

Jin-Soo Kim
[\(jinsoo.kim@snu.ac.kr\)](mailto:jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022

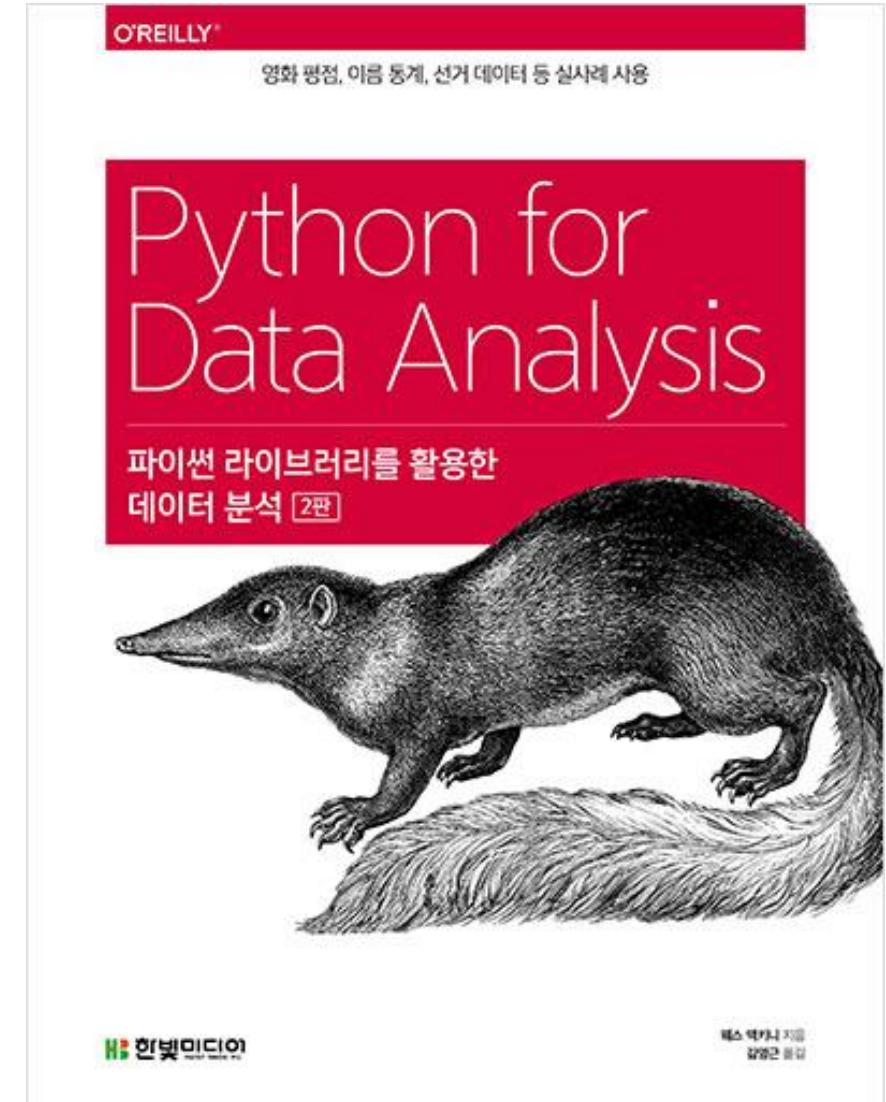


Python for Data Analytics

Python Basics

교재

- 파이썬 라이브러리를 활용한 데이터 분석
(2판)
- 김영근 옮김
- 한빛미디어, 2019.
- Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython (2nd Ed.)
- By Wes McKinney
- O'Reilly Media, 2017



일정 (I주차)

		1/3 (Mon)	1/4 (Tue)	1/5 (Wed)	1/6 (Thu)	1/7 (Fri)
오전	8:30	확률통계	확률통계	확률통계	확률통계	확률통계
	9:30					
	10:30					
	11:30					
	12:30	점심 시간 (12:30 ~ 1:30)				
오후	1:30	Python Basics	Python Sequence and Collections	NumPy I	NumPy II	Pandas I
	2:30					
	3:30	[실습]	[실습]	[실습]	[실습]	[실습]
	4:30					

일정 (2주차)

		1/10 (Mon)	1/11 (Tue)	1/12 (Wed)	1/13 (Thu)	1/14 (Fri)			
오전	8:30	Matplotlib I	확률통계	Matplotlib II	확률통계	Data Preprocessing II			
	9:30								
	10:30	[실습]				[실습]			
	11:30					시험			
	12:30	점심 시간 (12:30 ~ 1:30)							
오후	1:30	확률통계	Pandas II	확률통계	Data Preprocessing I	확률통계			
	2:30								
	3:30		[실습]		[실습]				
	4:30								

About Us

- 김진수 (Jin-Soo Kim)
 - Professor @ Dept. of Computer Science & Engineering, SNU
 - Systems Software & Architecture Laboratory
 - Operating systems, storage systems, parallel and distributed computing, embedded systems, ...
- E-mail: jinsoo.kim@snu.ac.kr
- <http://csl.snu.ac.kr>
- Tel: 02-880-7302
- Office: SNU Engineering Building #301-504
- TAs: 정성업(seongyeop.jeong@snu.ac.kr), 이준석(shawn159@snu.ac.kr)



Course Homepage

- <http://csl.snu.ac.kr/sleds>

- Lecture slides
- Lab. materials
- ID: lge
- PW: 6060

The screenshot shows a web browser window displaying the course homepage. The URL in the address bar is <http://csl.snu.ac.kr/sleds/>. The page header includes the Seoul National University logo, the text '서울대학교 SYSTEMS SOFTWARE & ARCHITECTURE LABORATORY', and a stylized blue 'ML' logo. The main content area features the title 'Python for Data Analytics (김진수)' and a list of course topics:

- Python basics
- Python sequence and collections
- NumPy I
- NumPy II
- Pandas I
- Matplotlib I
- Pandas II
- Matplotlib II
- Data Preprocessing
- Zoom link: <https://snu-ac-kr.zoom.us/j/88179526827?pwd=NzFSWDdXRmQxOVZ3UWtrV0oxNXdHUT09> (회의 ID: 881 7952 6827, 패스워드: 823823)

Outline

- Introduction to Python
- Basic data types
- Control Flow
- Functions

Introduction to Python

Why Python?

