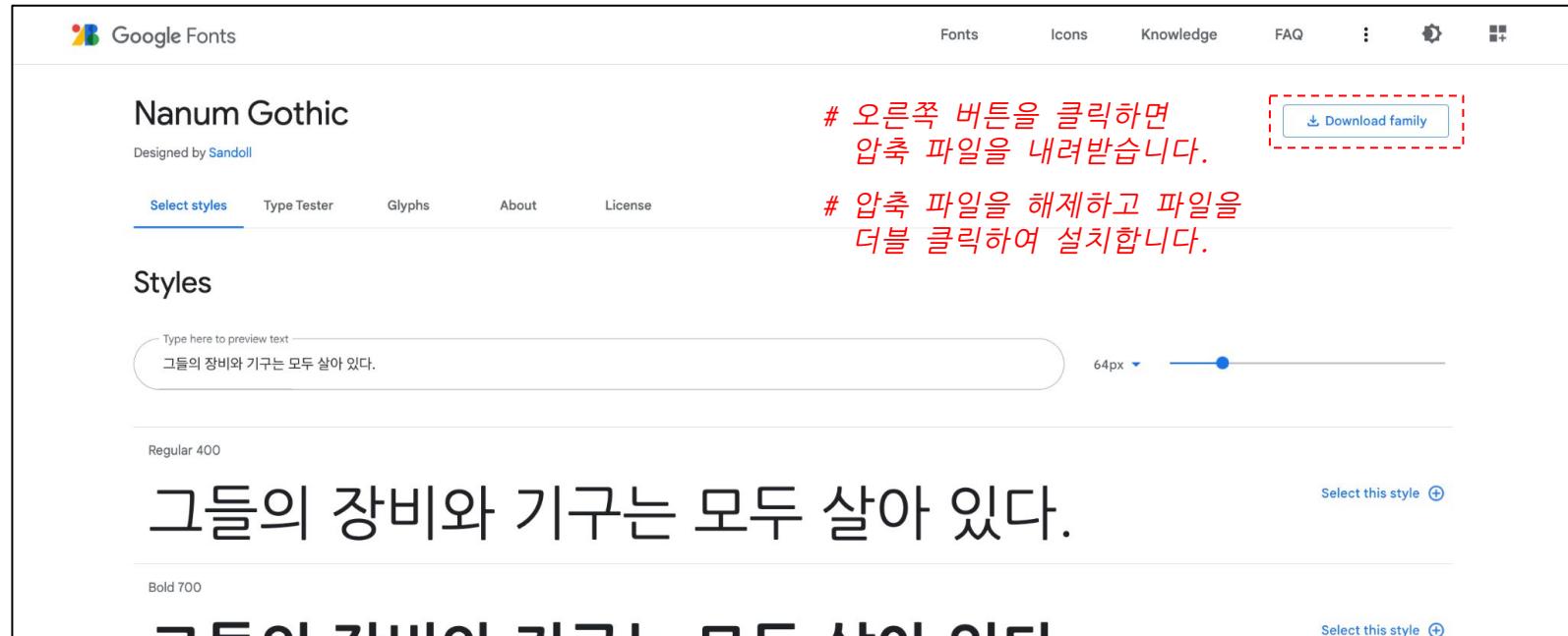
- 구글 폰트(<https://fonts.google.com/?subset=korean>)에서 한글폰트를 탐색합니다.

The screenshot shows the Google Fonts interface with the following details:

- Header:** Google Fonts, Fonts (highlighted), Icons, Knowledge, FAQ, settings icon.
- Search Bar:** Search fonts, Sentence dropdown, Type something input, 40px font size selector.
- Filter Options:** Categories, Korean, Font properties, Show only variable fonts.
- Results:** 31 of 1406 families listed.
 - Noto Sans Korean** by Google: 6 styles. Sample text: 모든 인류 구성원의 천부의 존엄성과 동등하고 양도할 수 없는 권리를 인정하는.
 - Nanum Gothic** by Sandoll: 3 styles. Sample text: 그들의 장비와 기구는 모두 살아 있다. (highlighted with a red dashed box).
 - Nanum Myeongjo** by Fontrix, Sandoll: 3 styles. Sample text: 그들의 장비와 기구는 모두 살아 있다.
- Annotations:** Red text overlay: # 관심 있는 폰트명을 클릭하면 상세 페이지로 이동합니다.

[참고] 한글폰트 설치(계속)

- 상세 페이지에서 오른쪽 위에 있는 Download family 버튼을 클릭합니다.



한글폰트명 탐색

- 현재 사용 중인 컴퓨터에 설치한 전체 폰트 파일명을 리스트로 반환합니다.

```
>>> fontList = fm.findSystemFonts(fontext = 'ttf'); fontList
```

- 리스트에서 특정 문자열(폰트명)을 포함하는 파일명만 선택합니다.

```
>>> fontPath = [font for font in fontList if 'Gamja' in font]; fontPath
```

- 반복문으로 컴퓨터에 설치된 폰트명을 출력합니다.

```
>>> for font in fontPath:
```

```
    print(fm.FontProperties(fname = font).get_name())
```

반복문을 실행한 결과에서 마음에 드는 폰트명을 선택하고 plt.rc() 함수에 지정합니다.
[참고] rc는 runtime configuration를 의미하며, pyplot을 실행하는 환경을 의미합니다.

그래픽 파라미터 설정

- 그래프 크기와 해상도를 설정합니다.

```
>>> plt.rc(group = 'figure', figsize = (8, 4), dpi = 150)
```

- 한글폰트와 글자 크기를 설정합니다.

```
>>> plt.rc(group = 'font', family = 'Gamja Flower', size = 10)
```

- 축에 유니코드 마이너스를 출력하지 않도록 설정합니다.

```
>>> plt.rc(group = 'axes', unicode_minus = False) # [참고] 왼쪽 코드를 설정하지 않으면  
음수 앞에 '−'를 출력합니다.
```

- 범례에 채우기 색과 테두리 색을 추가합니다. # fc는 facecolor(채우기), ec는 edgecolor(테두리)
관련 매개변수이고 '0'은 검정, '1'은 흰색입니다.

```
>>> plt.rc(group = 'legend', frameon = True, fc = '1', ec = '0')
```

[참고] 한글을 네모로 출력하는 문제 해결

- 한글폰트를 설정했음에도 한글을 네모로 출력하는 에러가 발생할 수 있습니다.
 - 에러 메시지: Font family ['폰트명'] not found. Falling back to DejaVu Sans
 - matplotlib 임시 폴더에 저장된 json 파일에 해당 한글폰트가 없기 때문입니다.
- json 파일을 삭제하고, Jupyter Notebook을 재실행하면 해결할 수 있습니다.

```
>>> import matplotlib, glob # 관련 라이브러리를 호출합니다.  
[참고] glob.glob() 함수는 조건에 맞는 파일명을 리스트로 반환합니다.  
>>> path = matplotlib.get_cachedir() # matplotlib 라이브러리 임시 폴더 경로를 path에 할당합니다.  
>>> fileName = glob.glob(f'{path}/fontlist-*.{json}')[0] # 폰트 정보를 갖는 json 파일명을  
fileName에 할당합니다.  
>>> os.remove(path = fileName) # matplotlib 라이브러리 임시 폴더에 있는 json 파일을 삭제합니다.  
Jupyter Notebook을 재실행하면 한글폰트를 설정할 수 있습니다.
```

[참고] 그래픽 파라미터 설정 관련 모듈 생성

- 시각화 라이브러리 호출, 스타일시트, 한글폰트 및 그래픽 파라미터를 설정하는 코드를 모듈(py 파일)로 저장하면 필요할 때마다 호출할 수 있으므로 편리합니다.
- Anaconda 메인에서 New Text File을 열고, 모듈(py 파일)로 저장할 시각화 설정 관련된 코드를 붙여넣습니다.
- 상단 메뉴에서 File → Save를 클릭하여 텍스트 파일을 저장합니다.
 - 파일명을 GraphicSetting.py로 변경합니다. # [주의] py 파일을 현재 사용 중인 Jupyter Notebook 파일과 같은 폴더에 저장하면 쉽게 호출할 수 있습니다.
- 시각화 설정 모듈을 호출합니다.

```
>>> from GraphicSetting import *
```

[참고] Python 파일 탐색 경로 확인

- 모듈(py 파일)을 호출하려면 아래 두 가지 조건 중 하나를 만족해야 합니다.
 - 현재 작업 중인 Jupyter Notebook 파일 경로에 py 파일을 저장했다.
 - Python 파일 탐색 경로 중 한 곳에 py 파일을 저장했다.
- 작업할 Jupyter Notebook 파일 경로가 바뀔 때마다 py 파일을 매번 옮겨야 하므로 첫 번째 조건은 불편합니다. 따라서 두 번째 조건을 따르는 것이 좋습니다.
- Python 파일 탐색 경로를 확인합니다.

```
>>> import sys # 관련 라이브러리를 호출합니다.
```

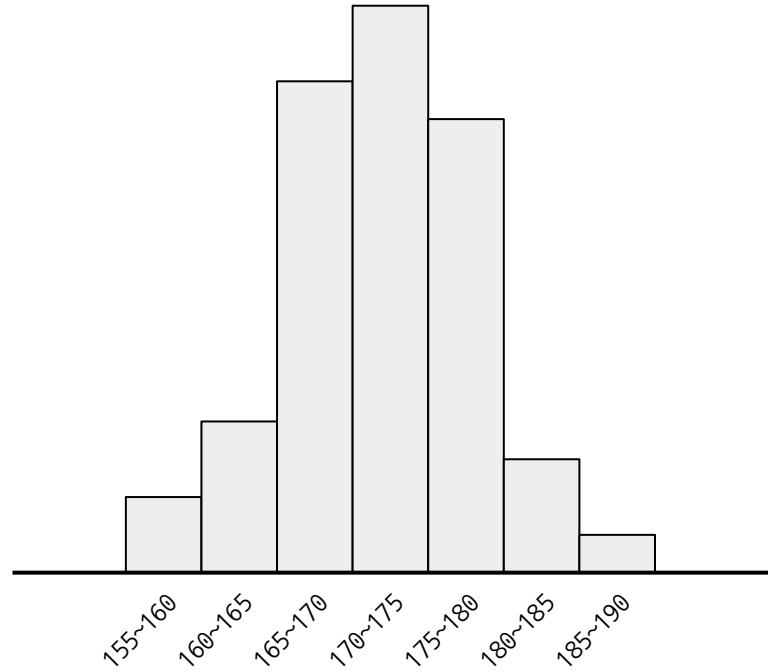
```
>>> sys.path # Python 파일 탐색 경로를 모두 출력합니다. 많은 경로 중 한 곳(마지막 경로 추천)에 py 파일을  
저장하면 해당 모듈을 항상 호출할 수 있습니다.
```

히스토그램

- 히스토그램은 일변량 연속형 변수의 도수분포표를 시각화한 것입니다.

계급	도수	상대도수
155 ~ 160	2	0.04
160 ~ 165	4	0.08
165 ~ 170	13	0.26
170 ~ 175	15	0.30
175 ~ 180	12	0.24
180 ~ 185	3	0.06
185 ~ 190	1	0.02
합계	50	1.00

도수분포표의 계급은 이상 ~ 미만입니다.



히스토그램 그리기

- 거래금액으로 히스토그램을 그립니다.

```
>>> sns.histplot(data = apt, x = '거래금액'); # data 매개변수에 데이터프레임을 지정합니다.  
          x 매개변수에 연속형 변수명을 지정합니다.
```

- 히스토그램에 그래픽 요소를 추가합니다.

```
>>> sns.histplot(data = apt, x = '거래금액', bins = 50, # bins 매개변수에 막대 개수를  
                  50으로 지정합니다.
```

```
          color = '1', edgecolor = '0'); # color 매개변수에 채우기 색, edgecolor  
          매개변수에 테두리 색을 지정합니다.
```

```
>>> sns.histplot(data = apt, x = '거래금액', binwidth = 2, # binwidth 매개변수에 막대  
                  너비를 2로 지정합니다.
```

```
          color = '1', edgecolor = '0');
```

[참고] bins와 binwidth 매개변수로 히스토그램 막대의 폭을 설정하면 최솟값부터 시작하므로 경계를 명확하게 구분할
수 없다는 단점이 있습니다. 이를 보완하려면 binrange 매개변수에 히스토그램 계급(막대 경계)을 지정합니다.

히스토그램에 계급 추가

- 거래금액 최솟값과 최댓값을 확인합니다.

```
>>> apt['거래금액'].describe()[['min', 'max']] # [참고] describe() 함수 실행 결과(시리즈)  
에서 일부 인덱스만 선택합니다.
```

- 히스토그램에 막대 개수와 범위(막대 경계)로 계급을 지정합니다.

```
>>> sns.histplot(data = apt, x = '거래금액', bins = 60, # [참고] binwidth = 2로 대신할  
수 있습니다.
```

`binrange = (0, 120),` # [주의] 히스토그램 계급은 최솟값보다 작거나 같은 값으로
시작하고 최댓값보다 크거나 같은 값으로 끝나야 합니다.

```
color = '1', edgecolor = '0');
```

[참고] 색상 목록

- matplotlib에서 제공하는 CSS Color 목록을 확인하고 원하는 색을 고릅니다.

```
>>> import matplotlib.colors as mcl # 관련 모듈을 호출합니다.
```

```
>>> mcl.CSS4_COLORS # 148가지 색이름과 Hex Code를 딕셔너리로 출력합니다.  
[출처] https://matplotlib.org/stable/gallery/color/named\_colors.html
```

CSS Colors

black	bisque	forestgreen	slategrey	firebrick	khaki	darkslategray	darkorchid
dimgray	darkorange	limegreen	lightsteelblue	maroon	palegoldenrod	darkslategrey	darkviolet
dimgrey	burlywood	darkgreen	cornflowerblue	darkred	darkkhaki	teal	mediumorchid
gray	antiquewhite	green	royalblue	red	ivory	darkcyan	purple
grey	tan	lime	ghostwhite	mistyrose	beige	aqua	thistle
darkgray	navajowhite	seagreen	lavender	salmon	lightyellow	cyan	plum
darkgrey	blanchedalmond	mediumseagreen	midnightblue	tomato	lightgoldenrodyellow	darkturquoise	violet
darkgray	papayawhip	springgreen	navy	darksalmon	olive	cadetblue	purple
darkgrey	moccasin	mintcream	darkblue	coral	yellow	powderblue	darkmagenta
lightgray	orange	mediumspringgreen	mediumblue	orangered	olivedrab	lightblue	fuchsia
lightgrey	wheat	mediumaquamarine	blue	lightsalmon	yellowgreen	deepskyblue	magenta
lightgray	oldlace	aquamarine	slateblue	sienna	darkolivegreen	skyblue	orchid
lightgrey	floralwhite	turquoise	darkslateblue	seashell	lightgreen	lightskyblue	mediumvioletred
gainsboro	darkgoldenrod	lightseagreen	mediumslateblue	chocolate	greenyellow	steelblue	deeppink
whitesmoke	goldenrod	mediumturquoise	mediumpurple	saddlebrown	chartreuse	aliceblue	hotpink
white	cornsilk	azure	rebeccapurple	mediumbrown	lawngreen	dodgerblue	lavenderblush
snow	gold	lightcyan	blueviolet	sandybrown	honeydew	lightslategray	palevioletred
rosybrown	lemonchiffon	paleturquoise	indigo	peachpuff	darkseagreen	lightslategray	crimson
lightcoral				peru	palegreen	lightgreen	pink
indianred				linen			lightpink
brown							

히스토그램 막대 채우기 색 변경

- 금액구분(범주형 변수)에 따라 채우기 색을 다르게 설정합니다.

```
>>> sns.histplot(data = apt, x = '거래금액', bins = 60, binrange = (0, 120),  
                 hue = '금액구분', edgecolor = '0');  
                 # hue 매개변수에 지정한 범주형 변수의 범주별로 채우기 색을 다르게 설정합니다.
```

- 채우기 색 배합을 범주형 변수에 맞는 팔레트로 변경합니다.

```
>>> sns.histplot(data = apt, x = '거래금액', bins = 60, binrange = (0, 120),  
                 hue = '금액구분', edgecolor = '0', palette = 'Set1');
```

[참고] 팔레트 탐색: Color Brewer

- Color Brewer는 미국의 지리학 교수인 Cynthia Brewer가 지도 제작을 위해 개발한 세 종류의 팔레트셋('sequential', 'diverging', 'qualitative')입니다.
- seaborn 라이브러리는 Color Brewer Palette를 탐색하는 함수를 제공합니다.

```
>>> sns.choose_colorbrewer_palette(data_type = 'qualitative');
```



[참고] 팔레트 설정

- 기본 팔레트 색을 출력합니다.

```
>>> sns.color_palette() # [참고] 기본 팔레트는 'deep'이며, plt.cm의 한 종류입니다.
```

- Color Brewer에서 탐색한 팔레트 색을 출력합니다.

```
>>> sns.color_palette(palette = 'Set1', n_colors = 9) # [참고] n_colors 매개변수에 색의  
개수를 변경할 수 있습니다.
```

- 기본 팔레트를 변경합니다.

```
>>> sns.set_palette(palette = 'Set1', n_colors = 9)
```

- 새로 설정한 기본 팔레트 색을 확인합니다.

```
>>> sns.color_palette()
```

[참고] 사용자 팔레트 생성

- 색이름을 원소로 갖는 리스트(사용자 팔레트)를 생성합니다.

```
>>> myPal = ['crimson', 'royalblue']
```

- 기존 그래프에 사용자 팔레트를 적용합니다.

```
>>> sns.histplot(data = apt, x = '거래금액', bins = 60, binrange = (0, 120),  
hue = '금액구분', edgecolor = '0', palette = myPal,  
hue_order = ['5천 미만', '5천 이상']);  
# hue_order 매개변수에 지정한 리스트 원소 순서대로 팔레트 색을 적용합니다.
```

히스토그램에 제목 및 축이름 추가

- 히스토그램에 제목, x축이름 및 y축이름을 추가합니다.

```
>>> sns.histplot(data = apt, x = '거래금액', bins = 60, binrange = (0, 120),
```

```
          hue = '금액구분', edgecolor = '0', palette = myPal,
```

```
          hue_order = [ '5천 미만', '5천 이상'])
```

```
>>> plt.title(label = '거래금액의 분포') # 그래프 제목을 추가합니다.
```

```
>>> plt.xlabel(xlabel = '매매가격') # x축이름을 추가합니다.
```

```
>>> plt.ylabel(ylabel = '거래건수'); # y축이름을 추가합니다.
```

히스토그램에 커널 밀도 추정 곡선 추가

- 히스토그램의 y축을 빈도수^{count} 대신 밀도^{density}로 변경합니다.

```
>>> sns.histplot(data = apt, x = '거래금액', bins = 60, binrange = (0, 120),  
                 color = '1', edgecolor = '0.8', # [참고] 커널 밀도 추정 곡선을 강조하고자  
                 stat = 'density') # stat 매개변수의 기본 인수는 'count'입니다.
```

- 히스토그램에 커널 밀도 추정 곡선을 추가합니다.

```
>>> sns.kdeplot(data = apt, x = '거래금액', color = 'red',  
                  linewidth = 0.5, linestyle = '-');
```

[참고] 커널 밀도 추정은 비모수적인 방법으로 밀도를 추정하는 것으로, 개별 관측값을 중심으로 하는 커널 함수의 평균을 계산합니다. 대표적인 커널 함수로는 가우시안 함수(정규분포 확률밀도함수)가 있습니다.

[참고] 선의 종류

- 선의 종류에는 다음과 같은 4가지를 주로 사용합니다.

구분	이름	기호	모양
실선	solid	' - '	
파선	dashed	' ___ '	
1점 쇄선	dashdot	' - . '	
점선	dotted	' : '	

관심 있는 자치구 선택

- apt에서 관심 있는 자치구를 선택하고 sub에 할당합니다.

```
>>> sub = apt[apt['자치구'].str.contains(pat = '강[남동북]')]
```

- 자치구별 빈도수를 확인합니다.

```
>>> sub['자치구'].value_counts()
```

- 자치구별 거래금액 평균(분포의 중심)을 확인합니다.

```
>>> sub.groupby(by = ['자치구'])['거래금액'].mean()
```

- 거래금액 최솟값과 최댓값을 확인합니다.

```
>>> sub['거래금액'].describe()[['min', 'max']]
```

히스토그램을 겹쳐서 그리기

- 자치구별 히스토그램을 겹쳐서 그립니다.

```
>>> sns.histplot(data = sub, x = '거래금액', bins = 60, binrange = (0, 120),  
                 hue = '자치구', edgecolor = '0');  
                 # palette 매개변수를 생략하면 기본 팔레트를 적용합니다.  
                 [참고] 현재 기본 팔레트는 'Set1'입니다.
```

- 커널 밀도 추정 곡선을 겹쳐서 그리는 것이 더 낫습니다.

```
>>> sns.kdeplot(data = sub, x = '거래금액', hue = '자치구', fill = True);  
                 # fill 매개변수에 True를 지정하면 커널 밀도 추정 곡선 아래를 기본 팔레트로 채웁니다.
```

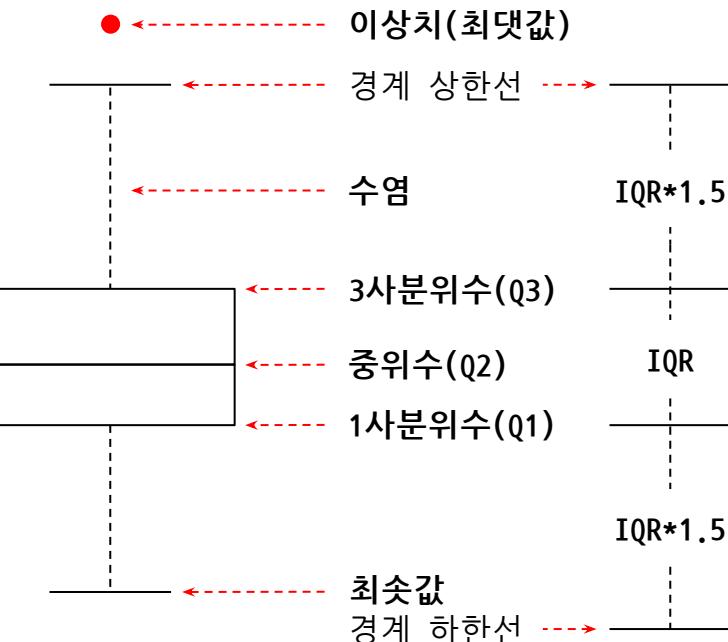
히스토그램을 나눠서 그리기

- 자치구별 히스토그램을 열(가로) 방향으로 나눠서 그립니다.

```
>>> sns.displot(data = sub, x = '거래금액', bins = 60, binrange = (0, 120),  
    hue = '자치구',  
    col = '자치구', # 자치구별 히스토그램을 열(column) 방향으로 출력합니다.  
                  # [참고] row 매개변수를 사용하면 행(row) 방향으로 출력합니다.  
    legend = False, # 범례를 추가하지 않습니다. (기본값: True)  
    height = 3, # 히스토그램 높이를 지정합니다. (기본값: 5)  
    aspect = 0.8); # 히스토그램 폭을 지정합니다. (기본값: 1)
```

상자 수염 그림

- 상자 수염 그림은 연속형 변수의 분포에 사분위수와 이상치를 시각화한 것입니다.
 - 상자 수염 그림은 세로로 그립니다.
- 상자 수염 그림은 사분범위^{Interquartile range}의 1.5배를 미달/초과하는 원소를 이상치로 판단합니다.
 - 사분범위는 3사분위수와 1사분위수의 간격 이므로 전체 데이터의 가운데 50%를 포함하는 범위입니다.



일변량 상자 수염 그림 그리기

- 거래금액으로 상자 수염 그림을 그립니다.

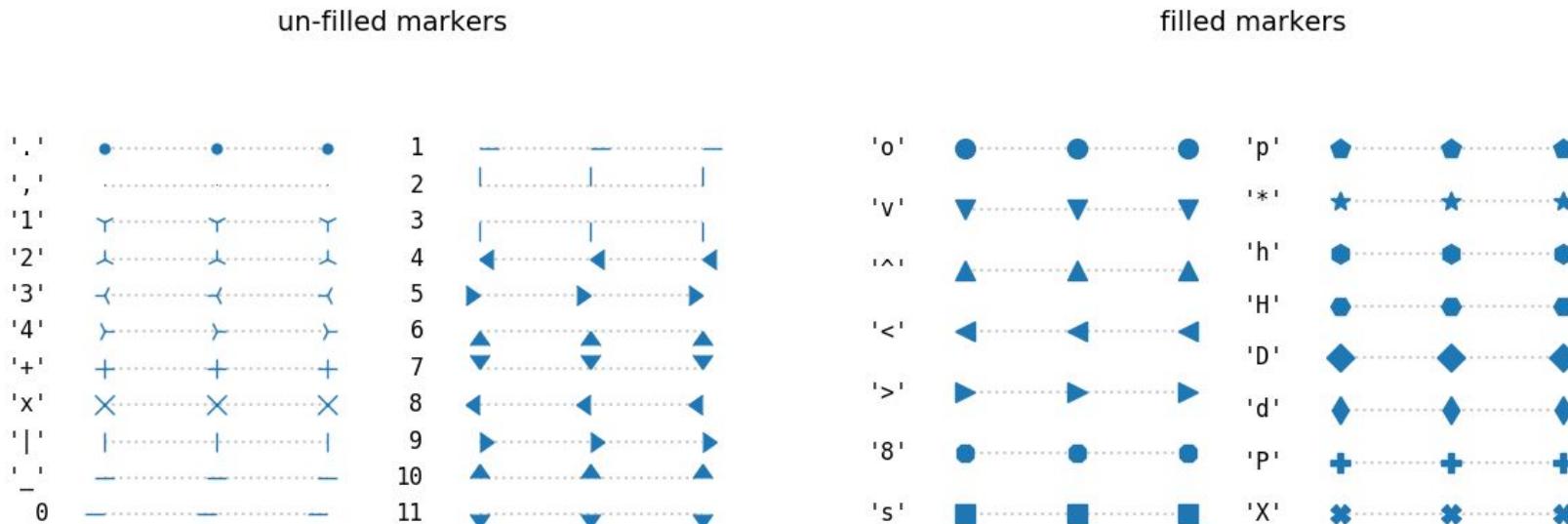
```
>>> sns.boxplot(data = apt,  
                 y = '거래금액', # y 매개변수에 연속형 변수를 지정합니다.  
                           [참고] x 매개변수를 사용하면 상자 수염 그림을 세워서 그립니다.  
                 color = '1', # 상자 채우기 색을 지정합니다.  
                           [참고] 생략하면 기본 팔레트의 첫 번째 색으로 채웁니다.  
                 linewidth = 0.5);
```

일변량 상자 수염 그림 그리기

- 이상치 관련 속성으로 딕셔너리를 생성하고 상자 수염 그림에 추가합니다.

```
>>> outProps = {'marker': 'o', # 이상치 모양을 지정합니다.(기본값: 'd')  
              'markersize': 3, # 이상치 크기를 지정합니다.(기본값: 5)  
              'markerfacecolor': 'pink', # 이상치 채우기 색을 지정합니다.(기본값: 'black')  
              'markeredgecolor': 'red', # 이상치 테두리 색을 지정합니다.(기본값: 'black')  
              'markeredgewidth': 0.2} # 이상치 테두리 두께를 설정합니다.(기본값: 1)  
  
>>> sns.boxplot(data = apt, y = '거래금액', color = '1', linewidth = 0.5,  
                  flierprops = outProps);
```

[참고] marker의 종류



출처: https://matplotlib.org/3.1.0/gallery/lines_bars_and_markers/marker_reference.html

이변량 상자 수염 그림 그리기

- apt의 자치구별 거래금액 중위수를 오름차순 정렬한 grp를 생성합니다.

```
>>> grp = apt.groupby(by = ['자치구'])['거래금액'].median()
```

```
>>> grp = grp.sort_values(); grp.head()
```

- x축에 자치구, y축에 거래금액을 지정하고 이변량 상자 수염 그림을 그립니다.

```
>>> sns.boxplot(data = apt, x = '자치구', y = '거래금액', linewidth = 0.5,
```

```
          flierprops = outProps, order = grp.index) # x축 순서를 grp 인덱스로  
          지정합니다.
```

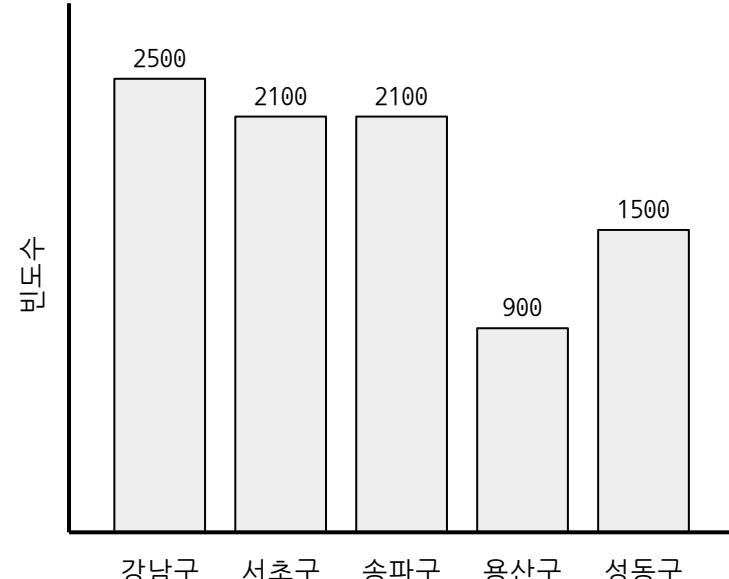
```
>>> plt.axhline(y = apt['거래금액'].median(), color = 'red', linewidth = 0.5)
```

```
>>> plt.xticks(rotation = 45); # x축 눈금명을 45도 회전시킵니다.
```

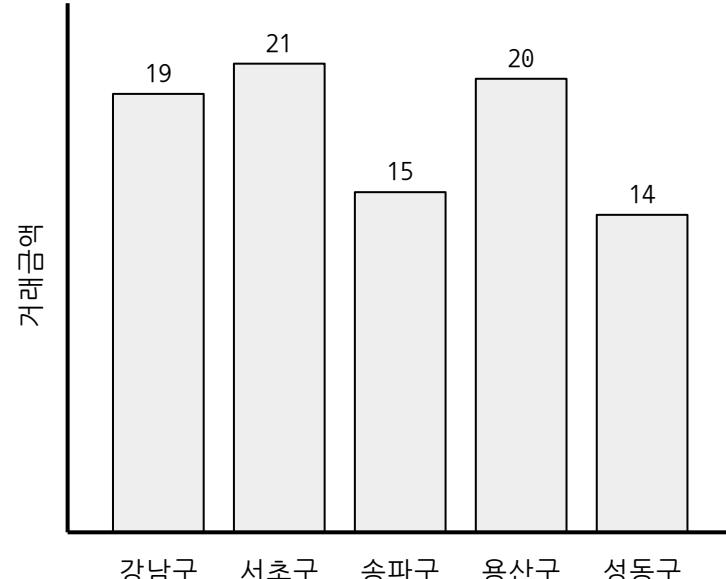
막대 그래프

- 막대 그래프는 범주형 변수를 요약하여 시각화한 것입니다.

일변량 막대 그래프



이변량 막대 그래프



일변량 막대 그래프 그리기

- apt의 자치구별 거래금액 빈도수를 내림차순 정렬한 grp를 생성합니다.

```
>>> grp = apt.groupby(by = ['자치구'])['거래금액'].count()
```

```
>>> grp = grp.sort_values(ascending = False); grp.head()
```

- 자치구별 빈도수로 일변량 막대 그래프를 그립니다.

```
>>> sns.countplot(data = apt, x = '자치구', order = grp.index)
```

```
>>> plt.ylim(0, 4000) # 막대 위에 텍스트를 출력할 공간을 확보하기 위해 y축 범위를 제한합니다.  
[주의] 데이터마다 빈도수 범위가 다르므로 미리 확인하고 설정합니다.
```

```
>>> plt.xticks(rotation = 45);
```

막대 위에 텍스트 추가

- 막대 위에 자치구별 빈도수를 텍스트로 추가합니다.

```
>>> for i, v in enumerate(grp): # enumerate() 함수는 객체의 인덱스와 원소 쌍을 튜플로 반환합니다.
```

```
    plt.text(x = i, y = v, s = v, # x, y 매개변수에 x, y 좌표를 지정합니다.  
            s 매개변수에 문자열로 출력할 값을 지정합니다.)
```

```
        ha = 'center', va = 'bottom', # ha 매개변수에 수평정렬 방식을 지정합니다.  
        va 매개변수에 수직정렬 방식을 지정합니다.
```

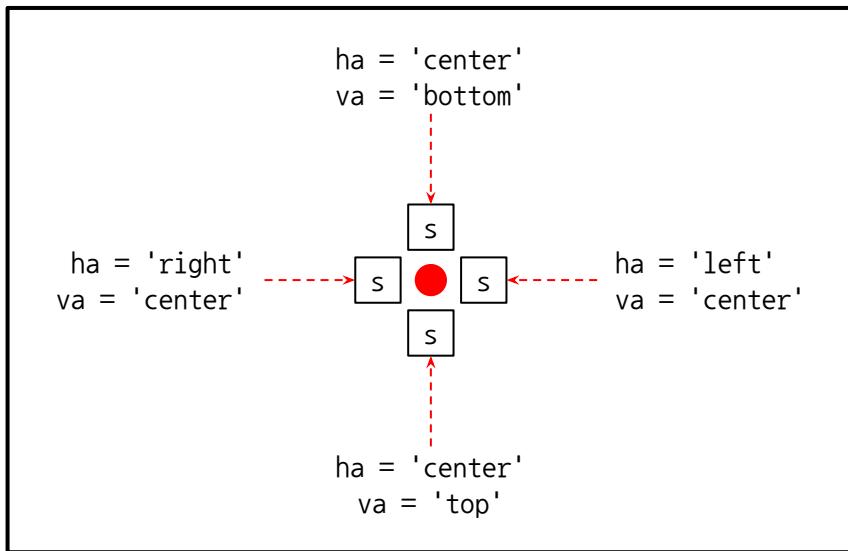
```
        color = 'black', fontsize = 9, # color 매개변수에 글자색을 지정합니다.  
        fontsize 매개변수에 크기를 지정합니다.
```

```
        fontweight = 'bold') # fontweight 매개변수에 글자 굵기를 지정합니다.  
        'light', 'regular', 'medium', 'bold', 'extra bold' 등  
        # [참고] Gamja Flower는 regular만 가능합니다!
```

[주의] 위 코드를 앞 페이지에서 작성한 막대 그래프 코드 셀에 추가해야 합니다.

[참고] plt.text() 함수

- plt.text() 함수는 지정한 위치(x, y 좌표와 ha, va 값)에 문자열을 출력합니다.
 - x, y 매개변수에 좌표를 지정합니다.
 - x, y 좌표를 빨간 점으로 가정합니다.
 - s 매개변수에 지정한 값을 문자열로 출력합니다.
 - ha, va 매개변수에 지정한 값에 따라 출력할 글자의 위치가 달라집니다.
 - 문자열 기준으로 빨간 점의 위치를 가로(ha)와 세로(va)로 조정합니다.



가로 막대 그래프 그리기

- 가로 막대 그래프를 그리고 막대 오른쪽에 빈도수를 출력합니다.

```
>>> plt.figure(figsize = (8, 6)) # [참고] Jupyter Notebook의 해당 Cell에 그래프 크기를 변경하려면  
plt.figure() 함수를 추가합니다.
```

```
>>> sns.countplot(data = apt, y = '자치구', order = grp.index) # y 매개변수에 범주형  
변수를 지정합니다.
```

```
>>> plt.xlim(0, 4000) # x축 범위를 제한합니다.
```

```
>>> for i, v in enumerate(grp):
```

```
    plt.text(x = v, y = i, s = v, # plt.text() 함수의 x와 y 매개변수에 지정했던 i, v을  
    맞바꿉니다.
```

```
        ha = 'left', va = 'center',
```

```
        color = 'black', fontsize = 9)
```

[참고] 파이 차트 그리기

- sub의 자치구별 빈도수를 내림차순 정렬한 grp를 생성합니다.

```
>>> grp = sub['자치구'].value_counts(); grp.head()
```

- grp로 파이 차트를 그립니다.

```
>>> plt.pie(x = grp.values, # 파이 차트의 쪄기 wedge 크기를 자치구별 빈도수로 지정합니다.  
           [참고] 파이 차트는 쪄기의 크기를 수치의 백분율로 계산합니다.
```

```
        explode = [0, 0, 0.2], # 쪄기별 시작 위치(실수)를 원소로 갖는 리스트로 지정합니다.  
                           [참고] 특정 쪄기를 강조하려면 0보다 큰 값을 지정합니다.
```

```
        labels = grp.index, # 쪄기별 라벨을 지정합니다.  
                           [참고] 라벨 출력 위치의 기본값은 1.2입니다.
```

```
        autopct = '%.1f%%'); # 쪄기 안에 백분율을 출력할 포맷을 지정합니다.  
                           [참고] 백분율 출력 위치의 기본값은 0.6입니다.
```

[참고] 파이 차트 꾸미기

- 파이 차트의 다양한 그래픽 요소를 변경합니다.

```
>>> plt.pie(x = grp.values, explode = [0, 0, 0.2],  
           labels = grp.index, autopct = '%.1f%%',  
           colors = ['white', 'white', 'orange'], # [주의] 팔레트명을 문자열로 지정하면  
                     # 에러를 반환합니다.  
           startangle = 90, counterclock = False, # 첫 번째 쪄기 시작 위치와 출력 방향을  
                                         # 변경합니다.  
           textprops = dict(color = '0', size = 12), # 글자 색과 크기를 지정합니다.  
           wedgeprops = dict(ec = '0', lw = 0.5)); # 쪄기의 테두리 색과 선의 두께를 지정  
                                         # 합니다.
```

이변량 막대 그래프 그리기

- apt의 자치구별 거래금액 평균을 내림차순 정렬한 grp를 생성합니다.

```
>>> grp = apt.groupby(by = ['자치구'])['거래금액'].mean()
```

```
>>> grp = grp.sort_values(ascending = False).round(1); grp.head()
```

- 자치구별 거래금액 평균으로 이변량 막대 그래프를 그립니다.

```
>>> sns.barplot(data = apt, x = '자치구', y = '거래금액', order = grp.index,  
estimator = np.mean, errorbar = None)
```

estimator 매개변수에 집계함수를 지정합니다. (기본값: 'mean')
errorbar 매개변수에는 95% 신뢰구간 출력 여부를 지정합니다.

```
>>> plt.ylim(0, 22) # [참고] 95% 신뢰구간이란 표본 평균이 모평균의 ±2 표준편차 안에  
# 95% 확률로 포함된다는 것을 의미합니다.  
>>> plt.xticks(rotation = 45);
```

막대 위에 텍스트 추가

- 막대 위에 거래금액 평균을 텍스트로 추가합니다.

```
>>> for i, v in enumerate(grp):  
  
    plt.text(x = i, y = v, s = v,  
  
              ha = 'center', va = 'bottom',  
  
              color = 'black', fontsize = 9,  
  
              fontweight = 'bold')
```

[주의] 위 코드를 앞 페이지에서 작성한 막대 그래프 코드 셀에 추가해야 합니다.

[참고] 묶음 막대 그래프 그리기

- sub의 자치구와 금액구분별 거래금액 평균으로 grp를 생성합니다.

```
>>> grp = sub.groupby(by = ['자치구', '금액구분'])['거래금액'].mean()
```

```
>>> grp = grp.round(1); grp.head() # [참고] grp는 자치구와 금액구분인 다중인덱스 MultiIndex를 갖는데,  
# [참고] grp는 자치구와 금액구분인 다중인덱스 MultiIndex를 갖는데,  
다중인덱스는 레벨로 구분할 수 있습니다.
```

- 이변량 막대 그래프의 x축에 범주형 변수를 추가한 묶음 막대 그래프를 그립니다.

```
>>> sns.barplot(data = sub, x = '자치구', y = '거래금액', hue = '금액구분',  
order = grp.index.levels[0], hue_order = grp.index.levels[1],  
estimator = np.mean, errorbar = None)
```

```
>>> plt.ylim(0, 27)
```

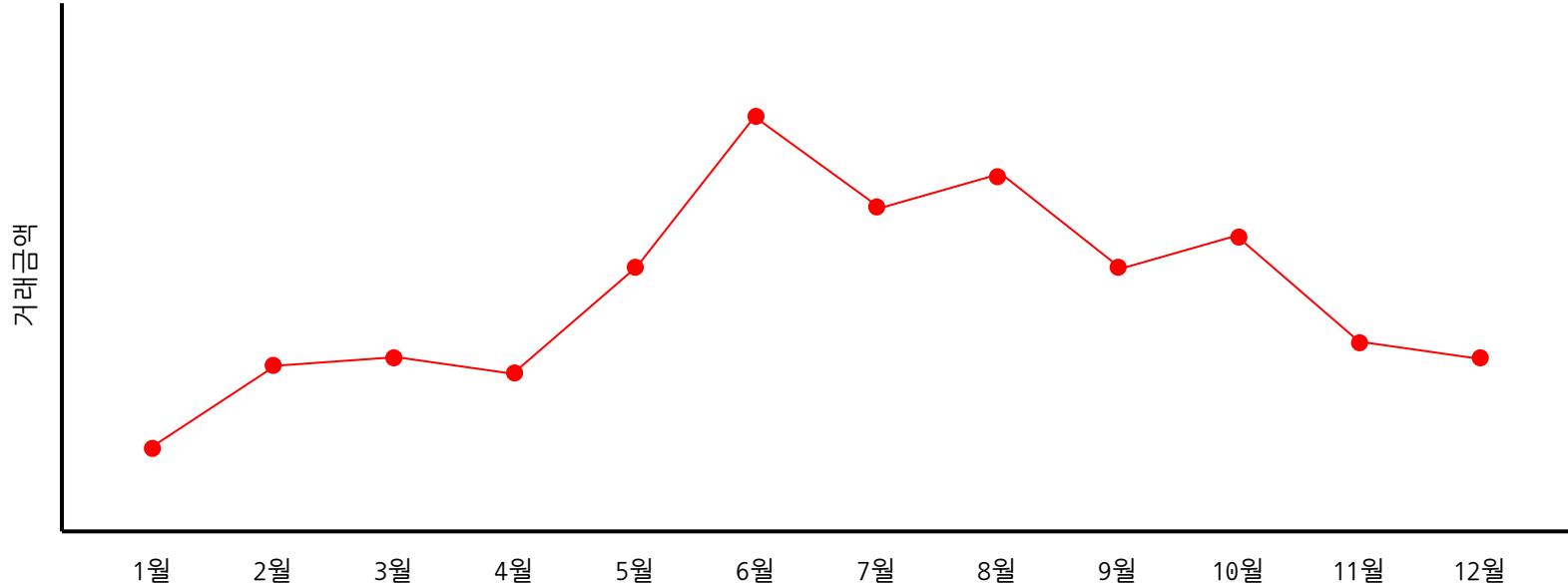
[참고] 묶음 막대 위에 텍스트 추가

- 묶음 막대 위에 거래금액 평균을 텍스트로 추가합니다.

```
>>> for i, v in enumerate(grp):  
  
    if i % 2 == 0:  
  
        i = i/2 - 0.2 # 묶음 막대가 2개일 때, x축 좌표(인덱스)에 ±0.2씩 간격을 줍니다.  
                    # 따라서 grp의 인덱스가 짝수면 인덱스를 2로 나누고 0.2를 뺍니다.  
  
    else:  
  
        i = (i-1)/2 + 0.2 # 만약 grp의 인덱스가 홀수면 인덱스에서 1을 뺀 값을 2로 나누고 0.2를 더합니다.  
  
    plt.text(x = i, y = v, s = v, ha = 'center', va = 'bottom')
```

선 그래프

- 선 그래프는 시간의 흐름에 따라 연속형 변수가 변하는 양상을 그린 그래프입니다.