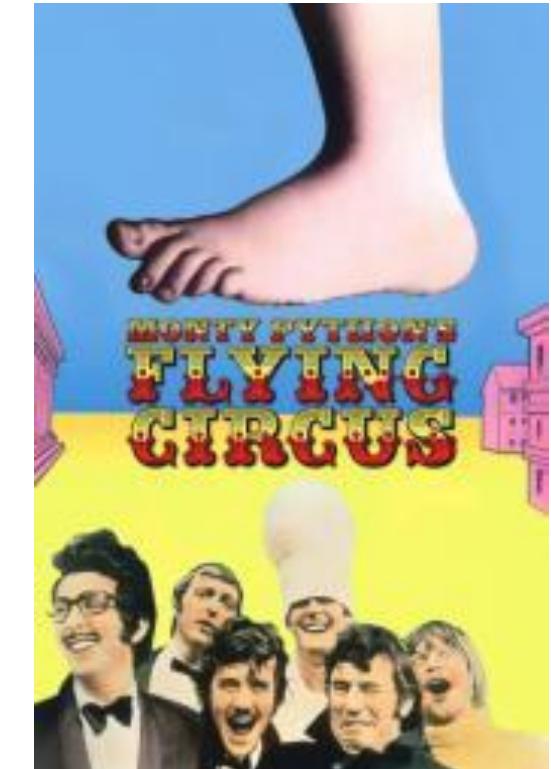
- 1st place among the “Top Programming Languages” (2017-2021, IEEE Spectrum)
- “Fastest growing major programming language” (stackoverflow.com, 2019)
- The 2nd most popular programming language in GitHub (Nov. 2021)
- “The language of AI”

Rank	Language	Type	Score
1	Python	🌐💻⚙️	100.0
2	Java	🌐📱💻	95.4
3	C	📱💻⚙️	94.7
4	C++	📱💻⚙️	92.4
5	JavaScript	🌐	88.1
6	C#	🌐📱💻⚙️	82.4
7	R	💻	81.7
8	Go	🌐💻	77.7
9	HTML	🌐	75.4
10	Swift	📱💻	70.4

The Birth of Python

- Developed by Guido van Rossum in 1990

Over six years ago, in December 1989, I was looking for a “hobby” programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python’s Flying Circus).



Python Goals

- “Computer programming for Everybody”
 - DARPA funding proposal
- An easy and intuitive language just as powerful as major competitors
- Open source, so anyone can contribute to its development
- Code that is as understandable as plain English
- Suitability for everyday tasks, allowing for short development times

Program like Plain English?

```
filename = input('Enter file: ')
f = open(filename)

counts = dict()
for line in f:
    words = line.strip().lower().split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

lst = sorted([(v,k) for k,v in counts.items()], reverse=True)

for v, k in lst[:10]:
    print(v, k)
```

Compare with this:

```

sys@sys:
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAX_CHARS 26
#define MAX_WORD_SIZE 30

struct TrieNode
{
    bool isEnd;
    unsigned frequency;
    int indexMinHeap;
    TrieNode* child[MAX_CHARS];
};

struct MinHeapNode
{
    TrieNode* root;
    unsigned frequency;
    char* word;
};

struct MinHeap
{
    unsigned capacity;
    int count;
    MinHeapNode* array;
};

TrieNode* newTrieNode()
{
    TrieNode* trieNode = new TrieNode;

    trieNode->isEnd = 0;
    trieNode->frequency = 0;
    trieNode->indexMinHeap = -1;
    for( int i = 0; i < MAX_CHARS; ++i )
        trieNode->child[i] = NULL;

    return trieNode;
}

MinHeap* createMinHeap( int capacity )
{
    MinHeap* minHeap = new MinHeap;
    minHeap->capacity = capacity;
    minHeap->count = 0;
    minHeap->array = new MinHeapNode[ minHeap->capacity ];

    return minHeap;
}

void swapMinHeapNodes ( MinHeapNode* a, MinHeapNode* b )
{
    MinHeapNode temp = *a;
    *a = *b;
    *b = temp;
}

void minHeapify( MinHeap* minHeap, int idx )
{
    int left, right, smallest;
    left = 2 * idx + 1;
    right = 2 * idx + 2;
    smallest = idx;
}

```

68,1-4 Top ▾

```

if ( left < minHeap->count &&
    minHeap->array[ left ].frequency <
    minHeap->array[ smallest ].frequency
)
    smallest = left;

if ( right < minHeap->count &&
    minHeap->array[ right ].frequency <
    minHeap->array[ smallest ].frequency
)
    smallest = right;

if( smallest != idx )
{
    minHeap->array[ smallest ].root->indexMinHeap = idx;
    minHeap->array[ idx ].root->indexMinHeap = smallest;

    swapMinHeapNodes ( &minHeap->array[ smallest ], &minHeap->array[ idx ] );
    minHeapify( minHeap, smallest );
}

void buildMinHeap( MinHeap* minHeap )
{
    int n, i;
    n = minHeap->count - 1;

    for( i = ( n - 1 ) / 2; i >= 0; --i )
        minHeapify( minHeap, i );
}

void insertInMinHeap( MinHeap* minHeap, TrieNode** root, const char* word )
{
    if( (*root)->indexMinHeap != -1 )
    {
        ++( minHeap->array[ (*root)->indexMinHeap ].frequency );
        minHeapify( minHeap, (*root)->indexMinHeap );
    }

    else if( minHeap->count < minHeap->capacity )
    {
        int count = minHeap->count;
        minHeap->array[ count ].frequency = (*root)->frequency;
        minHeap->array[ count ].word = new char [strlen( word ) + 1];
        strcpy( minHeap->array[ count ].word, word );

        minHeap->array[ count ].root = *root;
        (*root)->indexMinHeap = minHeap->count;

        ++( minHeap->count );
        buildMinHeap( minHeap );
    }

    else if( (*root)->frequency > minHeap->array[0].frequency )
    {
        minHeap->array[ 0 ].root->indexMinHeap = -1;
        minHeap->array[ 0 ].root = *root;
        minHeap->array[ 0 ].root->indexMinHeap = 0;
        minHeap->array[ 0 ].frequency = (*root)->frequency;

        delete [] minHeap->array[ 0 ].word;
        minHeap->array[ 0 ].word = new char [strlen( word ) + 1];
        strcpy( minHeap->array[ 0 ].word, word );
    }
}

```

136,2-8 50% ▾

```

}

void insertUtil ( TrieNode** root, MinHeap* minHeap,
                  const char* word, const char* dupWord )
{
    if ( *root == NULL )
        *root = newTrieNode();

    if ( *word != '\0' )
        insertUtil ( &(*root)->child[ tolower( *word ) - 97 ],
                     minHeap, word + 1, dupWord );
    else
    {
        if ( (*root)->isEnd )
            ++( (*root)->frequency );
        else
        {
            (*root)->isEnd = 1;
            (*root)->frequency = 1;
        }

        insertInMinHeap( minHeap, root, dupWord );
    }
}

void insertTrieAndHeap(const char *word, TrieNode** root, MinHeap* minHeap)
{
    insertUtil( root, minHeap, word, word );
}

void displayMinHeap( MinHeap* minHeap )
{
    int i;

    for( i = 0; i < minHeap->count; ++i )
    {
        printf( "%s : %d\n", minHeap->array[i].word,
                minHeap->array[i].frequency );
    }
}

void printKMostFreq( FILE* fp, int k )
{
    MinHeap* minHeap = createMinHeap( k );
    TrieNode* root = NULL;

    char buffer[MAX_WORD_SIZE];

    while( fscanf( fp, "%s", buffer ) != EOF )
        insertTrieAndHeap(buffer, &root, minHeap);

    displayMinHeap( minHeap );
}

int main()
{
    int k = 5;
    FILE *fp = fopen ("test.txt", "r");
    if (fp == NULL)
        printf ("File doesn't exist ");
    else
        printKMostFreq( fp, k );
    return 0;
}

```

204,0-1 Bot ▾

Python Versions

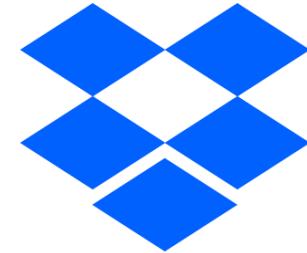
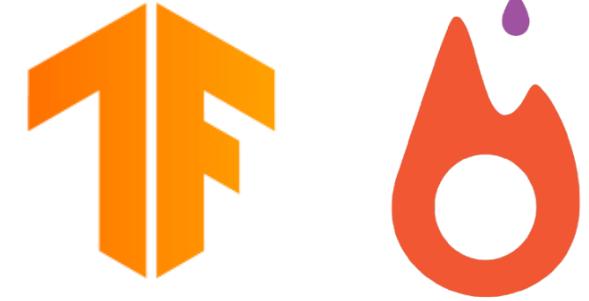
- Python 1.0 (1990)
- Python 2.0 (2000)
- Python 3.0 (2008) – Not backward compatible to 2.0
- The latest version: 3.10.1
- Official homepage: <https://www.python.org>
- Tutorial: <https://docs.python.org/ko/3/tutorial>

<https://docs.python.org>

The screenshot shows a web browser window displaying the Python 3.10.1 Documentation in Korean. The URL in the address bar is <https://docs.python.org/ko/3/>. The page title is "Python 3.10.1 문서". The left sidebar contains navigation links for "내려받기", "버전별 설명서" (with links to Python 3.11, 3.10, 3.9, 3.8, 3.7, 3.6, 3.5, 3.4, 3.3, 3.2, 3.1, 3.0, 2.9, 2.8, 2.7, and 2.6), and "기타 자원" (with links to PEP 색인, 초보자 가이드, 도서 목록, 오디오/비디오 토크, and 파이썬 개발자 지침서). The main content area starts with a welcome message: "Welcome! This is the official documentation for Python 3.10.1." It then lists several sections: "설명서의 파트들:", "파이썬 3.10 의 새로운 기능은?", "2.0 이후의 모든 "새로운 기능" 문서", "자습서", "여기에서 시작하세요", "라이브러리 레퍼런스", "배개 밑에 넣어 두세요", "언어 레퍼런스", "문법과 언어 요소들을 설명합니다", "파이썬 설정 및 사용법", "여러 플랫폼에서 파이썬을 사용하는 법", "파이썬 HOWTO", "특정 주제에 대한 심층적인 문서", "색인 및 표 목록:", "모듈 총 색인", "모든 모듈 조건표", "전체 색인", "함수, 클래스 및 용어 개관", "용어집", "가장 중요한 용어들을 설명합니다", "파이썬 모듈 설치하기", "파이썬 패키지 색인 및 기타 소스에서 설치하기", "파이썬 모듈 배포하기", "다른 사람들이 설치할 수 있도록 모듈을 게시하기", "확장 및 내장", "C/C++ 프로그래머를 위한 자습서", "파이썬/C API", "C/C++ 프로그래머를 위한 레퍼런스", "FAQs", "자주 나오는 질문들 (답도 있습니다!)", "검색 페이지", "문서 검색", and "종합 목차", "영역별 목차".

Python Applications

- Machine Learning (TensorFlow, PyTorch, etc.)
- GUI Applications (Kivy, Tkinter, PyQt, etc.)
- Web frameworks (Django used by YouTube, Instagram, Dropbox)
- Image processing (OpenCV, Pillow, etc.)
- Web scraping (Scrapy, BeautifulSoup, etc.)
- Text processing (NLTK, KoNLPy, Word2vec, etc.)
- Test frameworks
- Multimedia (audio, video, etc.)
- Scientific computing and many more ...



Welcome to the World of Spam!



Basic Data Types

Python Features

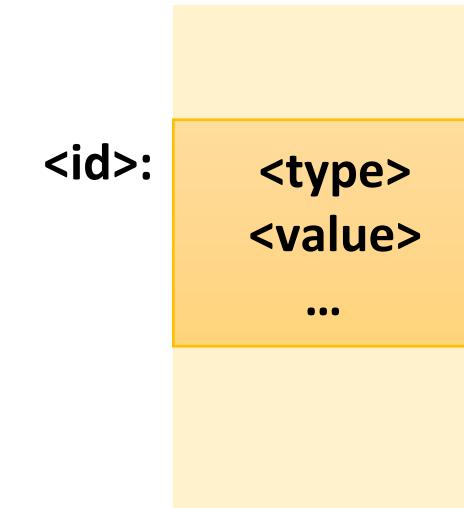
- Multi-paradigm platform-independent programming language
 - Structured
 - Object-oriented
 - Functional
 - ...
- Interpreted
- Dynamically-typed
- Highly extensible
 - Modules can be written in other languages such as C, C++, ...
- “Pythonic”

Data Types

- Basic data types
 - Boolean
 - Integer
 - Floating point
 - String
- Container data types
 - List
 - Dictionary
 - Tuple
 - Set
- Libraries
 - math
 - random
 - numpy
 - pandas
 - ...
- User-defined data types (classes)
 - Automobile
 - Monster
 - Pixel
 - ...

Object-oriented Data Model

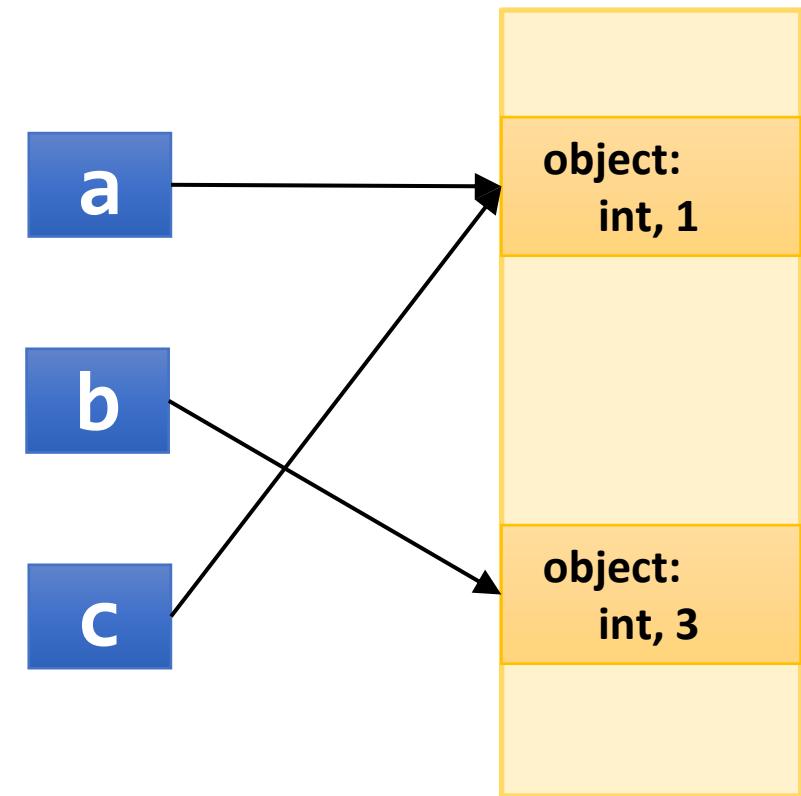
- Objects are Python's abstraction for data
- Each object has:
 - A type (or class) – `type(x)`
 - A value
 - An identity (e.g., memory address) – `id(x)`
 - A reference count – `sys.getrefcount(x)`
- The '`is`' operator compares the identity of two objects
- Objects can be **immutable** (e.g., numbers, strings, tuples, ...)
- Different variables can refer to the same object