선 그래프 그리기

- 거래월별 거래금액 평균으로 선 그래프를 그립니다.

```
>>> sns.lineplot(data = apt, x = '거래월', y = '거래금액', color = 'red',  
                 estimator = np.mean, errorbar = None);
```

- 선 그래프에 점을 추가합니다.

```
>>> sns.lineplot(data = apt, x = '거래월', y = '거래금액', color = 'red',  
                 estimator = np.mean, errorbar = None, marker = 'o');
```

선 그래프를 겹쳐서 그리기

- 자치구별 선 그래프를 겹쳐서 그립니다.

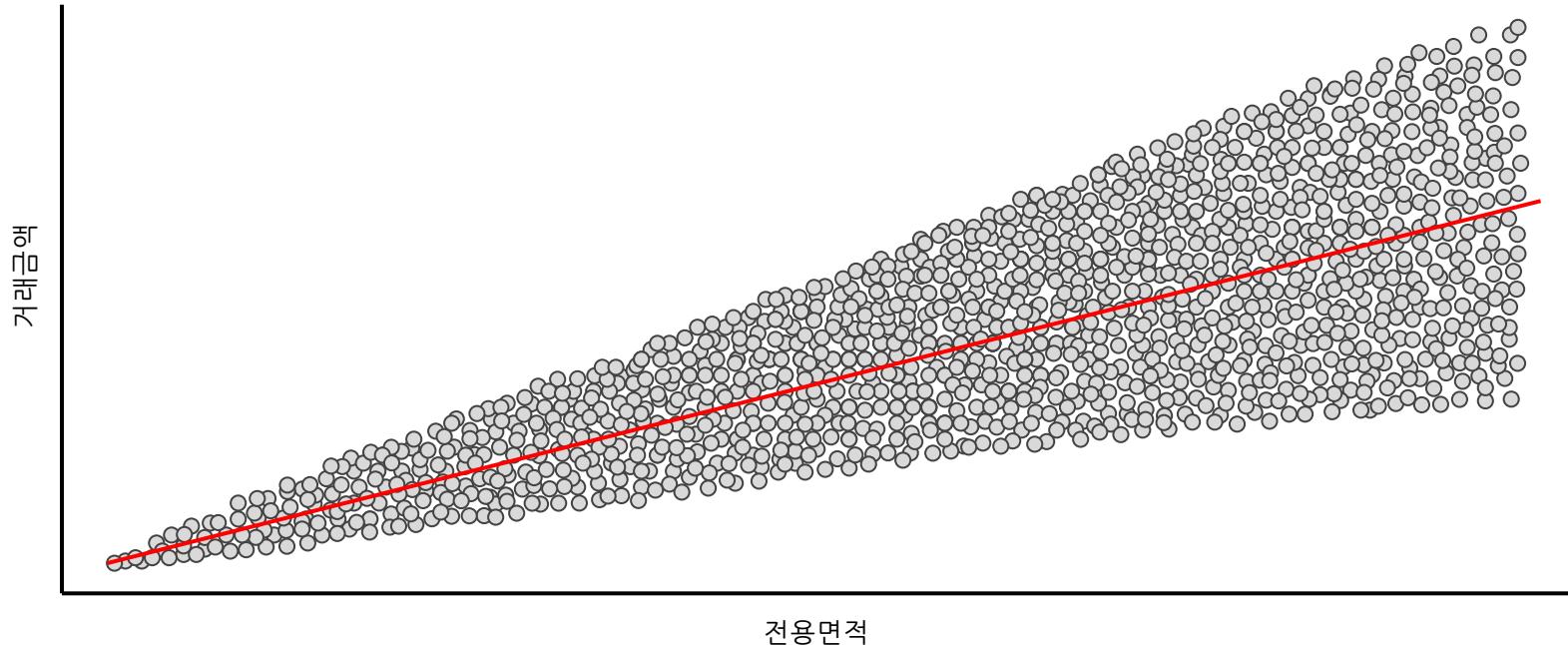
```
>>> sns.lineplot(data = sub, x = '거래월', y = '거래금액', hue = '자치구',  
                 estimator = np.mean, errorbar = None, marker = 'o');
```

- 자치구별로 점 모양을 다르게 설정합니다.

```
>>> sns.lineplot(data = sub, x = '거래월', y = '거래금액', hue = '자치구',  
                 estimator = np.mean, errorbar = None, markers = True,  
                 style = '자치구'); # markers 매개변수에 True, style 매개변수에 범주형 변수를  
                           # 지정하면 범주형 변수에 따라 점 모양을 다르게 설정합니다.
```

산점도

- 산점도는 이변량 연속형 변수의 선형관계를 점으로 표현한 그래프입니다.



산점도 그리기

- 전용면적과 거래금액으로 산점도를 그립니다.

```
>>> sns.scatterplot(data = apt, x = '전용면적', y = '거래금액',  
                    color = '0.3', # color 매개변수에 점의 채우기 색을 지정합니다.  
                           [참고] 생략하면 기본 팔레트의 첫 번째 색으로 적용합니다.  
                    ec = '0.8', # edgecolor 매개변수에 점의 테두리 색을 지정합니다.(기본값: '1')  
  
                    s = 15, # s 매개변수에 점의 크기를 지정합니다.(기본값: 50)  
  
                    alpha = 0.2); # alpha 매개변수에 채우기 색의 투명도를 0(투명) ~ 1(불투명)의  
                           실수로 지정합니다.(기본값: 1)
```

점의 채우기 색 변경

- apt를 세대수로 오름차순 정렬합니다.

```
>>> apt = apt.sort_values(by = ['세대수']) # apt를 세대수로 오름차순 정렬하고 아래 산점도를  
그리면 세대수가 큰 점이 더욱 뚜렷하게 보입니다.
```

- 세대수(연속형 변수)에 따라 채우기 색을 다르게 설정합니다.

```
>>> sns.scatterplot(data = apt, x = '전용면적', y = '거래금액',  
hue = '세대수', palette = 'RdYlGn',  
ec = '0.8', s = 15, alpha = 0.2);
```

강남구 데이터로 산점도 그리기

- apt에서 강남구만 선택하고 gng에 할당합니다.

```
>>> gng = apt[apt['자치구'].eq('강남구')]
```

- gng로 산점도를 그립니다.

```
>>> sns.scatterplot(data = gng, x = '전용면적', y = '거래금액',  
color = '0.3', ec = '0.8', s = 15, alpha = 0.2);
```

산점도에 회귀직선, 수직선 및 수평선 추가

- 점과 회귀직선 관련 그래픽 요소를 딕셔너리로 생성합니다.

```
>>> scatterProps = dict(color = '0.3', ec = '0.8', s = 15, alpha = 0.2)
```

```
>>> reglineProps = dict(color = 'red', lw = 1.5)
```

- 산점도에 회귀직선, 수직선(x 평균) 및 수평선(y 평균)을 추가합니다.

```
>>> sns.regplot(data = gng, x = '전용면적', y = '거래금액', ci = None,  
                scatter_kws = scatterProps, line_kws = reglineProps)
```

```
>>> plt.axvline(x = gng['전용면적'].mean(), lw = 0.5, ls = '--')
```

```
>>> plt.axhline(y = gng['거래금액'].mean(), lw = 0.5, ls = '--');
```

산점도 행렬 그리기

- 여러 산점도를 행렬 형태로 표현하면 여러 입력변수와 목표변수 간 관계를 빠르게 확인할 수 있습니다.
 - 산점도 행렬의 주대각에는 변수의 히스토그램, 삼각행렬에는 산점도를 그립니다.
 - 열 개수가 많으면 시간이 오래 걸리고 가독성이 떨어집니다.

- 산점도 행렬에 추가할 변수명으로 리스트를 생성합니다.

```
>>> cols = ['거래금액', '전용면적', '층', '세대수'] # [참고] 목표변수명과 입력변수명을 차례대로 지정합니다.
```

- 선택한 변수로 산점도 행렬을 그립니다.

```
>>> sns.pairplot(data = sub[cols], plot_kws = scatterProps);
```

산점도 행렬을 간결하게 그리기

- x축에 입력변수, y축에 목표변수를 지정하면 산점도 행렬을 간결하게 그립니다.

```
>>> sns.pairplot(data = sub,  
                  x_vars = ['전용면적', '층', '세대수'],  
                  y_vars = ['거래금액'], # [참고] 여러 변수명을 리스트로 지정하면 해당 개수만큼  
                           # 산점도 행렬의 행이 증가합니다.  
                  kind = 'reg', # kind 매개변수에 'reg'를 지정하면 회귀직선을 추가합니다.  
                           # [참고] 인수의 기본값은 'scatter'입니다.  
                  plot_kws = dict(scatter_kws = scatterProps,  
                                 line_kws = reglineProps));
```

탐색적 데이터 분석

탐색적 데이터 분석

- 데이터 분석 모델링에 앞서 탐색적 데이터 분석 Exploratory Data Analysis 을 실행함으로써 분석 데이터셋에 대한 이해도를 높일 수 있습니다.
- 탐색적 데이터 분석 과정에서 다양한 기술통계량을 계산하고 그래프로 데이터의 분포와 관계를 확인합니다.
- 결측값 NA 과 이상치 outlier 를 탐지하고 처리함으로써 데이터셋을 전처리하는 것 또한 탐색적 데이터 분석을 수행하는 목적 중 하나입니다.
 - 결측값은 입력된 값이 없는 상태입니다. 결측값을 대체하거나 삭제해야 합니다.
 - 이상치는 중심에서 멀리 떨어져 있는 값인데 전체 데이터의 관계를 왜곡할 수 있으므로 삭제하는 것이 좋습니다.

실습 데이터셋 소개

- 중고 자동차의 가격과 다양한 특성을 포함하는 텍스트 데이터입니다.
 - Price: 중고차 가격(달러)
 - Age: 중고차 나이(개월수)
 - KM: 주행거리(km)
 - FuelType: 연료의 종류(3종류)
 - HP: 마력
 - MetColor: 차량 색상(0/1)
 - Automatic: 미션의 종류(0/1)
 - CC: 엔진 배기량의 크기
 - Doors: 차량 문의 개수
 - Weight: 차량 무게(kg)

관련 라이브러리 호출

- 관련 라이브러리를 호출합니다.

```
>>> import os, joblib # [참고] 라이브러리를 콤마로 나열할 수 있지만 가독성 측면에는 좋지 않습니다.
```

```
>>> import numpy as np
```

```
>>> import pandas as pd
```

- 실수를 출력할 소수점 자리수를 설정합니다.

```
>>> %precision 3 # Jupyter Notebook에서 실수를 출력할 소수점 자리수를 3으로 설정합니다.
```

```
>>> pd.options.display.precision = 3 # pandas 옵션에서 실수를 출력할 소수점 자리수를 3으로 설정합니다.
```

작업 경로 확인 및 변경

- 현재 작업 경로를 확인합니다.

```
>>> os.getcwd()
```

- data 폴더로 작업 경로를 변경합니다.

```
>>> os.chdir(path = '../data')
```

- 현재 작업 경로에 있는 폴더명과 파일명을 출력합니다.

```
>>> os.listdir()
```

실습 데이터셋 준비

- 인터넷에 공유 중인 텍스트 데이터를 읽고 데이터프레임 df를 생성합니다.

```
>>> df = pd.read_csv(filepath_or_buffer = 'https://bit.ly/Used_Cars_Price')
```

- df의 정보를 확인합니다.

```
>>> df.info() # 행 개수, 열 개수, 열이름, 자료형, 열별 결측값 아닌 개수 및 자료형을 차례대로 확인합니다.
```

- df의 처음 5행을 출력합니다.

```
>>> df.head() # [참고] n 매개변수에 전달하는 인수의 기본값은 5입니다.
```

실습 데이터셋 전처리

- 범주형으로 변환할 열이름으로 리스트를 생성합니다.

```
>>> cols = ['MetColor', 'Automatic']
```

- 지정한 변수를 문자형으로 일괄 변환합니다.

```
>>> df[cols] = df[cols].astype(str)
```

- 열별 자료형을 확인합니다.

```
>>> df.dtypes # MetColor와 Automatic의 자료형이 object로 바뀌었습니다.
```

- 실수 및 정수형 변수의 기술통계량을 확인합니다.

```
>>> df.describe() # 실수 및 정수형 변수의 결측값 아닌 개수, 평균, 표준편차, 최솟값, 사분위수 및 최댓값을 확인합니다.
```

실습 데이터셋 전처리(계속)

- df를 KM로 오름차순 정렬하고 처음 5행을 출력합니다.

```
>>> df.sort_values(by = ['KM']).head()
```

- df에서 KM가 1보다 큰 행을 선택하고 행이름을 초기화합니다.

```
>>> df = df[df['KM'].gt(1)].reset_index(drop = True)
```

- df의 행 개수를 확인합니다.

```
>>> df.shape[0]
```

- 범주형 변수의 기술통계량을 확인합니다.

```
>>> df.describe(include = object) # 문자형 변수의 결측값 아닌 개수, 중복 제거한 원소 개수, 최빈값  
 및 최빈값의 빈도수를 확인합니다.
```

실습 데이터셋 전처리(계속)

- 범주형 변수의 범주별 빈도수를 출력합니다.

```
>>> df['FuelType'].value_counts().sort_index()
```

```
>>> df['MetColor'].value_counts().sort_index()
```

```
>>> df['Automatic'].value_counts().sort_index()
```

- 범주형 변수의 범주별 상대도수를 출력합니다.

```
>>> df['FuelType'].value_counts(normalize = True).sort_index()
```

```
>>> df['MetColor'].value_counts(normalize = True).sort_index()
```

```
>>> df['Automatic'].value_counts(normalize = True).sort_index()
```

시각화 설정: 라이브러리 호출

- 관련 라이브러리를 호출합니다.

```
>>> import seaborn as sns # 고급 시각화 함수를 포함하는 라이브러리입니다.
```

```
>>> import matplotlib.pyplot as plt # 그래프 크기, 제목, 축이름 등을 지정할 때 사용합니다.
```

```
>>> import matplotlib.font_manager as fm # 한글폰트를 지정할 때 사용합니다.
```

- 테스트용 그래프를 그립니다.

```
>>> sns.histplot(data = df, x = 'Price')
```

```
>>> plt.title(label = '중고차 가격 분포'); # [참고] 시각화 코드 마지막에 추가한 세미콜론(;)은  
plt.show() 함수와 같은 기능을 실행합니다.
```

위 코드를 실행하면 한글을 네모로 출력하므로 한글폰트를 설정해야 합니다.

[참고] 한글폰트 외 그래프 크기 및 해상도 등 다양한 그래픽 파라미터를 설정할 수 있습니다.

시각화 설정: 한글폰트명 탐색

- 현재 사용 중인 컴퓨터에 설치한 전체 폰트 파일명을 리스트로 반환합니다.

```
>>> fontList = fm.findSystemFonts(fontext = 'ttf'); fontList
```

- 리스트에서 특정 문자열(폰트명)을 포함하는 파일명만 선택합니다.

```
>>> fontPath = [font for font in fontList if 'Gamja' in font]; fontPath
```

- 반복문으로 컴퓨터에 설치된 폰트명을 출력합니다.

```
>>> for font in fontPath:
```

```
    print(fm.FontProperties(fname = font).get_name())
```

반복문을 실행한 결과에서 마음에 드는 폰트명을 선택하고 plt.rc() 함수에 지정합니다.
[참고] rc는 runtime configuration를 의미하며, pyplot을 실행하는 환경을 의미합니다.

시각화 설정: 그래픽 파라미터 설정

- 그래프 크기와 해상도를 설정합니다.

```
>>> plt.rc(group = 'figure', figsize = (4, 4), dpi = 150)
```

- 한글폰트와 글자 크기를 설정합니다.

```
>>> plt.rc(group = 'font', family = 'Gamja Flower', size = 10)
```

- 축에 유니코드 마이너스를 출력하지 않도록 설정합니다.

```
>>> plt.rc(group = 'axes', unicode_minus = False) # [참고] 왼쪽 코드를 설정하지 않으면  
음수 앞에 '‑'를 출력합니다.
```

- 범례에 채우기 색과 테두리 색을 추가합니다. # fc는 facecolor(채우기), ec는 edgecolor(테두리)
관련 매개변수이고 '0'은 검정, '1'은 흰색입니다.

```
>>> plt.rc(group = 'legend', frameon = True, fc = '1', ec = '0')
```

[참고] 그래픽 파라미터 설정 관련 모듈 생성

- 시각화 라이브러리 호출, 스타일시트, 한글폰트 및 그래픽 파라미터를 설정하는 코드를 모듈(py 파일)로 저장하면 필요할 때마다 호출할 수 있으므로 편리합니다.
- Anaconda 메인에서 New Text File을 열고, 모듈(py 파일)로 저장할 시각화 설정 관련 코드를 붙여넣습니다.
- 상단 메뉴에서 File → Save를 클릭하여 텍스트 파일을 저장합니다.
 - 파일명을 GraphicSetting.py로 변경합니다. # [주의] py 파일을 현재 사용 중인 Jupyter Notebook 파일과 같은 폴더에 저장하면 쉽게 호출할 수 있습니다.
- 시각화 설정 모듈을 호출합니다.

```
>>> from GraphicSetting import *
```

[참고] Python 파일 탐색 경로 확인

- 모듈(py 파일)을 호출하려면 아래 두 가지 조건 중 하나를 만족해야 합니다.
 - 현재 작업 중인 Jupyter Notebook 파일 경로에 py 파일을 저장했다.
 - Python 파일 탐색 경로 중 한 곳에 py 파일을 저장했다.
- 작업할 Jupyter Notebook 파일 경로가 바뀔 때마다 py 파일을 매번 옮겨야 하므로 첫 번째 조건은 불편합니다. 따라서 두 번째 조건을 따르는 것이 좋습니다.
- Python 파일 탐색 경로를 확인합니다.

```
>>> import sys # 관련 라이브러리를 호출합니다.
```

```
>>> sys.path # Python 파일 탐색 경로를 모두 출력합니다. 많은 경로 중 한 곳(마지막 경로 추천)에 py 파일을  
저장하면 해당 모듈을 항상 호출할 수 있습니다.
```

[참고] 시각화 함수 모듈 제공

- Python 통계 분석 및 머신러닝 강의에서 사용할 시각화 함수를 **HelloDataScience** 모듈(py 파일)로 제공합니다. 주요 함수는 아래와 같습니다.
 - 변수의 관계 확인: `plot_regression`, `plot_box_plot`, `plot_bar_dodge_freq` 등
 - 모형의 성능 평가: `regmetrics`, `plot_roc`, `clfmetrics`, `EpiROC` 등
- 시각화 및 통계 분석 관련 모듈을 호출합니다.

```
>>> import HelloDataScience as hds
```

```
# HelloDataScience 모듈의 의존성 라이브러리인 sklearn, statsmodels, graphviz를 먼저 설치해야 합니다.  
>>> !pip install sklearn # 왼쪽 코드에서 라이브러리 이름만 바꿔서 실행합니다.
```

```
# (아나콘다) pip로 graphviz 라이브러리를 설치할 수 없으면 아래 코드를 실행하세요. 설치 시간이 길니다.  
>>> !conda install python-graphviz
```

목표변수 분포 확인

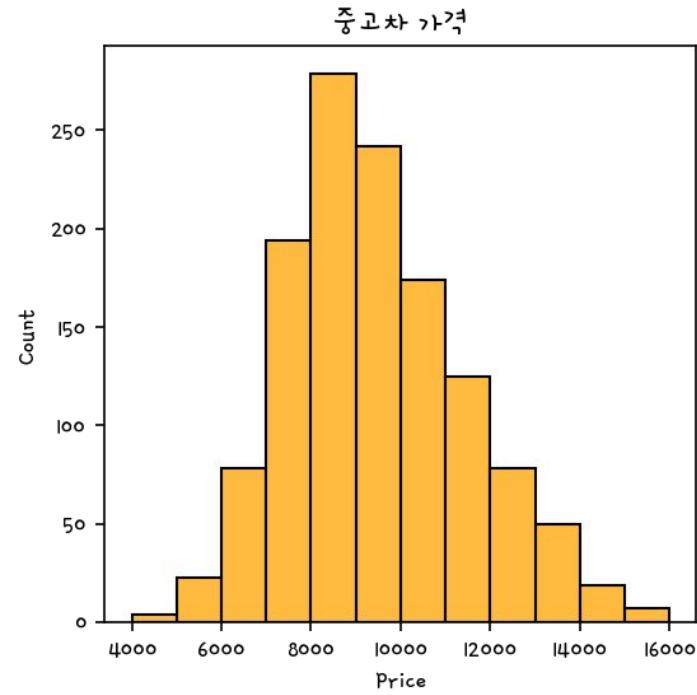
- 목표변수의 최솟값과 최댓값을 확인합니다.

```
>>> df['Price'].describe()[['min', 'max']]
```

- 히스토그램을 그립니다.

```
>>> sns.histplot(data = df, x = 'Price',  
                 binwidth = 1000,  
                 binrange = (4000, 16000),  
                 color = 'orange')
```

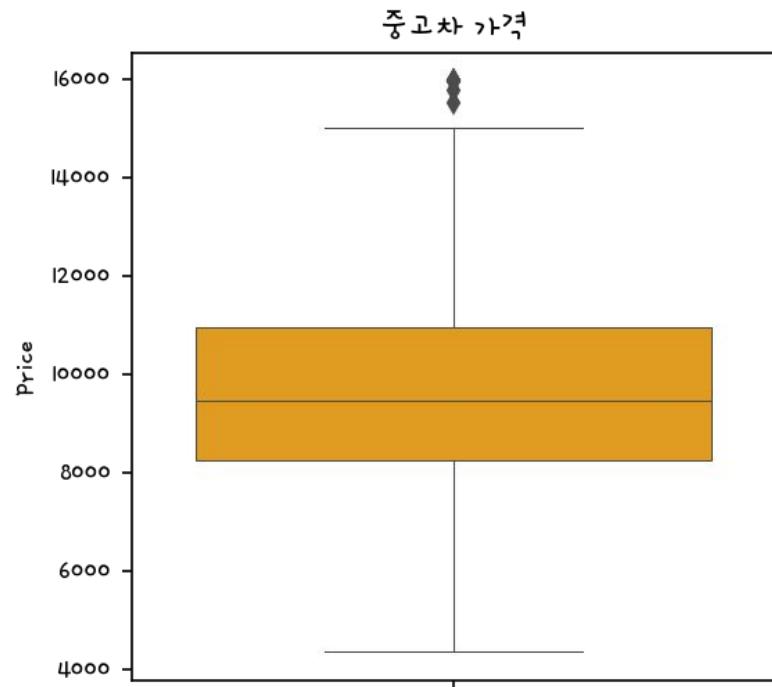
```
>>> plt.title(label = '중고차 가격');
```



목표변수 분포 확인(계속)

- 상자 수염 그림을 그립니다.

```
>>> sns.boxplot(data = df,  
                 y = 'Price',  
                 color = 'orange')  
  
>>> plt.title(label = '중고차 가격');
```



연속형 입력변수와 관계 파악: Age

- Age와 Price의 산점도를 그립니다.

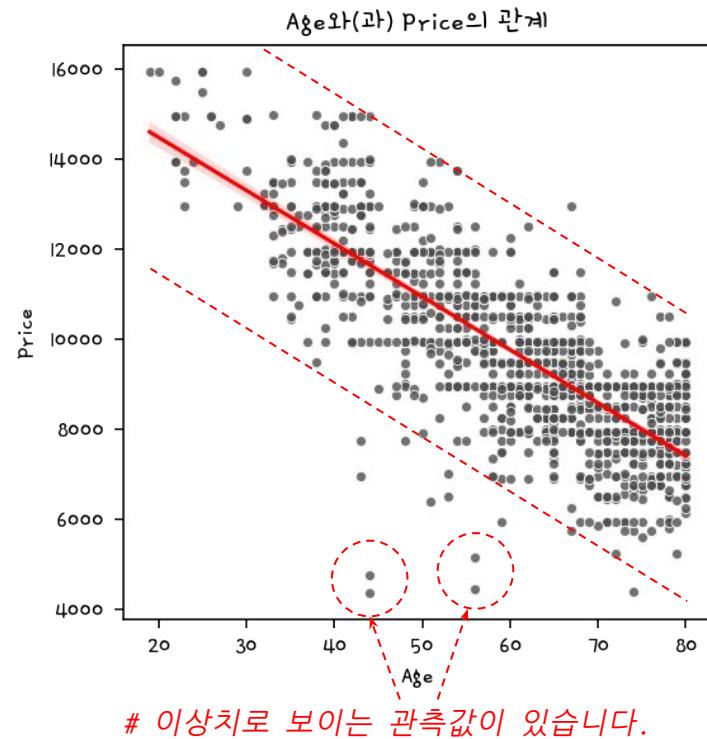
```
>>> hds.plot_regression(
```

```
    data = df,
```

```
    x = 'Age',
```

```
    y = 'Price'
```

```
)
```



연속형 입력변수와 관계 파악: KM

- KM와 Price의 산점도를 그립니다.

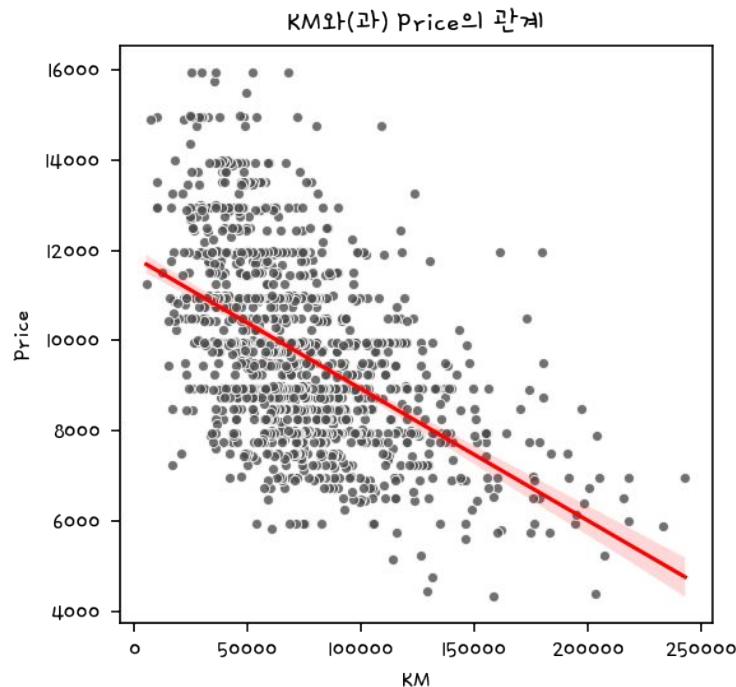
```
>>> hds.plot_regression(
```

```
    data = df,
```

```
    x = 'KM',
```

```
    y = 'Price'
```

```
)
```



연속형 입력변수와 관계 파악: HP

- HP와 Price의 산점도를 그립니다.

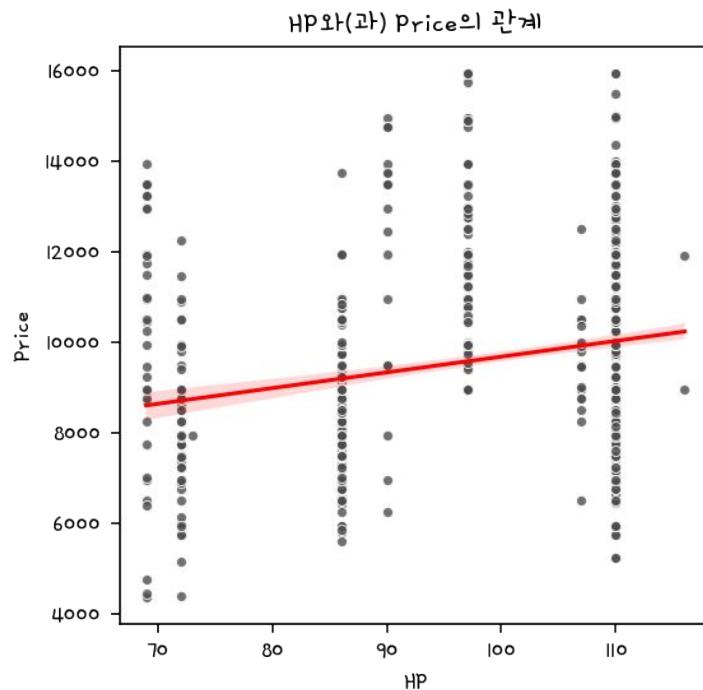
```
>>> hds.plot_regression(
```

```
    data = df,
```

```
    x = 'HP',
```

```
    y = 'Price'
```

```
)
```



연속형 입력변수와 관계 파악: CC

- CC와 Price의 산점도를 그립니다.

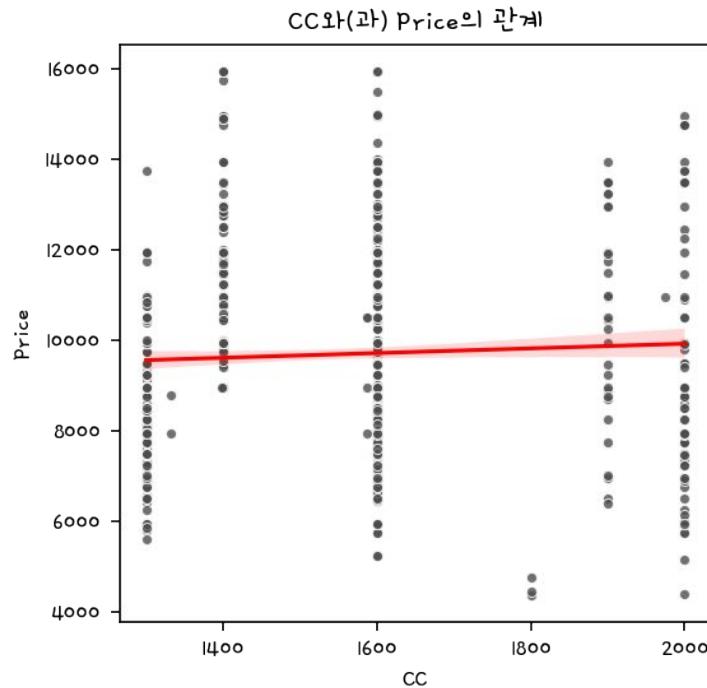
```
>>> hds.plot_regression(
```

```
    data = df,
```

```
    x = 'CC',
```

```
    y = 'Price'
```

```
)
```



연속형 입력변수와 관계 파악: Doors

- Doors와 Price의 산점도를 그립니다.

```
>>> hds.plot_regression(
```

```
    data = df,
```

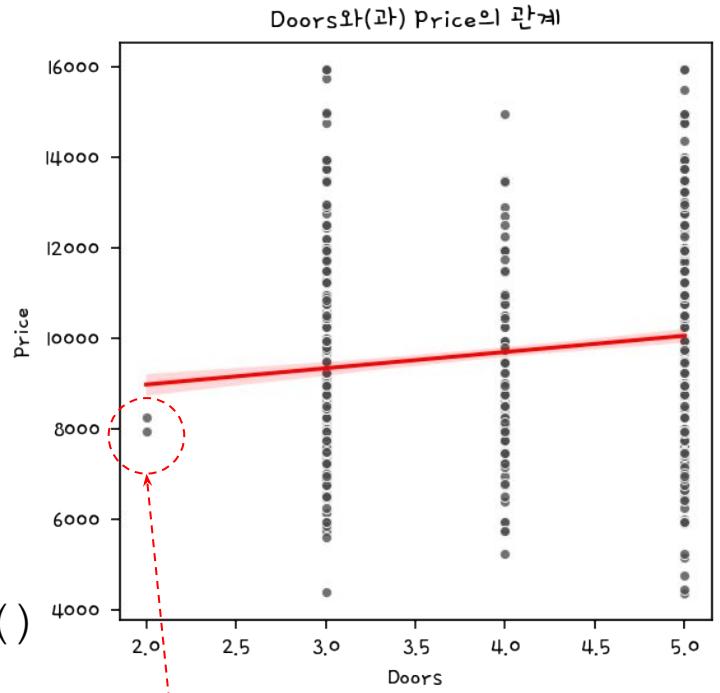
```
    x = 'Doors',
```

```
    y = 'Price'
```

```
)
```

```
>>> df['Doors'].value_counts().sort_index()
```

Doors의 원소별 빈도수를 확인합니다.



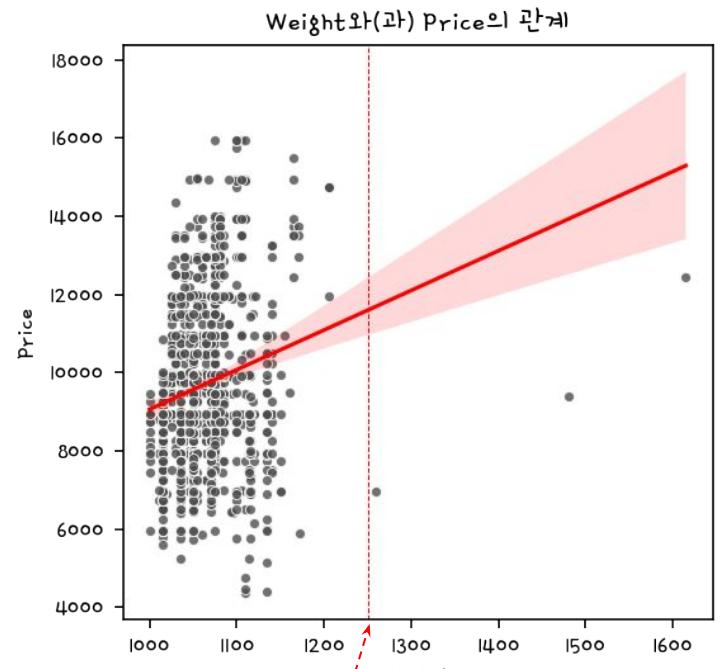
Doors가 2인 관측값은 소수라 삭제합니다.

연속형 입력변수와 관계 파악: Weight

- Weight와 Price의 산점도를 그립니다.

```
>>> hds.plot_regression(  
    data = df,  
    x = 'Weight',  
    y = 'Price'  
)
```

```
>>> plt.axvline(x = 1250, color = 'red',  
    # 세로선을  
    추가합니다.      lw = 0.5, ls = '--');
```

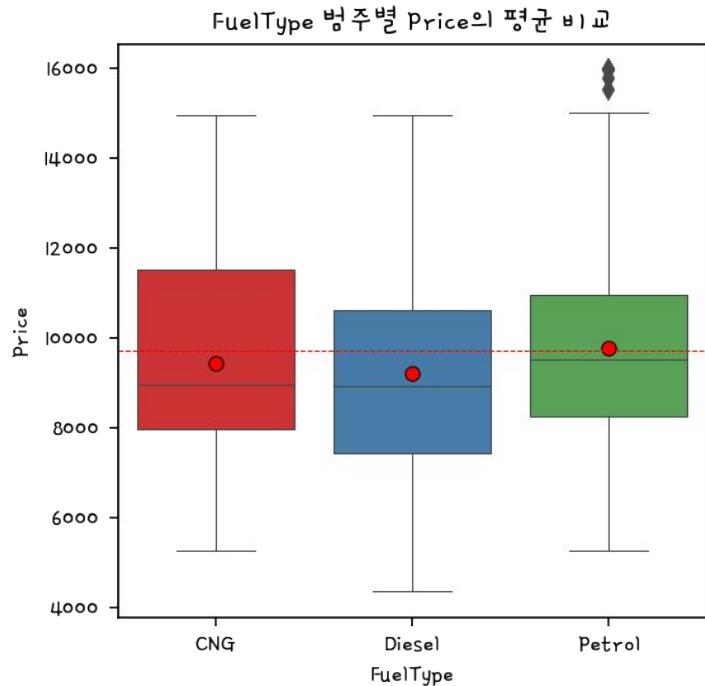


범주형 입력변수와 관계 파악: FuelType

- FuelType 범주별 Price의 상자 수염 그림을 그립니다.

```
>>> hds.plot_box_group(
```

```
    data = df,  
    x = 'FuelType',  
    y = 'Price',  
    pal = 'Set1'  
)
```



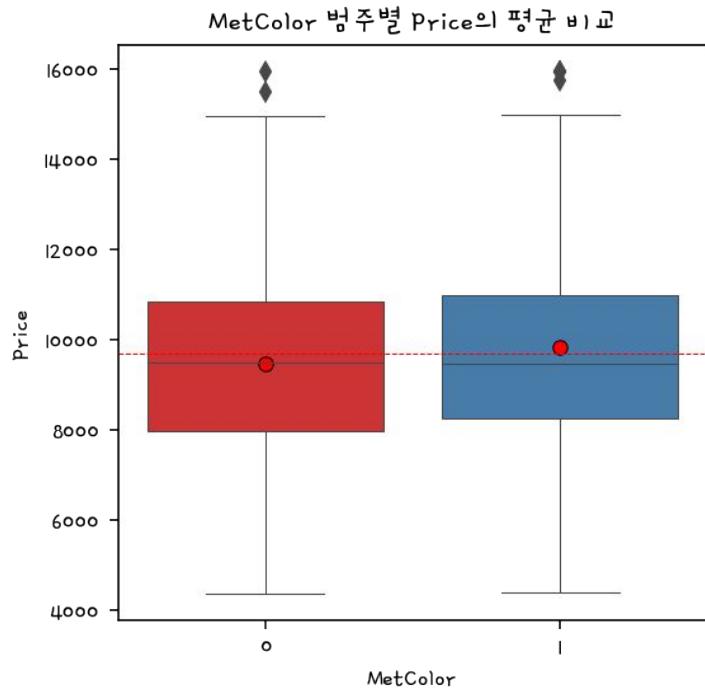
[참고] Price 범주별 평균(점)과 전체 평균
(수평선)을 추가합니다.

범주형 입력변수와 관계 파악: MetColor

- MetColor 범주별 Price의 상자 수염 그림을 그립니다.

```
>>> hds.plot_box_group(
```

```
    data = df,  
    x = 'MetColor',  
    y = 'Price',  
    pal = 'Set1'  
)
```

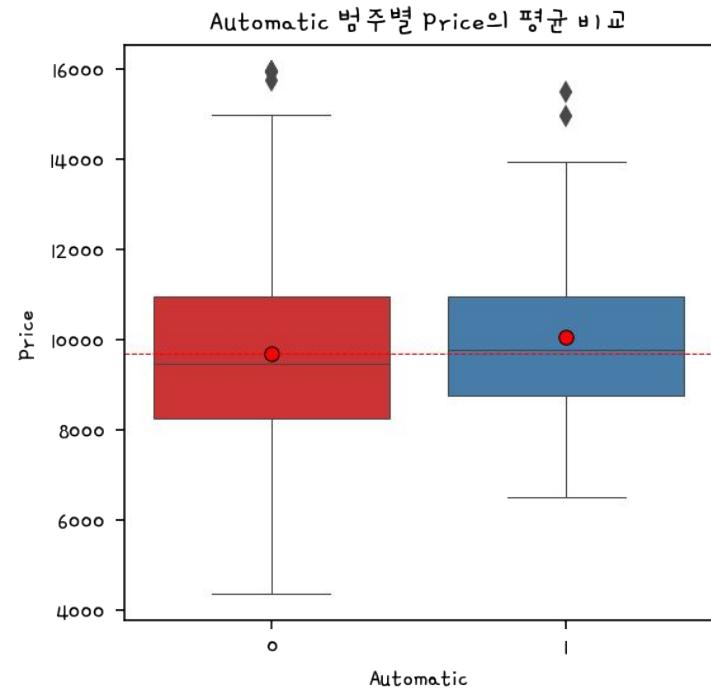


범주형 입력변수와 관계 파악: Automatic

- Automatic 범주별 Price의 상자 수염 그림을 그립니다.

```
>>> hds.plot_box_group(
```

```
    data = df,  
    x = 'Automatic',  
    y = 'Price',  
    pal = 'Set1'  
)
```



불필요한 행 삭제

- 시각화 결과 불필요하다고 판단하는 일부 행을 삭제합니다.

```
>>> df = df[df['Doors'].ne(2) & df['Weight'].le(1250)]
```

- df의 행이름을 초기화합니다.

```
>>> df = df.reset_index(drop = True)
```

- df의 행 개수를 확인합니다.

```
>>> df.shape[0]
```

- 열별 기술통계량을 확인합니다.

```
>>> df.describe()
```

외부 파일로 저장

- df를 xlsx 파일로 저장합니다.

```
>>> df.to_excel(excel_writer = 'Used_Cars_Price.xlsx', index = None)
```

- df를 csv 파일로 저장합니다.

```
>>> df.to_csv(path_or_buf = 'Used_Cars_Price.csv', index = None)
```

- df를 z 파일로 저장합니다.

```
>>> joblib.dump(value = df, filename = 'Used_Cars_Price.z')
```

기술통계 분석

통계 개요

- 통계^{statistic}는 데이터를 수집, 요약 및 추론 등을 통해 분석하는 방법론입니다.
 - 데이터 수집은 표본 추출이고 요약은 기술 통계, 추론은 모수 추정을 의미합니다.
- 통계와 관련된 주요 용어를 소개합니다.
 - 모집단^{population}: 조사 대상이 되는 개체의 전체 집합이며, 유한하거나 무한합니다.
 - 모수^{parameter}: 모집단을 대표하는 특성으로, 수치를 요약(기술 통계)한 값입니다.
 - 표본^{sample}: 모집단에서 추출한 부분 집합이며, 모집단을 대표해야 합니다.
 - 통계량^{statistics}: 표본의 수치를 요약한 값입니다.
- 통계 분석은 표본 통계량을 이용하여 모집단의 특성인 모수를 추정합니다.