Assignments

- Assignment operator (=) assigns a value (or an object) to a variable

```
>>> a = 1  
  
>>> b = 3  
  
>>> c = a  
  
>>> print(id(a), id(b), id(c))
```



Numbers

■ Integers (정수): unlimited range

- Decimal representation: 0 12 1_00 0001 
- Binary representation: 0b0 0b1100 0b110_0100
- Octal representation: 0o0 0o14 0o144
- Hexadecimal representation: 0x0 0xc 0x64

■ Floating-points (부동소수점수)

- Double-precision only ($4.9 \times 10^{-324} \sim 1.8 \times 10^{308}$)
- Always use a decimal representation (no binary/octal/hexadecimal)
- 12.0
- 0.0000_0000_001
- 3.14e-5

Arithmetic Operations

■ Addition:	$a + b$	$a += b$
■ Subtraction:	$a - b$	$a -= b$
■ Multiplication:	$a * b$	$a *= b$
■ Division:	a / b	\rightarrow floating-point
■ Floor division:	$a // b$	\rightarrow integer
■ Modulo:	$a \% b$	$a \%= b$
■ Power:	$a ** b$	$a **= b$

Bitwise Operations (Integers Only)

■ Invert:	$\sim a$	$a \sim= a$
■ Shift left:	$a \ll b$	$a \ll= b$
■ Shift right:	$a \gg b$	$a \gg= b$
■ Bitwise AND:	$a \& b$	$a \&= b$
■ Bitwise OR:	$a b$	$a = b$
■ Bitwise XOR:	$a ^ b$	$a ^= b$

math: Mathematical Functions

- `import math`
- `math.exp(x):` e^x
- `math.log(x):` $\log_e x$
- `math.log10(x):` $\log_{10} x$
- `math.log(x, b):` $\log_b x$
- `math.pow(x, y):` x^y
- `math.sqrt(x):` \sqrt{x}
- `math.sin(x):` $\sin x$
- `math.pi:` π
- ...

Floating-Point Example

```
>>> pi = 3.14159
>>> print(pi)
>>> print(2*pi)

>>> d = 0.1
>>> print(d+d+d+d+d+d+d+d+d)      # Use math.isclose()

>>> VeryLarge = 1e20
>>> x = (pi + VeryLarge) - VeryLarge
>>> y = pi + (VeryLarge - VeryLarge)
>>> print(x, y)
```

Boolean

- **bool**
 - `False` or `True`
 - A subtype of the integer type (Non-zero values are treated as `True`)
 - Boolean values behave like the integer values `0` and `1`, respectively
 - When converted to a string, '`False`' or '`True`' are returned, respectively

```
>>> t = False  
>>> print(t)  
  
>>> a = 100  
>>> b = bool(a)  
>>> print(a, b)
```

```
>>> t = True  
>>> f = False  
>>> x = 10  
>>> print(x + t)  
>>> print(t * f)  
>>> print(True == 1)
```

String

- A sequence of characters (immutable)

- Python 3 natively supports Unicode characters (even in identifiers)
- No difference in single (e.g., 'hello') or double-quoted strings (e.g., "hello")
- You can use raw strings by adding an **r** before the first quote

```
>>> print('I\'m your father')
>>> print("Where is 'spam'?")
>>> s1 = "What is the"
>>> s2 = 'spam'
>>> print(s1 + s2)
>>> print(len(s1))
```

```
>>> 이름 = '홍길동'
>>> print("안녕" , 이름)
>>> print("안녕" + 이름)
>>> print("안녕\n"+이름)
>>> print('안녕\n-\t'+이름)
>>> print(r'C:\abc\name')
```

ord() and chr()

■ ord(*c*)

- Return the Unicode for a character *c*

■ chr(*i*)

- Return a Unicode string of one character with ordinal *i*

```
>>> ord('a')  
97  
>>> ord('b')  
98  
>>> ord('A')  
65  
>>> ord('*')  
42  
>>> ord('\n')  
10  
>>> ord('가')  
44032  
>>> ord('각')  
44033
```

```
>>> chr(97)  
'a'  
>>> chr(97+1)  
'b'  
>>> c = 'x'  
>>> chr(ord(c)+1)  
'y'  
>>> chr(ord('A')+\\  
... ord(c)-ord('a'))  
'x'  
>>> chr(44032)  
'가'
```

Concatenating/Replicating Strings

- `str1 + str2` : Create a new string by adding two existing strings together
- Two or more string literals are automatically concatenated
- `str * n`: Create a new string by replicating the original string n times

```
>>> s1 = 'hello'  
>>> s2 = 'world'  
>>> s = s1 + s2  
>>> print(s)  
helloworld  
>>> print('hello' 'world' '!' )  
helloworld!
```

```
>>> s1 = 'hello'  
>>> s2 = 'world'  
>>> print(s1, s2)  
hello world  
>>> print(s1*3 + s2)  
hellohellohelloworld  
>>> print('-'*30)  
-----
```

String Indexing and Slicing

- `str[start:stop:step]`
- Same as list slicing

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> s = 'Monty Python'  
>>> print(s[1:2])  
o  
>>> print(s[8:])  
thon  
>>> print(s[:])  
Month Python  
>>> print(s[::-2])  
MnyPto
```

```
>>> print(s[-4:])  
thon  
>>> print(s[:-5])  
Monty P  
>>> print(s[:-6:-1])  
nohty  
>>> print(s[::-1])  
nohtyP ytnoM
```

The in Operator

- Check to see if one string is **in** another string
- Return **True** or **False**

```
>>> fruit = 'banana'  
>>> 'n' in fruit  
True  
>>> 'm' in fruit  
False  
>>> 'nan' in fruit  
True  
>>> if 'a' in fruit:  
...     print('Found it!')  
Found it!
```

String Methods

The screenshot shows a web browser displaying the Python documentation for string methods. The title 'String Methods' is at the top. Below it, the page content starts with a section about common sequence operations and two styles of string formatting: str.format() and printf-style. It then lists several methods with their descriptions:

- str.capitalize()**: Returns a copy of the string with its first character capitalized and the rest lowercased. A note says it's changed in version 3.8 to use titlecase instead of uppercase for characters like digraphs.
- str.casefold()**: Returns a casefolded copy of the string. Casefolded strings may be used for caseless matching. A note explains it's more aggressive than lower() for characters like German 'ß'.
- str.center(width[, fillchar])**: Returns centered in a string of length width. Padding is done using the specified fillchar (default is an ASCII space). The original string is returned if width is less than or equal to len(s).
- str.count(sub[, start[, end]])**: Returns the number of non-overlapping occurrences of substring sub in the range [start, end]. Optional arguments start and end are interpreted as in slice notation.
- str.encode(encoding="utf-8", errors="strict")**: Returns an encoded version of the string as a bytes object. Default encoding is 'utf-8'. errors may be given to set a different error handling scheme. The default for errors is 'strict', meaning that encoding errors raise a UnicodeError. Other possible values are 'ignore', 'replace', 'xmlcharrefreplace', etc.

Test

- **str.startswith(prefix[,start[,end]])**
 - True if string starts with the prefix
- **str.endswith(suffix [,start[,end]])**
 - True if string ends with the suffix
- **str.isalpha()**
 - True if all characters are alphabetic
- **str.isdigit()**
 - True if all characters are digits
- **str.isprintable()**
 - True if all characters are printable
- **str.islower()**
 - True if all characters are lower case
- **str.isupper()**
 - True if all characters are uppercase
- **str.isspace()**
 - True if there are only whitespace characters

Find / Replace

- `str.count(sub[,start[,end]])`
- `str.find(sub[,start[,end]])`
 - Return the lowest index where substring `sub` is found (-1 if `sub` is not found)
- `str.index(sub[,start[,end]])`
 - Like `find()`, but raise `ValueError` if `sub` is not found
- `str.replace(old,new[,count])`
 - Return a copy of the string with all occurrences of substring `old` replaced by `new`

```
>>> b = 'banana'  
>>> print(b.count('a'))  
3  
  
>>> print(b.find('x'))  
-1  
  
>>> print(b.index('na'))  
2  
  
>>> print(b.replace('a','x'))  
bxnxnx
```

Reformat (I)

■ **str.lower()**

- Return a copy of the string with all the characters converted to lowercase

■ **str.upper()**

- Return a copy of the string with all the characters converted to uppercase

■ **str.capitalize()**

- Return a copy of the string with its first character capitalized and the rest lowercased

```
>>> s = 'MoNtY PyThOn'
```

```
>>> print(s.lower())
monty python
```

```
>>> print(s.upper())
MONTY PYTHON
```

```
>>> print(s.capitalize())
Monty python
```

Reformat (2)

- **str.lstrip([chars])**

- Return a copy of the string with leading characters removed.
- If omitted, the *chars* argument defaults to whitespace characters

- **str.rstrip([chars])**

- Like `lstrip()`, but trailing characters are removed

- **str.strip([chars])**

- `str.lstrip([chars]) + str.rstrip([chars])`

```
>>> s = '-- monty python --'
```

```
>>> print(s.lstrip(' -'))  
monty python --
```

```
>>> print(s.rstrip(' - '))  
--- monty python
```

```
>>> print(s.strip(' -mno'))  
ty pyth
```

Split

- **`str.split(sep, maxsplit)`**
 - Return a list of the words in the string, using `sep` as the delimiter string
 - If `maxsplit` is given, at most `maxsplit` splits are done. Otherwise all possible splits are made
 - The `sep` argument may consist of multiple characters (None = whitespaces)
 - If `sep` is given, consecutive delimiters are NOT grouped together

```
>>> s = 'hi hello world'  
>>> s.split()  
['hi', 'hello', 'world']  
>>> t = '1:2:3'  
>>> t.split(':')  
['1', '2', '3']  
>>> t.split(':', 1)  
['1', '2:3']  
>>> t = '1:2::3'  
>>> t.split(':')  
['1', '2', '', '3']  
>>> t.split('::')  
['1:2', '3']
```

Join

- **str.join(*iterable*)**
 - Return a string which is the concatenation of the strings in *iterable*
 - *iterable*: List, Tuple, String, Dictionary, Set
 - The separator between elements is the string (*str*) providing this method

```
>>> menu = ['spam', 'ham', 'egg']
>>> ', '.join(menu)
'spam,ham,egg'
>>> ' '.join(menu)
'spam ham egg'
>>> ' * '.join(menu)
'spam * ham * egg'
>>> '#'.join('spam')
's#p#a#m'
```

input(): Getting User Input

- Read a line and convert it to a **string** with stripping a trailing newline

```
>>> name = input('Your name: ')
Your name: Spam ←
>>> age = input('Your age: ')
Your age: 20 ←
>>> print('Hello,', name)
Hello, Spam
>>> print('You will be', int(age)+1, 'next year!')
You will be 21 next year!
```

Formatting Strings

- Old string formatting with % operator

```
>>> print('%d %ss cost $%f' % (a, b, c))  
2 spams cost $4.990000
```

- The string `format()` method

```
>>> print('{0} {1}s cost ${2}'.format(a, b, c))  
2 spams cost $4.99
```

- f-strings (formatted string literals, since Python 3.6)

```
>>> print(f'{a} {b}s cost ${c}')  
2 spams cost $4.99
```

```
>>> a = 2  
>>> b = 'spam'  
>>> c = 4.99
```

Formatting with % Operator

- String formatting expression: '... %d ...' % (values)

```
>>> a = 2
>>> fruit = 'apples'

>>> print('I have %s.' % fruit)
I have apples.

>>> print('I have %d %s.' % (a, fruit))
I have 2 apples.

>>> print('sqrt(%d) is %f' % (a, a**0.5))
sqrt(2) is 1.414214
```

Formatting with % Operator: Type Code

- $\%[flags][width][.precision]typecode$

- *flags*
 - Left justification (-)
 - Numeric sign (+)
 - Zero fills (0)
 - ...
- *width*: a total minimum field width for the substitute text
- *precision*: the number of digits to display after a decimal point for floating-point numbers

Code	Meaning
%s	String (or most objects with str())
%c	Character
%d	Integer
%o	Octal integer
%x	Hexadecimal integer
%X	Same as %x, but with uppercase letters
%f	Floating-point decimal
%e	Floating point with exponent, lowercase
%E	Floating point with exponent, uppercase
%%	Literal %

Formatting with % Operator: Examples

```
>>> x = 1234
>>> s = 'integers: ...%d...%-6d...%06d' % (x, x, x)
>>> print(s)
integers: ...1234...1234    ...001234

>>> f = 3.14159265
>>> print('%e | %E | %f' % (f, f, f))
3.141593e+00 | 3.141593E+00 | 3.141593
>>> print('%-6.2f | %05.2f | %+06.1f' % (f, f, f))
3.14    | 03.14 | +003.1

>>> h = '0x%08x' % x
>>> print(hex(x), h)
0x4d2 0x0000004d2
```

Formatting with format()

- The string `format()` method: '`... {} ...`'.`format(values)`

```
>>> a = 2
>>> fruit = 'apples'

>>> print('I have {}'.format(fruit))
I have apples.

>>> print('I have {} {}'.format(a, fruit))
I have 2 apples.