기술통계

- 기술통계 Descriptive Statistics는 데이터의 주요 특징을 빠르게 파악할 때 사용합니다.
 - 'Descriptive'라는 단어에서 알 수 있듯이 데이터를 기술, 묘사한다는 것을 의미합니다.
- 기술통계의 주요 통계량에는 다음과 같습니다.
 - 대푯값: 평균, 절사평균, 중위수, 분위수, 사분위수, 최솟값, 최댓값, 최빈값
 - 산포: 범위, 사분범위, 분산, 표준편차, 중위수절대편차

관련 라이브러리 호출

- 관련 라이브러리를 호출합니다.

```
>>> import os, joblib
```

```
>>> import numpy as np
```

```
>>> import pandas as pd
```

- 실수를 출력할 소수점 자리수를 설정합니다.

```
>>> %precision 3
```

```
>>> pd.options.display.precision = 3
```

관련 라이브러리 호출(계속)

- statsmodels 라이브러리를 설치합니다.

```
>>> !pip install statsmodels
```

- 통계 관련 라이브러리를 호출합니다.

```
>>> from scipy import stats # 절사평균 관련 함수를 포함하는 모듈을 호출합니다.
```

```
>>> from statsmodels import robust # 중위수절대편차 관련 함수를 포함하는 모듈을 호출합니다.
```

작업 경로 확인 및 변경

- 현재 작업 경로를 확인합니다.

```
>>> os.getcwd()
```

- data 폴더로 작업 경로를 변경합니다.

```
>>> os.chdir(path = '../data')
```

- 현재 작업 경로에 있는 폴더명과 파일명을 출력합니다.

```
>>> os.listdir()
```

실습 데이터셋 준비

- xlsx 파일을 읽고 데이터프레임 df를 생성합니다.

```
>>> df = pd.read_excel(io = 'Used_Cars_Price.xlsx')
```

- df의 열별 자료형을 출력합니다.

```
>>> df.dtypes
```

MetColor와 Automatic의 자료형이 numpy.int64입니다.
[참고] xlsx 파일로 저장되면서 문자열 '0'과 '1'을 정수로 변환합니다.

- z 파일을 읽고 데이터프레임 df를 생성합니다.

```
>>> df = joblib.load(filename = 'Used_Cars_Price.z')
```

- df의 열별 자료형을 출력합니다.

```
>>> df.dtypes
```

MetColor와 Automatic의 자료형이 object입니다.
[참고] z 파일은 Python 객체를 그대로 저장하므로 자료형을 그대로 유지합니다.

평균

- 평균`mean`은 연속형 변수의 원소를 모두 더한 값을 원소 개수로 나눈 값입니다.
 - 평균은 전체 데이터를 포함하므로 이상치에 민감하게 영향을 받는다는 단점이 있습니다.
- 연속형 변수의 평균을 반환합니다.

```
>>> df['Price'].mean()
```

- 시리즈에 결측값^{NA}이 있으면 `mean()` 함수는 결측값을 제거한 평균을 반환합니다.

```
>>> nums = pd.Series(data = [1, np.nan, 2]) # 결측값을 포함하는 시리즈를 생성합니다.
```

```
>>> nums # nums의 1번 인덱스(두 번째) 원소가 결측값입니다.
```

```
>>> nums.mean() # nums의 평균을 반환합니다.
```

절사평균

- 절사평균^{trimmed mean}은 연속형 변수의 양 극단에서 일부를 제외한 평균입니다.
 - 아래 그림은 연속형 변수를 10등분한 십분위수를 표현한 것입니다.
 - 0%는 최솟값, 50%는 중위수, 그리고 100%는 최댓값입니다.
 - 10% 절사평균을 계산할 때 양 극단에서 10%씩 잘라내고 남은 80%로 평균을 계산합니다.



- 연속형 변수의 10% 절사평균을 반환합니다.

```
>>> stats.trim_mean(df['Price'], 0.1)
```

중위수

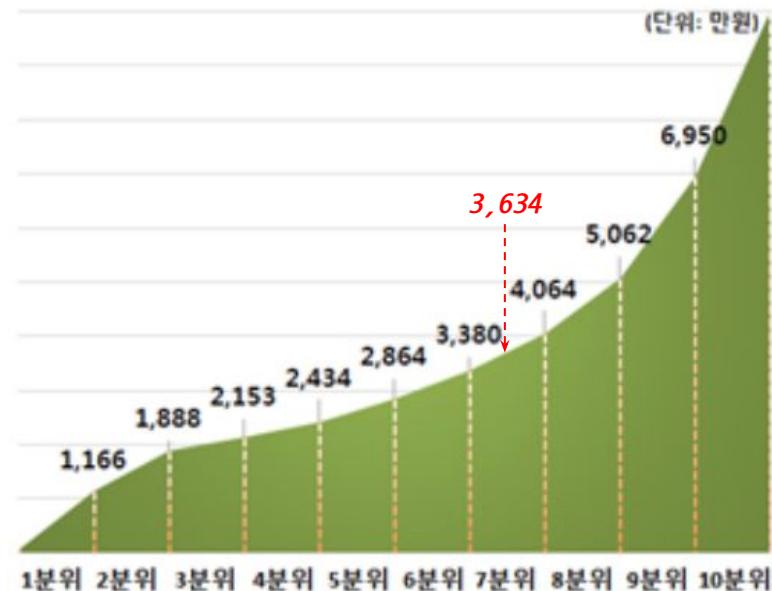
- 중위수^{median}는 연속형 변수를 오름차순 정렬했을 때 정가운데에 위치한 값입니다.
 - 원소 개수가 홀수면 정가운데 값을 반환합니다.
 - 원소 개수가 짝수면 정가운데에 있는 두 원소(수)의 평균을 반환합니다.
 - 중위수는 이상치에 민감한 평균에 비해 로버스트^{robust}한 통계량입니다.
 - 로버스트한 통계량은 이상치에 영향을 적게 받는 값을 의미합니다.
- 연속형 변수의 중위수를 반환합니다.

```
>>> df['Price'].median()
```

```
>>> stats.trim_mean(df['Price'], 0.5) # [참고] 50% 결사평균을 계산하면 원소 개수가 홀수일 때  
중위수, 짝수일 때 결측값을 반환합니다.
```

[참고] 우리나라 근로자 연봉 평균 및 중위수(2018년)

- 2018년 기준 우리나라 근로자 평균 연봉은 약 3,634만원이었습니다.
- 그러나 오른쪽 그림에서 보이는 바와 같이 중위수는 약 2,864만원입니다.
- 이와 같이 이상치가 많은 데이터의 대표값으로는 평균 대신 중위수를 사용하는 것이 좋습니다.
 - 좌우대칭하는 분포는 중위수 대신 평균을 사용합니다.



분위수와 사분위수

- 분위수^{quantile}는 오름차순 정렬한 연속형 변수를 n 등분하는 경계 값입니다.
 - `quantile()` 함수의 q 매개변수에 0~1의 값을 지정하면 해당 분위수를 반환합니다.
- 전체 데이터를 100 등분하는 경계 값인 백분위수^{percentile}를 반환합니다.

```
>>> df['Price'].quantile(q = np.linspace(start = 0, stop = 1, num = 100+1))
```
- 전체 데이터를 10 등분하는 경계 값인 십분위수^{decile}를 반환합니다.

```
>>> df['Price'].quantile(q = np.linspace(start = 0, stop = 1, num = 10+1))
```
- 전체 데이터를 4 등분하는 경계 값인 사분위수^{quartile}를 반환합니다.

```
>>> df['Price'].quantile(q = np.linspace(start = 0, stop = 1, num = 4+1))
```

범위와 사분범위

- 범위^{range}는 연속형 변수의 최댓값과 최솟값의 간격입니다.

```
>>> df['Price'].max() - df['Price'].min()
```

```
>>> df['Price'].quantile(q = [0, 1]).diff().iloc[-1] # diff() 함수는 원소간 차이를 반환합니다. (최댓값 - 최솟값)
```

- 사분범위^{interquartile range}는 연속형 변수의 3사분위수와 1사분위수의 간격입니다.

```
>>> df['Price'].quantile(q = [0.25, 0.75]).diff().iloc[-1]
```



최빈값

- 최빈값^{mode}은 이산형 또는 범주형 변수에서 빈도수가 가장 높은 원소를 의미합니다.

```
>>> df['FuelType'].mode() # FuelType의 최빈값을 반환합니다.
```

- 범주형 변수의 빈도수를 내림차순 정렬한 결과를 반환합니다.

```
>>> df['FuelType'].value_counts() # [참고] value_counts() 함수의 ascending 매개변수에 전달하는  
인수의 기본값이 False입니다.
```

- 범주형 변수의 빈도수를 인덱스로 오름차순 정렬한 결과를 반환합니다.

```
>>> df['FuelType'].value_counts().sort_index()
```

- 범주형 변수의 빈도수 대신 상대도수를 내림차순 정렬한 결과를 반환합니다.

```
>>> df['FuelType'].value_counts(normalize = True)
```

평균, 분산 및 표준편차의 관계

- 아래는 A 중학교 학생 n명의 수학 시험 점수를 수집하여 표로 정리한 것입니다.

순번(i)	데이터(X_i)	편차($X_i - \bar{X}$)	편차 제곱
1	85	85 - 78	$(85 - 78)^2$
2	72	72 - 78	$(72 - 78)^2$
:	:	:	:
n	97	97 - 78	$(97 - 78)^2$
합계	$\sum_{i=1}^n X_i$	$\sum_{i=1}^n (X_i - \bar{X}) = 0$	$\sum_{i=1}^n (X_i - \bar{X})^2$
평균	$\frac{1}{n} \sum_{i=1}^n X_i = \bar{X}$	$\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X}) = 0$	$\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2 = s^2$

편의상 X_i 의 평균을 78점으로
가정합니다.

분산 variance의 양의 제곱근은
표준편차 standard deviation입니다.

분산

- 분산^{variance}은 개별 관측값이 중심에서 떨어져 있는 정도를 나타내는 값입니다.
 - 개별 관측값이 중심(평균)에서 떨어져 있는 크기를 편차^{deviation}라고 합니다.
 - [참고] 편차를 단순하게 더하면 0이 되므로 제곱하여 부호를 통일해야 합니다.
 - 분산은 편차 제곱의 평균을 계산한 것입니다. 불편추정량이 되도록 $n-1$ 로 나눕니다.
 - 분산은 개별 관측값이 평균에서부터 평균적으로 떨어져 있는 정도를 나타냅니다.
 - 따라서 분산이 작을수록 개별 관측값이 평균에 밀집해 있다고 판단합니다.
- 연속형 변수의 분산을 반환합니다.

```
>>> df['Price'].var()
```

표준편차

- 표준편차 standard deviation 도 개별 관측값이 중심에서 떨어져 있는 정도를 나타냅니다.
 - 분산은 편차 제곱의 평균인데, 제곱했으므로 원래 변수의 척도와 다릅니다.
 - 예를 들어 단위가 cm인 연속형 변수의 분산을 계산하면 단위가 cm^2 로 바뀝니다.
 - 표준편차는 분산의 양의 제곱근이며, 원래 변수와 척도가 같아집니다.
 - 개별 관측값이 중심(평균)에서 떨어져 있는 정도를 파악할 때 표준편차를 확인합니다.
 - 표준편차도 작을수록 개별 관측값이 평균에 밀집해 있다고 판단합니다.
- 연속형 변수의 표준편차를 반환합니다.

```
>>> df['Price'].std()
```

중위수절대편차

- 중위수절대편차^{median absolute deviation}는 평균 대신 중위수를 사용하므로 표준편차보다 더욱 로버스트한 통계량이며 다음과 같이 계산합니다.
 - 연속형 변수의 중위수를 계산하고 개별 원소에서 중위수를 뺀 편차를 구합니다.
 - 중위수절대편차는 편차 절대값의 중위수를 계산하고 모수 추정 상수 1.4826을 곱합니다.

$$MAD = \text{median} \left(|X_i - \text{median}(X)| \right) \times 1.4826$$

- 연속형 변수의 중위수절대편차를 반환합니다.

```
>>> robust.mad(a = df['Price'])
```

여러 열의 기술통계량 확인

- 여러 연속형 변수의 평균, 표준편차 및 중위수를 반환합니다.

```
>>> df.apply(func = 'mean', numeric_only = True) # numeric_only는 pd.Series 통계 함수에 정의된 매개변수입니다.
```

```
>>> df.apply(func = 'std', numeric_only = True)
```

```
>>> df.apply(func = 'median', numeric_only = True)
```

- 여러 연속형/범주형 변수의 기술통계량을 반환합니다.

```
>>> df.describe() # 연속형 변수의 원소 개수, 평균, 표준편차, 최솟값, 사분위수 및 최댓값을 출력합니다.  
[참고] 표준편차를 계산할 때 분자를  $n-1$ 로 나눕니다.
```

```
>>> df.describe(include = object) # 범주형 변수의 원소 개수, 중복 제거한 원소의 종류, 최빈값 및 최빈값의 빈도수를 출력합니다.
```

확률의 개념과 특징

집합과 원소

- 집합은 어떤 조건을 만족하는 대상들이 모인 무리를 의미합니다.
 - 집합에 속한 대상을 원소라고 하며, 원소를 중복하여 표기하지 않습니다.
 - 집합은 어떤 원소가 그 집합에 속하는지 여부를 명확하게 구분합니다.
- 일반적으로 집합은 알파벳 대문자로 표기하고, 집합의 원소는 알파벳 소문자로 표기합니다.
 - 집합 A 에 원소 a 가 속한다면 다음과 같이 표기합니다. $a \in A$
 - 반대로 원소 a 가 집합 A 에 속하지 않는다면 다음과 같이 표기합니다. $a \notin A$
 - 원소가 없는 공집합은 \emptyset 로 표기합니다.

벤다이어그램

- 두 집합의 관계를 그림으로 표현한 것을 벤다이어그램이라고 합니다.
- 집합 A와 B는 교집합을 갖습니다.

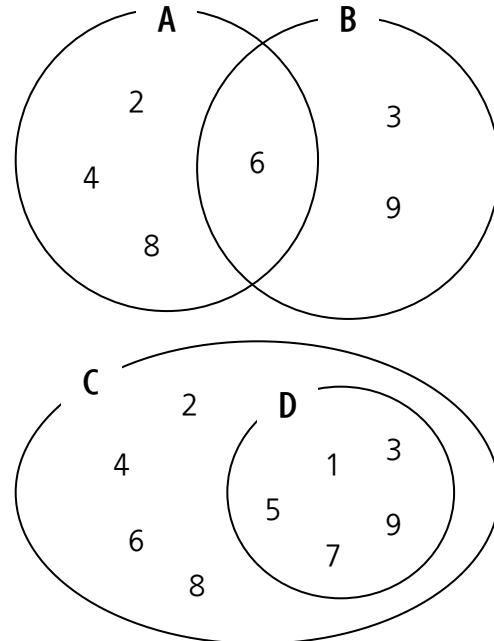
집합 A = {x | x는 10보다 작은 2의 배수}

집합 B = {x | x는 10보다 작은 3의 배수}

- 집합 D는 C의 부분집합입니다.

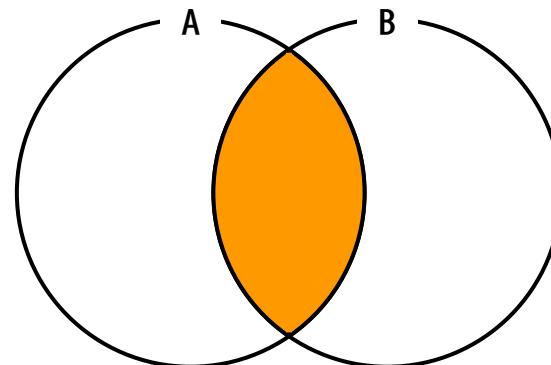
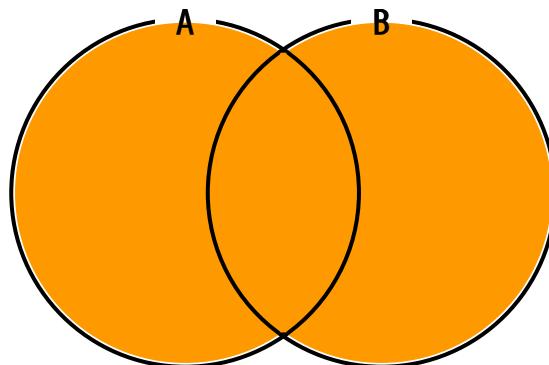
집합 C = {x | x는 10보다 작은 양의 정수}

집합 D = {x | x는 10보다 작은 양의 홀수}



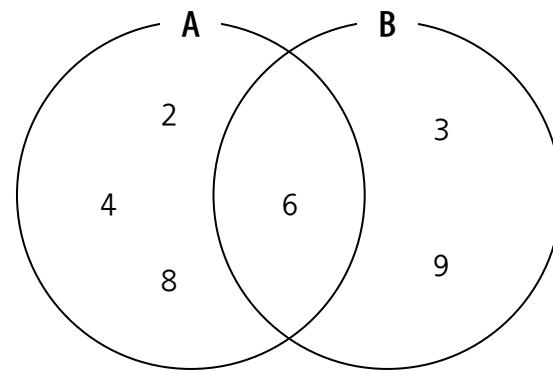
벤다이어그램(계속)

- 집합 A와 B의 합집합을 벤다이어 그램으로 그린 것입니다.
- 집합 A와 B의 교집합을 벤다이어 그램으로 그린 것입니다.



집합의 관계

- 집합 A와 B의 관계에 따라 다음과 같이 표현할 수 있습니다.
 - 두 집합에 모두 속하는 원소를 갖는 집합은 집합 A와 B의 교집합입니다.
 - 두 집합 중 어느 한 집합에 속하는 원소를 갖는 집합은 집합 A와 B의 합집합입니다.
 - 집합 A에서 교집합의 원소를 제외한 집합은 차집합입니다.



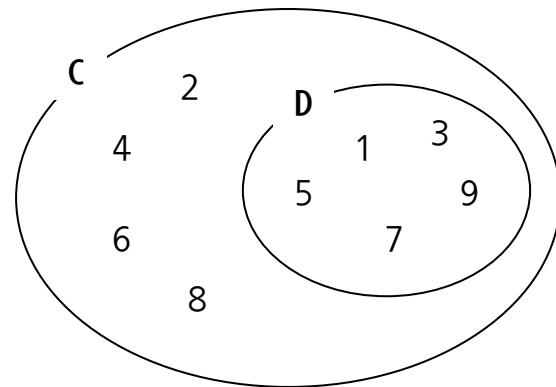
$$\text{교집합: } A \cap B = \{6\}$$

$$\text{합집합: } A \cup B = \{2, 3, 4, 6, 8, 9\}$$

$$\text{차집합: } A - B = \{2, 4, 8\}$$

집합의 관계(계속)

- 집합 C와 D의 관계에 따라 다음과 같이 표현할 수 있습니다.
 - 모든 원소를 갖는 집합은 전체집합입니다.
 - 전체집합에서 일부 원소를 갖는 집합은 부분집합입니다.
 - 부분집합 원소가 아닌 원소를 갖는 집합은 여집합입니다.
 - 원소가 하나도 없는 집합은 공집합입니다.
[참고] 공집합도 부분집합입니다.



전체집합은 자신의 부분집합: $C \subset C$

집합 D는 집합 C의 진부분집합: $D \subset C$

여집합: $D^c = C - D = \{2, 4, 6, 8\}$

공집합은 전체집합의 부분집합: $\emptyset \subset C$

확률 관련 용어

- 확률에서 자주 사용하는 용어에 대한 정의를 알아보겠습니다.
 - 시행^{trial}: 무작위 실험^{experiment} 또는 관찰^{observation}을 반복하여 결과물을 얻는 과정입니다.
 - 표본 공간^{sample space}: 시행에서 발생 가능한 모든 결과의 집합이며 보통 S 로 표기합니다.
 - 사건^{event}: 사전에 정의한 조건을 만족하는 시행의 결과입니다. # 알파벳 대문자로 표기합니다.
 - 표본 공간은 전체집합이고, 사건은 부분집합입니다.
 - 근원사건: 표본공간의 부분집합 중 원소가 1개인 사건입니다.
 - 전사건: 반드시 일어나는 사건이며, 표본 공간과 같습니다.
 - 공사건: 절대로 일어날 수 없는 사건입니다.

확률

- 확률은 일정한 조건 하에서 어떤 사건이 우연하게 발생할 가능성의 정도를 0~1의 실수로 나타낸 것입니다.
 - 확률은 0~1의 실수이지만 확률에 100을 곱한 백분율percentage로 표기하기도 합니다.
- 확률은 수학적 확률과 통계적 확률 등 두 가지로 구분합니다.

$$\text{수학적 확률} = \frac{\text{사건의 원소 개수}}{\text{표본 공간의 원소 개수}}$$

$$\text{통계적 확률} = \frac{\text{사건의 발생 횟수}}{\text{시행 횟수}}$$

주사위를 한 번 던졌을 때 짹수가 나올 확률

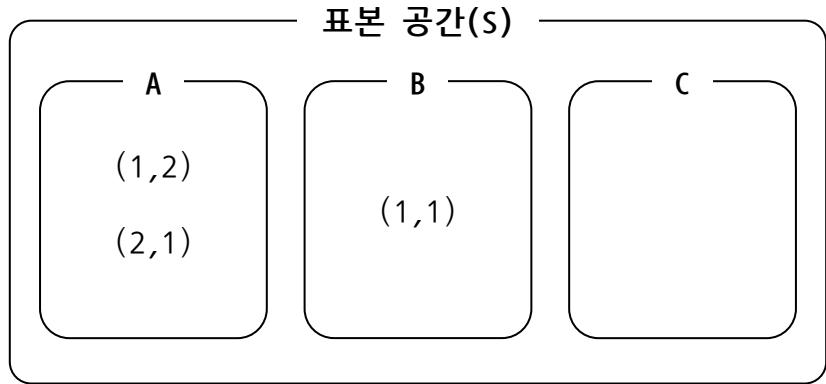
주사위 한 번 던지기의 표본 공간은 원소가 6개이며, 짹수일 사건은 원소가 3개입니다.

주사위를 천 번 던졌을 때 실제로 짹수가 나올 확률

수학적 확률을 계산할 수 있으면 시행 횟수를 반복 할수록 통계적 확률은 수학적 확률에 가까워집니다.

[참고] 수학적 확률의 시각적 예시

표본 공간(S)					
(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)
(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)



두 눈의 합이 3인 사건 A의 확률: $P(A) = 2/36$

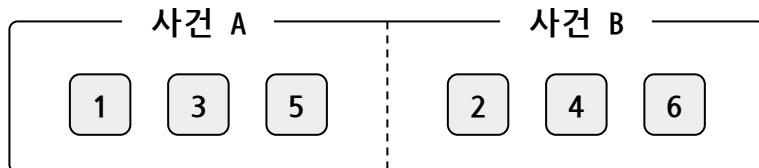
두 눈의 합이 2인 사건 B의 확률: $P(B) = 1/36$

두 눈의 합이 1인 사건 C의 확률: $P(C) = 0/36$

확률의 성질

- 표본 공간(전사건)의 확률은 1입니다.

[주사위 한 번 던지기의 표본 공간]



- 위와 같은 두 사건을 가정 개별 사건의 확률은 다음과 같습니다.

홀수인 사건 A의 확률: $P(A) = 1/2$

짝수인 사건 B의 확률: $P(B) = 1/2$

- 확률의 덧셈정리가 성립합니다.

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

- 배반사건이면 교집합이 없습니다.

○ 사건 A와 B는 배반사건입니다.

$$P(A \cup B) = P(A) + P(B) \because P(A \cap B) = \Phi$$

- 사건 A가 발생하지 않을 여사건의 확률은 1에서 A의 확률을 뺍니다.

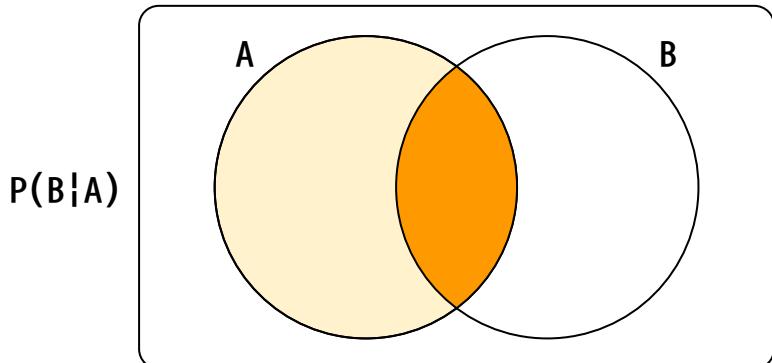
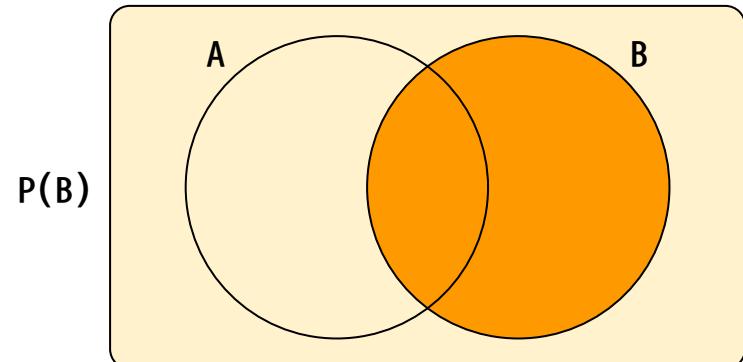
$$1 - P(A) = 1/2$$

조건부 확률

- 조건부 확률 Conditional Probability은 사건 A가 일어났을 때 사건 B가 일어날 확률입니다.
- 사건 B의 조건부 확률은 사건 A에 영향을 받습니다. 반대의 경우도 마찬가지입니다.
- 조건부 확률을 아래 수식으로 표현합니다.

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

$$P(A|B) = \frac{P(B \cap A)}{P(B)}$$



독립과 종속

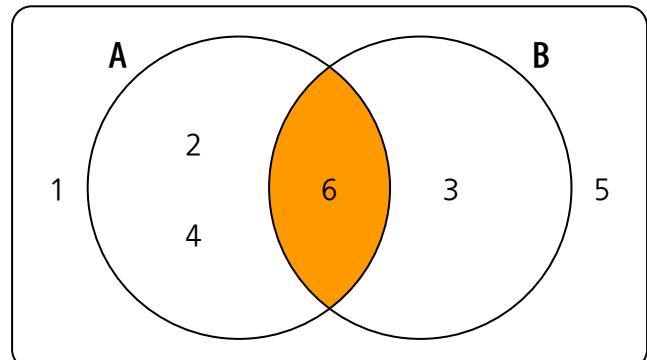
- 어떤 사건이 발생하는 것이 다른 사건의 발생에 영향을 주지 않는 것을 독립사건이라고 합니다. # [참고] 많은 머신러닝 알고리즘에서 개별 관측값의 발생을 독립사건으로 가정합니다.
- 사건 A와 B가 독립일 때 두 확률의 곱은 교집합일 확률과 같습니다.[결합확률]

$$P(A \cap B) = P(A) \times P(B)$$

- 사건 A와 B가 독립이 아닐 때(종속일 때) 두 확률의 곱은 아래 공식을 성립합니다.

$$P(A \cap B) = P(B|A) \times P(A)$$

$$P(A \cap B) = P(A|B) \times P(B) # [참고] 앞 페이지에서 소개한 조건부 확률 공식에서 분모를 양변에 곱한 결과입니다.$$



[주사위 던지기]

베이즈 정리

- 베이즈 정리 Bayes Theorem는 사전확률과 가능도로부터 사후확률을 추정합니다.
- 표본공간 S 에서 A_1, A_2, \dots, A_n 은 배반사건이며 $P(A) > 0$ 이고 $P(B) > 0$ 이면 다음을 성립합니다.

$$P(A_k|B) = \frac{P(A_k) \times P(B|A_k)}{P(B)} \quad \text{단, } P(B) = \sum P(A_i) \times P(B|A_i)$$

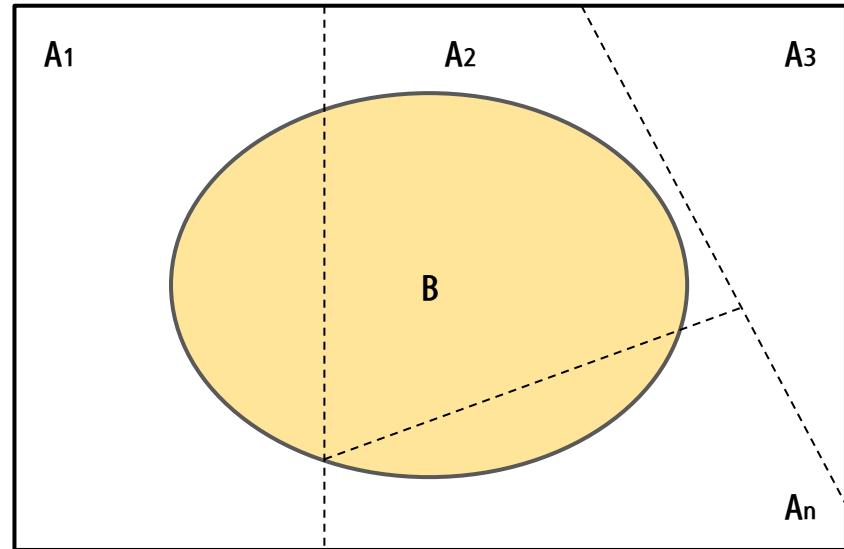
$P(A_k|B)$: 사건 B 가 발생했을 때, 사건 A_k 의 사후확률 Posterior Probability
 $P(A_k)$: 사건 A_k 의 사전확률 Prior Probability, Evidence
 $P(B|A_k)$: 사건 A_k 가 발생했을 때, 사건 B 의 가능도 Likelihood
 $P(B)$: $P(A_k|B)$ 를 확률로 만들기 위한 정규화 상수 Normalized Constant

베이즈 정리 관련 용어

- 사후확률: 우리가 알고자 하는 확률이며 추정의 대상이 되는 확률입니다.
- 사전확률: 우리가 경험적으로 이미 알고 있는 확률입니다.
 - 사전확률을 모를 때 임의로 설정할 수 있습니다.
- 가능성: 데이터에서 얻은 조건부 확률이며, 경험상 아는 것을 수치화한 것입니다.
 - 가능도를 확률과 비슷한 의미로 사용하지만 분포를 모르는 확률은 알 수 없습니다.
- 정규화 상수: 사전확률과 가능성의 곱(분자)을 전체집합 대신 부분집합 B일 때로 축소하는 상수입니다.
 - 정규화 상수는 계산 편의상 생략할 수 있습니다.

베이즈 정리의 시각적 예시

- 전체집합 S 가 n 개($A_1, A_2, A_3, \dots, A_n$)의 배반사건으로 구성되어 있고, 집합 B 가 부분집합 A_n 에 걸쳐 있다고 가정합니다.
- 부분집합 A_1 의 사전확률 $P(A_1)$ 은 집합 B 일 때 A_1 인 사후확률 $P(A_1|B)$ 와 서로 다릅니다.
- B 일 때 A_1 인 사후확률과 B 일 때 A_2 인 사후확률의 크기를 비교할 때 분모인 정규화 상수 $P(B)$ 를 생략해도 됩니다.



영화 리뷰로 베이즈 정리 이해하기

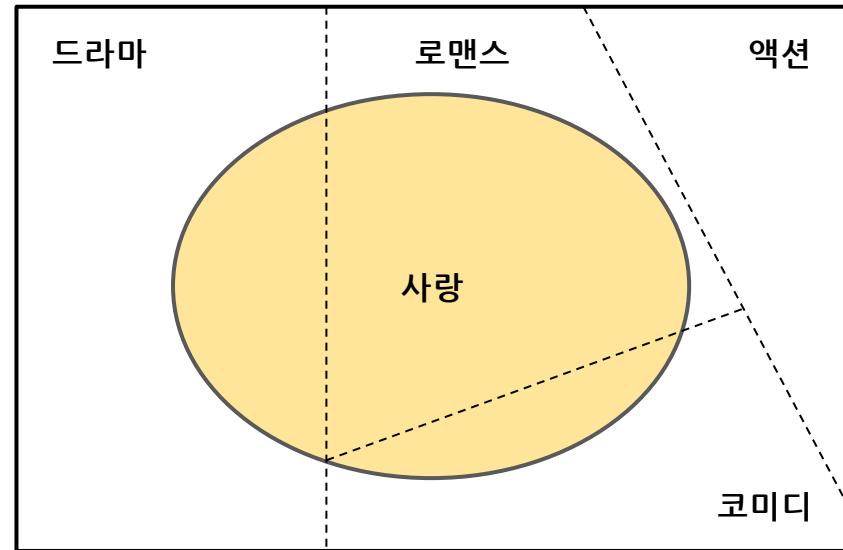
- 영화 리뷰가 작성된 전체 데이터에서 영화 장르가 4개라고 가정합니다.
 - 드라마(A_1), 로맨스(A_2), 액션(A_3), 코미디(A_4)
 - 전체 데이터에서 각 영화 장르의 상대도수는 사전확률입니다.
- 어떤 영화의 리뷰를 읽었을 때, 해당 리뷰에서 '사랑'이라는 단어가 사용되었다면 그 영화는 로맨스와 액션 중 어떤 장르에 가까울까요?
 - 이것이 우리가 궁금한 사후확률이며, 아래 공식으로 계산할 수 있습니다.
 - $P(\text{로맨스}|\text{사랑}) = P(\text{로맨스}) \times P(\text{사랑}|\text{로맨스}) \div P(\text{사랑})$
 - $P(\text{액션}|\text{사랑}) = P(\text{액션}) \times P(\text{사랑}|\text{액션}) \div P(\text{사랑})$

영화 리뷰로 베이즈 정리 이해하기(계속)

- 전체 데이터에 사용된 모든 단어를 정리하고 개별 영화 리뷰별로 각 단어의 출현 여부를 계산합니다.
 - 전체 데이터에서 각 단어별 상대도수를 계산하면 해당 단어의 정규화 상수가 됩니다.
 - 예를 들어 '사랑'이라는 단어의 상대도수인 $P(\text{사랑})$ 이 분모인 정규화 상수입니다.
- 가능도는 전체 데이터에서 조건부 확률을 계산합니다.
 - 장르별 영화 리뷰 데이터에서 '사랑'이라는 단어의 상대도수를 계산합니다.
 - 가능도는 아래 공식으로 계산합니다.
 - $P(\text{사랑}|\text{드라마}), P(\text{사랑}|\text{로맨스}), P(\text{사랑}|\text{액션}), P(\text{사랑}|\text{코미디})$

영화 리뷰로 베이즈 정리 이해하기(계속)

- $P(\text{드라마})$ 는 사전확률입니다.
 - 전체 리뷰 데이터에서 드라마 장르의 상대도수를 의미합니다.
- $P(\text{사랑}|\text{드라마})$ 는 가능성입니다.
 - 데이터로부터 계산할 수 있는 조건부 확률입니다.
- $P(\text{사랑})$ 은 정규화 상수입니다.
- 오른쪽 그림에서 '사랑'이라는 단어가 사용된 장르별 사후확률의 최댓값은?



확률분포의 이해

확률변수

- 확률변수는 일정한 확률로 발생하는 사건에 수치를 부여한 함수이며, 변수가 갖는 값의 형태에 따라 이산확률변수와 연속확률변수로 구분합니다. # [참고] 확률변수는 대문자, 같은 소문자로 표기합니다.
 - 확률변수 X 가 가질 수 있는 모든 값을 셀 수 있을 때 이산확률변수라고 합니다.
 - 확률변수 X 가 가질 수 있는 모든 값을 셀 수 없을 때 연속확률변수라고 합니다.
 - 정밀도 문제로 연속형을 이산형으로 처리했다면 이산형을 연속형으로 다루어야 합니다.
- 두 확률변수의 가장 큰 차이점은 값을 확률로 표현할 수 있는지 여부입니다.
 - 이산확률변수는 $P(X = x)$ 를 계산할 수 있습니다. # 이산형: 주사위 눈금 1이 나올 확률
연속형: 20대 남자 키가 170~175cm일 확률
 - 연속확률변수에서 $P(X = x)$ 는 0이므로, 두 지점의 면적을 계산하여 확률로 사용합니다.

이산확률분포

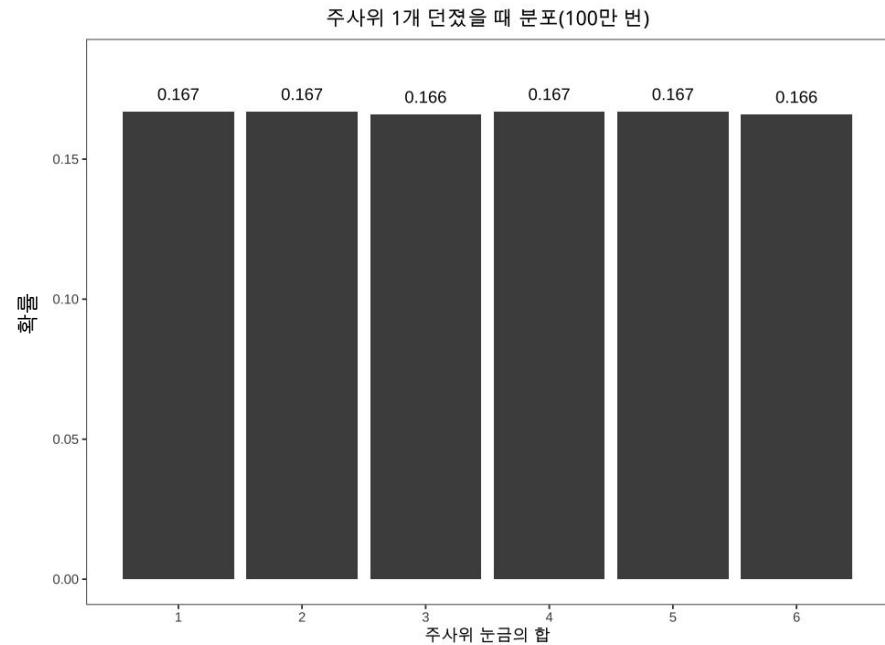
- 확률분포는 확률변수가 갖는 값마다 대응하는 확률을 표 또는 그래프로 표현한 것입니다. # [참고] 연속확률분포는 표로 나타낼 수 없습니다!

- 정상적인 주사위를 한 번 굴렸을 때 각 눈금이 나올 확률은 $1/6$ 입니다.

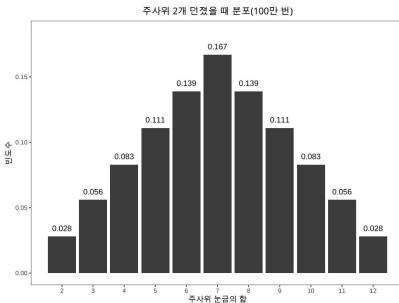
눈금	1	2	3	4	5	6
확률	$1/6$	$1/6$	$1/6$	$1/6$	$1/6$	$1/6$

- 이산확률변수를 나타내는 함수 $f(x)$ 를 확률질량함수라고 합니다.

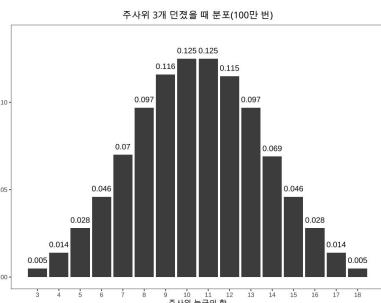
$$P(X) = f(x) \text{ # 함수값이 확률입니다.}$$



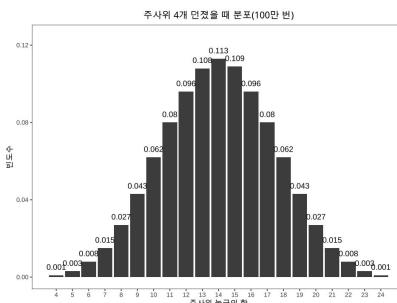
[참고] 주사위 던진 횟수를 늘리면?



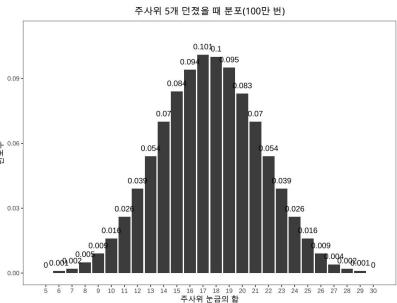
[주사위 2개 눈금의 합]



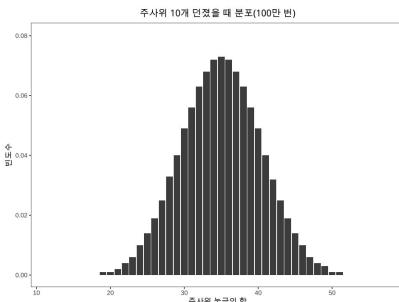
[주사위 3개 눈금의 합]



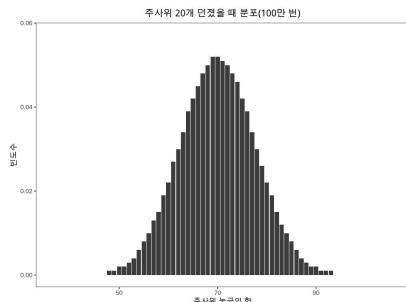
[주사위 4개 눈금의 합]



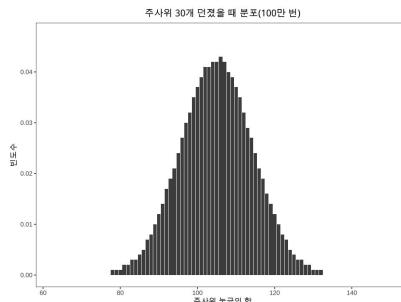
[주사위 5개 눈금의 합]



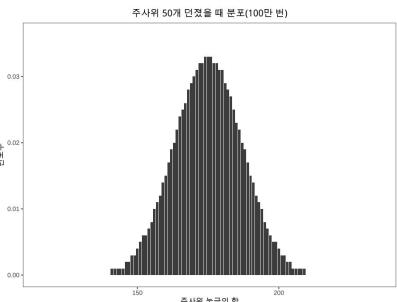
[주사위 10개 눈금의 합]



[주사위 20개 눈금의 합]



[주사위 30개 눈금의 합]



[주사위 50개 눈금의 합]

[참고] 표본평균에 관한 두 개의 정리

- 대수의 법칙 Law of Large Numbers 은 표본 크기가 증가하면 참값인 모평균에 가까워진다는 정리이고, 중심극한정리 Central Limit Theorem 는 모집단의 분포와 상관 없이 표본 크기가 증가하면 표본 통계량(ex. 표본평균)은 정규분포에 가까워진다는 정리입니다.
- 중심극한정리는 표본 통계량을 사용하는 가설검정(추론통계)에서 중요한 이론적인 근거를 제시하고 있으므로 매우 중요하며, 부트스트래핑으로 확인할 수 있습니다.
 - 모평균이 μ 이고, 모표준편차가 σ 인 분포를 따릅니다.
 - 모집단에서 n 개의 표본을 추출합니다. n 이 30개 이상이면 충분히 크다고 판단합니다.
 - 표본평균의 평균은 모평균, 표본평균의 분산은 모분산을 n 으로 나눈 값에 가까워집니다.

[참고] 부트스트래핑

- 부트스트래핑은 단일표본으로 표본 통계량의 분포를 추정하는 방법입니다.
 - 단일표본을 모집단으로 가정하고 작은 표본(크기 n)을 여러 번 무작위 복원추출합니다.
 - 모집단의 분포와 상관 없이 표본평균의 분포는 정규분포에 가까워집니다.
 - 작은 표본평균의 평균(기댓값)은 모평균, 표본평균의 분산은 모분산을 n 으로 나눈 값에 가까워집니다.

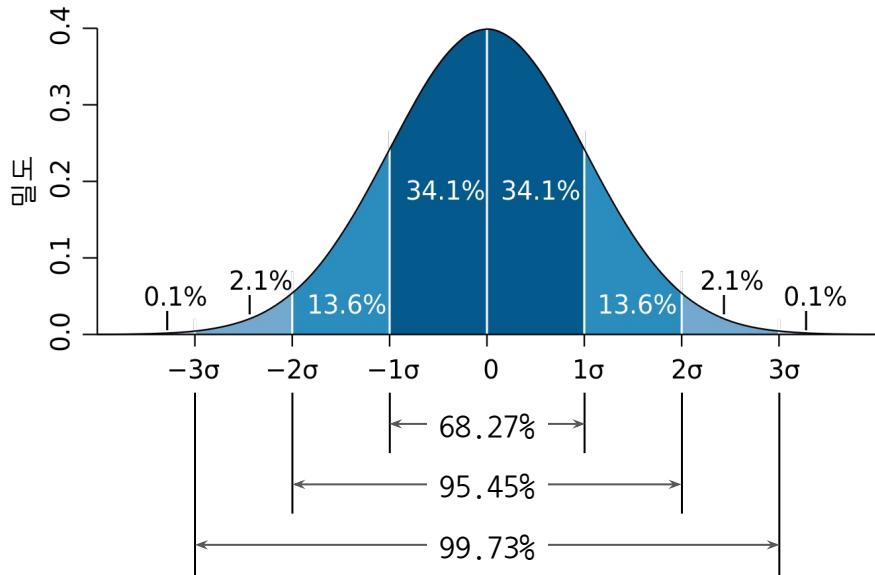


연속확률분포

- 연속확률변수를 나타내는 함수 $f(x)$ 를 확률밀도함수라고 합니다.
 - 누적분포함수를 미분한 것입니다.
- 연속확률변수는 셀 수 없는 값을 가지므로 확률 $P(X = x)$ 는 0이 됩니다.
- 확률밀도함수 $f(x)$ 를 $a \sim b$ 구간에서 적분한 값(곡선 아래 면적)을 계산하여 해당 구간에서의 확률로 사용합니다.

$$P(a \leq X \leq b) = \int_a^b f(x) d(x) \quad \# \text{ 함수의 면적이 확률입니다.}$$

[평균이 0, 표준편차가 σ 인 정규분포]



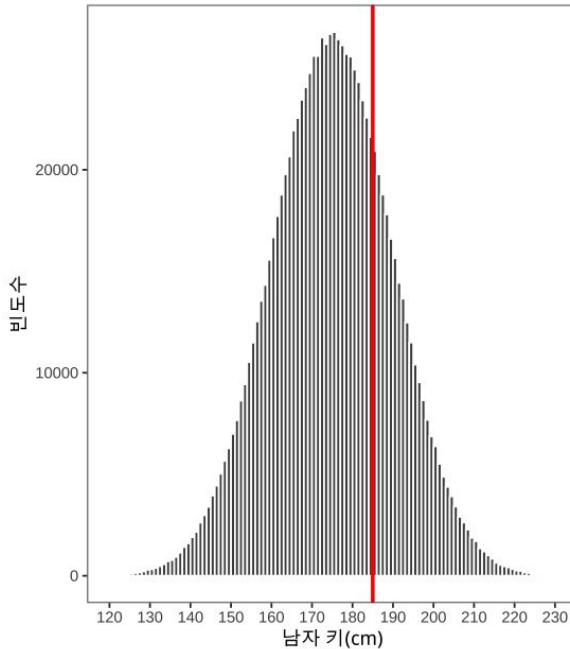
출처: https://ko.wikipedia.org/wiki/확률_분포

정규분포

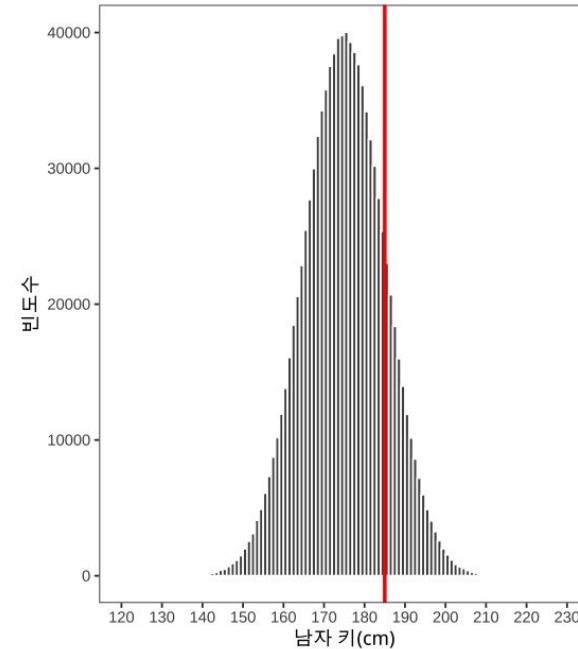
- 정규분포는 우리 주변에서 쉽게 발견할 수 있는 연속확률분포 중 하나입니다.
 - 분포의 형태는 좌우 대칭인 종모양의 곡선이고 분포의 중앙은 평균입니다.
 - 정규분포 확률밀도 곡선을 영어로는 Bell Curve라고 합니다.
 - 예를 들어 우리나라 20~59세 성인 남자를 모집단이라 가정하고 모집단에 속한 사람들의 몸무게를 측정하면 평균 몸무게 근처에 많은 사람들이 모여 있습니다.
- 모평균과 표준편차를 알면 정규분포 곡선을 그릴 수 있습니다.
 - 모평균은 중심이동, 표준편차는 분포의 폭을 결정합니다.
 - 표준편차가 작으면 평균 주변에 많은 원소가 있으므로 뾰족한 종모양을 그리지만 반대로 표준편차가 크면 넓게 퍼진 종모양으로 그려집니다.

정규분포의 예시

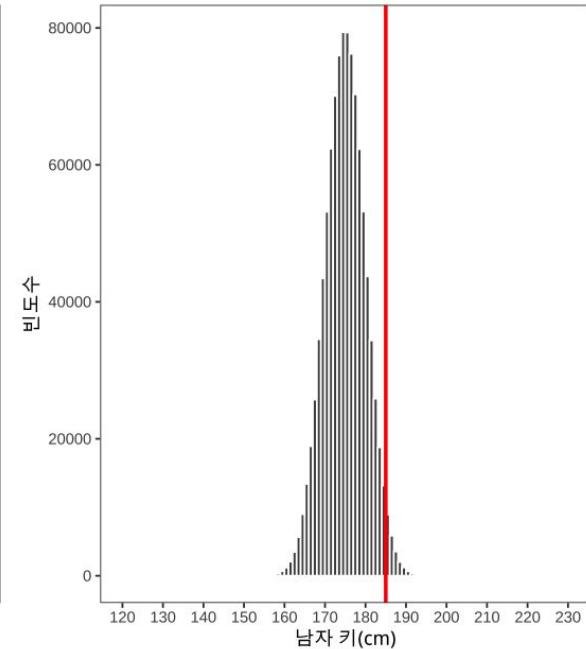
평균 175, 표준편차 15인 정규분포



평균 175, 표준편차 10인 정규분포



평균 175, 표준편차 5인 정규분포



정규분포를 따르는 세 집단의 평균은 같지만 표준편차가 다르기 때문에 정규분포의 폭이 다릅니다.

관련 라이브러리 호출

- 관련 라이브러리를 호출합니다.

```
>>> import os, joblib
```

```
>>> import numpy as np
```

```
>>> import pandas as pd
```

- 실수를 출력할 소수점 자리수를 설정합니다.

```
>>> %precision 3
```

```
>>> pd.options.display.precision = 3
```

관련 라이브러리 호출(계속)

- 통계 관련 라이브러리를 호출합니다.

```
>>> from scipy import stats
```

```
>>> from statsmodels import robust
```

- 시각화 및 통계 분석 관련 모듈을 호출합니다.

```
>>> from GraphicSetting import *
```

```
>>> import HelloDataScience as hds
```

정규분포를 따르는 무작위 값 생성

- 시드를 고정합니다.

```
>>> np.random.seed(seed = 1234) # 시드에 같은 값을 지정하면 항상 같은 결과를 얻습니다.  
[참고] 아래 함수에 random_state = 1234를 추가한 것과 같습니다.
```

- 평균이 175, 표준편차가 5인 정규분포를 따르는 무작위 표본을 생성합니다.

```
>>> heights = stats.norm.rvs(loc = 175, scale = 5, size = 5000)
```

- heights(표본)의 평균과 표준편차를 출력합니다.

```
>>> heights.mean() # heights(표본)의 평균은 모평균인 175에 가깝습니다.  
[참고] 표본 크기가 증가할수록 표본평균은 모평균에 더 가까워집니다.
```

```
>>> heights.std() # heights(표본)의 표준편차는 모표준편차인 5에 가깝습니다.  
[참고] 표본 크기가 증가할수록 표본표준편차도 모표준편차에 더 가까워집니다.
```

[참고] 정규분포 시각화

- 평균이 같고 표준편차가 다른 정규분포 확률밀도곡선을 시각화합니다.

```
>>> sp1 = stats.norm.rvs(loc = 175, scale = 15, size = 1000000)
```

```
>>> sp2 = stats.norm.rvs(loc = 175, scale = 10, size = 1000000)
```

```
>>> sp3 = stats.norm.rvs(loc = 175, scale = 5, size = 1000000)
```

```
>>> sns.kdeplot(x = sp1, label = 'Std: 15')
```

```
>>> sns.kdeplot(x = sp2, label = 'Std: 10')
```

```
>>> sns.kdeplot(x = sp3, label = 'Std: 5')
```

```
>>> plt.legend();
```

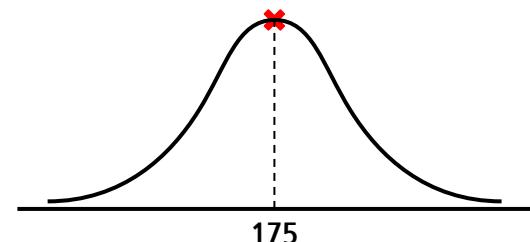
정규분포 확률밀도

- 다양한 정규분포에서 확률변수값 175의 확률밀도(높이)를 비교합니다.

```
>>> stats.norm.pdf(x = 175, loc = 175, scale = 15)
```

```
>>> stats.norm.pdf(x = 175, loc = 175, scale = 10)
```

```
>>> stats.norm.pdf(x = 175, loc = 175, scale = 5)
```



- 정규분포 확률밀도는 표본에 대한 모수의 가능도 likelihood입니다. # 모수의 가능도는 모수를 따르는 분포에서 표본에 대해 부여하는 확률입니다.

```
>>> stats.norm.pdf(x = [174, 175, 176], loc = 175, scale = 5).prod()
```

```
>>> np.log(stats.norm.pdf(x = [174, 175, 176], loc = 175, scale = 5)).sum()
```

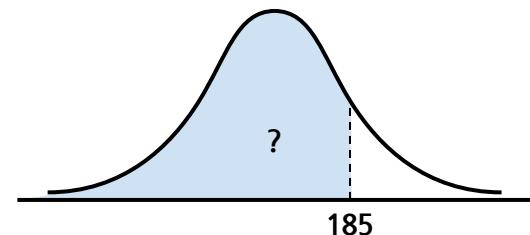
정규분포 누적확률

- 다양한 정규분포에서 확률변수값 185의 누적확률을 반환합니다.

```
>>> stats.norm.cdf(x = 185, loc = 175, scale = 15)
```

```
>>> stats.norm.cdf(x = 185, loc = 175, scale = 10)
```

```
>>> stats.norm.cdf(x = 185, loc = 175, scale = 5)
```



- 두 확률변수값 사이의 확률을 계산합니다.

x 매개변수에 원소가 2개인 리스트를 할당하면
해당 확률변수값의 누적확률을 반환합니다.

```
>>> cdfs = stats.norm.cdf(x = [165, 185], loc = 175, scale = 5); cdfs
```

```
>>> np.diff(a = cdfs)[0] # 평균에서 ±2 표준편차 구간의 확률을 계산합니다.
```

[참고] *cdfs*는 *numpy.ndarray*이므로 *diff()* 방식이 없습니다.

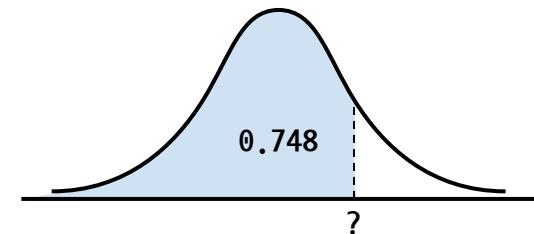
정규분포 확률변수값

- 다양한 정규분포에서 지정한 누적확률에 해당하는 확률변수값을 반환합니다.

```
>>> stats.norm.ppf(q = 0.748, loc = 175, scale = 15)
```

```
>>> stats.norm.ppf(q = 0.841, loc = 175, scale = 10)
```

```
>>> stats.norm.ppf(q = 0.977, loc = 175, scale = 5)
```



- (재미삼아) 20대 후반 남자 키가 평균이 175, 표준편차는 5인 정규분포를 따를 때 상위 5%인 남자 키는 몇 cm 이상일까요? 상위 1%인 남자 키도 알아봅시다.

```
>>> stats.norm.ppf(q = 0.95, loc = 175, scale = 5)
```

```
>>> stats.norm.ppf(q = 0.99, loc = 175, scale = 5)
```

왜도와 첨도

- 왜도로 확률밀도곡선의 중심이 한 쪽으로 치우친 정도를 알 수 있습니다.

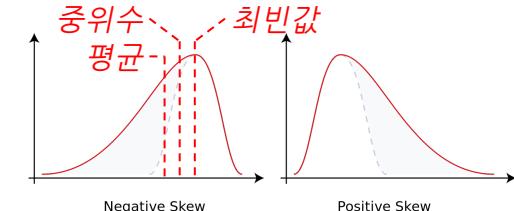
- 정규분포하는 데이터의 왜도는 0입니다.
 - 표본 왜도가 ± 2 일 때 정규분포한다고 판단합니다.

```
>>> stats.skew(a = heights) # 왜도가 0보다 작으면 오른쪽, 크면 왼쪽으로 치우친 분포를 의미합니다.
```

- 첨도로 확률밀도곡선의 봉우리가 뾰족한지 완만한지 여부를 알 수 있습니다.

- 정규분포하는 데이터의 첨도는 3입니다.
 - 표본 첨도가 1~5일 때 정규분포한다고 판단합니다.

```
>>> stats.kurtosis(a = heights) # 첨도가 3보다 크면 뾰족, 작으면 완만한 봉우리를 의미합니다.  
[참고] 이 함수는 정규분포하는 데이터의 첨도를 0으로 반환합니다.
```



정규성 검정

- 5천 건 이하인 데이터의 정규성 검정은 사피로-윌크 검정을 실행합니다.

```
>>> stats.shapiro(x = heights) # 정규성 검정의 귀무가설은 '데이터가 정규분포한다'입니다.  
# 유의확률p-value이 유의수준 0.05 보다 크면 정규분포한다고 판단합니다.
```

- 5천 건을 초과하는 가상의 키 데이터를 생성합니다.

```
>>> np.random.seed(seed = 1234)
```

```
>>> heights = stats.norm.rvs(loc = 175, scale = 5, size = 10000)
```

```
>>> stats.shapiro(x = heights) # [주의] shapiro() 함수에 5천 건을 초과하는 데이터를 넣고 실행하면  
# 반환되는 유의확률이 정확하지 않다는 경고를 출력합니다.
```

- 5천 건을 초과하는 데이터의 정규성 검정은 앤더슨-달링 검정을 실행합니다.

```
>>> stats.anderson(x = heights) # 앤더슨-달링 검정의 귀무가설도 '데이터가 정규분포한다'입니다.  
# [참고] 유의확률 대신 임계치critical values를 반환합니다.
```

[참고] 정규성 검정

- 사피로-윌크 Shapiro-Wilks 검정은 표본 데이터를 오름차순 정렬하고, 표준정규분포에서 추출한 순서 통계량과의 상관계수를 통해 데이터의 정규성을 확인합니다.
 - 귀무가설: 데이터가 정규분포를 따른다.
 - 대립가설: 데이터가 정규분포를 따르지 않는다.
 - 검정통계량: $W = \frac{(\sum_{i=1}^n a_i x_{(i)})^2}{\sum_{i=1}^n (x_i - \bar{x})^2}$ # $x_{(i)}$ 는 i 번째 순서 통계량이고, 계수 a_i 는 표준정규분포에서 추출한 순서 통계량의 기댓값입니다.
- 앤더슨-달링 Anderson-Darling 검정은 표본 데이터가 특정 확률분포에서 추출되었는지를 확인하며, 특히 정규성 검정에 주로 사용합니다.
 - 검정통계량: $A^2 = -n - S$, $S = \sum_{i=1}^n \frac{2i-1}{n} [\ln(F(Y_i)) + \ln(1 - F(Y_{n+1-i}))]$ # Y_i 는 오름차순 정렬한 데이터이고 $F(x)$ 는 누적분포함수입니다.

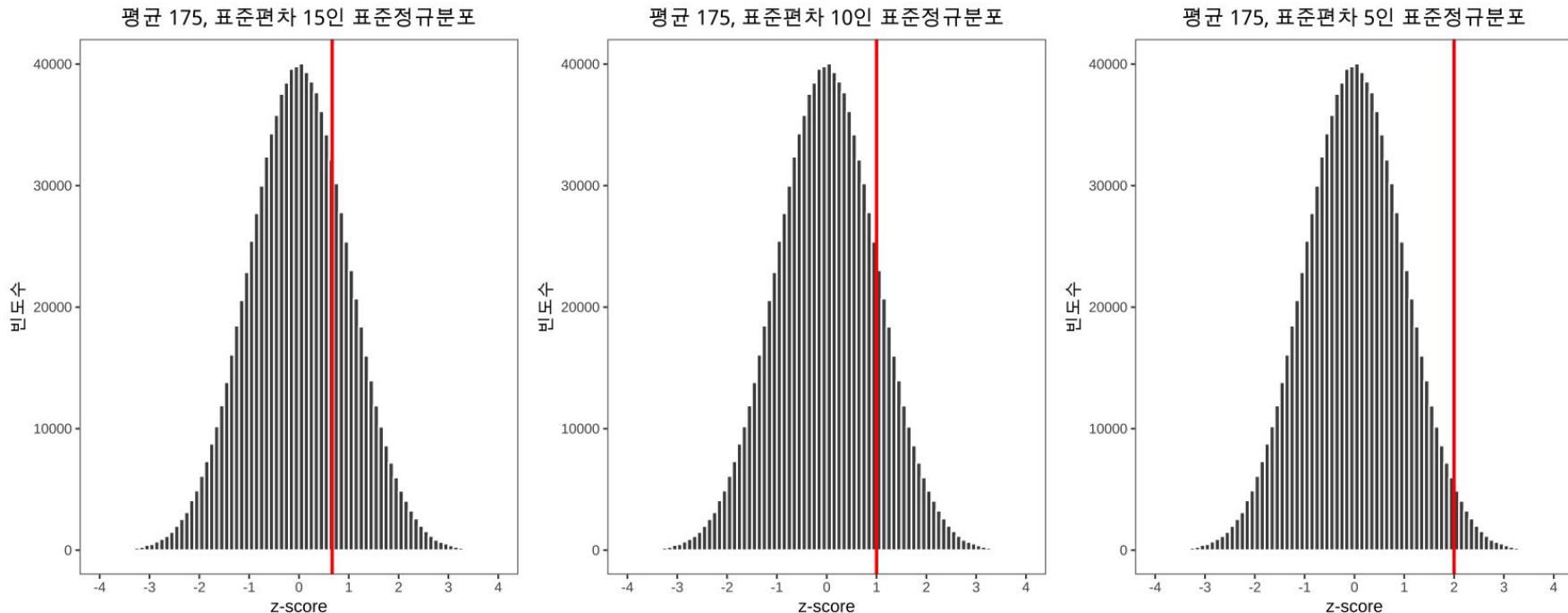
데이터 정규화

- 홍길동이 최근에 받은 수학과 영어 점수입니다. 어떤 과목을 더 잘하나요?
 - 수학: 90점(평균 75, 표준편차 15)
 - 영어: 55점(평균 40, 표준편차 10)
- 홍길동이 받은 두 과목의 점수 중에서 절대값은 수학이 크지만, 상대적인 크기를 비교할 수 없습니다.
 - 분포가 다른 두 값의 상대적인 크기를 비교하려면 척도를 통일해야 합니다.
 - 데이터 정규화^{normalization}는 두 값(스칼라, 벡터)의 척도를 같아지도록 변환하는 것입니다.
 - 다양한 머신러닝 알고리즘에서 입력 데이터셋의 정규화를 요구합니다.

표준정규분포

- 만약 두 확률변수가 따르는 정규분포의 평균과 표준편차가 다르면 확률변수값을 비교하는 것이 어렵습니다.
- 하지만 확률변수를 평균이 0, 표준편차가 1인 표준정규분포를 따르는 z-score로 변환하면 확률변수값을 비교할 수 있습니다.
 - z-score는 확률변수값에서 평균을 빼고 표준편차로 나눈 값입니다.
- z-score = $\frac{X - \mu}{\sigma}$ # 데이터를 표준화하면 개별 원소가 표준편차의 몇 배에 해당하는지 쉽게 알 수 있습니다.
- 확률변수값을 z-score로 변환하는 것을 데이터 표준화^{standardization}라고 합니다.
 - 데이터 표준화는 데이터를 정규화하는 대표적인 방법 중 하나입니다.

표준정규분포의 예시



정규분포를 따르는 세 집단을 데이터 표준화한 결과, 185cm인 사람의 z-score가 서로 다른 위치에 있음을 알 수 있습니다.

데이터 표준화

- 데이터를 표준화하는 함수를 생성합니다.

```
>>> def scale(x, loc, scale):  
    return (x - loc) / scale
```

- 다양한 정규분포를 따르는 관측값을 표준화하고 크기를 비교합니다.

```
>>> scale(x = 185, loc = 175, scale = 15)  
>>> scale(x = 185, loc = 175, scale = 10)  
>>> scale(x = 185, loc = 175, scale = 5)
```

[참고] 이상치 탐지 방법 비교

- [std] 표준화된 데이터의 절대값이 3을 초과할 때 이상치로 판단합니다.

```
>>> locs = np.where(np.abs(stats.zscore(heights)) > 3) # 표준화된 절대값이 3을 초과하는  
# 인덱스를 locs에 할당합니다.  
>>> heights[locs] # 평균과 표준편차 기준에서 이상치로 보이는  
# 원소를 출력합니다. # np.where() 함수는 조건이 True인  
# 인덱스를 정수로 반환합니다.
```

- [mad] 평균과 표준편차 대신 중위수와 중위수절대편차를 이용한 방법입니다.

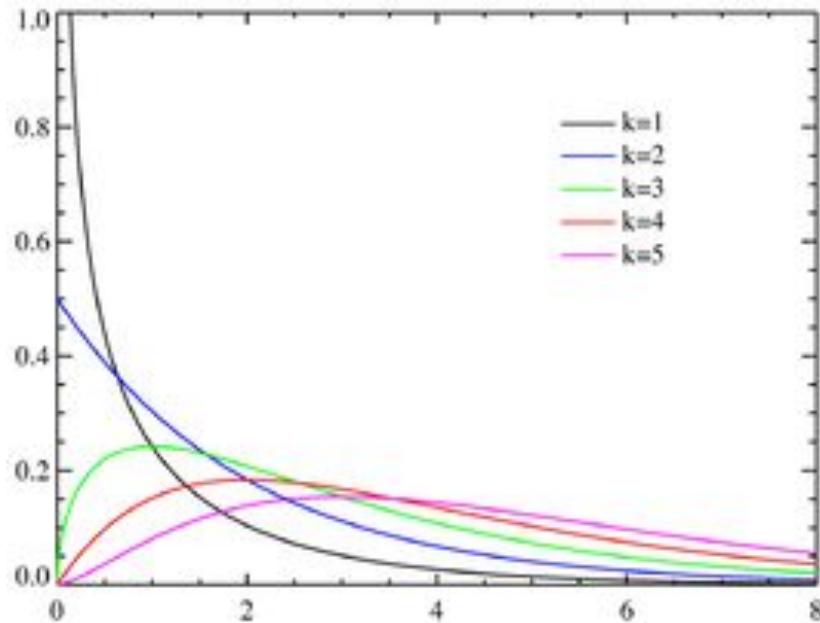
```
>>> med = np.median(a = heights) # 중위수를 med에 할당합니다.  
  
>>> mad = robust.mad(a = heights) # 중위수절대편차를 mad에 할당합니다.  
  
>>> locs = np.where(np.abs((heights - med) / mad) > 3) # 중위수와 중위수절대편차로  
# 정규화된 값이 3을 초과하는  
# 인덱스를 locs에 할당합니다.  
>>> heights[locs] # 중위수와 중위수절대편차 기준에서 이상치로 보이는 원소를 출력합니다.
```

χ^2 분포

- 카이제곱^{chi-square} 분포는 서로 독립인 z-score의 제곱을 k개 더한 분포이며 모분산을 추정할 때 사용합니다.

$$\chi_{(k)}^2 = z_1^2 + z_2^2 + \cdots + z_k^2 = \sum_{i=1}^k z_i^2$$

- 자유도(k)에 따라 분포 모양이 달라지며 자유도가 증가할수록 중심이 오른쪽으로 이동하면서 좌우대칭에 가까워집니다.
- 카이제곱 분포는 교차분석 및 로지스틱 회귀모형의 유의성 검정에서 사용합니다.

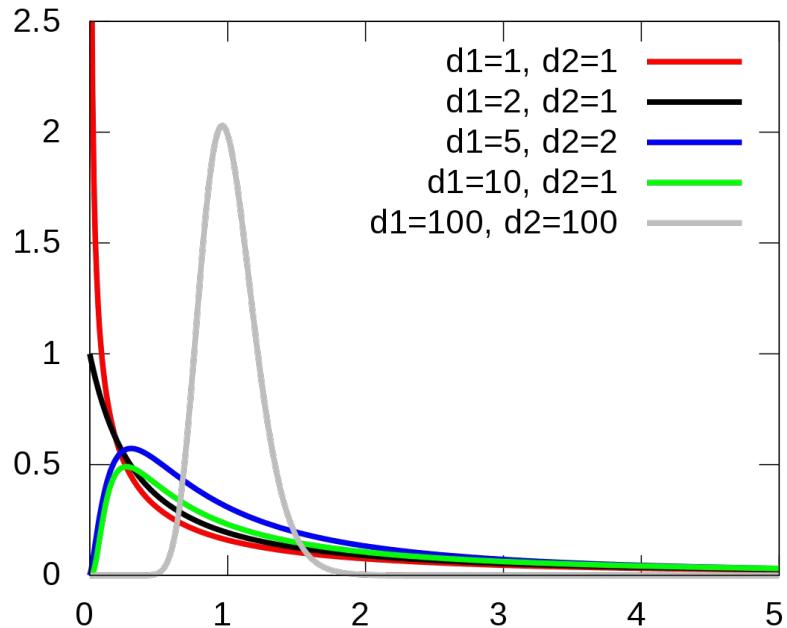


F 분포

- F 분포는 카이제곱 분포를 따르는 두 확률값을 각각의 자유도로 나눈 값의 비^{ratio}의 분포입니다.

$$F = \frac{\chi_1^2/k_1}{\chi_2^2/k_2}$$

- F 분포는 분산분석에서 집단 간 변동과 집단 내 변동의 비^{ratio}가 1인지(분산이 같은지) 확인할 때 사용합니다.
- 또한 선형 회귀모형의 유의성 검정에서 F 분포를 사용합니다.

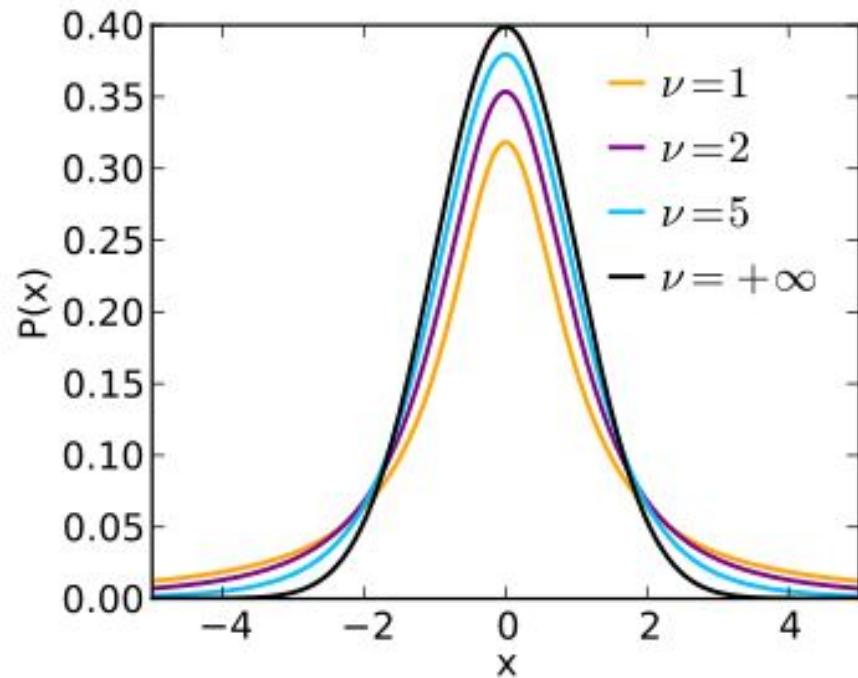


스튜던트 t 분포

- 스튜던트 t 분포는 모분산을 모르는 크기가 작은 표본평균의 분포입니다.

$$t_{\bar{X}} = \frac{\bar{X} - \mu}{\frac{s}{\sqrt{n-1}}}$$

- t 분포의 자유도는 $n-1$ 이고, 커질수록 표준정규분포에 가까워집니다.
- t 분포는 두 집단의 평균 차이가 서로 같은지 확인하는 t-검정 및 선형 회귀 계수의 유의성 검정에서 사용합니다.



[참고] ν 는 그리스 문자 $N(nu)$ 의 소문자입니다.

[참고] 자유도의 이해

- 자유도^{degree of freedom}는 말 그대로 자유로운 정도인데, 통계량을 계산할 때 표본에서 값을 자유롭게 가질 수 있는 표본의 개수를 의미합니다.
- 예를 들어 20대 후반 남자 키의 평균을 추정하기 위해 모집단에서 n 개의 표본을 추출한다고 가정하면 모든 표본은 자유롭게 값을 가질 수 있으므로 표본 평균의 자유도는 n 이 됩니다.
- 하지만 표본 평균이 특정 값(예를 들어 175cm)이어야 한다고 가정한다면 $n-1$ 개의 표본은 자유롭게 값을 가질 수 있지만 마지막 1개는 조건을 만족해야 하기 때문에 값이 고정될 수밖에 없습니다.
- 따라서 자유도는 표본 크기에서 제약 조건의 수를 차감하는 방식으로 계산합니다.

가설검정의 이해

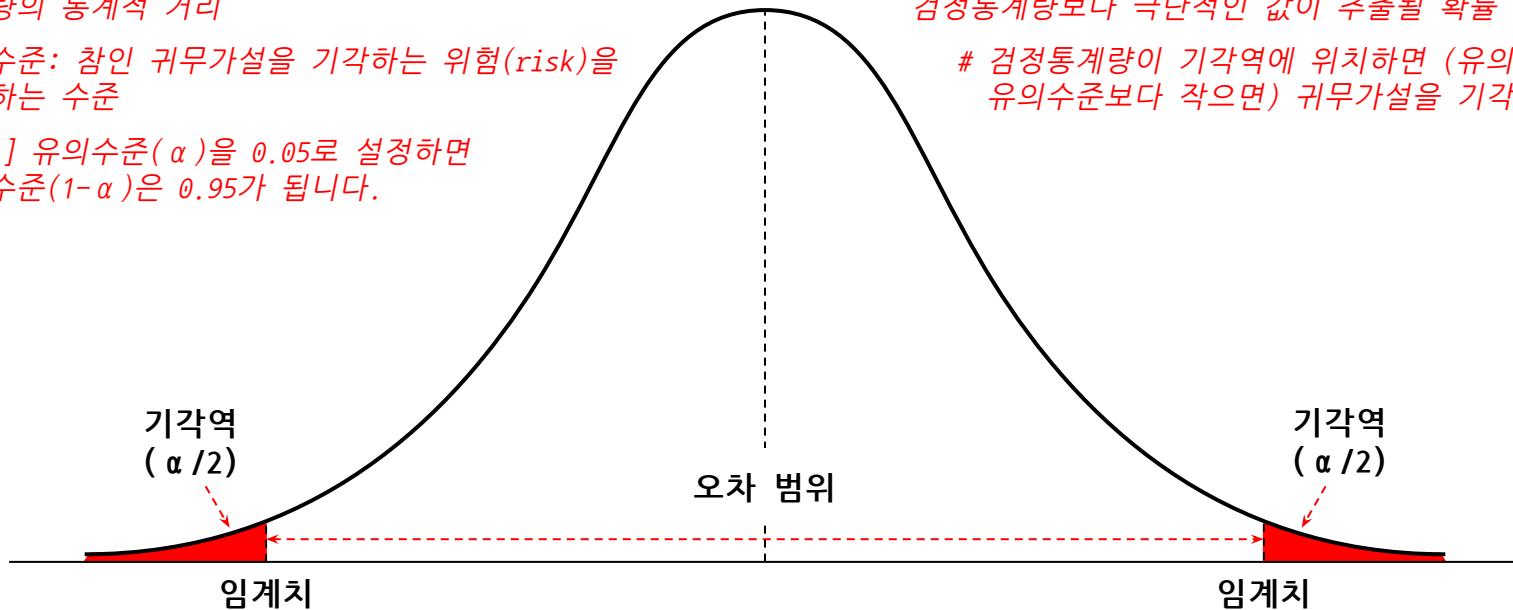
통계적 가설검정의 이해

- 통계적 가설검정은 어떤 주장(가설)에 대해 모수와 표본 통계량을 이용하여 해당 주장이 합당한 것인지를 판단하는 과정을 의미하며 아래 절차를 거칩니다.
 - 귀무가설^{null hypothesis}과 대립가설^{alternative hypothesis}을 설정합니다.
 - 유의수준(α)을 설정하면 양측검정과 단측검정 여부에 따라 기각역이 결정됩니다.
 - 모집단을 대표하는 표본 통계량과 모수의 통계적 거리인 검정통계량을 계산합니다.
 - 대표적인 검정통계량으로는 z-score(모분산 사용) 또는 t-value(표본분산 사용)가 있습니다.
 - 검정통계량으로 유의확률^{p-value}을 계산합니다.
 - 유의확률이 유의수준보다 작으면(검정통계량이 기각역에 속하면) 귀무가설을 기각합니다.

가설검정 관련 용어

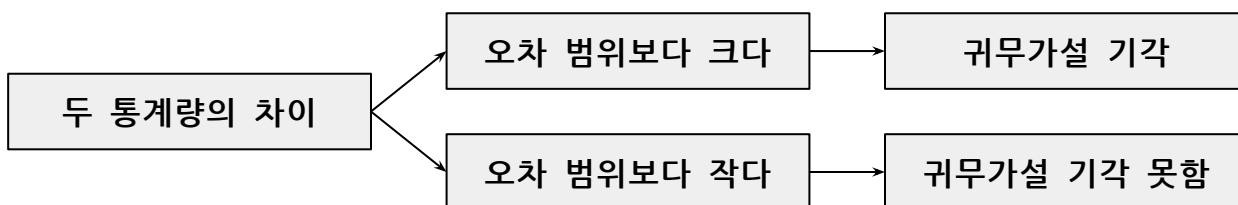
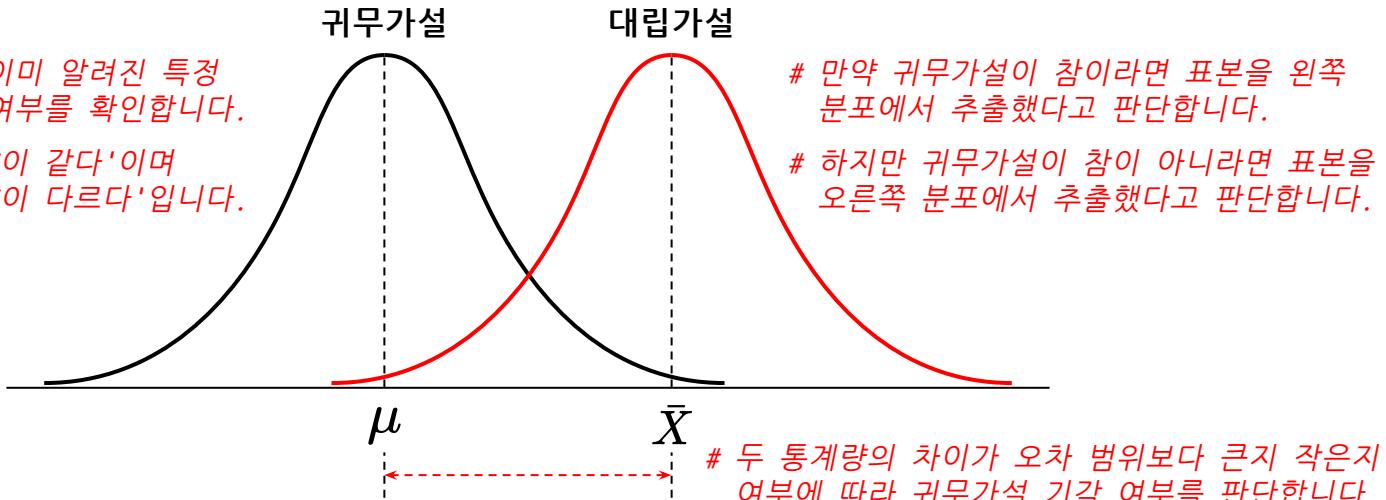
- # 귀무가설: (차이 없는) 기각하기 위해 설정한 가설
대립가설: 분석가가 채택하려는 가설
- # 검정통계량: 대립가설을 뒷받침해주는 모수와 표본 통계량의 통계적 거리
- # 유의수준: 참인 귀무가설을 기각하는 위험(risk)을 감내하는 수준
- # [참고] 유의수준(α)을 0.05로 설정하면 신뢰수준($1 - \alpha$)은 0.95가 됩니다.

- # 양측검정일 때 기각역 크기는 양 극단에 0.025씩, 단측검정일 때 기각역 크기는 해당 방향으로 0.05
- # 유의확률: 귀무가설이 참이라는 가정 하에 수집한 검정통계량보다 극단적인 값이 추출될 확률
- # 검정통계량이 기각역에 위치하면 (유의확률이 유의수준보다 작으면) 귀무가설을 기각합니다.

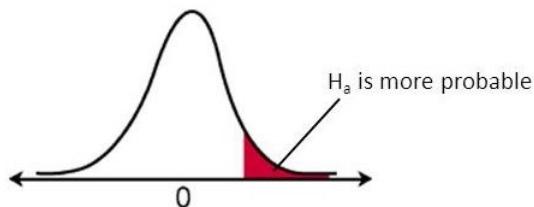


가설검정 판단 기준

- # 표본 통계량(평균)이 이미 알려진 특정 값(모평균)과 같은지 여부를 확인합니다.
- # 귀무가설은 '두 통계량이 같다'이며 대립가설은 '두 통계량이 다르다'입니다.