>>> print('sqrt({}) is {}'.format(a, a**0.5))
sqrt(2) is 1.414214
```

Formatting with format(): Examples

```
>>> print('{}, {} and {}'.format('spam', 'ham', 'eggs'))
spam, ham and eggs
>>> print('{0}, {2}, and {1}'.format('spam', 'ham', 'eggs'))
spam, eggs, and ham
>>> print('{x}, {y}, and {z}'.format(z='spam', x='ham', y='eggs'))
ham, eggs, and spam
>>> print('{x}, {0}, and {y}'.format('spam', x='ham', y='eggs'))
ham, spam, and eggs

>>> print('{}, {} and {}'.format(-1, 3.14159265, [1, 2, 3, 4]))
-1, 3.14159265 and [1, 2, 3, 4]
>>> x = 3.14156295
>>> print('{0:<10.2f}, {1:>10.5f}, and {val:+10.2f}'.format(x, x, val=x))
3.14      ,     3.14156, and      +3.14
```

Formatting with f-strings

- Formatted string literals: `f'... {value} ...'`

```
>>> a = 2
>>> fruit = 'apples'

>>> print(f'I have {fruit}.')
I have apples.

>>> print(f'I have {a} {fruit}.')
I have 2 apples.

>>> print(f'sqrt({a}) is {a**0.5}')
sqrt(2) is 1.4142135623730951
```

Formatting with f-strings: Examples

```
>>> val = 1234
>>> print(f'...{val}...{val:6}...{val:+6}...{val:06}')
...1234... 1234... +1234...001234
>>> print(f'...{val:x}...{val:o}...{val:b}...{val:e}')
...4d2...2322...10011010010...1.234000e+03
>>> st = 'hello'
>>> print(f'|{val:<10}|{val:>10}|{st:<10}|{st:>10}|')
|1234      |      1234|hello      |      hello|
>>> sq2 = 2**0.5
>>> print(f'{sq2:f}...{sq2:e}...{sq2:E}')
1.414214...1.414214e+00...1.414214E+00
>>> print(f'{sq2:<10.4f}...{sq2:010.4f}...{sq2:+6.1f}')
1.4142      ...00001.4142... +1.4
```

Control Flow

Evaluating Conditions

- Boolean expressions using comparison operators evaluate to **True** or **False**
- Several Boolean expressions can be combined using logical **and** / **or** / **not** operators
- Comparison operators do not change the variables

Notation	Meaning
<code>a < b</code>	True if a is less than b
<code>a <= b</code>	True if a is less than or equal to b
<code>a == b</code>	True if a is equal to b
<code>a != b</code>	True if a is not equal to b
<code>a >= b</code>	True if a is greater than or equal to b
<code>a > b</code>	True if a is greater than b
<code>A and B</code>	True if both A and B are True
<code>A or B</code>	True if either A or B (or both) is True
<code>not A</code>	True if A is False
<code>a is b</code>	True if a and b point to the same object
<code>a is not b</code>	True if a and b point to the different object
<code>a in b</code>	True if a is in the sequence b
<code>a not in b</code>	True if a is not in the sequence b

Branching

- <condition> has a value **True** or **False**
- Evaluate expressions in that block if <condition> is **True**

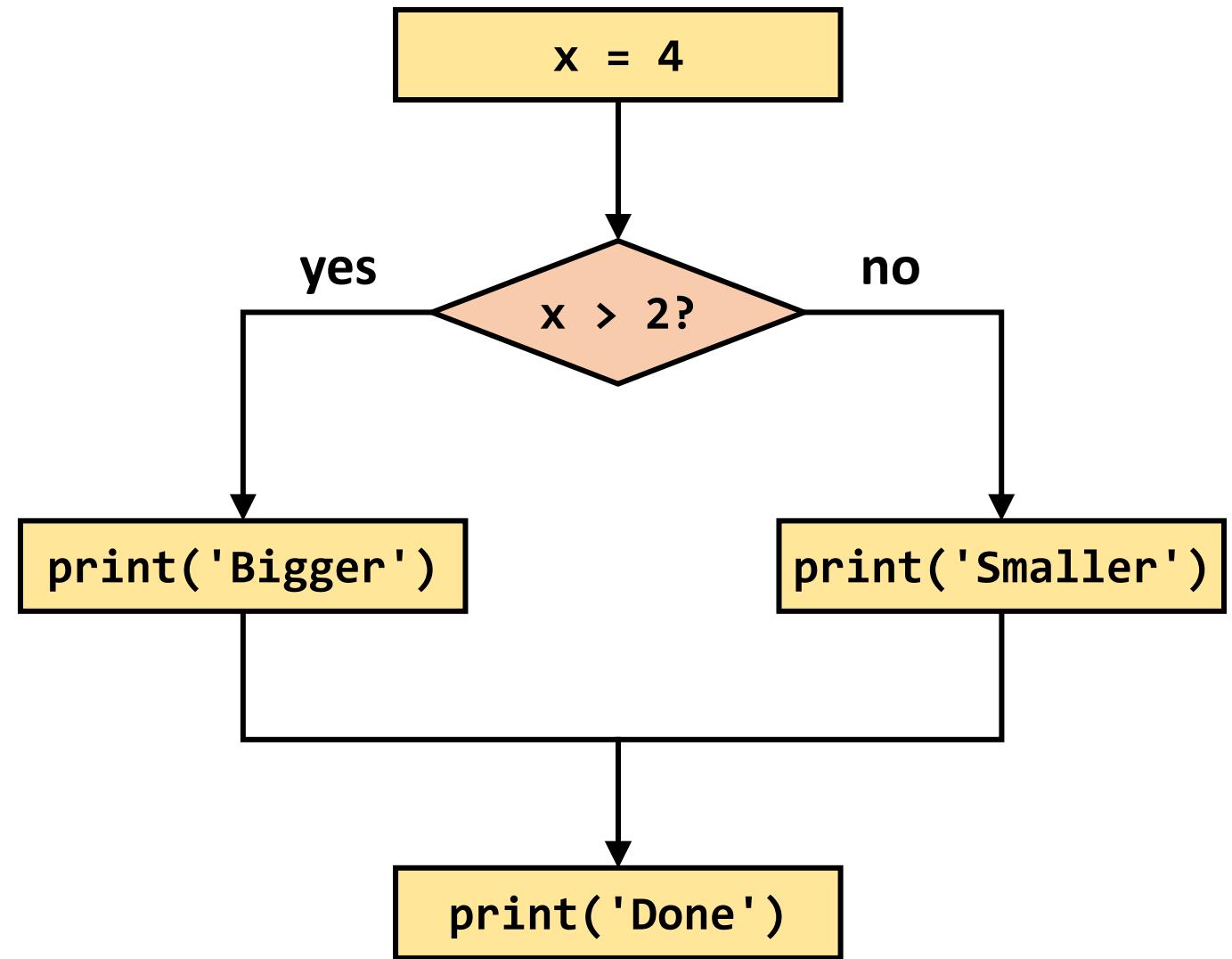
```
if <condition>:  
    <expression>  
    <expression>  
    ...
```

```
if <condition>:  
    <expression>  
    ...  
else:  
    <expression>  
    ...
```