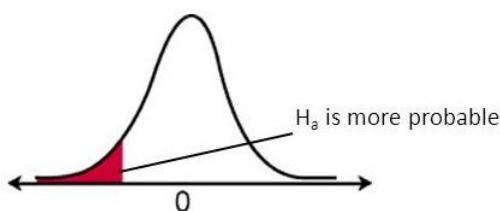


단측검정과 양측검정의 차이



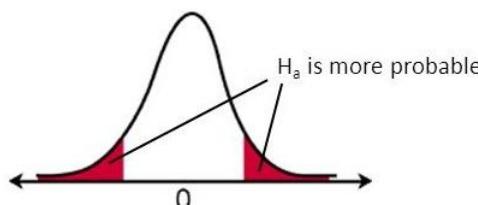
Right-tail test

H_a: $\mu > \text{value}$ # 한국 성인 남자의 평균 키는 173cm 보다 크다.



Left-tail test

H_a: $\mu < \text{value}$ # 한국 성인 남자의 평균 키는 173cm 보다 작다.



Two-tail test

H_a: $\mu \neq \text{value}$ # 한국 성인 남자의 평균 키는 173cm가 아니다.

출처: <https://www.fromthegenesis.com/why-hypothesis-testing/>

가설검정의 종류

구분		목표변수	
		연속형	범주형
입력변수	연속형	상관분석 <small>correlation analysis</small> (무상관 검정)	-
	범주형	독립표본 t-검정 <small>t-test</small> 분산분석 <small>ANOVA</small>	교차분석 <small>chisquared-test</small> (독립성 검정)

관련 라이브러리 호출

- 관련 라이브러리를 호출합니다.

```
>>> import os, joblib
```

```
>>> import numpy as np
```

```
>>> import pandas as pd
```

- 실수를 출력할 소수점 자리수를 설정합니다.

```
>>> %precision 3
```

```
>>> pd.options.display.precision = 3
```

관련 라이브러리 호출(계속)

- 통계 관련 라이브러리를 설치합니다.

```
>>> !pip install pingouin # [참고] 아나콘다 사용자는 터미널에서 아래 코드를 실행합니다.  
% conda install -c conda-forge pingouin
```

```
>>> !pip install scikit_posthocs
```

- 통계 관련 라이브러리를 호출합니다.

```
>>> from scipy import stats
```

```
>>> import pingouin as pg
```

```
>>> import scikit_posthocs as sp
```

작업 경로 확인 및 변경

- 현재 작업 경로를 확인합니다.

```
>>> os.getcwd()
```

- data 폴더로 작업 경로를 변경합니다.

```
>>> os.chdir(path = '../data')
```

- 현재 작업 경로에 있는 폴더명과 파일명을 출력합니다.

```
>>> os.listdir()
```

실습 데이터셋 준비

- z 파일을 읽고 데이터프레임 df를 생성합니다.

```
>>> df = joblib.load(filename = 'Used_Cars_Price.z')
```

- df의 정보를 확인합니다.

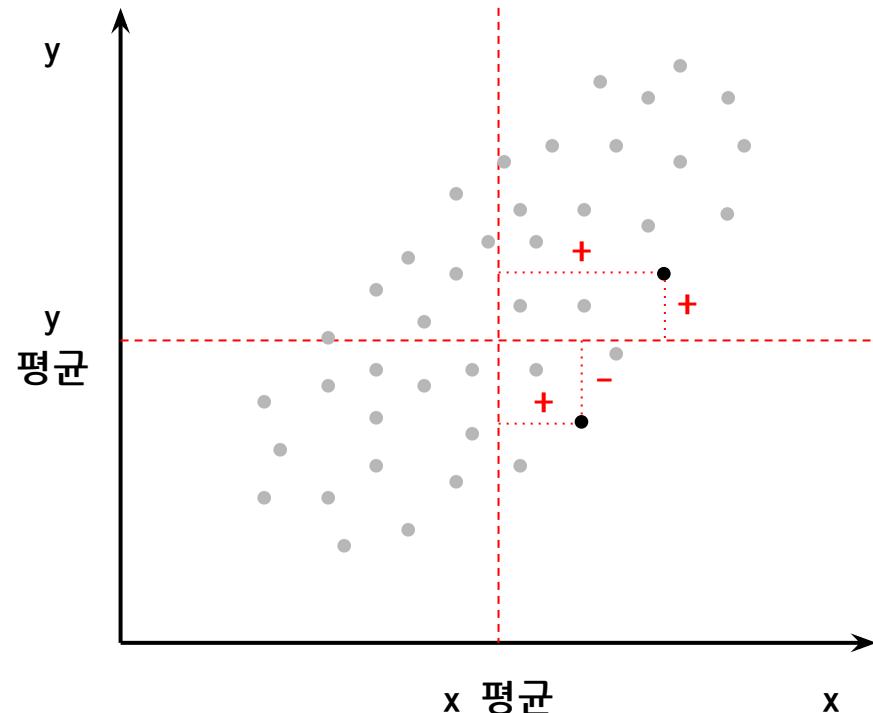
```
>>> df.info()
```

- df의 처음 5행을 출력합니다.

```
>>> df.head()
```

분산, 공분산 및 상관계수의 관계

구분	공식
분산	$\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2$
공분산	$\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})$
상관계수	$\frac{1}{n} \sum_{i=1}^n \frac{(X_i - \bar{X})(Y_i - \bar{Y})}{s_X s_Y}$



공분산

- 공분산 covariance 은 두 연속형 변수의 상관관계 방향을 나타내는 통계량입니다.
 - x가 증가할 때 대응하는 y가 증가하면 공분산은 양수입니다. # [참고] 상관관계는 직선의 관계를 의미합니다.
 - x가 증가할 때 대응하는 y는 감소하면 공분산은 음수입니다.
 - x와 y의 척도가 다르므로 공분산은 상관관계의 방향만 제시합니다.
- 두 연속형 변수의 공분산을 반환합니다.

```
>>> df['Age'].cov(df['Price'])
```

```
>>> df.cov() # df에서 수치형 변수만 선택하고 분산-공분산 행렬을 반환합니다.
```

상관계수

- 상관계수^{correlation coefficient}는 공분산을 각 변수의 표준편차로 나눈 것입니다.
 - 상관계수는 두 연속형 변수에 대한 상관관계의 방향과 강도를 함께 나타냅니다.
 - 상관계수는 $-1 \sim 1$ 의 값을 갖는데 ± 1 에 가까울수록 강한 상관관계가 있다고 판단합니다.

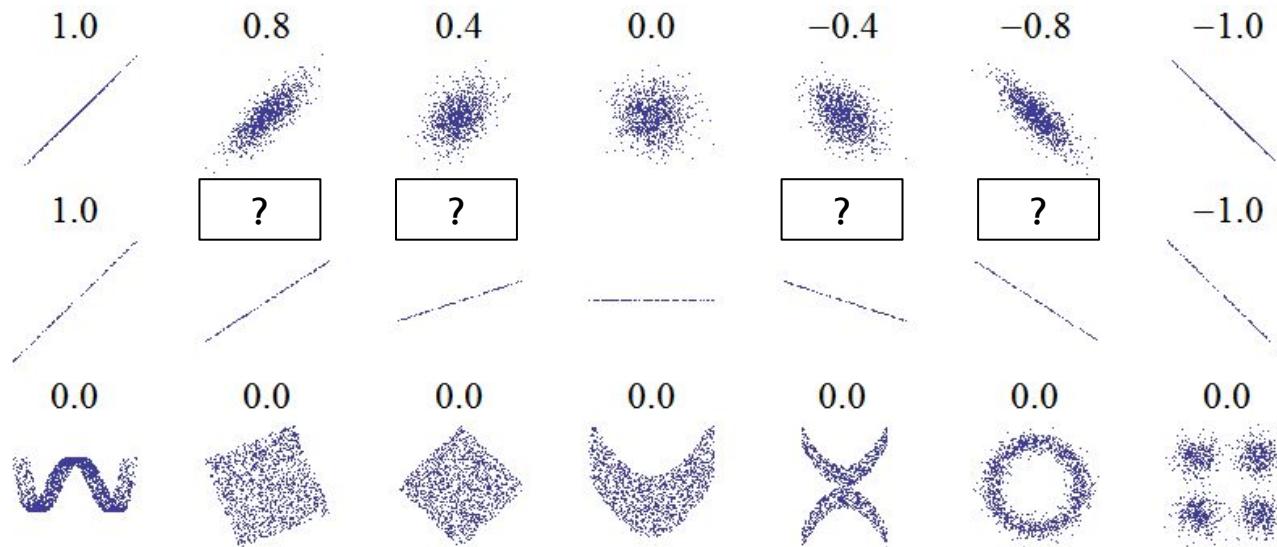
상관계수	$ r < 0.10$	$0.10 \leq r < 0.29$	$0.29 \leq r < 0.46$	$0.46 \leq r $
해석	상관관계 거의 없음	약한(낮은) 상관관계	보통 상관관계	강한(높은) 상관관계

- 두 연속형 변수의 피어슨 상관계수를 반환합니다.

```
>>> df['Age'].corr(df['Price']) # [참고] method 매개변수에 상관계수 종류를 문자열로 지정합니다.  
# 기본값은 'pearson'이고, 'spearman'과 'kendall'도 가능합니다.
```

```
>>> df.corr() # df에서 수치형 변수만 선택하고 피어슨 상관계수 행렬을 반환합니다.
```

[참고] 상관계수 관련 그래프



출처: https://en.wikipedia.org/wiki/Distance_correlation

피어슨 상관분석

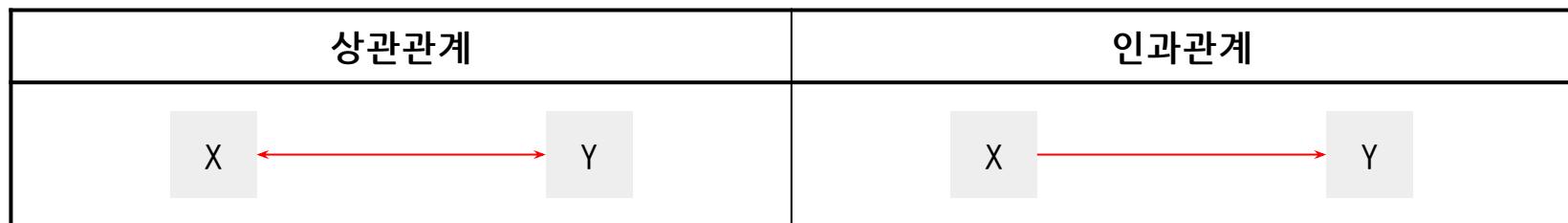
- 피어슨 상관분석은 두 연속형 변수에 상관관계가 있는지 측정합니다.
 - 피어슨 상관계수는 정규분포를 따르는 두 연속형 변수의 공분산을 각 표준편차로 나누어 표준화한 값입니다.(모수적 방법)
 - 예를 들어 국어시험 점수와 영어시험 점수와의 관계를 의미합니다.
 - 피어슨 상관계수는 $-1 \leq \rho \leq 1$ 을 만족합니다.
 - 피어슨 상관계수가 -1 또는 1에 가까워질수록 강한 상관관계를 갖습니다.
 - 피어슨 상관분석 관련 함수는 상관계수와 유의확률을 반환합니다.
 - 유의확률이 유의수준 0.05 보다 작으면 두 변수에 상관관계가 있다고 판단합니다.

스피어만 상관분석

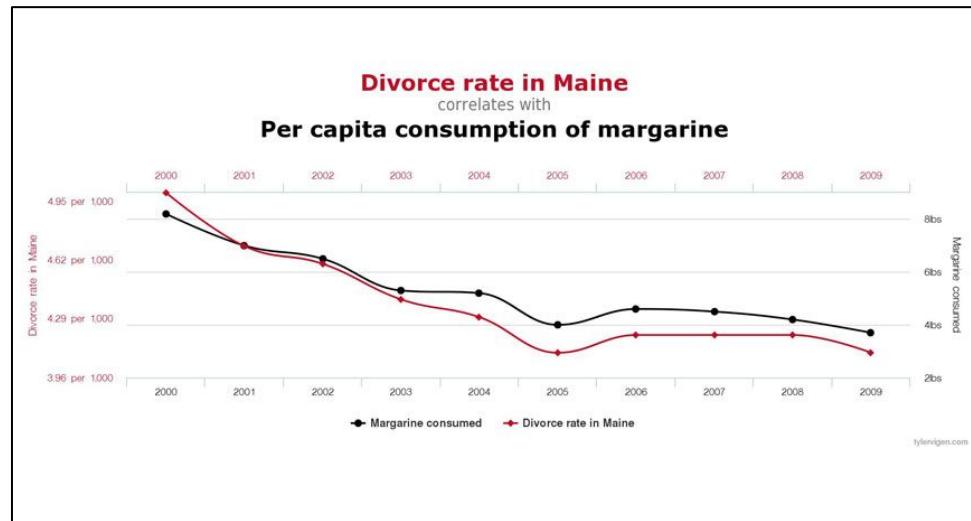
- 스피어만 상관분석은 이산형/순서형 변수의 의존성을 측정합니다.
 - 스피어만 상관계수는 정규분포하지 않는 두 변수에서 한 변수가 증가할 때 다른 변수도 증가하는지 확인합니다.(순위를 고려한 비모수적인 방법)
 - 예를 들어 국어시험 등수와 영어시험 등수와의 관계를 의미합니다.
 - 스피어만 상관계수의 공식은 피어슨 상관계수의 공식과 비슷합니다.
 - 데이터의 순서로 피어슨 상관계수를 계산합니다.
 - 스피어만 상관계수가 1이면 한 변수의 값이 증가할 때 다른 변수의 값도 증가한다는 것을 의미합니다.
 - 유의확률이 유의수준 0.05 보다 작으면 두 변수의 순서에 상관관계가 있다고 판단합니다.

상관관계 vs 인과관계

- 두 변수에 상관관계가 있다는 것이 인과관계가 있다는 것을 의미하지 않습니다.
 - 상관관계는 두 변수가 서로 영향을 주고 받는 관계입니다. 우연의 일치도 있습니다.
 - 같은 시점에 측정한 두 변수의 크기가 같은 방향으로 움직일 때 상관관계가 높게 나타납니다.
 - 인과관계는 원인 변수가 결과 변수에 영향을 주는 관계입니다.
 - 두 변수에 시점의 차이가 있습니다. 예를 들어 식사량에 따라 체중 증가량이 달라집니다.



[참고] 인과관계 판단 오류



출처: <http://www.ssacstat.com/>

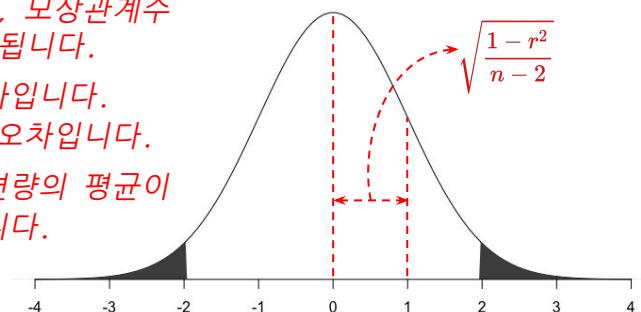
출처: <https://www.sciencenewsforstudents.org>

피어슨 상관분석의 가설 및 검정통계량

- 피어슨 상관분석의 가설 및 검정통계량은 다음과 같습니다.

- 귀무가설(H_0): $\rho = 0$ (모상관계수는 0이다)
- 대립가설(H_1): $\rho \neq 0$ (모상관계수는 0이 아니다)
- 유의수준: 5% \rightarrow 양측검정이므로 각 2.5%

- 검정통계량: $t = \frac{r - \rho}{\sqrt{\frac{1-r^2}{n-2}}}$
 - # 분자에서 r 은 표본상관계수이며, 모상관계수 $\rho(rho)$ 는 귀무가설에 의해 0이 됩니다.
 - # 분모는 표본상관계수의 표준편차입니다.
표본 통계량의 표준편자는 표준오차입니다.
 - # 표본상관계수를 계산할 때 두 변량의 평균이 사용되므로 자유도는 $n-2$ 가 됩니다.
 - # 검정통계량이 기각역(검정색)에 있으면 귀무가설을 기각합니다.



피어슨 상관분석

- 두 연속형 변수의 피어슨 상관분석을 실행하고 유의확률을 확인합니다.

```
>>> pg.corr(x = df['Age'], y = df['Price']) # method 매개변수의 기본 인수는 'pearson'입니다.
```

```
>>> pg.corr(x = df['KM'], y = df['Price'])
```

```
>>> pg.corr(x = df['HP'], y = df['Price'])
```

```
>>> pg.corr(x = df['CC'], y = df['Price'])
```

```
>>> pg.corr(x = df['Doors'], y = df['Price'])
```

```
>>> pg.corr(x = df['Weight'], y = df['Price'])
```

[참고] 피어슨 상관분석 유의확률 출력 함수 생성

- 변수 x에 df의 연속형 변수를 할당합니다.

```
>>> x = df[ 'Age' ]
```

- 변수 x와 Price의 피어슨 상관분석 실행 결과에서 유의확률만 출력합니다.

```
>>> pg.corr(x = x, y = df[ 'Price' ])['p-val']
```

- 연속형 입력변수와의 상관분석 유의확률을 출력하는 람다 표현식을 생성합니다.

```
>>> corr = lambda x: pg.corr(x = x, y = df[ 'Price' ])['p-val']
```

- 람다 표현식 함수로 피어슨 상관분석 유의확률을 출력합니다.

```
>>> corr(x = df[ 'Age' ])
```

[참고] apply() 함수를 활용한 상관분석 실행

- df의 열별 자료형을 확인합니다.

```
>>> df.dtypes # [참고] Windows는 정수를 numpy.int32로 생성합니다.
```

- 열별 자료형이 정수형 또는 실수형이면 True, 아니면 False인 벡터를 생성합니다.

```
>>> locs = df.dtypes.astype(str).isin(values = ['float64', 'int64'])
```

```
>>> locs
```

- df의 정수형 또는 실수형 변수만 선택하여 상관분석을 실행하고 유의확률이 0.05 보다 작은지 여부를 데이터프레임으로 반환합니다.

```
>>> df.loc[:, locs].apply(func = corr).lt(0.05)
```

t-검정의 종류

- t-검정은 다음과 같이 세 가지 종류가 있습니다.

구분	단일표본 t-검정	대응표본 t-검정	독립표본 t-검정
내용	표본 통계량과 모평균 비교	짝을 이루는 두 집단의 평균 비교	서로 독립인 두 집단의 평균 비교
예시	A 중학교 남학생의 평균 키는 170cm 이상인가?	다이어트 약 복용 전/후 평균 체중이 다른가?	B 중학교 남/여학생 간 평균 체중이 다른가?
기본 가정	정규성	정규성	독립성, 정규성, 등분산성
정규성 가정 만족 안함	윌콕슨 부호순위 검정	윌콕슨 부호순위 검정	윌콕슨 순위합 검정

독립표본 t-검정

- 독립표본 t-검정은 두 집단의 평균이 같은지(두 집단의 평균에 차이가 없는지) 확인할 때 실행합니다.
 - 세 개 이상의 집단에 대한 평균 비교는 분산분석 또는 다중비교를 실행합니다.
- 독립표본 t-검정은 두 집단 내에서의 변화량(잔차)으로 두 집단의 평균 차이가 통계적으로 유의한지 확인합니다.
 - 두 집단의 평균 차이가 작으면 두 집단의 평균이 같다고(차이가 없다고) 판단합니다.
 - 두 집단의 평균 차이의 표준오차가 작으면 두 집단의 평균은 같지 않다고 판단합니다.
 - t-검정은 두 집단의 분산이 같다는 가정 하에 합동표본분산을 사용합니다.

[참고] t-검정 자료구조

- 범주형 입력변수와 연속형 목표변수를 표로 나타내면 아래와 같습니다.

MetColor	Price(y_{ij})	범주별 평균 (\bar{y}_i)	잔차 ($y_{ij} - \bar{y}_i$)
0	8500	9460.8	-960.8
0	10450		989.2
:	:		:
1	9450	9814.1	-364.1
1	10050		235.9
:	:		:

[참고] 목표변수에서 범주별 평균을 차감한 잔차가 정규분포해야 합니다.

독립표본 t-검정의 가설 및 검정통계량

- 독립표본 t-검정의 가설 및 검정통계량은 다음과 같습니다.
 - 귀무가설(H_0): $\mu_1 = \mu_2$ (두 집단의 모평균이 같다.)
 - 대립가설(H_1): $\mu_1 \neq \mu_2$ (두 집단의 모평균이 같지 않다.)
 - 유의수준: 5%(양측검정)
 - 검정통계량: $t = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{s_p^2 \left(\frac{1}{n_1} + \frac{1}{n_2} \right)}}$
- # 두 표본평균의 차이가 작을수록 t 통계량은 0에 수렴합니다.
- # 귀무가설에 의해 두 모평균의 차이는 0이 됩니다.
- # 분모는 두 표본으로부터 계산한 편차 제곱합을 자유도 합계로 나눈 합동표본분산 *pooled sample variance*을 사용한 표준오차입니다.
- $s_p^2 = \frac{(n_1 - 1) s_1^2 + (n_2 - 1) s_2^2}{(n_1 - 1) + (n_2 - 1)}$ # 합동표본분산은 모분산의 불편추정량이며 두 표본분산이 동일하다는 가정 하에서 계산된 가중평균입니다.

독립표본 t-검정의 가정

- 입력변수가 두 개의 범주를 갖는 범주형이고, 목표변수는 연속형일 때 입력변수의 범주별 목표변수 평균이 같은지 확인하기 위해 독립표본 t-검정을 실행합니다.
 - 입력변수 범주별 목표변수 평균이 다르면 선형 회귀모형의 입력변수로 추가합니다.
- 독립표본 t-검정은 독립성, 정규성 및 등분산성 가정을 만족해야 합니다.
 - 독립성 가정은 데이터를 수집하는 과정에서 두 집단 간 차이에 영향을 주는 외부 요인이 없어야 한다는 것을 의미합니다. 즉, 수집한 데이터는 독립표본이어야 합니다.
 - [참고] 실험 데이터는 무작위화^{randomization}를 통해 외부 요인을 상쇄시킬 수 있습니다.
 - 입력변수 범주에 관계 없이 잔차는 정규분포하고 모분산이 같아야 합니다.

정규성 검정

- t-검정은 잔차의 정규성 가정을 만족해야 합니다.
 - 목표변수 개별 값에서 범주별 평균을 차감한 잔차가 정규분포하는지 확인합니다.
 - 귀무가설은 '데이터가 정규분포한다'입니다.
 - 유의확률이 유의수준 0.05 보다 크면 정규성 가정을 만족합니다.
- MetColor 범주별 Price의 정규성 검정을 실행합니다.

```
>>> pg.normality(data = df, dv = 'Price', # 데이터프레임과 연속형 목표변수를 지정합니다.  
           [참고] 'dv'는 'dependent variable'을 의미합니다.
```

```
           group = 'MetColor', # 범주형 입력변수를 지정합니다.
```

```
           method = 'shapiro') # 정규성 검정 방식을 지정합니다.(기본값: 'shapiro')  
           [참고] 대용량 데이터는 'normaltest'를 지정합니다.
```

등분산성 검정

- 정규성 가정을 만족하는 두 집단의 등분산성 검정을 실행합니다.
 - 등분산성 검정의 귀무가설은 '두 집단의 모분산을 나눈 비^{ratio}가 1이다'입니다.
 - 유의확률이 유의수준 0.05 보다 크면 두 그룹의 모분산이 같다고 판단합니다.
- (정규성 가정 만족) MetColor 범주별 Price의 등분산성 검정을 실행합니다.

```
>>> pg.homoscedasticity(data = df, dv = 'Price', group = 'MetColor',  
                         method = 'levene') # 등분산 검정 방식을 지정합니다.(기본값: 'levene')  
                           [참고] 정규분포하면 'bartlett'을 지정합니다.
```

- 등분산이면 t-검정, 이분산이면 Welch의 t-검정을 실행합니다.
 - 실행 함수는 같지만 correction 매개변수에 전달하는 인수를 다르게 설정해야 합니다.

범주형 변수로 데이터프레임 분할

- MetColor 원소의 중복을 제거한 values를 생성합니다.

```
>>> values = df['MetColor'].unique(); values
```

- MetColor의 원소별 Price로 시리즈를 생성합니다.

```
>>> sp1, sp2 = [df['Price'][df['MetColor'].eq(v)] for v in values]
```

- 두 시리즈의 평균을 각각 확인합니다.

```
>>> print(sp1.mean()) # sp1은 MetColor가 '0'인 Price입니다.
```

```
>>> print(sp2.mean()) # sp2는 MetColor가 '1'인 Price입니다.
```

t-검정

- (정규성 가정 만족) 두 집단의 평균이 같은지 확인할 때 t-검정을 실행합니다.
 - 귀무가설은 '두 집단의 모평균이 같다'입니다.
 - 유의확률이 유의수준 0.05 보다 작으면 두 집단의 모평균이 다르다고 판단합니다.
 - 등분산성 가정 만족 여부에 따라 correction 매개변수에 지정하는 값이 달라집니다.
- 등분산 가정된 t-검정을 실행합니다.

```
>>> pg.ttest(x = sp1, y = sp2, correction = False)
```

- 이분산 가정된 t-검정을 실행합니다.

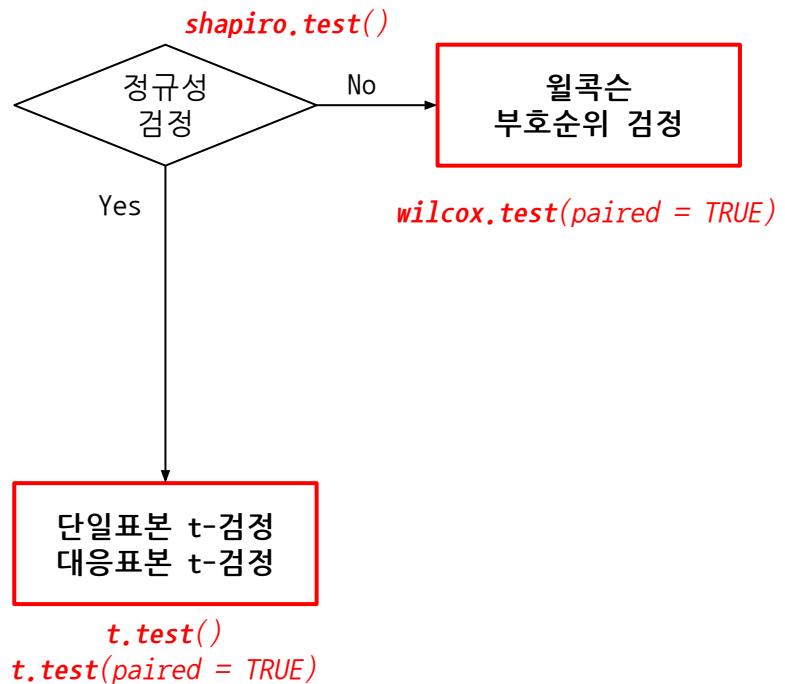
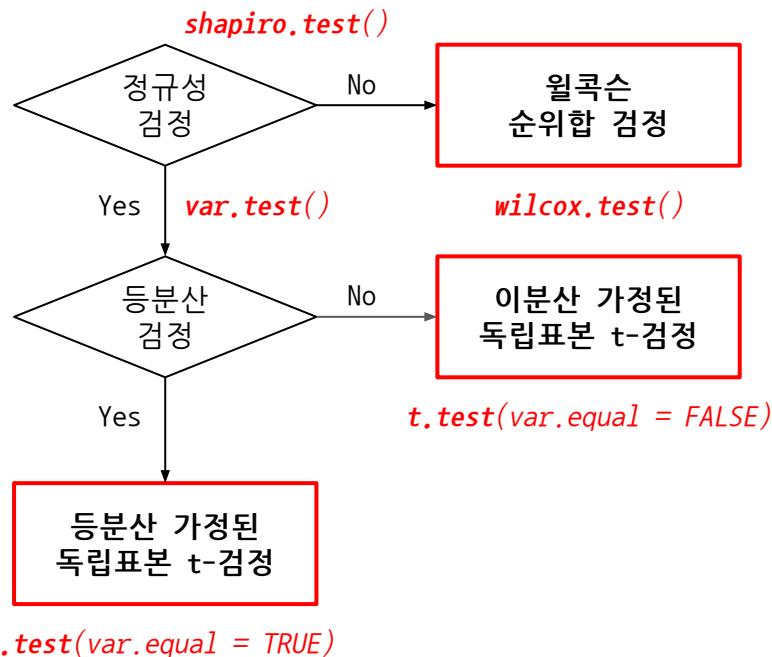
```
>>> pg.ttest(x = sp1, y = sp2, correction = True)
```

윌콕슨 순위합 검정 또는 맨-휘트니 U 검정

- 윌콕슨 순위합 검정은 두 집단 분포 위치가 같은지 여부를 판단하는 비모수적인 검정 방법입니다. 맨-휘트니 U 검정과 같습니다.
 - 귀무가설은 '두 집단 간 분포의 위치가 같다'입니다. # [참고] 순위 데이터의 검정 대상은 분포의 전반적인 위치를 나타내는 중위수입니다.
- 두 집단의 순위 배치가 비슷하면 검정통계량은 U 분포의 중앙에 위치합니다.
 - 두 집단을 포함한 전체 데이터의 순위를 구합니다.
 - 검정통계량은 한 집단의 순위보다 작은 다른 집단의 순위 개수를 모두 더한 값입니다.
- (정규성 가정 불만족) 맨-휘트니 U 검정을 실행합니다.

```
>>> pg.mwu(x = sp1, y = sp2)
```

[참고] t-검정 프로세스



분산분석

- 분산분석은 세 개 이상의 집단 간 평균이 같은지 여부를 확인할 때 실행합니다.
 - 일원배치 분산분석은 하나의 요인(입력변수)에 속한 수준(범주)별 변동을 이용합니다.
 - 요인^{factor}에 속한 모든 수준^{level}의 결합을 처리^{treatment}라고 합니다.
 - 요인의 수준이 2개일 때, 분산분석과 t-검정 실행 결과가 같습니다.
 - 요인의 수준이 3개 이상일 때, 분산분석은 t-검정을 여러 번 실행한 결과와 다릅니다!
- 분산분석은 집단 간 변동과 집단 내 변동 비^{ratio}로 평균의 차이를 확인합니다.
 - 귀무가설(H_0): $\mu_1 = \mu_2 = \dots = \mu_k$ (모든 집단의 모평균이 같다.)
 - 대립가설(H_1): 최소한 한 집단의 모평균이 다르다.

[참고] 분산분석 자료구조

- i번째 FuelType(수준)의 j번째 Price(반응)를 표로 나타내면 아래와 같습니다.

FuelType	Price(y_{ij})	i번째 수준 평균(\bar{y}_i)	전체 평균(\bar{y})	총변동($y_{ij} - \bar{y}$)	설명 가능($\bar{y}_i - \bar{y}$)	설명 불가능($y_{ij} - \bar{y}_i$)
CNG	9450	9421.2	9694.7	-244.7	-273.5	28.8
CNG	9250			-444.7		-171.2
:	:			:		:
Diesel	9250	9227.7	9694.7	-444.7	-467.0	22.3
Diesel	9400			-294.7		172.3
:	:			:		:
Petrol	9795	9751.1	9694.7	100.3	56.4	43.9
Petrol	9799			104.3		47.9
:	:			:		:

분산분석표

- 관측값 y_{ij} 는 i번째 수준의 평균(\bar{y}_i)과 오차(ϵ_{ij})의 합으로 표현할 수 있습니다.
 - $y_{ij} = \bar{y}_i + \epsilon_{ij}$ (단, i는 수준, j는 관측값(행) 개수)
 - 오차는 서로 독립이고 평균이 0인 정규분포를 따르며 등분산성 가정을 만족해야 합니다.
- 총변동은 수준으로 설명 가능한 부분과 설명 불가능한 부분의 합계입니다.
 - 위 식의 양변에서 전체 평균(\bar{y})을 차감하면 아래와 같이 변경할 수 있습니다.
 - $y_{ij} - \bar{y} = (\bar{y}_i - \bar{y}) + (y_{ij} - \bar{y}_i)$ # 오른쪽 두 번째 항은 위 식을 오차항으로 정리한 것입니다.
 - $\bar{y}_i - \bar{y}$: 수준 i로 설명할 수 있는 부분입니다.
 - $y_{ij} - \bar{y}_i$: 수준 i로 설명할 수 없는 부분입니다.

분산분석표(계속)

- 각 항의 제곱합을 정리하면 아래 수식이 성립합니다.(뒷 페이지 참조)

$$\sum_{i=1}^k \sum_{j=1}^{n_i} (y_{ij} - \bar{y})^2 = \sum_{i=1}^k n_i (\bar{y}_i - \bar{y})^2 + \sum_{i=1}^k \sum_{j=1}^{n_i} (y_{ij} - \bar{y}_i)^2$$

*Sum of Squared
Total* *Sum of Squared
Treatment* *Sum of Squared
Error*

- 처리와 오차의 제곱합을 나눈 비^{ratio}로 F-통계량과 유의확률을 확인합니다.

구분	제곱합(ss)	자유도(df)	평균제곱합(MS)	F-통계량	유의확률
처리(TR)	SSTR	k-1	MSTR = SSTR ÷ (k-1)	MSTR / MSE	p-value
오차(E)	SSE	n-k	MSE = SSE ÷ (n-k)		
전체(T)	SST	n-1			

[참고] 분산분석 증명

$$\begin{aligned}\sum_{i=1}^k \sum_{j=1}^{n_i} (y_{ij} - \bar{y})^2 &= \sum_{i=1}^k \sum_{j=1}^{n_i} (y_{ij} - \bar{y}_i + \bar{y}_i - \bar{y})^2 \\&= \sum_{i=1}^k \sum_{j=1}^{n_i} (y_{ij} - \bar{y}_i)^2 + \sum_{i=1}^k \sum_{j=1}^{n_i} 2(y_{ij} - \bar{y}_i)(\bar{y}_i - \bar{y}) + \sum_{i=1}^k \sum_{j=1}^{n_i} (\bar{y}_i - \bar{y})^2\end{aligned}$$

그런데, $\sum_{j=1}^{n_i} (y_{ij} - \bar{y}_i) = \sum_{j=1}^{n_i} y_{ij} - n_i \bar{y}_i = \sum_{j=1}^{n_i} y_{ij} - n_i \times \frac{1}{n_i} \sum_{j=1}^{n_i} y_{ij} = 0$

따라서, $\sum_{i=1}^k \sum_{j=1}^{n_i} (y_{ij} - \bar{y})^2 = \boxed{\sum_{i=1}^k n_i (\bar{y}_i - \bar{y})^2} + \boxed{\sum_{i=1}^k \sum_{j=1}^{n_i} (y_{ij} - \bar{y}_i)^2}$

수준 i 로 설명 # 수준 i 로 설명
가능한 부분 불가능한 부분

분산분석의 가정

- 분산분석을 실행하려면 독립성, 정규성 및 등분산성 가정을 만족해야 합니다.
- 독립성 가정은 집단 간 차이에 영향을 주는 외부 요인이 없어야 한다는 것입니다.
- 정규성 가정은 잔차가 정규분포하는지 확인합니다.
 - 정규성 가정을 만족하면 등분산성 검정 결과를 확인하고 분산분석을 실행합니다.
 - 정규성 가정을 만족하지 못하면 크루스칼-왈리스 순위합 검정을 실행합니다.
- 범주가 세 개 이상인 데이터의 등분산성 검정은 바틀렛 검정을 실행합니다.
 - 바틀렛 검정은 정규분포하는 연속형 변수의 모분산이 같은지 확인합니다.
 - 연속형 변수가 정규분포하지 않으면 레벤 검정을 대신 실행합니다.

정규성 검정

- 분산분석은 잔차의 정규성 가정을 만족해야 합니다.
 - 목표변수 개별 값에서 범주별 평균을 차감한 잔차가 정규분포하는지 확인합니다.
 - 귀무가설은 '데이터가 정규분포한다'입니다.
 - 유의확률이 유의수준 0.05 보다 크면 정규성 가정을 만족합니다.
- FuelType 범주별 Price의 정규성 검정을 실행합니다.

```
>>> pg.normality(data = df, dv = 'Price', # 데이터프레임과 연속형 목표변수를 지정합니다.  
                  [참고] 'dv'는 'dependent variable'을 의미합니다.  
                  group = 'FuelType', # 범주형 입력변수를 지정합니다.  
                  method = 'shapiro') # 정규성 검정 방식을 지정합니다.(기본값: 'shapiro')  
                  [참고] 대용량 데이터는 'normaltest'를 지정합니다.
```

등분산성 검정

- 정규성 가정을 만족하는 셋 이상 집단의 등분산성 검정을 실행합니다.
 - 바틀렛 검정의 귀무가설은 '모든 집단의 모분산이 같다'입니다.
 - 유의확률이 유의수준 0.05 보다 크면 모든 집단의 모분산이 같다고 판단합니다.
- (정규성 가정 만족) FuelType 범주별 Price의 등분산성 검정을 실행합니다.

```
>>> pg.homoscedasticity(data = df, dv = 'Price', group = 'FuelType',  
                         method = 'levene') # 등분산 검정 방식을 지정합니다.(기본값: 'levene')  
                           [참고] 정규분포하면 'bartlett'을 지정합니다.
```

- 등분산이면 분산분석, 이분산이면 Welch의 분산분석을 실행합니다.
 - [참고] 분산분석과 Welch의 분산분석을 실행하는 함수가 다릅니다.

분산분석

- (정규성 가정 만족) 셋 이상의 집단 간 평균이 같은지 분산분석을 실행합니다.
 - 귀무가설은 '모든 집단의 모평균이 같다'입니다.
 - 유의확률이 유의수준 0.05 보다 작으면 최소한 한 집단의 모평균이 다르다고 판단합니다.
 - 등분산성 가정 만족 여부에 따라 분산분석을 실행하는 함수가 달라집니다.
- 등분산 가정된 분산분석을 실행합니다.

```
>>> pg.anova(data = df, dv = 'Price', between = 'FuelType')
```
- 이분산 가정된 분산분석을 실행합니다.

```
>>> pg.welch_anova(data = df, dv = 'Price', between = 'FuelType')
```

크루스칼-왈리스 검정

- 크루스칼-왈리스 검정은 셋 이상 집단 간 분포의 위치가 같은지 여부를 판단하는 비모수적인 검정 방법이며 맨-휘트니 U 검정의 확장판입니다.
 - 귀무가설은 '모든 집단 간 분포의 위치가 같다'입니다.
- 집단 간 분포의 위치가 다를수록 검정통계량이 커집니다. # 검정통계량을 보정하면 근사적으로 카이제곱 분포를 따릅니다.
 - 모든 집단을 포함한 전체 데이터의 순위를 구합니다.
 - 검정통계량은 각 집단의 순위합을 제곱하고 개수로 나눈 값을 모두 더한 값입니다.
- (정규성 가정 불만족) 크루스칼-왈리스 순위합 검정을 실행합니다.

```
>>> pg.kruskal(data = df, dv = 'Price', between = 'FuelType')
```

[참고] 검정을 반복하면 안 되는 이유

- 같은 데이터로 검정을 반복하면 유의수준이 증가하여 제 1종 오류를 범할 위험이 커지기 때문입니다.
 - 제 1종 오류는 '참인 귀무가설을 기각하는 오류'입니다.
 - 유의수준을 α 로 설정하면 제 1종 오류를 범하지 않을 신뢰수준은 $1-\alpha$ 가 됩니다.
- 검정을 n 번 반복했을 때 동시에 올바른 결론을 내릴 수 있는 확률은 신뢰수준을 n 번 제곱한 값으로 감소합니다. 아울러 유의수준은 $1-(1-\alpha)^n$ 으로 증가합니다.
 - 예를 들어 검정을 3번 반복하면 신뢰수준은 약 0.857($= 0.95^3$)로 감소합니다.
 - 따라서 검정을 3번 반복하면 유의수준은 약 0.143($= 1 - 0.857$)이 됩니다.

사후분석(다중비교)

- 분산분석 결과 귀무가설을 기각했다면 사후분석 Post-Hoc을 통해 어떤 집단 간 평균이 다른지 확인합니다. 검정을 반복하므로 다중비교라고도 합니다.

구분	상세 내용
Bonferroni	<ul style="list-style-type: none">- 유의수준을 반복횟수로 나눈 값으로 줄여서 전체 유의수준이 커지지 않도록 합니다.- 집단 개수가 5 이상이면 유의수준이 낮아져서 귀무가설을 잘 기각시키지 못합니다.
Scheffe	<ul style="list-style-type: none">- 집단별 크기가 다를 때 사용하며, 검정통계량을 작게 줄이는 방법입니다.- 사회과학에서 많이 사용합니다.
Tukey	<ul style="list-style-type: none">- 집단별 크기가 같을 때 사용하며, t-검정을 변형한 방법입니다.- 자연과학에서 많이 사용합니다. 집단별 크기가 다르면 Tukey-Kramer를 사용합니다.
Dunnett	<ul style="list-style-type: none">- 대조군을 기준으로 다양한 실험군과 쌍대비교 pairwise comparison 합니다.- 집단 개수가 많을 때 결과를 쉽게 확인할 수 있습니다.

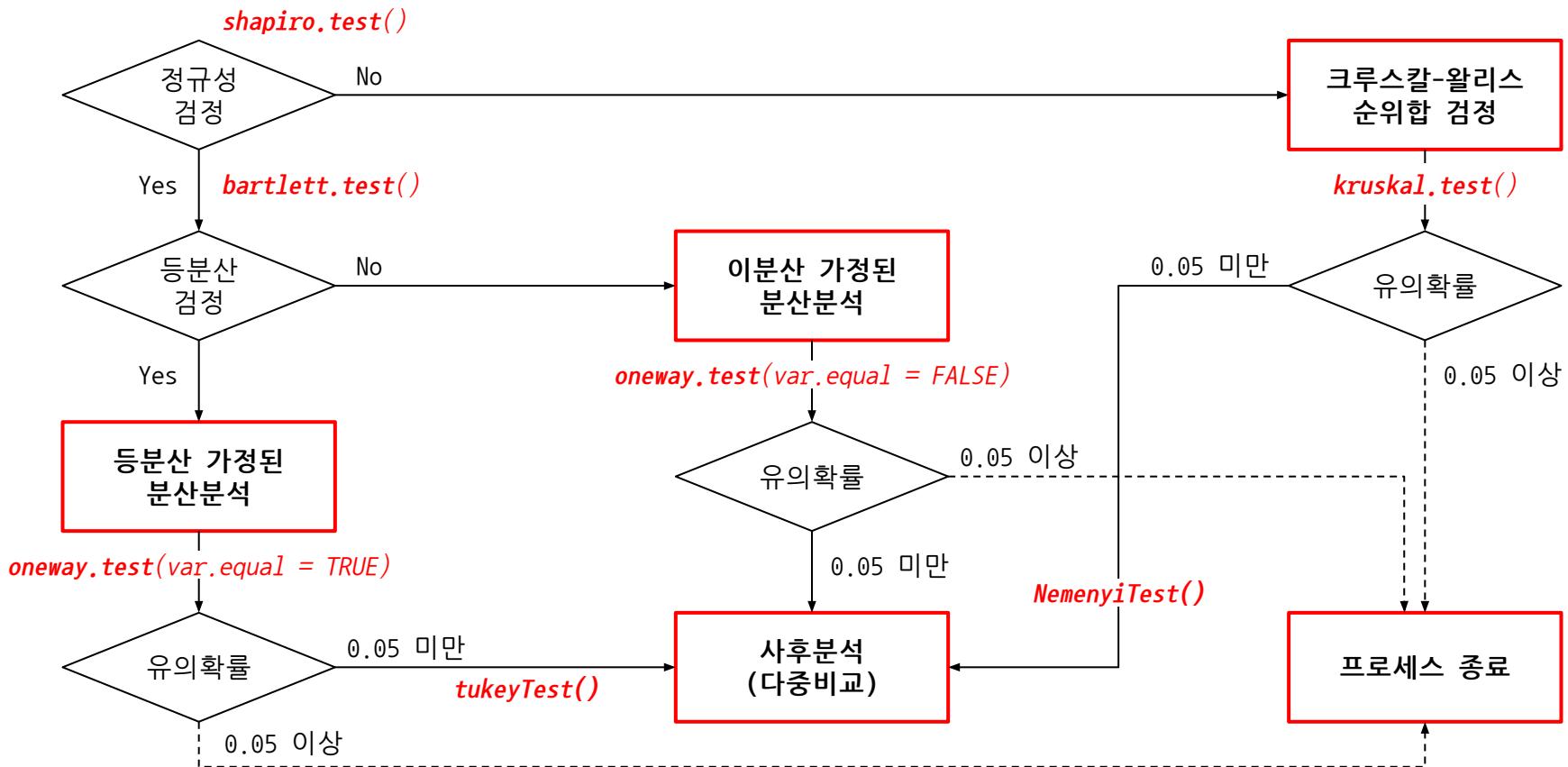
[참고] 위 방식은 등분산일 때 사용하며, 이분산이면 Tamhane의 T2, Games-Howell 또는 Dunnett의 T3를 사용합니다.
순위 데이터를 사용하는 비모수적 방식으로는 Nemenyi, Conover 또는 Dunn 등이 있습니다.

사후분석(다중비교) 실행

- 다양한 사후분석을 실행하고 유의확률을 확인합니다.

```
>>> sp.posthoc_ttest(a = df, val_col = 'Price', group_col = 'FuelType',  
                     p_adjust = 'bonferroni')  
  
>>> sp.posthoc_tukey(a = df, val_col = 'Price', group_col = 'FuelType')  
  
>>> sp.posthoc_scheffe(a = df, val_col = 'Price', group_col = 'FuelType')  
  
>>> sp.posthoc_tamhane(a = df, val_col = 'Price', group_col = 'FuelType')  
  
>>> sp.posthoc_nemenyi(a = df, val_col = 'Price', group_col = 'FuelType')
```

[참고] 분산분석 프로세스



교차분석: 카이제곱 검정

- 입력변수 범주별로 목표변수의 빈도수 차이가 같은지 여부를 확인할 때 교차 분석(카이제곱 검정)을 실행합니다.
- 입력변수 범주별로 목표변수 실제값과 기대값의 차이가 크면 카이제곱 통계량 또한 커집니다.(오른쪽 표 참조)
 - 카이제곱 통계량이 증가하면 유의확률이 감소하며 유의확률이 유의수준 0.05 보다 작으면 귀무가설을 기각합니다.

구분		목표변수		소계
		범주a	범주b	
입력 변수	범주1	O_{1a}	O_{1b}	$O_{1\cdot}$
	범주2	O_{2a}	O_{2b}	$O_{2\cdot}$
소계		$O_{\cdot a}$	$O_{\cdot b}$	Total

$$\chi^2 = \sum \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \quad \begin{cases} O_{ij} : \text{관측값 개수} \\ E_{ij} : \text{기댓값 개수} \end{cases}$$

$$\text{단, } E_{ij} = \text{Total} \times \frac{O_{i\cdot}}{\text{Total}} \times \frac{O_{\cdot j}}{\text{Total}}$$

교차테이블 출력

- 두 범주형 변수의 빈도수를 출력합니다.

```
>>> pd.crosstab(index = df['MetColor'], columns = df['Automatic'])  
# [참고] index 매개변수에 입력변수, columns 매개변수에 목표변수를 각각 지정하면 결과를  
해석하기 좋습니다.
```

- 두 범주형 변수의 상대도수를 출력합니다.

```
>>> pd.crosstab(index = df['MetColor'], columns = df['Automatic'],  
normalize = 'index', # normalize 매개변수에 'index'를 지정하면 각 행별 합계가  
1이 되도록 상대도수를 반환합니다.  
margins = True) # margins 매개변수에 True를 지정하면 맨 아래에 합계를 추가합니다.  
# [참고] normalize 매개변수에 지정한 값에 따라 결과가 달라집니다.
```

카이제곱 검정

- 카이제곱 검정은 두 범주형 변수의 범주별로 빈도수 차이가 있는지 확인합니다.
 - 귀무가설이 '집단 간 빈도수 차이가 없다'입니다.
 - 유의확률이 유의수준 0.05 보다 작으면 집단 간 빈도수 차이가 있다고 판단합니다.
- 두 범주형 변수로 교차분석(카이제곱 검정)을 실행합니다.

```
>>> test = pg.chi2_independence(data = df, x = 'MetColor', y = 'Automatic',  
                                correction = False) # [참고] 예이츠의 연속성 보정을 적용  
                                하지 않도록 설정합니다.
```

- 교차분석 결과를 확인합니다.

```
>>> test[2].iloc[[0]] # test는 기댓값 행렬, 관측값 행렬 및 카이제곱 검정 결과를 원소로 갖는 튜플입니다.  
                    세 번째 원소의 첫 번째 행인 피어슨 카이제곱 검정 유의확률을 확인합니다.
```

[참고] 피셔의 정확검정

- 표본 개수가 매우 작거나(약 40 미만) 또는 교차테이블에서 기대도수가 5 미만인 셀이 있으면 유의확률이 작아지므로 제 1종 오류를 과소평가합니다.
 - 예이츠 Yates 연속성 보정은 분자(관측값-기댓값)에서 0.5를 차감하여 검정통계량을 작게 보정합니다. 하지만 검정통계량을 지나치게 작게 보정하는 경향이 있습니다.
 - 초기하분포 기반의 유의확률을 계산하는 피셔의 정확검정 Fisher's Exact Test 을 이용하는 것이 좋습니다.
- 피셔의 정확검정을 실행하면 오즈비와 유의확률을 반환합니다.

```
>>> stats.fisher_exact(table = test[1]) # test의 두 번째 원소인 관측값 행렬을 지정합니다.
```

변수 제거 및 압축 파일로 저장

- 가설검정 결과를 반영하여 데이터프레임을 전처리합니다.

```
>>> df = df.drop(columns = ['CC', 'Automatic']) # CC와 Automatic을 삭제합니다.
```

```
>>> df = df[df['FuelType'].ne('CNG')] # FuelType이 'CNG'인 건수가 매우 적고 Petrol, Diesel과의  
목표변수 평균에서 유의한 차이가 없으므로 삭제합니다.
```

- df의 행이름을 초기화합니다.

```
>>> df = df.reset_index(drop = True)
```

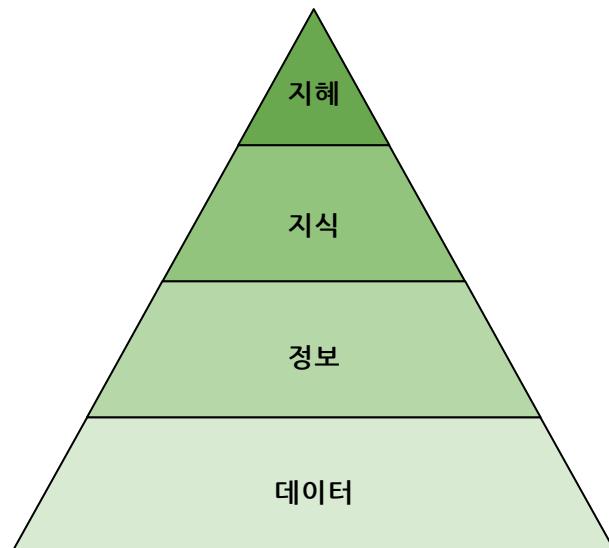
- df를 z 파일로 저장합니다.

```
>>> joblib.dump(value = df, filename = 'Used_Cars_Price_Prep.z')
```

데이터 분석 개요

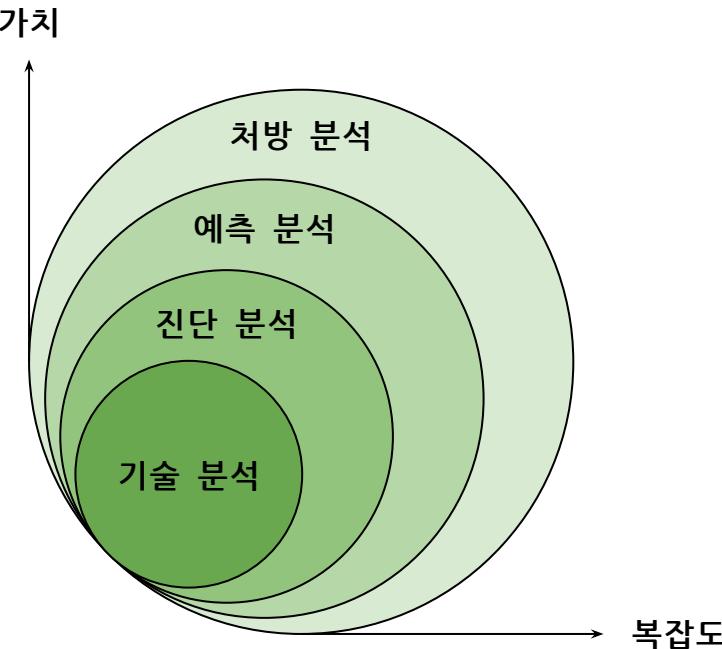
DIKW 피라미드

- DIKW 피라미드는 데이터^{data}, 정보^{information}, 지식^{knowledge}, 지혜^{wisdom}의 관계를 표현한 것입니다.



- 지혜: 축적한 지식 기반으로부터 도출한 창의적인 아이디어
- 지식: 유용한 정보에 개인의 경험을 결합하여 내재화한 것
- 정보: 데이터를 가공하여 패턴 및 의미를 확인한 것
- 데이터: 가공하기 전 상태의 객관적인 사실(수, 문자열, 기호)

데이터 분석의 4가지 유형



- 기술 분석: 무엇이 발생했는가?
 - 과거에 발생한 사건을 다양한 관점에서 정리
- 진단 분석: 원인이 무엇인가?
 - 과거에 발생한 사건의 인과관계 패턴에 초점
- 예측 분석: 앞으로 어떻게 될 것인가?
 - 데이터의 패턴을 파악하여 가까운 미래 예측
- 처방 분석: 무엇을 해야 하는가?
 - 미래 예측에 대한 다양한 대응 방안 검토

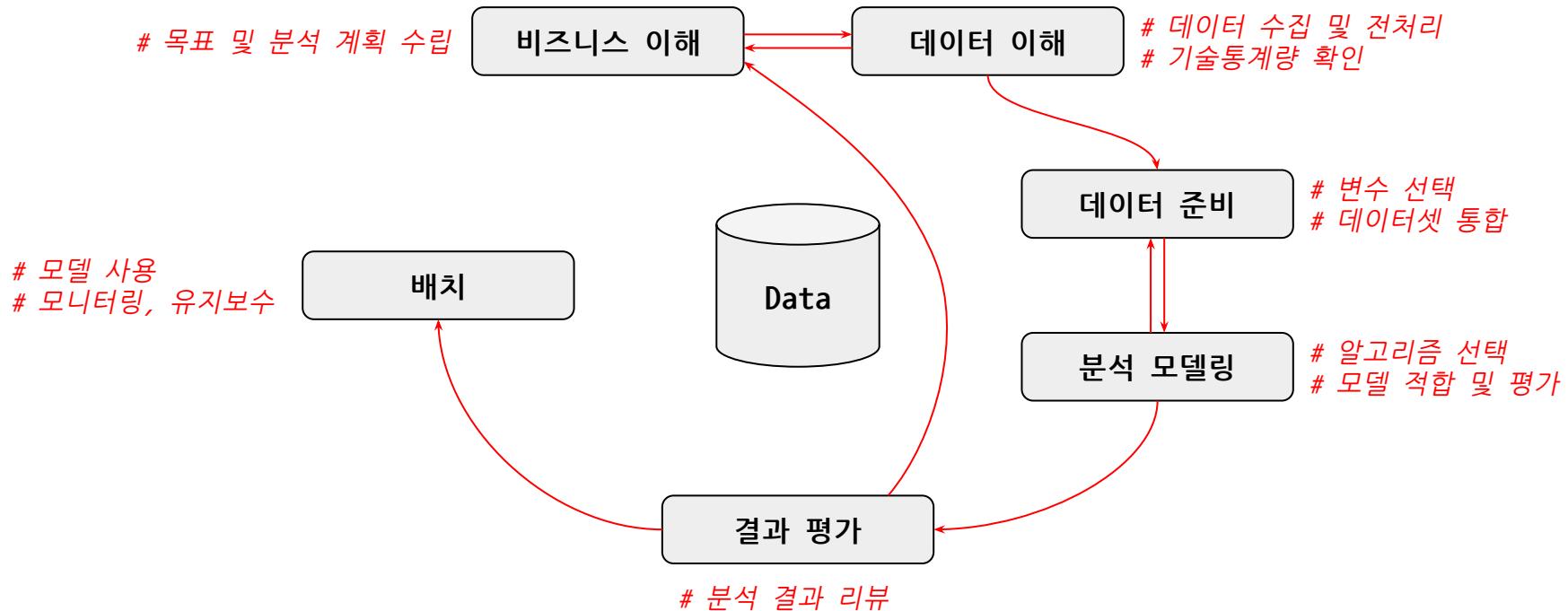
데이터 분석의 4가지 유형

구분	분석 방법	예시
기술 분석 <small>descriptive analysis</small>	<ul style="list-style-type: none">데이터 시각화, 기술통계탐색적 데이터 분석	<ul style="list-style-type: none">신규고객 유입 채널별 비율과 매출액 변화 확인당사 고객의 인구통계학적 특성(성비, 연령 등)
진단 분석 <small>diagnostic analysis</small>	<ul style="list-style-type: none">상관관계 분석, 가설 검정데이터 마이닝(원인 파악)	<ul style="list-style-type: none">신규고객 매출액이 감소한 이유는 최근 시행한 이벤트로 유입된 고객 비중이 증가했기 때문
예측 분석 <small>predictive analysis</small>	<ul style="list-style-type: none">머신러닝(원인보다 예측력)딥러닝	<ul style="list-style-type: none">신규고객 유입 채널 비중을 현재 수준으로 유지한다면 신규고객 매출액은 계속 감소할 것임
처방 분석 <small>prescriptive analysis</small>	<ul style="list-style-type: none">예측 상황별 시뮬레이션최적화 분석	<ul style="list-style-type: none">매출액 감소를 예상하는 상황에서 현재 적용한 이벤트 예산을 어디에 지출해야 하는지 검토

데이터 분석 방법

구분	상세 내용
데이터 시각화	<ul style="list-style-type: none">일변량 데이터의 분포, 이변량 데이터의 관계를 시각적으로 확인함으로써 분석할 데이터에 대한 이해의 폭을 넓힐 수 있습니다.
기술통계 분석	<ul style="list-style-type: none">기술통계는 데이터의 주요 특징을 빠르게 파악할 때 사용하는 통계기법입니다. 평균, 중위수 등 대푯값과 분산, 표준편차 등 흩어진 정도를 파악합니다.
통계적 가설 검정	<ul style="list-style-type: none">통계적 가설검정은 모수를 이용하여 어떤 가설에 대해 합당한 것인지를 판단하는 과정입니다. 피어슨 상관분석, t-검정, 분산분석 및 교차분석 등을 실행합니다.
데이터 마이닝	<ul style="list-style-type: none">입력변수(원인)와 목표변수(결과)의 관계를 확인하는 분석방법입니다. 목표변수의 형태에 따라 선형 회귀분석(연속형)과 로지스틱 회귀분석(범주형)을 실행합니다.
머신러닝/딥러닝	<ul style="list-style-type: none">목표변수의 원인 규명보다 결과에 대한 정확한 예측에 중점을 둔 분석방법입니다. 대표적인 알고리즘으로 의사결정나무, 랜덤 포레스트 등이 있습니다.

데이터 마이닝 프로세스: CRISP-DM Framework



CRISP-DM: Cross Industry Standard Process for Data Mining

머신러닝의 이해

- 머신러닝은 컴퓨터로 데이터를 학습할 때 사용하는 다양한 알고리즘을 포함하는 기법을 의미합니다.

인공지능 Artificial Intelligence

인간에게는 어려운
과업을 컴퓨터가
지능적으로 수행하는
일련의 서비스

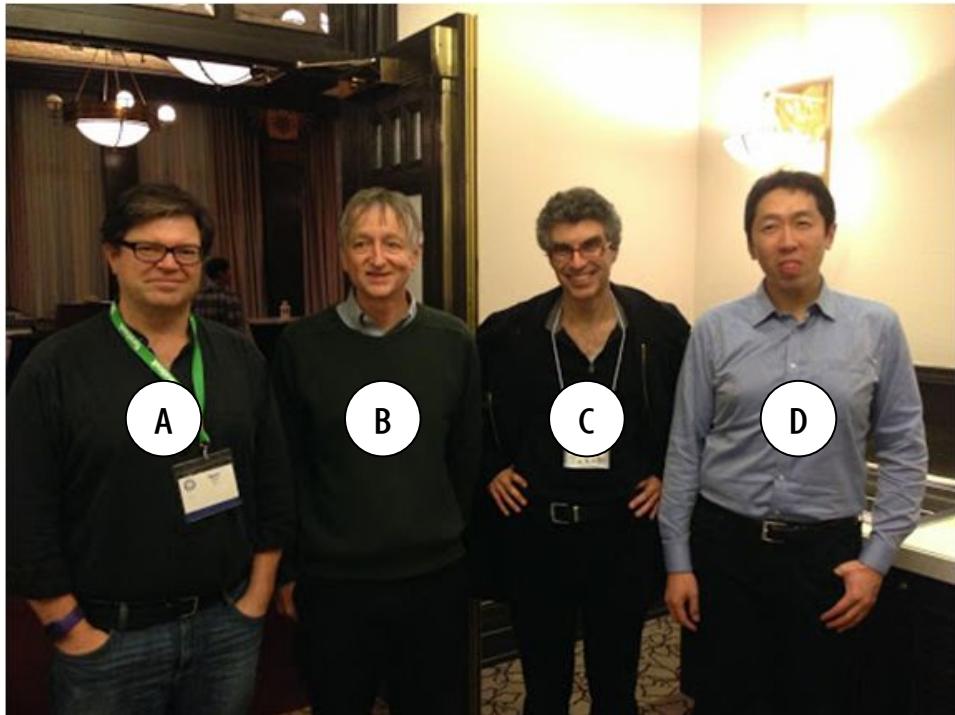
머신러닝 Machine Learning

컴퓨터가 다양한
알고리즘을 이용하여
데이터에 내재된 패턴을
학습하고, 정보를 제공

딥러닝 Deep Learning

인간의 뇌(뉴런)가
학습하는 방법을 모사한
인공신경망^{ANN} 알고리즘을
복잡한 형태로 개선한 것

[참고] 인공지능 4대천왕



A. 얀 르쿤

- 뉴욕대 교수
- 페이스북 AI 수장
- CNN 모델 개발

B. 제프리 힌튼

- 토론토대 교수
- 구글 AI 수장
- 딥러닝 개념 창시자

C. 요슈아 벤지오

- 몬트리올대 교수
- 밀라 MILA 연구소 설립
- 딥러닝 자연어처리 창안

D. 앤드류 응

- 스탠포드대 교수
- 구글 브레인, 바이두를 거쳐 랜динAI 창업

출처 : <https://www.kdnuggets.com/2015/03/talking-machine-deep-learning-gurus-p1.html>

머신러닝의 개념과 한계

- 머신러닝은 데이터에 잠재된 패턴을 학습함으로써 실생활에서의 문제를 해결하고 왔습니다.
 - 로지스틱 회귀분석은 제왕절개 추천 여부에 관한 0~1의 추정 확률을 제시합니다.
 - 나이브 베이즈는 이메일 제목/본문에 포함된 단어로 스팸 메일을 분류합니다.
- 분석가가 목표변수^{label}와 입력변수^{feature}를 지정해주어야 한다는 단점이 있습니다.
 - 머신러닝 알고리즘은 문제 해결을 위해 어떤 입력변수가 필요한지 모릅니다.
 - 머신러닝의 성과는 데이터 품질에 크게 의존합니다.
 - Garbage in, garbage out

머신러닝의 도전과 해결책

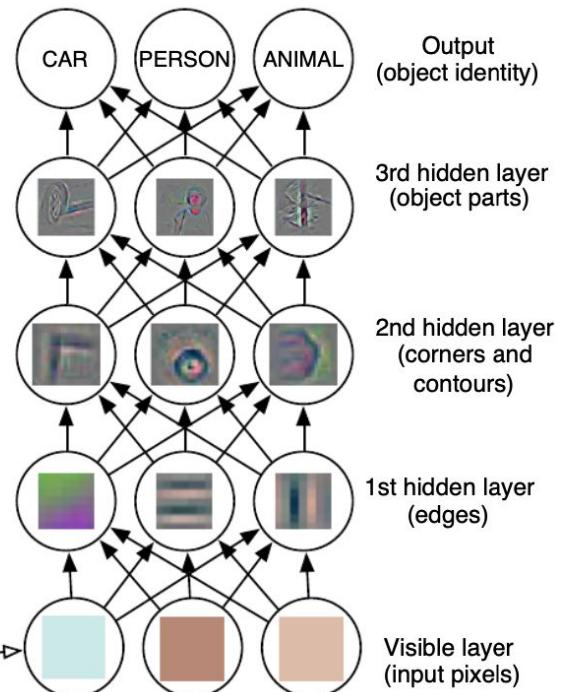
- 머신러닝은 수학적인 규칙을 통해 인간이 풀기 어려운 형식적인 문제를 컴퓨터로 쉽게 해결해왔습니다.
- 그런데 인간은 직관적으로 해결할 수 있지만 형식적으로 서술하기 어려운 문제들, 예를 들어 글자나 이미지를 인식하는 문제는 기호와 내용을 연결하지 못하는 특성 때문에 쉽게 해결하지 못했습니다.
- 하지만 컴퓨터가 개념의 계통구조를 이용하여 과거 데이터로부터 학습하는 구조를 설정함으로써 인간이 문제를 형식적으로 지정해주지 않아도 스스로 개념을 배울 수 있게 되었습니다.
 - 개념의 연결 관계를 도식화하면 여러 층^{layers}을 형성하는데 이를 딥러닝이라고 합니다.

딥러닝의 출현

- 딥러닝의 본질적인 모형은 순방향 심층 신경망 또는 다층 퍼셉트론입니다.
 - 다층 퍼셉트론은 인간의 뇌에 존재하는 뉴런을 모방한 것인데, 입력층과 출력층 사이에 다수의 은닉층을 설정하고 입력층 - 은닉층 - 출력층 간 연결고리를 간단한 수학 함수로 표현한 것입니다.
 - 각 연결고리마다 가중치^{weight}를 부여하며 이전 단계의 출력은 이후 단계의 입력이 됩니다.
 - 각 단계마다 가중치와 입력값으로 출력값을 계산하며 출력값이 활성함수^{activation function}에 미리 설정해놓은 임계점^{threshold}을 넘으면 다음 단계로 자극을 전달합니다.
- 딥러닝은 머신러닝의 한 종류이며 원시 데이터로부터 스스로 특징 집합을 추출할 수 있는 능력이 있으므로 머신러닝이 풀지 못한 많은 문제를 해결하고 있습니다.

딥러닝 모델의 예시

- 입력층에 이미지의 픽셀 데이터(1차원 배열)를 지정합니다.
 - 2차원 배열을 1차원으로 변환합니다.
- 여러 겹의 은닉층을 거치면서 추상화된 특징을 추출합니다.
- 각 연결고리에 부여한 가중치에 따라 출력값이 달라집니다.
- 활성함수가 출력값을 전달할지 여부를 결정합니다.



[참고] ImageNet Result

Classification Results (CLS)

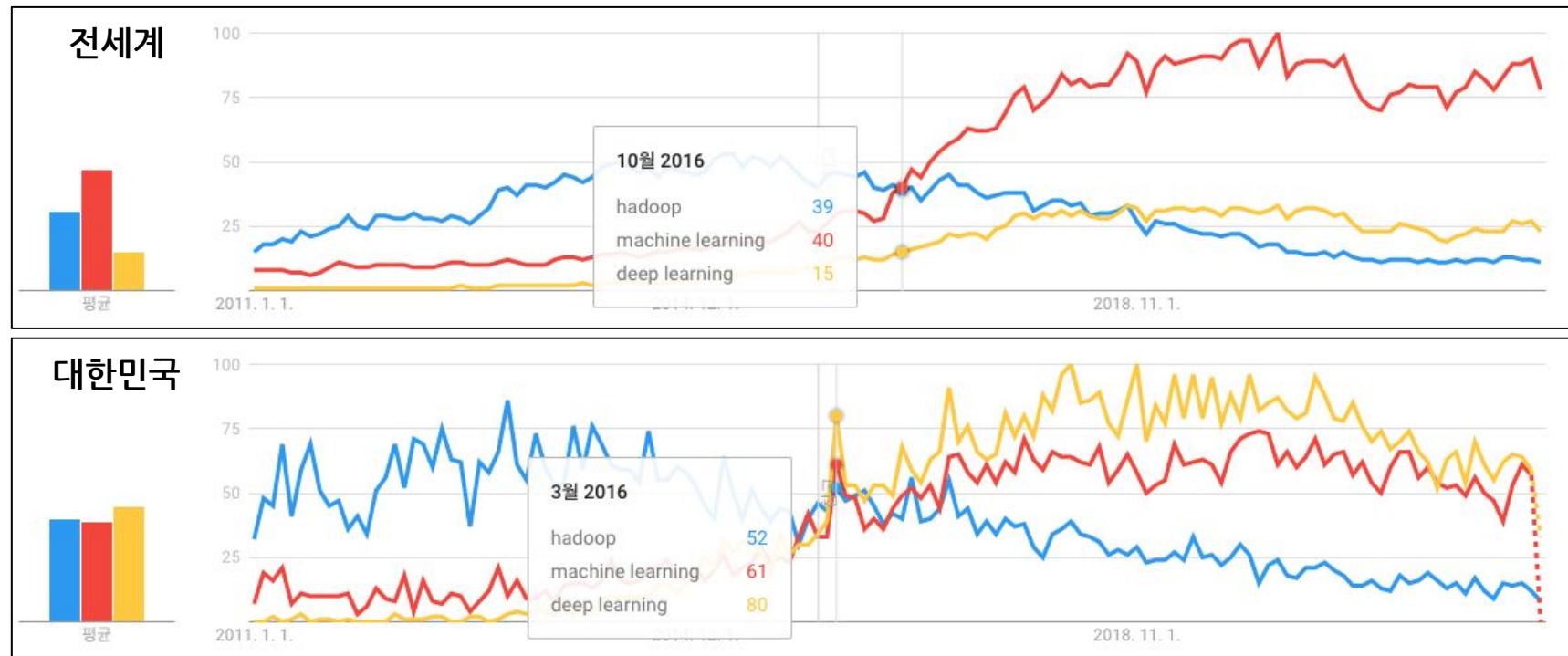


출처: <https://www.bulentsiyah.com/imagenet-winning-cnn-architectures-ilsvrc>

딥러닝 활용 사례

- 딥러닝은 음성 인식 분야에서 오류율을 크게 낮췄습니다.
- 보행자 인식 및 영상 분야에서 주목할만한 성공을 거두었습니다.
- 교통 표지판 분류에서 인간을 능가하는 성과를 보였습니다.
- 제시된 이미지에서 하나의 형체가 아닌 모든 문자를 인식합니다.
- 순환 신경망 Recurrent Neural network, RNN과 LSTM Long Short-Term Memory 알고리즘은 시퀀스 관계를 모형화하므로 시계열 분석 및 기계번역에서 혁신적인 발전을 가능하게 했습니다.
 - GPT-n 시리즈는 기계번역에 괄목할만한 혁신을 이루었습니다.
- 딥마인드는 Atari와 AlphaGo 등 게임 분야에서 딥러닝 강화학습을 활용했습니다.

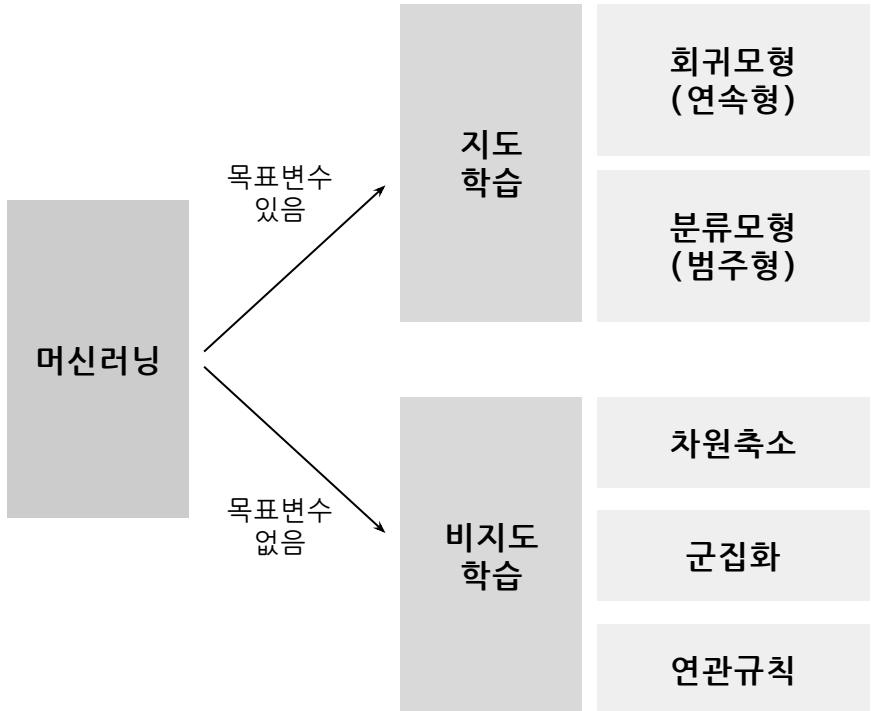
구글 트렌드로 살펴본 관심도 차이(전세계 vs 대한민국)



출처: 구글 트렌드(전세계/대한민국, 2022.6.1 기준)

머신러닝의 종류

- 머신러닝은 목표변수의 유무에 따라 지도학습과 비지도학습으로 구분하며 강화학습도 포함합니다.
- 지도학습은 목표변수의 형태에 따라 회귀모형(연속형)과 분류모형(범주형)으로 구분합니다.
- 비지도학습은 목표변수가 없으므로 데이터의 패턴을 파악하는 차원축소, 군집화 및 연관규칙 등이 있습니다.



[참고] 머신러닝 알고리즘의 종류

분류

- Logistic Regression
- Decision Tree
- Naive Bayes
- KNN(K-nearest neighbors)
- Random Forest
- GBM, XGBoost
- Support Vector Machine
- ANN(Artificial Neural Network)

회귀

- Linear Regression(Stepwise)
- Regularized Linear Regression
- Regression Tree
- KNN(K-nearest neighbors)
- Random Forest
- GBM, XGBoost
- Support Vector Machine
- ANN(Artificial Neural Network)

차원축소

- PCA(Principal Component Analysis)
- Factor Analysis
- MDS(Multi-Dimensional Scaling)

군집화

- Hierarchical Clustering
- K-means Clustering
- K-medoids Clustering
- SOM(Self-Organizing Map)

연관규칙

- MBA(Market Basket Analysis)
- Sequence MBA
- Collaborative Filtering

지도학습

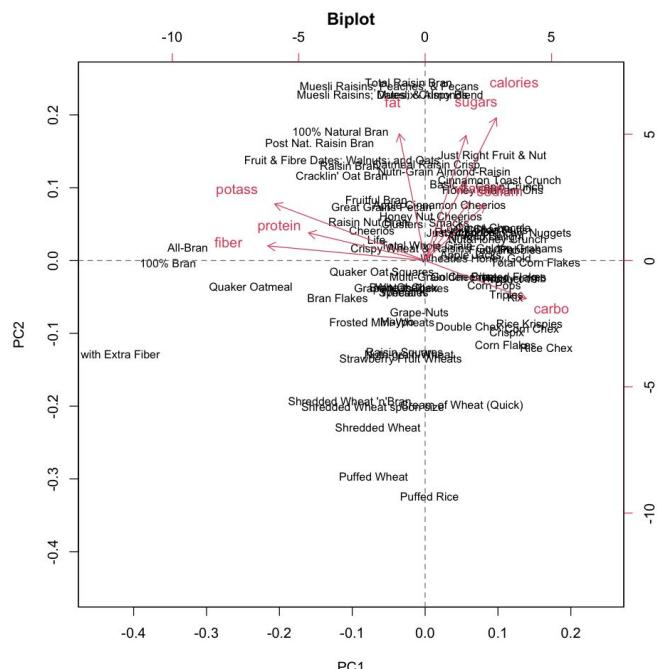
- 목표변수가 연속형이면 회귀모형 Regression을 적합합니다. # [참고] '적합한다'는 모형을 생성한다는 동사 'fit'을 번역한 것입니다.
 - 목표변수가 연속형이므로 회귀모형은 연속형 추정값을 반환합니다.
 - 목표변수를 범주형으로 변환하면 회귀모형을 분류모형으로 바꿀 수 있습니다.
 - 목표변수가 0~100인 실수형 변수를 구간화하여 '합격' 또는 '불합격'인 범주형 변수로 변환하면 분류모형을 적합하는 문제로 바뀝니다.
- 목표변수가 범주형이면 분류모형 Classification을 적합합니다.
 - 목표변수의 범주(레벨)가 두 가지(예를 들어 '예' 또는 '아니오')인 이진 분류와 범주가 여러 개인 다항 분류가 있습니다.
 - 분류모형은 실수형 추정확률(0~1) 및 범주형 추정값(라벨)을 반환합니다.

비지도학습

- 차원축소 Dimension Reduction는 p 열 column 을 m 개로 축소합니다. ($p > m$)[#] [참고] 차원은 열 개수를 의미합니다.
 - 차원축소는 주성분분석 PCA 을 많이 사용하며, $m = 2$ 이면 주성분점수로 산점도를 그립니다.
- 군집화 Clustering 는 n 행 row 을 k 개의 세부 군집으로 나눕니다.
 - 군집화는 행 row 을 축소한다는 점에서 차원축소와 다릅니다.
 - p 차원의 특성(열)이 유사한(거리가 짧은) 관측값(행)을 같은 군집으로 묶습니다.
- 연관규칙 Association Rule 은 조건부 확률을 이용하여 연관성이 높은 규칙을 발견함으로써 추천 recommendation 등 마케팅 활동에 활용합니다.
 - 아빠들이 대형마트에서 기저귀와 맥주를 함께 구매할 확률이 높다는 사례가 있습니다.

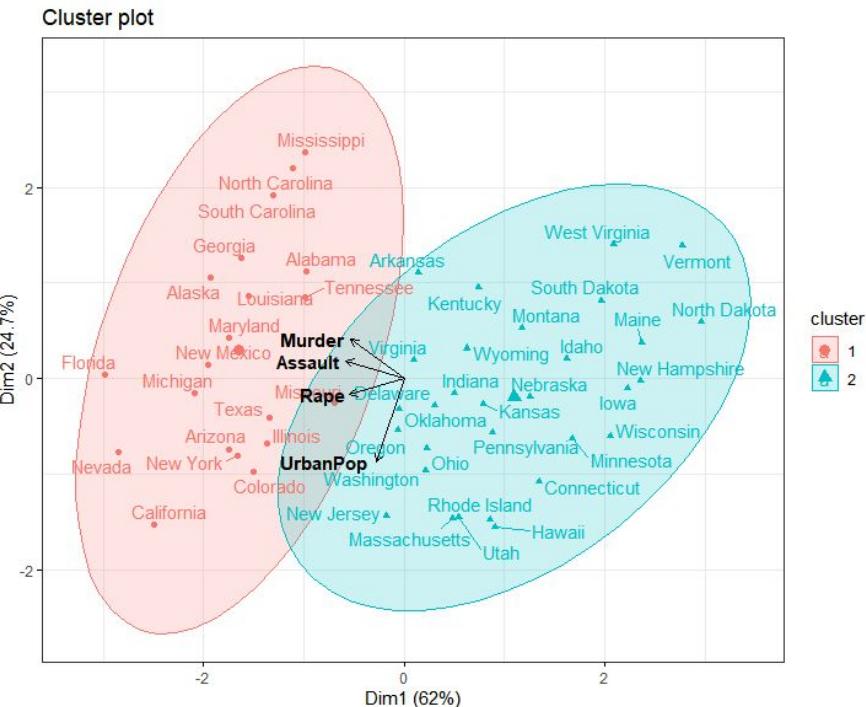
[참고] 차원축소: Principal Component Analysis

- 주성분분석은 p 차원의 변수로 구성된 데이터셋을 m 차원의 주성분으로 축소합니다.
 - 2차원으로 축소하면 산점도를 그릴 수 있으므로 전체 데이터를 한 눈에 파악할 수 있습니다.
- 주성분은 상관관계가 높은 변수의 선형결합이며 계수 coefficients 가 큰 변수들로부터 주성분의 특징을 파악합니다.
- 주성분은 직교하기 때문에 상관계수는 0입니다.
 - 회귀모형의 다중공선성 문제를 해결할 수 있습니다.



[참고] 군집분석: K-means Clustering

- k-means는 각 군집의 중심에서 가까운 관측값을 같은 군집으로 묶습니다.
 - 군집의 중심과 관측값 간 유클리드 거리를 계산합니다.
 - 거리를 계산하기 전 반드시 데이터 표준화를 실행해야 합니다.
- 군집별로 변수의 평균을 계산하거나 행렬도로 시각화하면 군집의 특징을 쉽게 파악할 수 있습니다.



[참고] 연관규칙: Market Basket Analysis

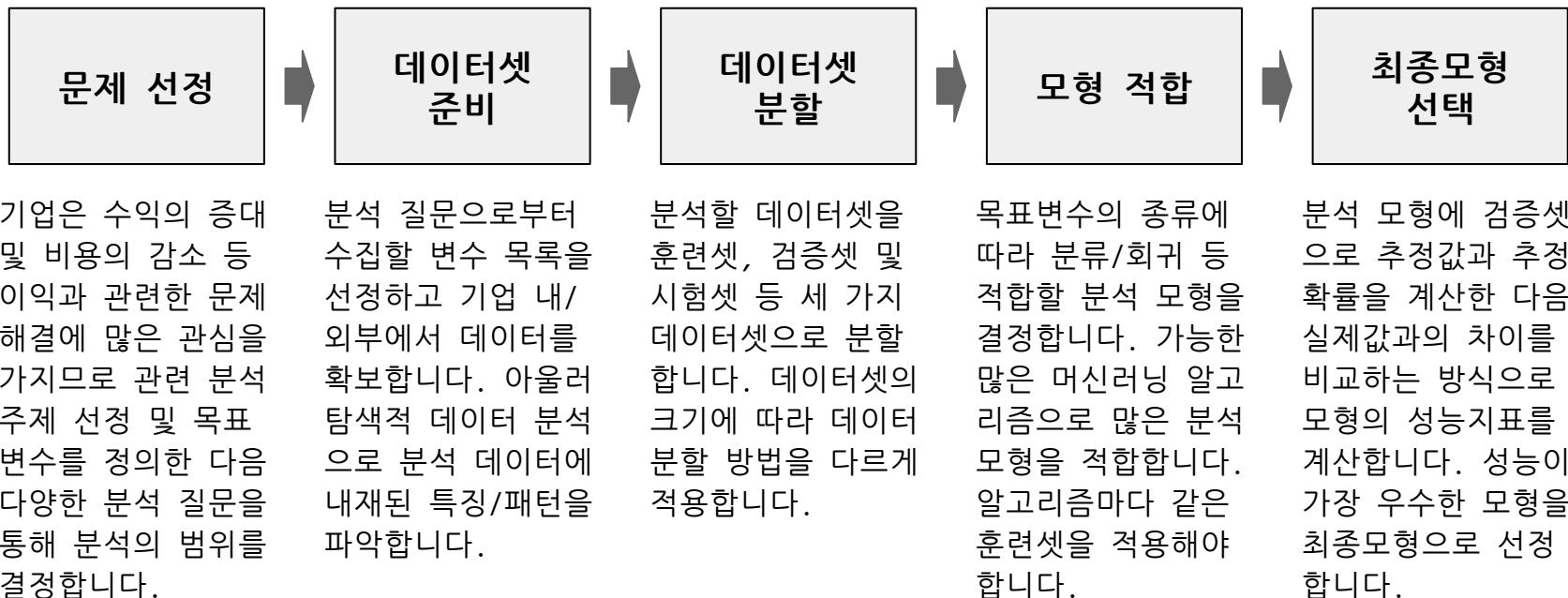
- 연관규칙은 같은 거래에서 함께 구매된 상품 정보로 동시 구매 패턴을 찾습니다.
- 연관규칙을 탐색하는 3가지 기준입니다.
 - 지지도: $P(A \cap B)$
 - 신뢰도: $\frac{P(A \cap B)}{P(A)} = P(B|A)$
 - 향상도: $\frac{P(A \cap B)}{P(A) \times P(B)} = \frac{P(B|A)}{P(B)}$
- 맥주와 기저귀 사례는 장바구니 분석의 대표적인 사례입니다.