```
if <condition>:  
    <expression>  
    ...  
elif <condition>:  
    <expression>  
    ...  
elif <condition>:  
    <expression>  
    ...  
else:  
    <expression>  
    ...
```

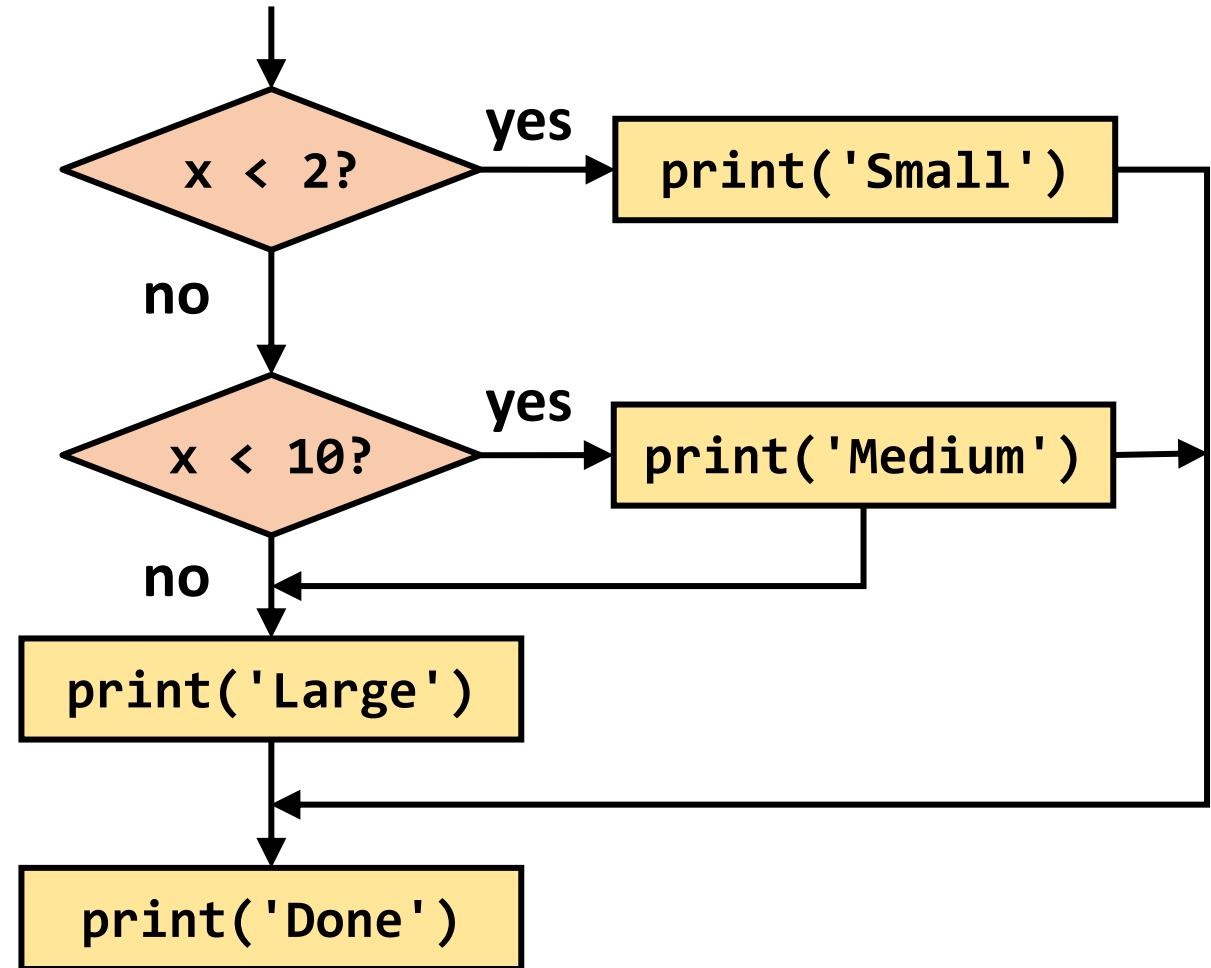
Two-way Decisions

```
x = 4
if x > 2:
    print('Bigger')
else:
    print('Smaller')
print('Done')
```



Multi-way Decisions

```
if x < 2:  
    print('Small')  
elif x < 10:  
    print('Medium')  
else:  
    print('Large')  
print('Done')
```



Multi-way Puzzles

- What's wrong with these programs?

```
if x < 2:  
    print('Below 2')  
elif x >= 2:  
    print('Two or more')  
else:  
    print('Something else')
```

```
if x < 2:  
    print('Below 2')  
elif x < 20:  
    print('Below 20')  
elif x < 10:  
    print('Below 10')  
else:  
    print('Something else')
```

Conditional Expression

```
if score >= 90:  
    grade = 'A'  
elif score >= 80:  
    grade = 'B'  
elif score >= 70:  
    grade = 'C'  
elif score >= 60:  
    grade = 'D'  
else:  
    grade = 'F'
```

```
grade = 'A' if score >= 90 else \  
        'B' if score >= 80 else \  
        'C' if score >= 70 else \  
        'D' if score >= 60 else \  
        'F'
```

Loops

- **while loop**

- Keep running the loop body while expression is **True**

```
while (expression):  
    <statement_1>  
    <statement_2>  
    ...  
    <statement_n>
```

- **for loop**

- Run the loop body for the specified range

```
for <element> in <object>:  
    <statement_1>  
    <statement_2>  
    ...  
    <statement_n>
```

Loops Example

- **while loop**

- Keep running the loop body while expression is **True**

```
i = 0  
  
while i < 5:  
    print(i)  
    i += 1
```

- **for loop**

- Run the loop body for the specified range