출처: <https://www.korea.kr/news/top50View.do>

지도학습 프로세스

- 지도학습 알고리즘은 아래 과정을 거칩니다.

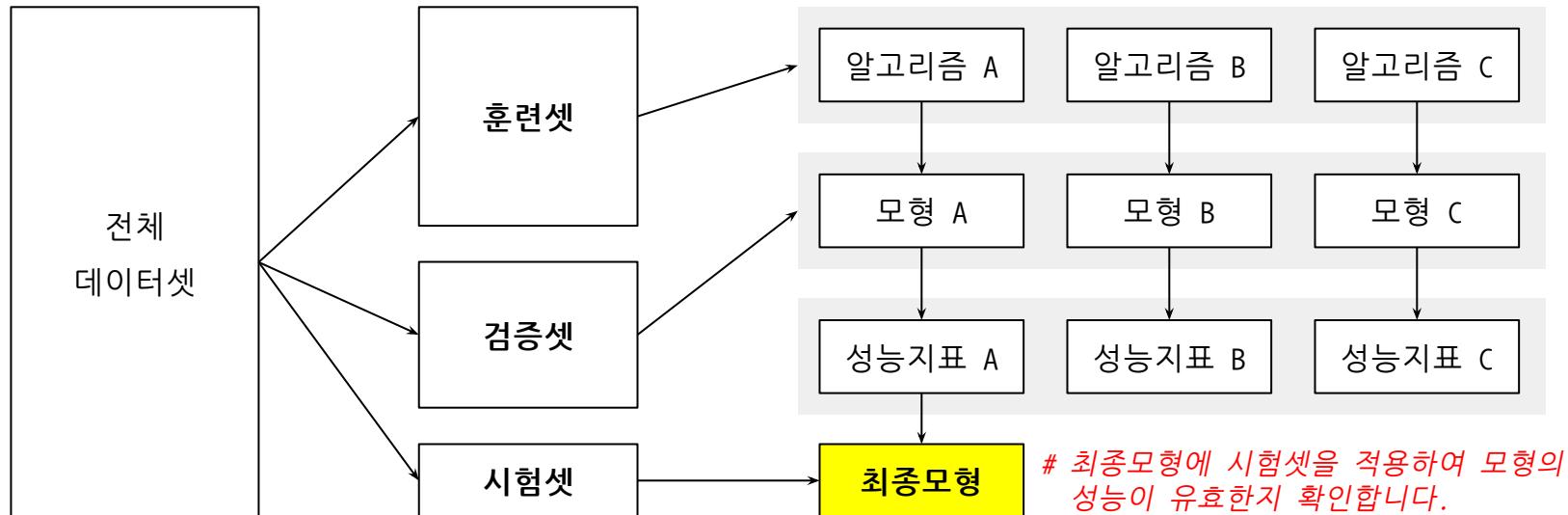


데이터셋 분할 방법

- 데이터셋이 충분히 크다고 판단하면 자료분할^{Hold-out Validation}을 사용합니다.
 - 전체 데이터셋을 훈련셋^{training}, 검증셋^{validation} 및 시험셋^{testing}으로 분할합니다.
 - 자료분할은 단순임의추출방식을 주로 사용하지만 데이터셋을 계층으로 분리해야 한다면 층화추출방식을 사용합니다.
- 데이터셋이 작다고 판단하면 k겹-교차검증^{k-folds Cross Validation}을 사용합니다.
 - 전체 데이터셋을 훈련셋과 시험셋으로 분할한 다음, 훈련셋을 k개로 등분합니다.
 - k-1개로 분류모형을 적합하고, 남은 1개로 추정값을 생성하여 성능지표를 계산합니다.
 - 위 과정을 k번 반복하여 얻은 성능지표의 평균을 해당 모형의 성능지표로 반영합니다.

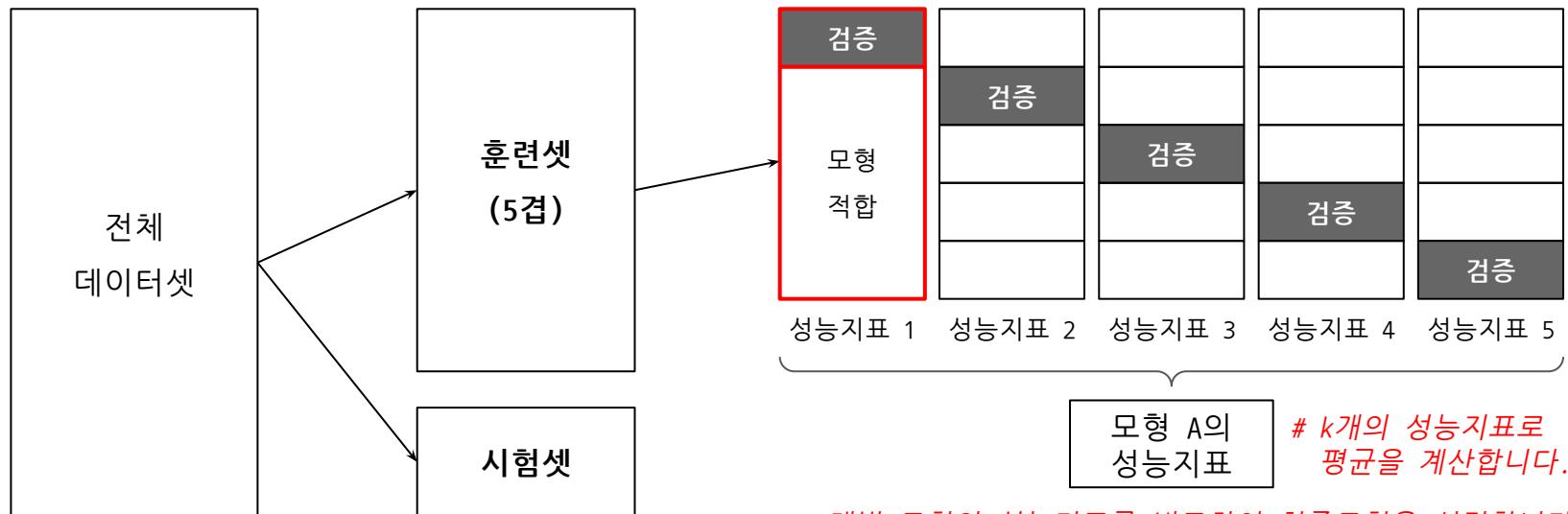
자료분할의 시각적 표현

- 자료분할은 전체 데이터셋을 훈련셋, 검증셋 및 시험셋으로 분할하는 방법입니다. 정해진 비율이 없으므로 스스로 결정합니다.



k-겹 교차검증의 시각적 표현

- 훈련셋을 k개로 등분하고 k-1개로 모형을 적합하고 남은 1개로 검증하는 것을 k번 반복하여 k개 성능지표의 평균이 우수한 최종모형을 선택합니다.



회귀모형 성능 평가 기준

- 회귀모형은 목표변수가 연속형 벡터이므로 실제값과 추정값 간의 차이인 오차를 계산하여 모형의 성능을 평가합니다.

$$\text{오차}^{\text{Error}} = \text{실제값}^{\text{Actual Value}} - \text{추정값}^{\text{Predicted Value}}$$

- 모형 전체의 오차 크기를 계산할 때 개별 오차를 단순하게 합산하면 부호가 서로 다른 오차로 인해 0에 수렴하므로 오차를 제곱하거나 절대값을 취하는 방식으로 부호를 통일합니다.
- 회귀모형 성능을 평가하는 지표는 다양하지만 결과는 비슷하므로 한 가지 지표를 정하여 사용하는데 머신러닝 알고리즘은 MSE를, 사람은 RMSE를 많이 사용합니다.

회귀모형 성능 평가 지표(5종)

성능지표	세부 내용	공식
MSE ^{Mean Squared Error}	<ul style="list-style-type: none"> - 오차 제곱의 평균입니다. - 오차가 클수록 MSE는 크게 증가합니다. 	$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$
RMSE ^{Root Mean Squared Error}	<ul style="list-style-type: none"> - MSE의 양의 제곱근입니다. - 실제값과 척도가 같습니다. 	$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$
RMSLE ^{Root Mean Squared Log Error}	<ul style="list-style-type: none"> - 추정값과 실제값의 상대적 차이를 나타내며 과소추정에 높은 벌점을 부여합니다. 	$\sqrt{\frac{1}{n} \sum_{i=1}^n \log \left(\frac{\hat{y}_i + 1}{y_i + 1} \right)^2}$
MAE ^{Mean Absolute Error}	<ul style="list-style-type: none"> - 오차 절대값의 평균입니다. 	$\frac{1}{n} \sum_{i=1}^n y_i - \hat{y}_i $
MAPE ^{Mean Absolute Percentage Error}	<ul style="list-style-type: none"> - 오차를 실제값으로 나눈 절대값 평균입니다. 	$\frac{1}{n} \sum_{i=1}^n \frac{ y_i - \hat{y}_i }{ y_i }$

분류모형 성능 평가 기준

- 머신러닝 알고리즘은 우수한 분류모형을 선택할 때 실제값과 추정값이 같은 정도(정확도) 또는 다른 정도(오분류율)를 사용합니다.
 - 그런데 정확도와 오분류율은 다수 범주에 초점을 맞춘 가중평균된 값이므로 관심 있는 소수의 범주를 제대로 분류하지 못하는 분류모형도 높은 값을 가질 수 있습니다.
 - 따라서 정확도와 오분류율은 목표변수 범주별 백분율이 비슷할 때 효과적인 지표입니다.
- 해결해야 하는 문제의 종류와 데이터셋의 특징에 따라 성능 평가 지표가 다를 수 있다는 점에 유의해야 합니다.
 - 산업별로 분석 주제가 다양하므로 분류모형의 성능에 대한 요구사항도 달라집니다.
 - 목표변수 범주별 백분율이 크게 다르면 정확도보다 민감도와 정밀도가 더 중요합니다.

분류모형 성능 평가 지표(3종)

- 혼동행렬 Confusion Matrix 은 실제값 Actual Value 과 추정값 Predicted Value 의 빈도수를 성분으로 갖는 행렬이며, 분류모형이 실제값을 얼마나 다르게 추정했는지 확인합니다.
 - 정확도 accuracy , 민감도 sensitivity , 정밀도 precision , 특이도 specificity 등 성능지표를 계산합니다.
 - 민감도와 정밀도가 중요한데, 두 지표의 방향이 서로 엇갈리는 경우가 자주 발생합니다.
- F1 점수는 민감도와 정밀도의 조화평균이며, 두 지표가 높아야 큰 값을 갖습니다.
- ROC Receiver Operating Characteristic 곡선은 x축이 1-특이도, y축이 민감도인 그래프입니다.
 - ROC 곡선을 그렸을 때 왼쪽 위 모서리에 가장 가까운 분류모형을 찾습니다.
 - ROC 곡선 아래의 면적을 AUC Area Under Curve 라고 하며, AUC로 분류모형의 성능을 비교합니다.

혼동행렬

		실제값	
		긍정 Positive	부정 Negative
추정값	긍정 Positive	True Positive	False Positive
	부정 Negative	False Negative	True Negative

Positive는 관심 있는 소수 범주로 설정합니다.
예를 들어 '대출 연체', '고객 이탈' 등입니다.

- 혼동행렬의 4가지 성분입니다.

- TP: 모형이 긍정으로 추정, 실제값도 긍정인 건수입니다.
- FP: 모형은 긍정으로 추정, 실제값은 부정인 건수입니다.
- FN: 모형은 부정으로 추정, 실제값은 긍정인 건수입니다.
- TN: 모형이 부정으로 추정, 실제값도 부정인 건수입니다.

혼동행렬(계속)

- 실제값이 긍정인 건수

$$P = TP + FN$$

- 실제값이 부정인 건수

$$N = FP + TN$$

- 전체 건수

$$T = TP + FN + FP + TN$$

$$= P + N$$

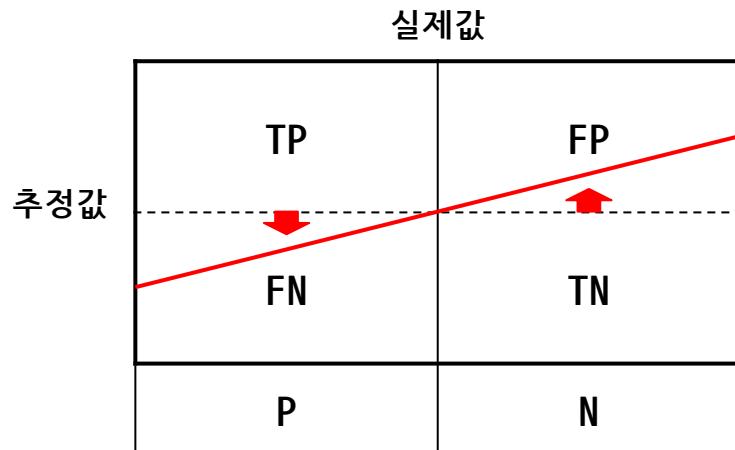
		실제값	
		TP	FP
추정값	TP		
	FN		TN
	P		N

분류모형의 성능에 따라 긍정과 부정으로 분류하는 빈도수가 달라집니다.

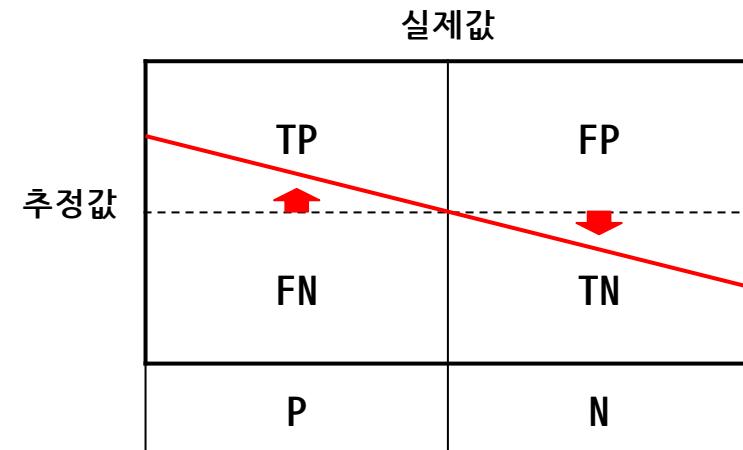
따라서 분류모형에 따라 가로선의 위치가 바뀝니다.

시험셋의 실제값 빈도수는 고정이므로 세로선의 위치는 바뀌지 않습니다.

[참고] 분류모형 성능에 따른 가로선의 변화



[분류 성능이 좋은 모형]



[분류 성능이 나쁜 모형]

혼동행렬 성능 평가 지표

성능지표	세부 내용	공식
정확도 (accuracy)	- 실제값과 추정값이 같은 건수를 전체 건수로 나눈 값입니다.	$\frac{TP + TN}{P + N}$
민감도 (sensitivity)	- 실제값이 긍정인 건에서 분류모형이 맞춘 비율입니다. 재현율 ^{recall} 또는 참긍정비율 ^{True Positive Rate, TPR} 이라고도 합니다.	$\frac{TP}{P}$
정밀도 (precision)	- 분류모형이 긍정이라고 한 건에서 분류모형이 맞춘 비율입니다.	$\frac{TP}{TP + FP}$
특이도 (specificity)	- 실제값이 부정인 건에서 분류모형이 맞춘 비율입니다.	$\frac{TN}{N}$
1-특이도	- 실제값은 부정인데 분류모형이 긍정으로 오분류한 비율입니다. 거짓긍정비율 ^{False Positive Rate, FPR} 이라고도 합니다.	$\frac{FP}{N}$

혼동행렬에서 중요한 지표

- 분류모형의 성능을 판단할 때 정확도는 좋은 지표일까요?

A

250	250	500
250	250	500
500	500	1000

B

5	45	50
45	905	950
50	950	1000

$$\text{정확도} = (250 + 250) \div 1000 = 0.50$$

$$\text{민감도} = 250 \div 500 = 0.50$$

$$\text{정밀도} = 250 \div 500 = 0.50$$

$$\text{정확도} = (5 + 905) \div 1000 = 0.91$$

$$\text{민감도} = 5 \div 50 = 0.10$$

만약 이상거래
탐지모형이라면?

$$\text{정밀도} = 5 \div 50 = 0.10$$

혼동행렬에서 중요한 지표(계속)

- 분류모형의 성능에 따라 민감도와 정밀도의 방향이 엇갈릴 수 있습니다.

A

40	20	60
10	930	940
50	950	1000

B

45	25	70
5	925	930
50	950	1000

$$\text{민감도} = 40 \div 50 = 0.80$$

$$\text{정밀도} = 40 \div 60 = 0.67$$

$$\text{정확도} = (40 + 930) \div 1000 = 0.97$$

$$\text{민감도} = 45 \div 50 = 0.90$$

$$\text{정밀도} = 45 \div 70 = 0.64$$

$$\text{정확도} = (45 + 925) \div 1000 = 0.97$$

혼동행렬에서 중요한 지표(계속)

- 민감도와 정밀도가 함께 높을수록 성능이 좋은 모형입니다.

A

40	20	60
10	930	940
50	950	1000

B

45	15	60
5	935	940
50	950	1000

$$\text{민감도} = 40 \div 50 = 0.80$$

$$\text{정밀도} = 40 \div 60 = 0.67$$

$$\text{정확도} = (40 + 930) \div 1000 = 0.97$$

$$\text{민감도} = 45 \div 50 = 0.90$$

$$\text{정밀도} = 45 \div 60 = 0.75$$

$$\text{정확도} = (45 + 935) \div 1000 = 0.98$$

혼동행렬에서 중요한 지표(계속)

- 민감도와 특이도는 반비례 관계일 수 있습니다.

A	0	0	0
	50	950	1000
	50	950	1000

$$\text{민감도} = \text{TP} \div \text{P} = 0.00$$

$$\text{특이도} = \text{TN} \div \text{N} = 1.00$$

$$1 - \text{특이도} = 0.00$$

ROC 곡선의
점(0, 0)을
의미합니다.

B	50	950	1000
	0	0	0
	50	950	1000

$$\text{민감도} = \text{TP} \div \text{P} = 1.00$$

$$\text{특이도} = \text{TN} \div \text{N} = 0.00$$

$$1 - \text{특이도} = 1.00$$

ROC 곡선의
점(1, 1)을
의미합니다.

F1 점수

- F1 점수는 민감도와 정밀도의 조화평균입니다.
 - 조화평균으로 계산하는 이유는 둘 중 하나라도 낮으면 F1 점수가 낮아지기 때문입니다.
- 조화평균은 평균속력을 계산할 때 사용합니다.
 - 같은 거리를 왕복할 때 가는 속력과 오는 속력의 차이가 클수록 평균속력이 낮아집니다.
 - 예를 들어 약 100km 거리(서울시청~천안시청)를 갈 때 100km/h, 올 때 25km/h 속력으로 왕복했다면 평균속력은 얼마가 될까요?
 - 분류모형의 민감도가 1.00이고 정밀도는 0.25라면 F1 점수는 얼마가 될까요?

[참고] 조화평균의 이해

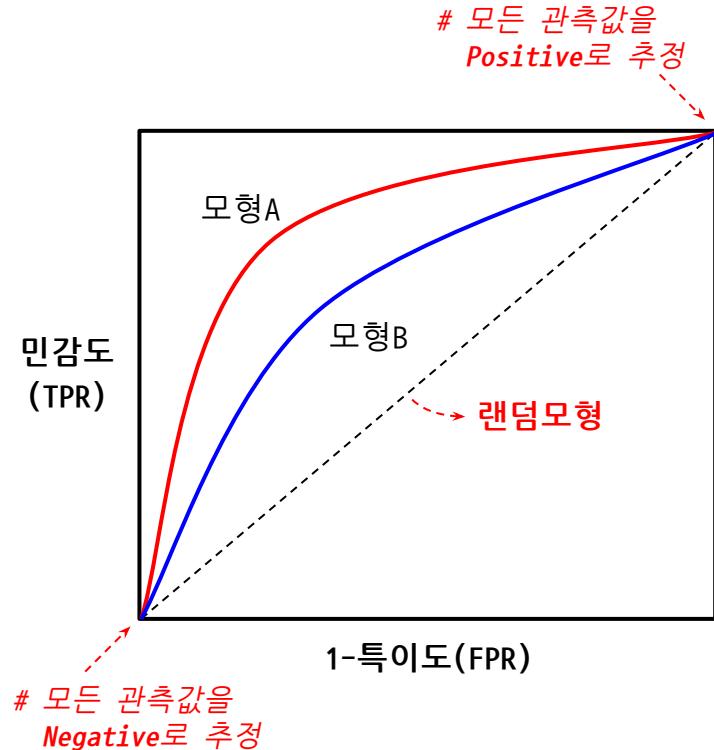
- 조화평균은 '역수로 계산한 산술평균의 역수'입니다.

$$\frac{2}{\frac{1}{a} + \frac{1}{b}} = \frac{2}{\frac{a+b}{ab}} = \frac{2ab}{a+b}$$

- 조화평균은 평균속력을 계산할 때 사용합니다.(속력 = 거리 ÷ 시간)
 - 같은 거리를 왕복할 때 만약 가는 속력과 오는 속력이 다르면 투입된 시간이 다르므로 평균 속력을 계산하려면 속력의 산술평균이 아닌 시간의 산술평균을 계산해야 합니다.
 - 시간(거리 ÷ 속력)의 산술평균은 '속력의 역수'로 계산한 평균입니다.
 - 평균시간에 역수를 취하면 원래 차원(평균속력)으로 되돌아옵니다.

ROC 곡선

- ROC 곡선은 X축에 1-특이도, Y축에 민감도를 놓고 분류모형의 성능을 그린 그래프입니다.
 - ROC 곡선은 항상 두 점 $(0, 0)$ 과 $(1, 1)$ 을 지납니다.
- 성능이 우수한 분류모형은 1-특이도가 낮고 민감도가 높은 모형입니다.
 - ROC 곡선이 왼쪽 위 모서리에서 가까울수록 성능이 좋은 모형입니다.
- [중요] ROC 곡선은 추정확률로 그립니다!



[참고] ROC 곡선이 그려지는 원리

- 분류모형으로부터 추정확률을 생성하고 분리 기준점마다 추정값으로 변환합니다.
추정값으로 계산한 점(1-특이도, 민감도)들을 이어 붙이면 ROC 곡선을 그립니다.

행이름	추정확률		분리 기준점 <small>cut-off</small> 으로 추정값 생성					
	Negative	Positive	0.00	...	0.50	...	1.00	
1	0.0023	0.9977	Positive	...	Positive	...	Negative	
2	0.9783	0.0217	Positive	...	Negative	...	Negative	
3	0.8912	0.1088	Positive	...	Negative	...	Negative	
4	0.1432	0.8568	Positive	...	Positive	...	Negative	
:	:	:	:	:	:	:	:	

FPR: 1

TPR: 1

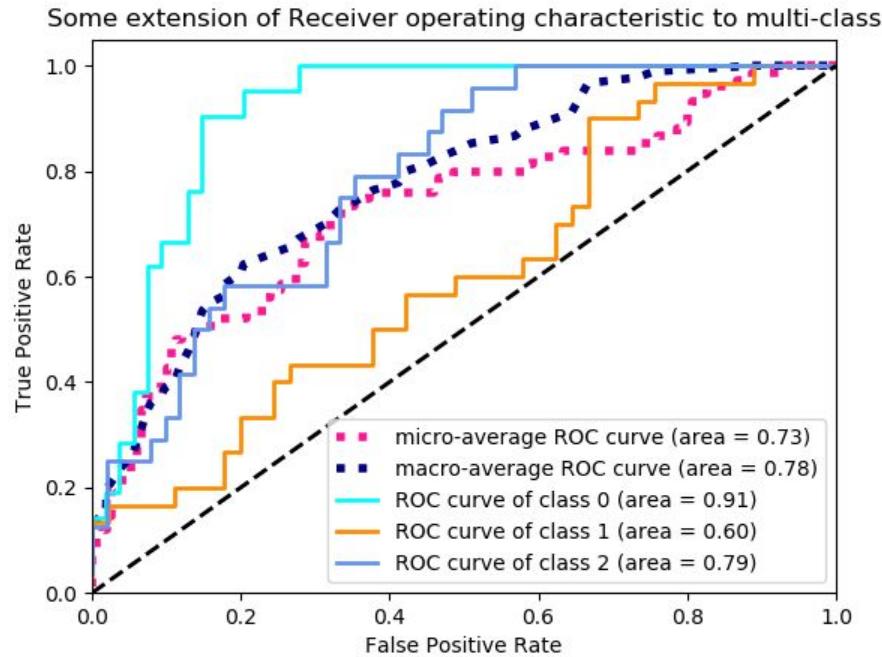
FPR: 0.2

TPR: 0.8

FPR: 0

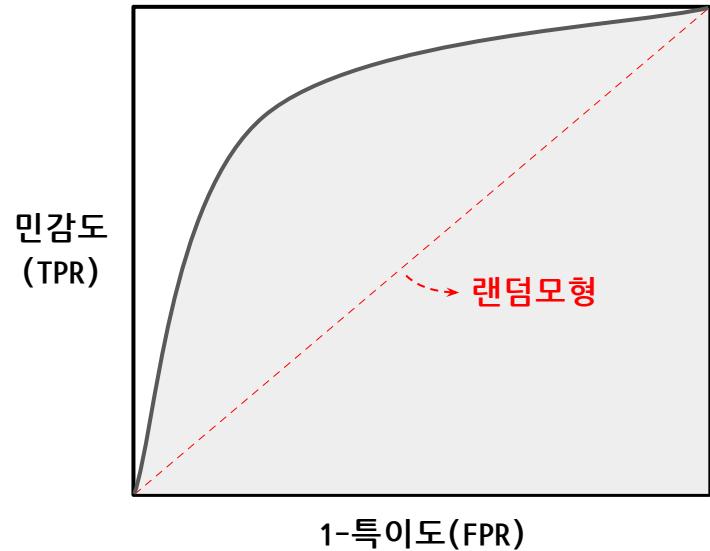
TPR: 0

[참고] ROC 곡선 예시



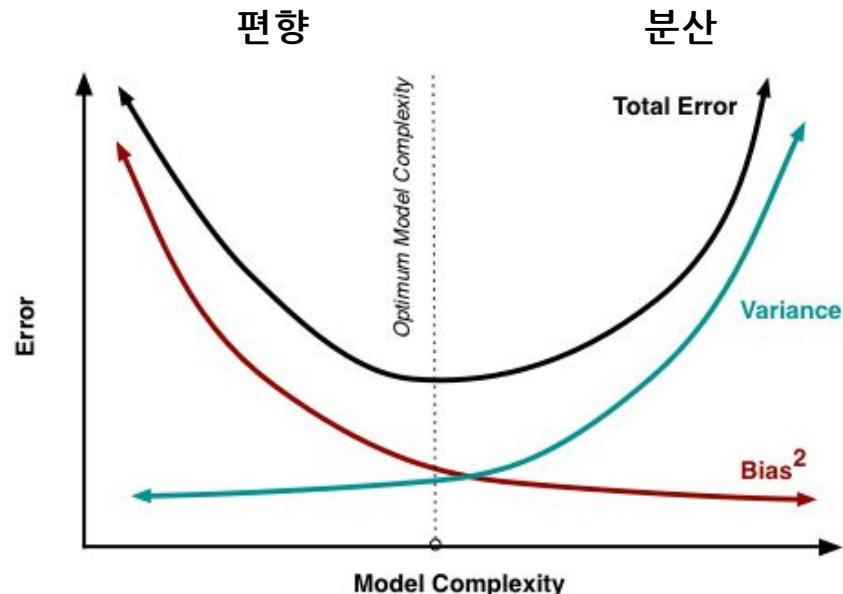
AUC

- AUC^{Area Under Curve}는 ROC 곡선 아래 면적입니다.
 - ROC 곡선이 왼쪽 모서리에 가까울수록 AUC는 1에 가까워집니다.
 - 추정값을 랜덤하게 반환하는 분류모형의 ROC 곡선은 오른쪽 그림의 빨간색 점선과 같고, 이 모형의 AUC는 0.5입니다.
 - AUC가 0.5 미만인 모형은 랜덤모형보다 성능이 낮으므로 가치가 없습니다.
- [중요] AUC도 추정확률로 계산합니다.



모형의 복잡도와 오차의 관계

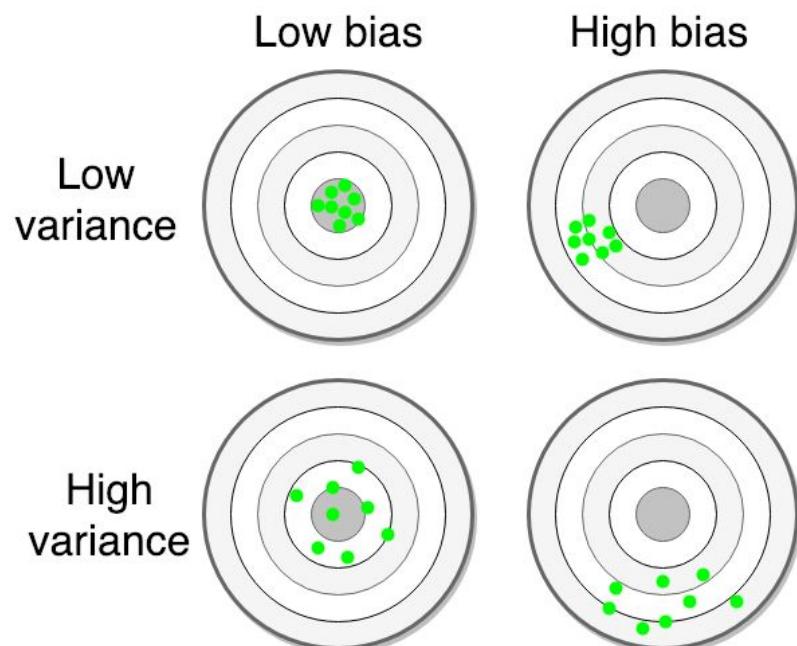
- 모형의 오차는 편향 제곱과 분산의 합입니다.
 - $\text{error} = \text{bias}^2 + \text{variance}$
- 학습 초기에 오차가 상당히 큰 이유는 과소적합에 의한 편향 때문입니다.
- 학습을 진행하면 총오차는 감소하다가 어느 시점을 지나면 다시 증가하는데 그 이유는 모델이 과적합하여 분산이 증가하기 때문입니다.



출처: <https://stats.stackexchange.com/questions/336433/bias-variance-tradeoff-math>

[참고] 편향과 분산

- 학습 초기 과소적합^{underfitting}된 모형은 충분하게 학습하지 않았으므로 편향을 가집니다.
 - 아직 모형이 단순하다는 의미입니다.
- 과적합^{overfitting}된 모형은 훈련셋을 외울 정도로 학습했으므로 모형을 적합할 때 사용하지 않은 데이터를 맞추지 못하기 때문에 분산이 증가합니다.
 - 모형이 복잡해졌다는 것을 의미합니다.



출처: <http://www.machinelearningtutorial.net/2017/01/26/the-bias-variance-tradeoff/>

과적합 및 과소적합 판단 기준

- 머신러닝 모형을 적합하면 훈련셋과 검증셋으로 성능지표를 계산하고 아래 표를 참고하여 과적합 또는 과소적합 여부를 판단합니다.

훈련셋의 성능	시험셋의 성능	판단 결과
높음	높음	성능 좋은 모형
높음	낮음	과적합 의심
낮음	낮음	과소적합 의심
낮음	높음	(알고리즘, 샘플링, 샘플 크기)

선형 회귀분석

선형 회귀분석의 개요

- 선형 회귀분석은 하나 이상의 입력변수와 목표변수 간 관계를 파악하기에 유용한 데이터 마이닝 기법의 일종이며 아래 관계식으로 표현합니다.
 - 모집단: $y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_p x_{pi} + \epsilon_i$ (마지막 항은 오차)
 - 표본: $y_i = b_0 + b_1 x_{1i} + b_2 x_{2i} + \cdots + b_p x_{pi} + e_i$ (마지막 항은 잔차)
 - 목표변수의 추정값(\hat{y}_i)은 입력변수의 선형결합에 y절편(b_0)을 더한 관계식으로 표현할 수 있으며 실제값과 추정값 간 차이를 잔차(e_i)라고 합니다.
 - 회귀분석은 입력변수별 회귀계수(b_j)를 추정합니다. 회귀계수는 다른 입력변수를 고정한 상태에서 해당 입력변수가 1단위 증가할 때 목표변수에 미치는 크기를 의미합니다.
 - 선형 회귀모형은 목표변수의 변동을 설명할 수 있는 크기를 결정계수(R^2)로 표현합니다.

선형 회귀분석의 기본 가정

- 목표변수와 입력변수는 직선의 관계를 가져야 합니다.
 - 두 변수로 산점도를 그렸을 때 직선의 관계가 있는지 확인합니다.
 - 피어슨 상관분석을 실행하여 두 변수 간 상관관계가 있는지 확인합니다.
 - 선형 회귀모형의 입력변수-잔차 산점도를 그려 잔차의 패턴을 확인합니다.
- 입력변수끼리 강한 상관관계가 없어야 합니다.
 - 다른 입력변수에 종속된 입력변수를 다중공선성 입력변수라고 합니다.
 - 다중공선성 입력변수는 목표변수에 대한 설명력을 다른 입력변수와 나눠가집니다.
 - 그 결과로 회귀계수의 설명력이 낮아지므로 다중공선성 변수를 제거해야 합니다.

선형 회귀모형 진단

- 회귀모형의 잔차항에 대한 가정은 아래와 같습니다.
 - 잔차는 평균이 0이고 표준편차가 σ인 정규분포를 따라야 합니다.[정규성]
 - 모든 입력변수 값에서 동일한 분산을 갖습니다.[등분산성]
 - 잔차끼리 자기 상관이 없어야 합니다.[독립성]
 - 잔차의 산점도를 그렸을 때 일정한 패턴이 없어야 합니다.[잔차의 패턴]
- 선형 회귀직선은 입력변수의 전체 영역에서 목표변수와의 관계를 설명하는 유일한 직선이므로 만약 관측값에 이상치가 있으면 전체 관계가 왜곡될 수 있습니다.
 - 따라서 이상치가 있으면 제거해야 합니다.[쿡의 거리로 이상치 진단]

[참고] 선형 회귀모형의 시각적 이해

아들의 키

잔차

아버지의 키

입력변수가 증가함에 따라 목표변수의 값도 지속적으로 증가하거나 감소해야 합니다.

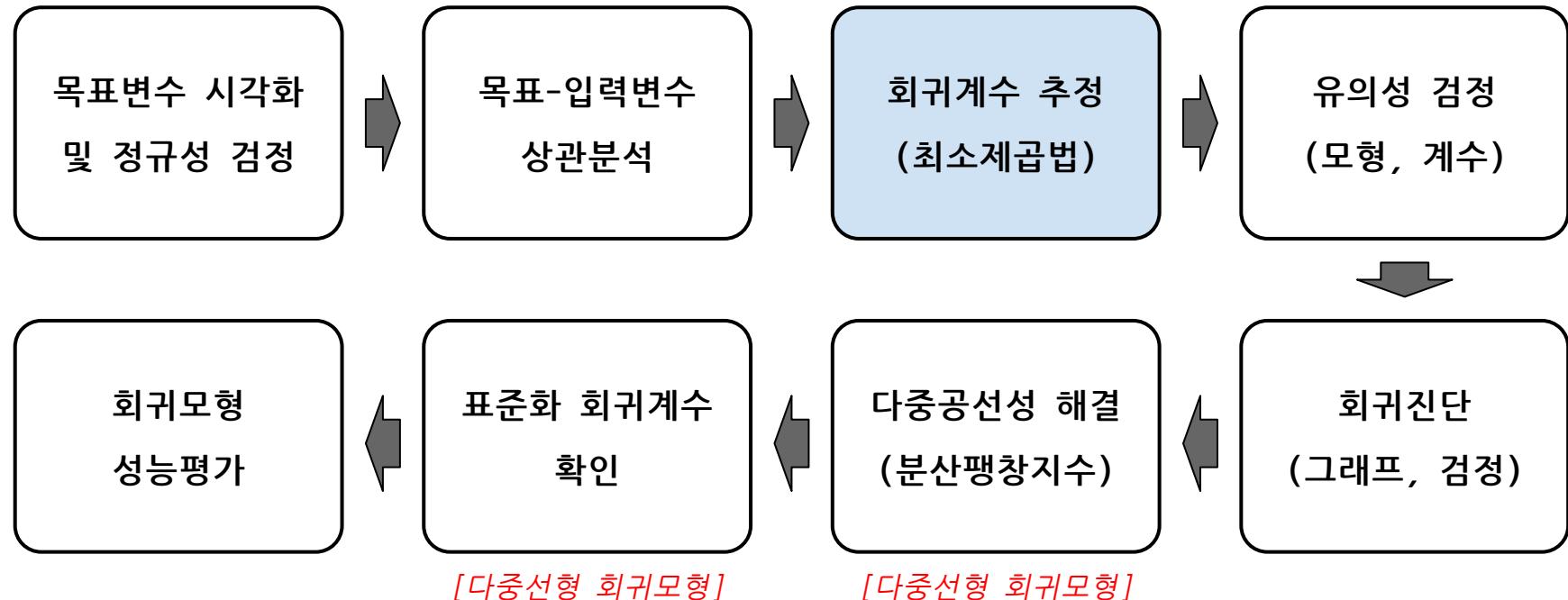
회귀직선은 입력변수의 값이 변함에 따라 목표변수의 평균을 이은 직선입니다.

어떤 입력변수 값에서 목표변수 실제값(회색점)과 회귀직선 위 추정값(빨간점) 간 차이를 잔차라고 합니다.

잔차는 평균이 0, 표준편차가 σ 인 정규분포를 따라야 합니다.(정규성, 등분산성)

만약 잔차가 어떤 패턴을 보이면 입력변수를 추가해야 합니다.(선형성)

선형 회귀분석 프로세스



관련 라이브러리 호출

- 관련 라이브러리를 호출합니다.

```
>>> import os, joblib
```

```
>>> import numpy as np
```

```
>>> import pandas as pd
```

- 실수를 출력할 소수점 자리수를 설정합니다.

```
>>> %precision 3
```

```
>>> pd.options.display.precision = 3
```

관련 라이브러리 호출(계속)

- 통계 관련 라이브러리를 호출합니다.

```
>>> from scipy import stats
```

```
>>> import pingouin as pg
```

- 시각화 및 통계 분석 관련 모듈을 호출합니다.

```
>>> from GraphicSetting import *
```

```
>>> import HelloDataScience as hds
```

작업 경로 확인 및 변경

- 현재 작업 경로를 확인합니다.

```
>>> os.getcwd()
```

- data 폴더로 작업 경로를 변경합니다.

```
>>> os.chdir(path = '../data')
```

- 현재 작업 경로에 있는 폴더명과 파일명을 출력합니다.

```
>>> os.listdir()
```

실습 데이터셋 준비

- z 파일을 읽고 데이터프레임 df를 생성합니다.

```
>>> df = joblib.load(io = 'Used_Cars_Price_Prep.z')
```

- df의 정보를 확인합니다.

```
>>> df.info()
```

- df의 처음 5행을 출력합니다.

```
>>> df.head()
```

- y절편 역할을 수행할 상수 1을 df의 두 번째 열로 삽입합니다.

```
>>> df.insert(loc = 1, column = 'const', value = 1)
```

범주형 입력변수의 더미변수 변환

- 범주형 입력변수는 범주 개수 - 1개의 더미변수로 변환합니다.
 - 왜냐하면 범주 개수만큼 더미변수를 생성하면 다중공선성 문제가 발생하기 때문입니다.
- 이번 예제에서는 FuelType으로 더미변수 Petrol을 생성합니다.

FuelType	Diesel	Petrol
Diesel	1	0
Petrol	0	1

FuelType이 'Diesel'일 때 더미변수는 0이므로 목표변수 추정값에 영향을 미치지 않지만 'Petrol'일 때는 회귀계수만큼 추정값에 영향을 미칩니다.

회귀계수가 양수면 'Petrol' 차량의 가격이 'Diesel'에 비해 평균적으로 회귀계수만큼 높다는 것을 의미합니다.

- MetColor도 범주형 입력변수지만 원소가 '0' 또는 '1'인 문자형이므로 정수형으로 변환하면 더미변수와 같은 형태입니다.

더미변수 생성

- 범주형 입력변수로 더미변수를 생성합니다.

```
>>> df = pd.get_dummies(data = df, columns = ['FuelType'], drop_first = True)
```

- df의 처음 10행을 출력합니다.

```
>>> df.head(n = 10)
```

- 더미변수명을 변경합니다.

```
>>> df = df.rename(columns = {'FuelType_Petrol': 'Petrol'})
```

- df의 열별 자료형을 확인합니다.

```
>>> df.dtypes # [참고] 더미변수의 자료형은 numpy.uint8(부호 없는 8비트 정수)입니다.
```

더미변수 생성(계속)

- 더미변수로 변환할 문자형 열이름으로 리스트를 생성합니다.

```
>>> cols = ['MetColor'] # [참고] 원소가 '0' 또는 '1'인 문자형 변수는 자료형을 정수형으로 변환해주어도  
더미변수로 변환한 것과 같은 결과를 얻을 수 있습니다.
```

- 지정한 변수를 정수형으로 일괄 변환합니다.

```
>>> df[cols] = df[cols].astype(np.uint8) # [참고] 'uint8'로 지정해도 같은 결과를 반환합니다.
```

- df의 열별 자료형을 확인합니다.

```
>>> df.dtypes
```

실습 데이터셋 분할

- 관련 라이브러리를 호출합니다.

```
>>> from sklearn.model_selection import train_test_split
```

- 전체 데이터셋의 70%를 훈련셋, 30%를 시험셋으로 분할합니다.

```
>>> trSet, teSet = train_test_split(df, test_size = 0.3, random_state = 0)
```

- 훈련셋의 목표변수 평균을 확인합니다.

```
>>> trSet['Price'].mean()
```

- 시험셋의 목표변수 평균을 확인합니다.

```
>>> teSet['Price'].mean()
```

입력변수와 목표변수 분리

- 목표변수명을 변수에 할당합니다.

```
>>> yvar = 'Price'
```

- 훈련셋을 목표변수 벡터와 입력변수 행렬로 분리합니다.

```
>>> trReal = trSet[yvar].copy()
```

```
>>> trSetX = trSet.drop(columns = [yvar])
```

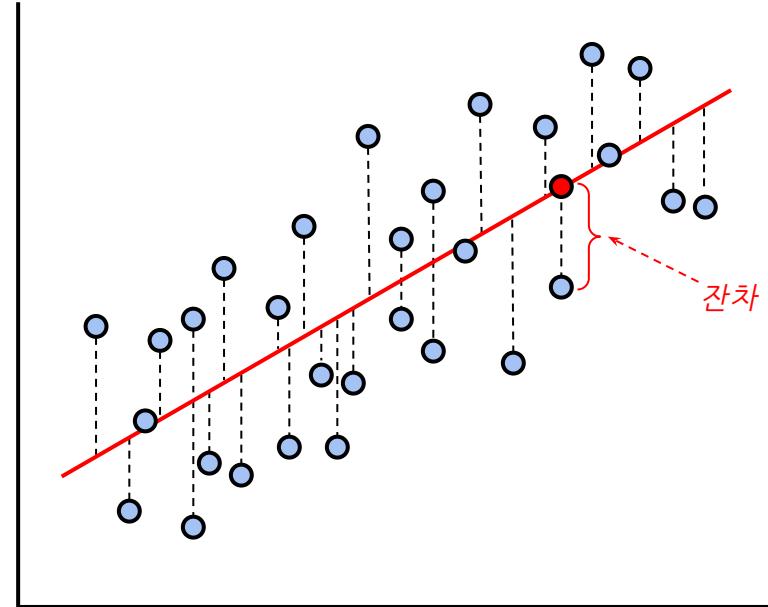
- 시험셋을 목표변수 벡터와 입력변수 행렬로 분리합니다.

```
>>> teReal = teSet[yvar].copy()
```

```
>>> teSetX = teSet.drop(columns = [yvar])
```

선형 회귀계수 추정: 최소제곱법

- x축에 입력변수, y축에 목표변수를 놓고 산점도를 그립니다.
 - 산점도는 상관관계를 시각화한 것입니다.
- 산점도에서 두 변수의 관계를 표현하는 회귀직선을 찾습니다.
$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$
- 회귀직선의 기울기와 y절편은 목표변수의 실제값과 추정값의 차이를 최소로 만드는 모델 파라미터입니다.



선형 회귀계수 추정: 최소제곱법(계속)

- 입력변수가 1개인 단순선형 회귀모형은 아래와 같습니다.

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

- 회귀계수를 추정하기 위해 최소제곱법(잔차 제곱합) 공식으로 변형합니다.

$$\sum (y_i - \hat{y}_i)^2 = \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$$

- 위 식을 각 회귀계수로 편미분한 방정식을 풁니다.

$$\frac{\partial}{\partial \beta_0} \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 = -2 \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

$$\frac{\partial}{\partial \beta_1} \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 = -2 \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) x_i = 0$$

선형 회귀계수 추정(계속)

- β_0 으로 편미분 방정식을 풀면 다음과 같습니다.

$$\frac{\partial}{\partial \beta_0} \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 = -2 \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

$$\sum y_i - \sum \hat{\beta}_0 - \sum \hat{\beta}_1 x_i = 0$$

$$n\hat{\beta}_0 = \sum y_i - \hat{\beta}_1 \sum x_i$$

$$\hat{\beta}_0 = \frac{1}{n} \left(\sum y_i - \hat{\beta}_1 \sum x_i \right) = \bar{y} - \hat{\beta}_1 \bar{x}$$

y 의 평균 - (기울기) x 의 평균

선형 회귀계수 추정(계속)

- β_1 으로 편미분 방정식을 풀면 다음과 같습니다.

$$\frac{\partial}{\partial \beta_1} \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 = -2 \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) x_i = 0$$

$$\sum x_i y_i = \hat{\beta}_0 \sum x_i + \hat{\beta}_1 \sum x_i^2$$

$$\hat{\beta}_1 \sum x_i^2 = \sum x_i y_i - \hat{\beta}_0 \sum x_i \quad \# \hat{\beta}_0 = \frac{1}{n} \left(\sum y_i - \hat{\beta}_1 \sum x_i \right) \text{을 대입하고 } \hat{\beta}_1 \text{으로 정리합니다.}$$

$$\hat{\beta}_1 = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2} = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} \quad \begin{array}{l} \# \text{ 분자: } x \text{와 } y \text{의 공분산} \\ \# \text{ 분모: } x \text{의 분산} \end{array}$$

위 식에서 분자와 분모를 n 으로 나누었을 때 분모는 $\sum x_i^2 - \frac{1}{n}(\sum x_i)^2 = \sum (x_i - \bar{x})^2$ 을 성립합니다.

선형 회귀모형의 유의성 검정

- 회귀모형이 의미를 가지려면 최소 한 개 이상의 회귀계수는 0이 아니어야 합니다.
 - 만약 모든 회귀계수가 0이면 회귀모형은 추정값으로 목표변수의 평균을 반환합니다.
 - 입력변수의 값이 바뀌어도 목표변수의 추정값은 바뀌지 않는다는 것을 의미합니다.
- 회귀모형의 유의성 검정은 분산분석을 실행합니다.
 - 귀무가설은 '모든 모회귀계수가 0이다'입니다.
 - 대립가설은 '최소한 하나 이상의 모회귀계수는 0이 아니다'입니다.
 - 선형 회귀모형의 총분산 SST 은 회귀제곱합 SSR 과 잔차제곱합 SSE 의 합입니다.
 - 평균회귀제곱합 MSR 을 평균잔차제곱합 MSE 로 나눈 F 통계량의 유의확률로 판단합니다.

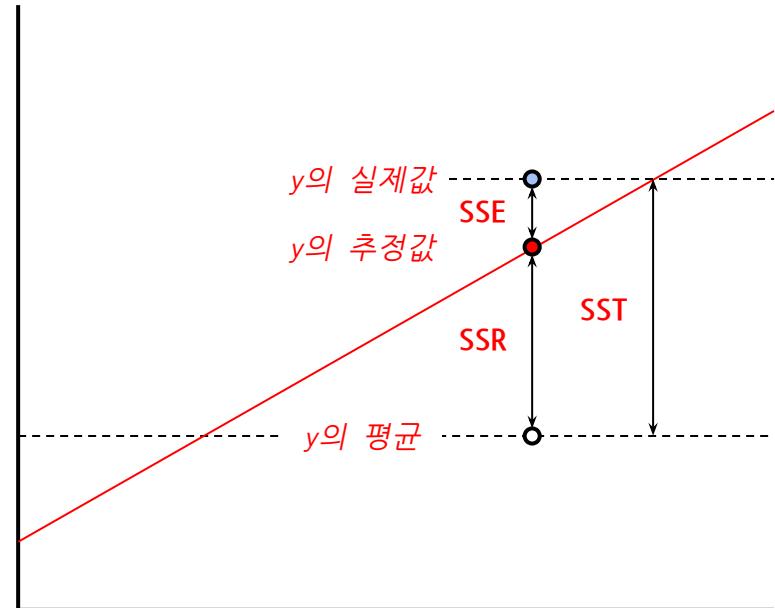
선형 회귀모형의 유의성 검정(계속)

- 회귀제곱합 SSR은 추정값과 평균 차이를 제곱하여 모두 더한 것입니다.

$$SSR = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$$

- 잔차제곱합 SSE은 실제값과 추정값 차이를 제곱하여 모두 더한 것입니다.

$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$



목표변수에 대한 입력변수의 설명력이 증가할수록
잔차가 감소하므로 SSR은 증가, SSE는 감소합니다.

선형 회귀모형의 유의성 검정(계속)

- 회귀모형의 유의성 검정통계량은 회귀제곱합과 잔차제곱합을 각각의 자유도로 나눈 평균제곱합의 비^{ratio}입니다.
 - 평균회귀제곱합^{MSR}은 회귀제곱합을 자유도인 p 로 나눈 것입니다. (p 는 입력변수 개수)
 - 평균잔차제곱합^{MSE}은 잔차제곱합을 자유도인 $n-p-1$ 로 나눈 것입니다. (n 은 행 개수)
- 회귀모형의 분산분석표는 아래와 같습니다.

구분	제곱합(SS)	자유도(df)	평균제곱(MS)	F-통계량	유의확률
모형(R)	SSR	p	$MSR = SSR \div p$	MSR / MSE	$p\text{-value}$
잔차(E)	SSE	$n-p-1$	$MSE = SSE \div (n-p-1)$		
전체(T)	SST	$n-1$			

선형 회귀모형의 유의성 검정(계속)

- 선형 회귀모형을 적합했다면 가장 먼저 확인해야 할 것은 F-통계량의 유의확률이 유의수준(α)보다 작은지 여부입니다.
- F-통계량의 유의확률이 유의수준 0.05 보다 작으면 '모든 회귀계수가 0'이라는 귀무가설을 기각할 수 있으므로 회귀모형에 있는 회귀계수 중 최소한 하나는 0이 아니라고 판단합니다.
- 만약 F-통계량의 유의확률이 유의수준 0.05 보다 크면 모든 회귀계수가 0이라고 판단할 수 있으므로 회귀모형을 사용할 수 없습니다.
- 관측값(행)을 늘리거나 입력변수(열)를 추가하고 회귀모형을 다시 적합합니다.

선형 회귀계수의 유의성 검정

- 회귀모형에서 어떤 입력변수가 의미를 가지려면 회귀계수가 0이 아니어야 합니다.
 - 만약 어떤 입력변수의 회귀계수가 0이면 입력변수의 값이 바뀌어도 목표변수의 추정값은 바뀌지 않는다는 것을 의미합니다.
- 회귀계수의 유의성 검정은 t-검정으로 실행합니다.
 - 귀무가설은 '모회귀계수(β_j)가 0이다'입니다.
 - 대립가설은 '모회귀계수(β_j)는 0이 아니다'입니다.
 - 선형 회귀모형에서 추정한 회귀계수로 모회귀계수가 0인지 여부를 판단합니다.
 - 유의확률이 유의수준 0.05 보다 작으면 모회귀계수는 0이 아니라고 판단합니다.

선형 회귀계수의 유의성 검정(계속)

- 선형 회귀계수의 t-검정통계량은 다음과 같습니다.

$$t = \frac{b_j - \beta_j}{s_{b_j}}$$

b_j 는 회귀모형에서 추정한 회귀계수입니다.
귀무가설에서 모회귀계수(β_j)를 0으로 가정합니다.
s_{b_j} 는 회귀계수의 표준오차입니다.

- 선형 회귀계수의 표준오차는 다음과 같이 추정합니다.

$$s_{b_j} = \frac{s_\epsilon}{\sqrt{\sum(x_i - \bar{x})^2}}$$

분자는 회귀모형 잔차의 표준오차입니다.

$$s_\epsilon = \sqrt{\frac{\text{SSE}}{n - p - 1}} = \sqrt{\text{MSE}}$$

잔차의 표준오차는 실제값과 추정값이 서로 가까울수록 작아집니다.
[참고] MSE 또는 SSE가 감소하면 회귀계수의 표준오차는 감소하고 t-검정통계량의 절대값은 커지므로 결과적으로 유의확률이 감소합니다.

결정계수

- 결정계수^{R-squared}는 선형 회귀모형의 적합도^{goodness of fit}를 나타내는 지표입니다.
 - 결정계수는 총변동^{SST}에서 회귀모형에 의해 설명할 수 있는 변동^{SSR}의 비율을 의미합니다.

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

- 결정계수는 0~1의 값을 가지며 1에 가까울수록 적합도가 좋은 모형입니다.
- 단순선형 회귀모형의 결정계수는 입력변수와 목표변수 간 피어슨 상관계수를 제곱입니다.
- 따라서 입력변수와 목표변수 간 강한 상관관계를 가질수록 결정계수는 증가합니다.

[참고] 단순선형 회귀모형의 결정계수

- 단순선형 회귀모형의 결정계수가 두 변수의 상관계수 제곱임을 증명합니다.

$$\begin{aligned} \text{SSR} &= \sum (\hat{y}_i - \bar{y})^2 = \sum (\hat{\beta}_0 + \hat{\beta}_1 x_i - \bar{y})^2 \\ &= \sum (\bar{y} - \hat{\beta}_1 \bar{x} + \hat{\beta}_1 x_i - \bar{y})^2 = \sum (\hat{\beta}_1 x_i - \hat{\beta}_1 \bar{x})^2 \end{aligned}$$

$$\hat{\beta}_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}, \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

$$r_{xy} = \frac{\frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{n-1}}{\sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}} \sqrt{\frac{\sum (y_i - \bar{y})^2}{n-1}}}$$

$$\begin{aligned} R^2 &= \frac{\text{SSR}}{\text{SST}} = \frac{\hat{\beta}_1^2 \sum (x_i - \bar{x})^2}{\sum (y_i - \bar{y})^2} \\ &= \frac{\left[\sum (x_i - \bar{x})(y_i - \bar{y}) \right]^2}{\left[\sum (x_i - \bar{x})^2 \right]^2} \frac{\sum (x_i - \bar{x})^2}{\sum (y_i - \bar{y})^2} \end{aligned}$$

$$= \frac{\left[\sum (x_i - \bar{x})(y_i - \bar{y}) \right]^2}{\left[\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2 \right]} = r_{xy}^2$$

조정된 결정계수

- 선형 회귀모형의 결정계수는 입력변수의 개수가 많아질수록 계속 증가합니다.
 - 최적의 회귀모형을 선택하는 기준으로 결정계수를 사용하지 않습니다.
- 조정된 결정계수^{adjusted R-squared}는 입력변수의 개수로 벌점^{penalty}을 부여합니다.

$$\text{adj.}R^2 = 1 - \frac{(n - 1)(1 - R^2)}{n - p - 1}$$

n : 훈련셋의 행(관측값) 개수
 p : 훈련셋의 열(입력변수) 개수

- 조정된 결정계수는 0~1의 값을 가지며, 분수가 0으로 수렴하면 1에 가까워집니다.
- 입력변수를 추가하면 분모는 감소하고, 결정계수 증가로 분자도 감소합니다.
- 만약 목표변수와 상관계수가 낮은 입력변수를 추가하면 결정계수가 증가하지 않으므로 분자는 고정인데 분모가 감소했기 때문에 조정된 결정계수는 오히려 감소합니다.

선형 회귀모형 적합 함수

- statsmodels 라이브러리에 선형 회귀모형을 적합하는 두 가지 함수를 소개합니다.
 - `statsmodels.api.OLS()` 함수에 목표변수와 입력변수를 각각 지정합니다.
 - `statsmodels.formula.api.ols()` 함수에 '목표변수 ~ 입력변수' 관계식을 문자열로 지정합니다. 입력변수가 여러 개일 때 입력변수 사이에 '+' 기호를 추가합니다.
- 선형 회귀모형을 적합할 때 `statsmodels.api.OLS()` 함수를 사용합니다.
 - endog: 목표변수를 지정합니다.
 - exog: 입력변수를 지정합니다. 함수에 지정하기 전에 상수 1을 추가해야 합니다.

선형 회귀모형 적합 함수 생성

- 관련 라이브러리를 호출합니다.

```
>>> import statsmodels.api as sa
```

- 선형 회귀모형을 반환하는 함수를 생성합니다.

```
>>> def ols(y, X):  
    model = sa.OLS(endog = y, exog = X)  
    return model.fit()
```

선형 회귀모형 적합 및 결과 확인

- 훈련셋으로 선형 회귀모형을 적합하고 결과를 확인합니다.

```
>>> fit1 = ols(y = trReal, X = trSetX)
```

```
>>> fit1.summary() # Prob(F-statistic), 회귀계수의 P>|t|, Adj.R-squared를 차례대로 확인합니다.
```

Dep. Variable:	Price	R-squared:	0.733	coef	std err	t	P> t	[0.025	0.975]
Model:	OLS	Adj. R-squared:	0.730	const	-4832.6730	2260.499	-2.138	0.033	-9269.362 -395.984
Method:	Least Squares	F-statistic:	339.3	Age	-94.4704	2.961	-31.902	0.000	-100.283 -88.658
Date:	Sun, 26 Jun 2022	Prob (F-statistic):	2.57e-243	KM	-0.0170	0.001	-12.892	0.000	-0.020 -0.014
Time:	13:21:49	Log-Likelihood:	-7326.4	HP	2.2961	4.971	0.462	0.644	-7.460 12.052
No. Observations:	875	AIC:	1.467e+04	MetColor	14.4447	75.854	0.190	0.849	-134.434 163.324
Df Residuals:	867	BIC:	1.471e+04	Doors	-5.0942	46.181	-0.110	0.912	-95.733 85.545
Df Model:	7			Weight	18.9523	2.256	8.401	0.000	14.524 23.380
Covariance Type:	nonrobust			Petrol	1283.4546	325.808	3.939	0.000	643.990 1922.919

회귀진단: 잔차 가정 확인

- fit1 모형의 잔차 가정을 확인합니다.

Omnibus 값이 0에 가까우면 정규분포한다고 판단합니다.

왜도가 0이고 첨도가 3이면 정규분포한다고 판단합니다.

Omnibus: 54.450	Durbin-Watson: 1.962
Prob(Omnibus): 0.000	Jarque-Bera (JB): 140.503
Skew: -0.314	Prob(JB): 3.09e-31
Kurtosis: 4.860	Cond. No. 5.13e+06

Durbin-Watson 값이 1~3이면 독립성 가정을 만족합니다.

Jarque-Bera 값이 크면 정규성 가정을 만족하지 못합니다.

Condition number가 큰 값이면 다중공선성을 의심합니다.

- fit1 모형 잔차의 등분산성 검정을 실행합니다.

```
>>> hds.breushpagan(model = fit1)
```

Breush-Pagan 검정으로 회귀모형 잔차의 이분산성을 확인합니다.
F-통계량 유의확률이 유의수준 0.05 보다 크면 등분산성 가정을 만족하는 것으로 판단합니다.

회귀진단: 잔차 그래프

- 회귀모형의 잔차 그래프를 출력하고 회귀진단을 실행합니다.
 - 선형성: 추정값과 잔차의 산점도에서 가로 점선에 빨간색 직선이 일치해야 합니다.
 - 정규성: Normal Q-Q 그래프에서 모든 점(잔차)들이 45도 점선 위에 있는지 확인합니다.
 - Normal Q-Q 그래프는 이상적인 잔차의 분위수와 표준화 잔차의 실제 분위수로 그린 것입니다.
 - 모든 점이 45도 점선에 있다면 두 잔차의 상관계수가 1에 가깝다는 의미입니다.
 - 등분산성: 추정값과 표준화 잔차 제곱근의 산점도에서 두께가 일정해야 합니다.
 - 산점도가 점점 커지는 나팔 모양, 점점 작아지는 깔때기 모양이면 이분산으로 판단합니다.
 - 영향점 진단: Cook's distance가 큰 관측값은 이상치로 판단할 수 있습니다.

회귀진단: 잔차 그래프

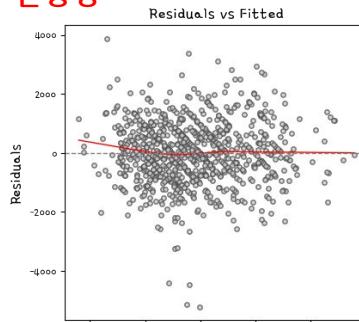
- fit1 모형의 잔차 그래프를 그립니다.

```
>>> hds.regressionDiagnosis(model = fit1)
```

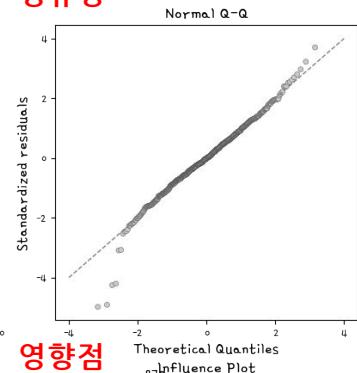
- 오른쪽 그래프로 잔차 가정을 확인합니다.

- 선형성: 빨간 직선에서 패턴이 없어야 합니다.
- 정규성: 모든 점이 대각선 위에 있어야 합니다.
- 등분산성: 두께(높이)가 일정해야 합니다.
- 영향점: 표준화 잔차와 레버리지 값이 큰 점이 없어야 합니다.

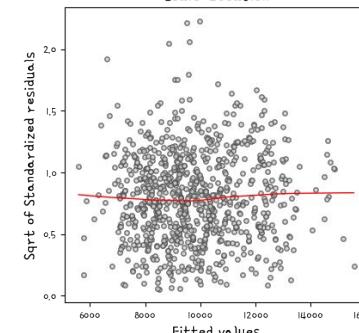
선형성



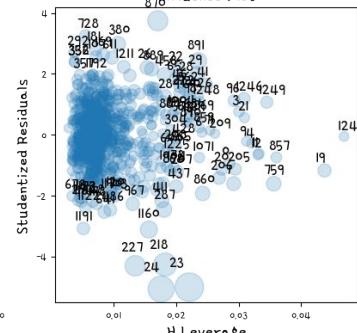
정규성



등분산성



영향점



회귀진단: 잔차의 정규성 검정

- Normal Q-Q 그래프를 그렸을 때 모든 점(잔차)들이 45도 점선 위에 있으면 잔차의 정규성 가정을 만족하는 것으로 판단합니다.
 - 오른쪽 위 꼬리가 올라가고 왼쪽 아래 꼬리가 내려간 것은 정규분포의 양 극단 관측값이 이상적인 분포에 비해 많다는 것을 의미합니다.
- 잔차의 히스토그램을 그려서 분포를 확인합니다.

```
>>> sns.histplot(x = fit1.resid, bins = 50, stat = 'density');
```

- 잔차의 정규성 검정을 실행합니다.

```
>>> stats.shapiro(x = fit1.resid) # 유의확률이 유의수준 0.05 보다 크면 잔차가 정규분포한다고  
판단합니다.
```

회귀진단: 잔차의 독립성 검정

- 잔차의 독립성 가정은 잔차끼리 상관관계가 없어야 한다는 것을 의미합니다.
 - 잔차끼리 상관관계가 있으면 유의성 검정통계량과 결정계수를 과대추정할 수 있습니다.
- 대표적인 잔차의 독립성 문제는 시계열 자료를 회귀분석으로 다룰 때 나타납니다.
 - 이전 시점의 잔차항이 다음 시점의 잔차항에 영향을 미칠 수 있습니다.
 - 시차를 두고 잔차항이 상관관계를 보이는 것을 자기상관^{autocorrelation}이라고 합니다.
- 자기상관 여부를 확인할 수 있는 방법은 더빈-왓슨^{Durbin-Watson} 검정입니다.
 - 더빈-왓슨 검정통계량은 0~4의 값을 가지며 2에 가까우면 자기상관이 없고, 0 또는 4에 가까우면 자기상관이 있는 것으로 판단합니다. [참고] 유의확률은 확인하지 않습니다.

[참고] 더빈-왓슨 검정

- 선형 회귀모형을 적합할 때 표본은 같은 모집단으로부터 독립적으로 추출되어야 한다는 i.i.d 조건을 만족해야 합니다.
 - i.i.d는 independent and identically distributed의 머리글자로 독립 항등분포입니다.
 - 횡단면 ^{cross-sectional} 데이터와 다르게 시계열 ^{time-series} 데이터의 경우 이전 관측값의 영향력을 배제하기 어려우므로 lag 1 시점의 잔차와 상관관계를 확인해야 합니다.
- 더빈-왓슨 검정통계량(d)은 $t-1$ 시점과 t 시점의 잔차 간 상관관계를 계산합니다.
 - 더빈-왓슨 표에서 관측값 및 입력변수와 유의수준으로 dL 과 dU 를 찾고 검정통계량(d)이 $dL \sim dU$ 에 있으면 잔차 간 상관관계가 없다는 귀무가설을 기각할 수 없습니다.

$$\hat{\rho} = \frac{\sum_{t=2}^n e_t e_{t-1}}{\sum_{t=1}^n e_t^2}, \quad d = \frac{\sum_{t=2}^n (e_t - e_{t-1})^2}{\sum_{t=1}^n e_t^2} = \frac{\sum_{t=2}^n (e_t^2 + e_{t-1}^2 - 2e_t e_{t-1})}{\sum_{t=1}^n e_t^2} \cong 2 - 2\hat{\rho} = 2(1 - \hat{\rho})$$

[참고] 영향점 판단 기준

구분	세부 내용
스튜던트 잔차	<ul style="list-style-type: none">- 잔차는 목표변수의 실제값과 추정값의 차이가 클수록 큰 값을 갖습니다.- 잔차의 크기는 입력변수에 영향을 받으므로, 입력변수의 영향을 제거하려면 표준화해야 합니다.- 스튜던트 잔차는 잔차를 표준편차와 레버리지로 나눈 것입니다.- 스튜던트 잔차가 2~4보다 크면 이상치로 판단합니다.
레버리지	<ul style="list-style-type: none">- 레버리지는 목표변수의 실제값이 추정값에 미치는 영향을 나타낸 값인데, 목표변수의 추정값은 Hat Matrix의 대각원소와 실제값의 내적으로 계산합니다.- 레버리지는 입력변수의 평균에서 멀어질수록 큰 값을 가지며, 회귀모형에 영향을 미칩니다.- 레버리지의 합계는 회귀모형의 입력변수 개수(p)와 같으므로, 평균은 p/n과 같습니다.- 레버리지가 평균의 2~4배일 때 영향점으로 판단합니다.
쿡의 거리	<ul style="list-style-type: none">- 잔차와 레버리지를 모두 고려하여 이상치를 판단하는 지표가 쿡의 거리입니다.- 쿡의 거리가 $4/n$(행 개수)보다 큰 관측값 중에서 이상치 여부를 확인합니다.

[참고] 영향점 확인

- 훈련셋의 관측값마다 영향점 정보를 갖는 데이터프레임을 생성합니다.

```
>>> aug = hds.augment(model = fit1)
```

```
>>> aug.head() # 목표변수 실제값, 추정값(fitted), 잔차(resid), 레버리지(hat), 해당 관측값을 제거한 모형의  
잔차 표준편차 추정값(sigma), 쿡의 거리(cooksd) 및 표준화잔차(std_resid)를 출력합니다.
```

- 스튜던트 잔차의 절대값이 3을 초과하는 행 개수를 확인합니다.

```
>>> aug['std_resid'].abs().gt(3).sum()
```

- 레버리지 평균의 3배를 초과하는 행 개수를 확인합니다.

```
>>> hatAvg = aug['hat'].mean(); hatAvg
```

```
>>> aug['hat'].gt(hatAvg * 3).sum()
```

훈련셋에서 이상치 제거

- 쿠의 거리가 $4/n$ (행 개수)를 초과하는 행 개수를 확인합니다.

```
>>> n = trSet.shape[0] # 훈련셋의 행 개수 n을 생성합니다.
```

```
>>> locs = aug['cooksd'].gt(4/n) # 쿠의 거리가  $4/n$ (행 개수)를 초과하면 True, 미만이면 False인 원소를 갖는 부울형 시리즈 locs를 생성합니다.
```

```
>>> locs.sum() # 쿠의 거리가  $4/n$ (행 개수)를 초과하는 원소 개수를 확인합니다.
```

- 훈련셋에서 locs가 True(이상치)인 행을 제거합니다.

```
>>> trSetX = trSetX.loc[~locs] # 훈련셋의 입력변수 행렬에서 locs가 False인 원소를 남깁니다.  
[참고] ~ 기호는 논리부정이므로 True/False를 반전합니다.
```

```
>>> trReal = trReal.loc[~locs] # 훈련셋의 목표변수 벡터에서 locs가 False인 원소를 남깁니다.
```

```
>>> trSetX.shape[0] # 이상치를 제거한 훈련셋의 행 개수를 확인합니다.
```

선형 회귀모형 재적합 및 결과 확인

- 이상치를 제거한 훈련셋으로 선형 회귀모형을 적합합니다.

```
>>> fit2 = ols(y = trReal, X = trSetX)
```

- fit2 모형의 적합 결과를 확인합니다.

```
>>> fit2.summary()
```

- fit2 모형 잔차의 등분산성 검정을 실행합니다.

```
>>> hds.breushpagan(model = fit2)
```

- fit2 모형의 잔차 그래프를 그립니다.

```
>>> hds.regressionDiagnosis(model = fit2)
```

[참고] 더미변수의 시각적 이해

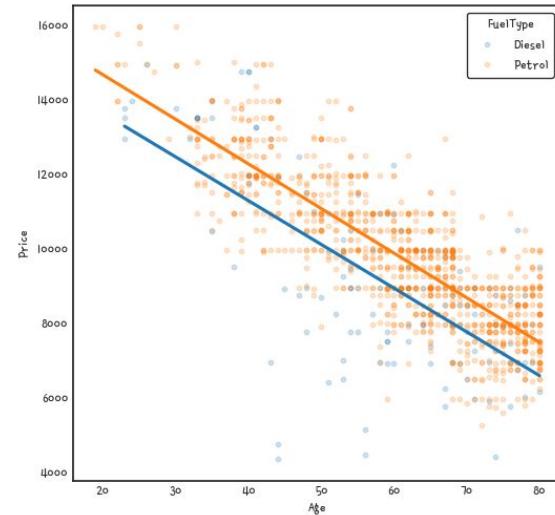
- Age와 Price의 회귀직선은 FuelType에 따라 달라집니다.

```
>>> labels = ['Diesel', 'Petrol']

>>> for i, v in enumerate(labels):

    sns.regplot(data = df[df['Petrol'].eq(i)],
                 x = 'Age', y = 'Price',
                 ci = None, label = v,
                 scatter_kws = dict(s = 10, alpha = 0.2))

>>> plt.legend(loc = 'best', title = 'FuelType');
```



목표변수의 추정값 생성

- 훈련셋으로 fit1과 fit2 모형의 추정값을 생성하고 실제값과 비교합니다.

```
>>> trPred1 = fit1.predict(exog = trSetX)
```

```
>>> trPred2 = fit2.predict(exog = trSetX) # fit1보다 fit2의 추정값이 실제값에 더 가깝습니다.
```

```
>>> pd.DataFrame(data = {'Real': trReal, 'Pred1': trPred1, 'Pred2': trPred2})
```

- 시험셋으로 fit1과 fit2 모형의 추정값을 생성하고 실제값과 비교합니다.

```
>>> tePred1 = fit1.predict(exog = teSetX)
```

```
>>> tePred2 = fit2.predict(exog = teSetX)
```

```
>>> pd.DataFrame(data = {'Real': teReal, 'Pred1': tePred1, 'Pred2': tePred2})
```

선형 회귀모형 성능 평가

- 훈련셋으로 fit1과 fit2 모형의 성능지표를 출력합니다.

```
>>> hds.regmetrics(y_true = trReal, y_pred = trPred1)
```

```
>>> hds.regmetrics(y_true = trReal, y_pred = trPred2)
```

- 시험셋으로 fit1과 fit2 모형의 성능지표를 출력합니다.

```
>>> hds.regmetrics(y_true = teReal, y_pred = tePred1)
```

```
>>> hds.regmetrics(y_true = teReal, y_pred = tePred2)
```

[참고] 회귀모형 성능지표 관련 함수

- 관련 라이브러리를 호출합니다.

```
>>> from sklearn import metrics
```

- 회귀모형 성능지표를 출력합니다. # [참고] metrics 모듈에는 RMSE와 RMSLE를 반환하는 함수는 없으므로 MSE와 MSLE의 제곱근을 계산합니다.

```
>>> metrics.mean_squared_error(y_true = teReal, y_pred = tePred2)
```

```
>>> metrics.mean_squared_log_error(y_true = teReal, y_pred = tePred2)
```

```
>>> metrics.mean_absolute_error(y_true = teReal, y_pred = tePred2)
```

```
>>> metrics.mean_absolute_percentage_error(y_true = teReal, y_pred = tePred2)
```

다중공선성

- 현실세계에서 관찰하는 데이터에는 정도의 차이가 있지만 입력변수 간 상관관계를 피할 수 없습니다.
- 입력변수 간 강한 상관관계가 있으면 다중공선성 Multicollinearity 문제가 발생합니다.
 - 다중선형 회귀계수는 다른 입력변수를 고정한 상태에서 해당 입력변수가 1단위 증가할 때마다 목표변수에 미치는 크기를 의미합니다.
 - 상관관계가 높은 입력변수로 다중선형 회귀모형을 적합하면 목표변수에 대한 설명력을 입력변수가 나누어 가지므로 회귀계수의 절대값과 부호가 달라질 수 있습니다.
 - 또한 회귀계수의 표준오차가 증가하므로 회귀계수의 설명력도 낮아집니다.
 - 결국 회귀모형을 해석할 수 없는 문제가 발생합니다.

분산팽창지수

- 다중선형 회귀모형에서 사용한 p 개의 입력변수마다 다른 입력변수에 종속되었는지 여부를 판단할 때 분산팽창지수 Variance Inflation Factor를 계산합니다.

- j 번째 입력변수를 목표변수, 나머지 $p-1$ 개를 입력변수로 설정하여 회귀모형을 적합하고 반환되는 결정계수로 j 번째 입력변수의 분산팽창지수를 계산합니다.

$$vif_j = \frac{1}{1 - R_j^2}$$

- 만약 j 번째 입력변수를 목표변수로 설정한 회귀모형의 결정계수가 0.9이면 위 공식으로 계산한 분산팽창지수는 10입니다.
- 분산팽창지수가 10 이상이면 다중공선성 입력변수로 판단하고 삭제합니다.
 - [참고] 다중공선성을 판단하는 절대적인 기준은 아닙니다.

다중공선성 확인

- 분산팽창지수를 출력하고 다중공선성 입력변수를 확인합니다.

```
>>> hds.vif(X = trSetX)
```

- **vif()** 함수 실행 결과, 분산팽창지수가 10 이상인 입력변수가 없으므로 이번 예제에서는 다중공선성 입력변수는 없는 것으로 판단합니다.
- 만약 분산팽창지수가 10 이상인 입력변수가 다수일 때 분산팽창지수가 가장 큰 입력변수 하나만 훈련셋에서 삭제하고 회귀모형을 다시 적합합니다.
- 새로 적합한 회귀모형의 분산팽창지수를 확인합니다.
- 분산팽창지수가 10 이상인 변수가 없을 때까지 위 과정을 반복합니다.

다중공선성 입력변수 제거

- 다중공선성 입력변수가 있다고 가정하고 훈련셋에서 삭제합니다.

```
>>> trSetX1 = trSetX.drop(columns = ['Petrol'])
```

- 입력변수별 분산팽창지수를 다시 출력합니다.

```
>>> hds.vif(X = trSetX1) # [참고] 모든 입력변수의 분산팽창지수가 감소했습니다.
```

변수선택법

- 입력변수가 p 개일 때 모든 경우의 수를 고려하여 성능이 우수한 최종 회귀모형을 선택한다고 가정하면 총 2^p 개의 후보모형을 생성해야 합니다.
 - 예를 들어 입력변수가 세 개(x_1, x_2, x_3)일 때 가능한 조합은 다음과 같습니다.
 - $y = \underline{x_1}, y = \underline{x_2}, y = \underline{x_3}, y = \underline{x_1 + x_2}, y = \underline{x_1 + x_3}, y = \underline{x_2 + x_3}, y = \underline{x_1 + x_2 + x_3}$
(모든 입력변수를 추가한 Full 모형)과 $\underline{y = \beta_0}$ (입력변수 없는 Null 모형)이 있습니다.
- 입력변수 개수가 많아질수록 모든 후보모형을 고려하는 것이 어렵습니다.
 - 입력변수 개수가 10개일 때 가능한 조합은 1024개로 증가합니다.
- 따라서 전진선택법 forward selection, 후진소거법 backward elimination, 단계적방법 stepwise method 을 사용하여 여러 후보모형 중에서 AIC가 가장 작은 모형을 선택합니다.

최적의 회귀모형 선택 기준: AIC

- AIC^{Akaike Information Criteria}는 로그 SSE(잔차제곱합)에 $2p$ (벌점)를 더한 값입니다.

$$AIC = n \ln \left(\frac{SSE}{n} \right) + 2p$$

- AIC의 특징은 아래와 같습니다.
 - 목표변수와 상관계수가 큰 입력변수를 추가하면 실제값과 추정값의 차이가 작아지므로 SSE(잔차제곱합)은 감소하는데, $2p$ (벌점)보다 감소량이 크면 AIC는 감소합니다.
 - 목표변수와 상관계수가 낮거나 상관관계가 없는 입력변수를 추가하면 SSE(잔차제곱합)은 거의 감소하지 않는데, $2p$ (벌점)보다 감소량이 작으면 AIC는 오히려 증가합니다.
 - 따라서 AIC가 작을수록 좋은 모형이라고 판단합니다.

전진선택법

- 전진선택법은 입력변수를 하나씩 추가하면서 최적의 회귀모형을 탐색합니다.
 - Null 모형에서 입력변수를 하나씩 추가한 후보모형을 적합한 다음, Null 모형보다 AIC가 작은 후보모형 중에서 AIC가 최소인 후보모형을 선택합니다.
 - 이전 단계에서 선택한 회귀모형으로 위 과정을 반복하면서 Full 모형까지 전진합니다.
 - 이전 단계에서 선택한 회귀모형보다 AIC가 작은 후보모형이 없으면 중단합니다.
- 전진선택법의 장점과 단점은 다음과 같습니다.
 - 장점: 입력변수가 p 개일 때 최대 $\frac{p(p+1)}{2}$ 개의 후보모형을 적합합니다.
 - 단점: 한 번 선택한 입력변수를 제거하지 않습니다.

후진소거법

- 후진소거법은 입력변수를 하나씩 제거하면서 최적의 회귀모형을 탐색합니다.
 - Full 모형에서 입력변수를 하나씩 제거한 후보모형을 적합한 다음, Full 모형보다 AIC가 작은 후보모형 중에서 AIC가 최소인 후보모형을 선택합니다.
 - 이전 단계에서 선택한 회귀모형으로 위 과정을 반복하면서 Null 모형까지 후진합니다.
 - 이전 단계에서 선택한 회귀모형보다 AIC가 작은 후보모형이 없으면 중단합니다.
- 후진소거법의 장점과 단점은 다음과 같습니다.
 - 장점: 입력변수가 p 개일 때 최대 $\frac{p(p+1)}{2}$ 개의 후보모형을 적합합니다.
 - 단점: 한 번 제거한 입력변수를 추가하지 않습니다.

단계적 방법

- 단계적 방법은 전진선택법과 후진소거법의 단점을 보완한 방법입니다.
 - 시작 모형과 범위(Null 모형 ~ Full 모형)를 지정합니다.
 - 시작 모형에서 입력변수를 하나씩 추가한 후보모형과 하나씩 제거한 후보모형을 적합하고 시작 모형보다 AIC가 작은 후보모형 중에서 AIC가 최소인 후보모형을 선택합니다.
 - 이전 단계에서 선택한 회귀모형보다 AIC가 작은 후보모형이 없으면 중단합니다.
- 단계적 방법의 장점과 단점은 다음과 같습니다.
 - 장점: 전진선택법과 후진소거법보다 성능이 좋은 회귀모형을 탐색할 가능성이 있습니다.
 - 단점: 전진선택법과 후진소거법보다 적합해야 할 후보모형 개수가 많습니다.

단계적방법으로 선형 회귀모형 적합

- 단계적방법으로 선형 회귀모형을 적합합니다.

```
>>> fit3 = hds.stepwise(y = trReal, X = trSetX, direction = 'both')
```

direction 매개변수에 변수선택법 방향('forward',
'backward', 'both')을 문자열로 지정합니다.

- fit3 모형의 적합 결과를 확인합니다.

```
>>> fit3.summary()
```

- fit3 모형 잔차의 등분산성 검정을 실행합니다.

```
>>> hds.breushpagan(model = fit3)
```

표준화 회귀계수

- 입력변수가 두 개 이상인 다중선형 회귀모형은 입력변수별 회귀계수를 추정하지만 회귀계수로는 입력변수의 상대적인 영향력을 비교할 수 없습니다.
 - 왜냐하면 입력변수 간 척도가 서로 다르기 때문입니다.
- 표준화 회귀계수^{Standardized Regression Coefficient}로 목표변수에 대한 입력변수의 상대적인 영향력을 비교합니다.
- 표준화 회귀계수의 절대값이 클수록 목표변수에 대한 영향력이 크다고 판단합니다.
 - 표준화 회귀계수의 부호는 방향을 의미할 뿐 상대적인 영향력과는 무관합니다.

$$\beta_j^* = \beta_j \times \frac{\sigma_{x_i}}{\sigma_y}$$

표준화 회귀계수 확인

- fit3 모형의 회귀계수를 출력합니다.

```
>>> fit3.params
```

- 표준화 회귀계수를 생성합니다.

```
>>> beta_z = hds.std_coefs(model = fit3)
```

```
>>> beta_z
```

- 표준화 회귀계수의 절대값을 오름차순 정렬한 결과를 출력합니다.

```
>>> beta_z.abs().sort_values()
```

회귀모형 성능 평가

- 시험셋으로 fit3 모형의 추정값을 생성합니다.

```
>>> tePred3 = fit3.predict(exog = teSetX)
```

- 시험셋으로 fit3 모형의 성능지표를 출력합니다.

```
>>> hds.regmetrics(y_true = teReal, y_pred = tePred3)
```

- 시험셋으로 fit2 모형의 성능지표와 비교합니다.

```
>>> hds.regmetrics(y_true = teReal, y_pred = tePred2)
```

End of Document