```
for i in range(5):  
    print(i)
```

While vs. For

- Indefinite loop – **while**
 - **while** is natural to loop an indeterminate number of times until a logical condition becomes **False**
- Definite loop – **for**
 - **for** is natural to loop through a **list**, characters in a **string**, **tuples**, etc. (anything of determinate size)
 - Run the loop once for each of the items

Specifying an Integer Range

- `range([start,] stop[, step])`
 - Represents an immutable sequence of numbers
 - If the `step` argument is omitted, it defaults to 1 (`step` should not be zero)
 - If the `start` argument is omitted, it defaults to 0

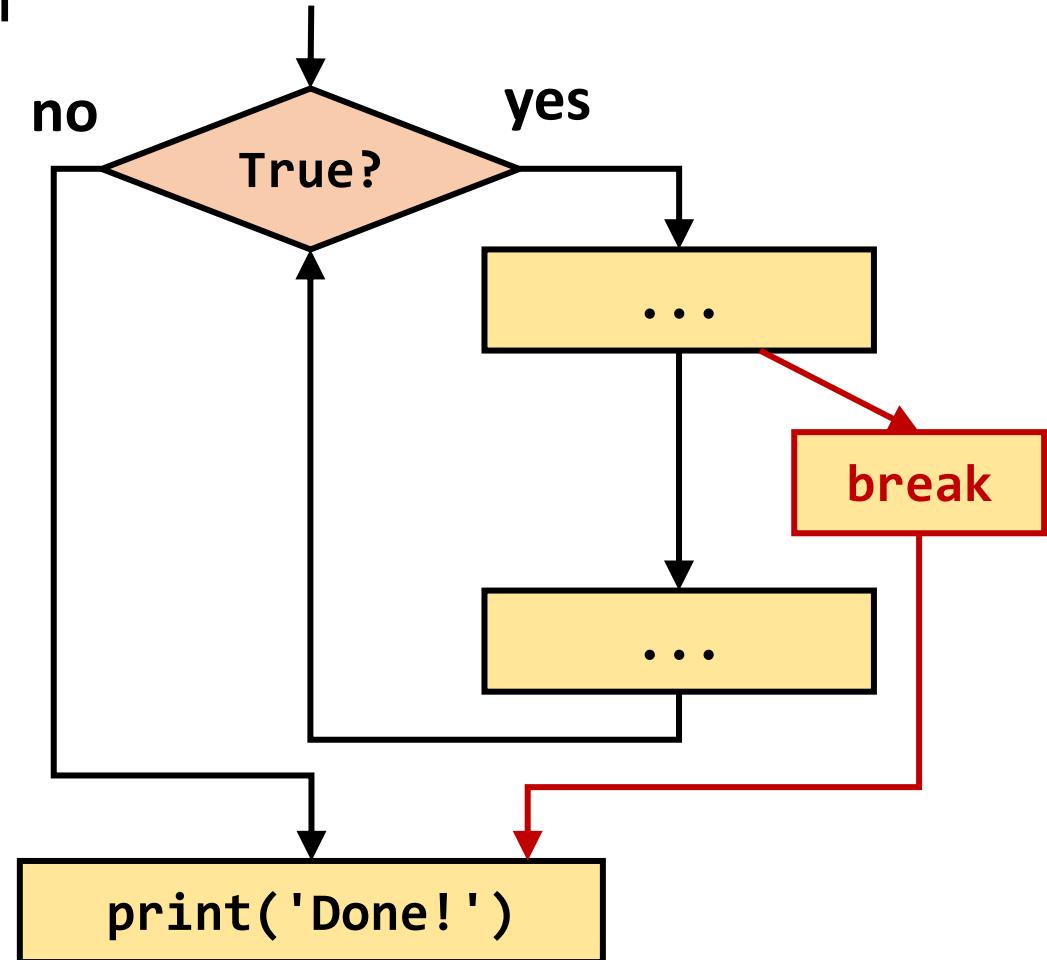
```
range(5)           # 0, 1, 2, 3, 4  
range(-1, 4)      # -1, 0, 1, 2, 3  
range(0,10,2)     # 0, 2, 4, 6, 8  
range(5,0,-1)     # 5, 4, 3, 2, 1  
range(10,2)       # ???
```

- `list(range(100)) → [0, 1, 2, ..., 99]`

break: Breaking Out of a Loop

- Immediately exists whatever loop it is in
 - Skips remaining expressions in code block
 - Exits only innermost loop!

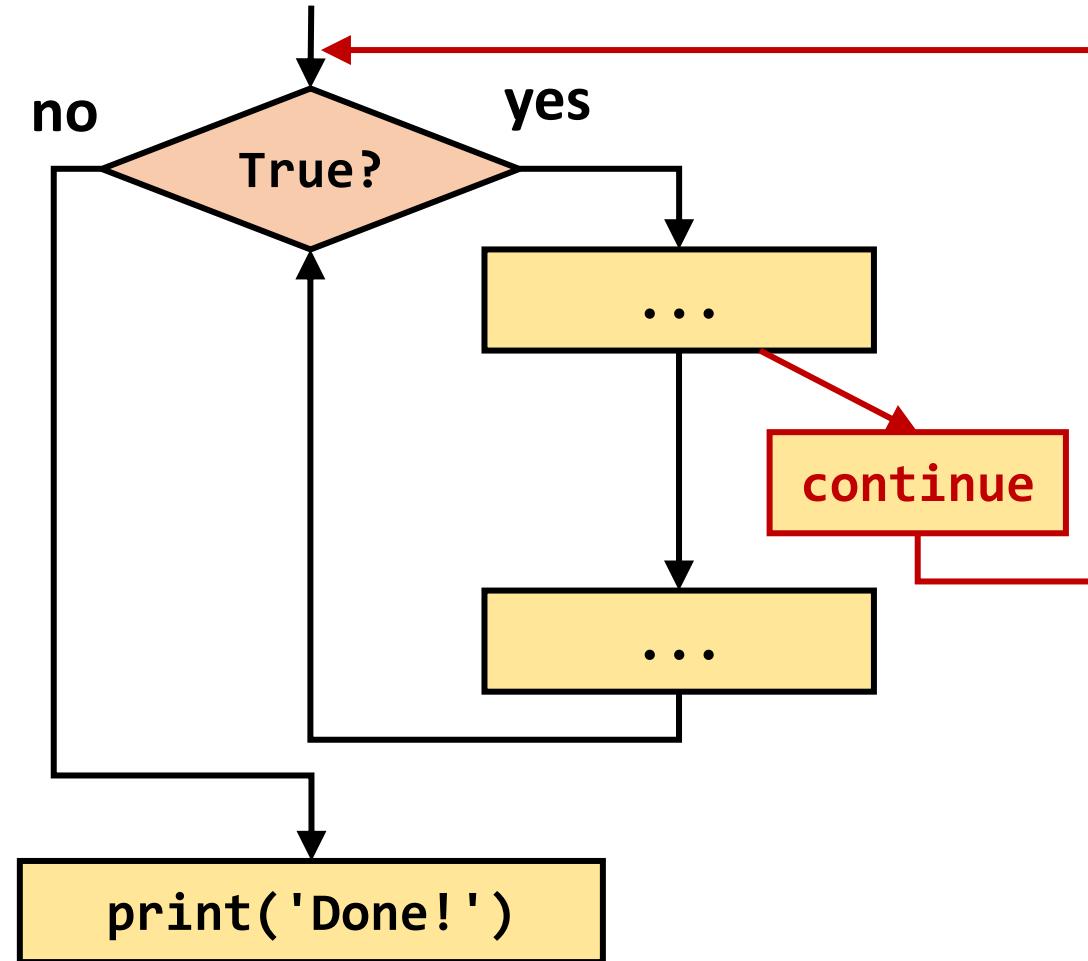
```
while True:  
    line = input('> ')  
    if line == 'done':  
        break  
    print(line)  
print('Done!')
```



continue: Finishing an Iteration

- Ends the current iteration and jumps to the top of the loop and starts the next iteration

```
while True:  
    line = input('> ')  
    if line[0] == '#':  
        continue  
    if line == 'done':  
        break  
    print(line)  
print('Done!')
```



Looping Through a List

```
friends = ['Harry', 'Sally', 'Tom', 'Jerry']

for friend in friends:
    print('Merry Christmas,', friend)

for i in range(len(friends)):
    print('Merry Christmas,', friends[i])
```

Exceptions

- Errors detected during execution even if a statement or expression is syntactically correct
 - `ZeroDivisionError`
 - `NameError`
 - `TypeError`
 - `ValueError`
 - `IndexError...`

```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> int('what')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with
base 10: 'what'
```

Handling Exceptions

- Surround a dangerous section of code with `try` and `except`
- If the code in the `try` works – the `except` is skipped
- If the code in the `try` fails – it jumps to the `except` code block

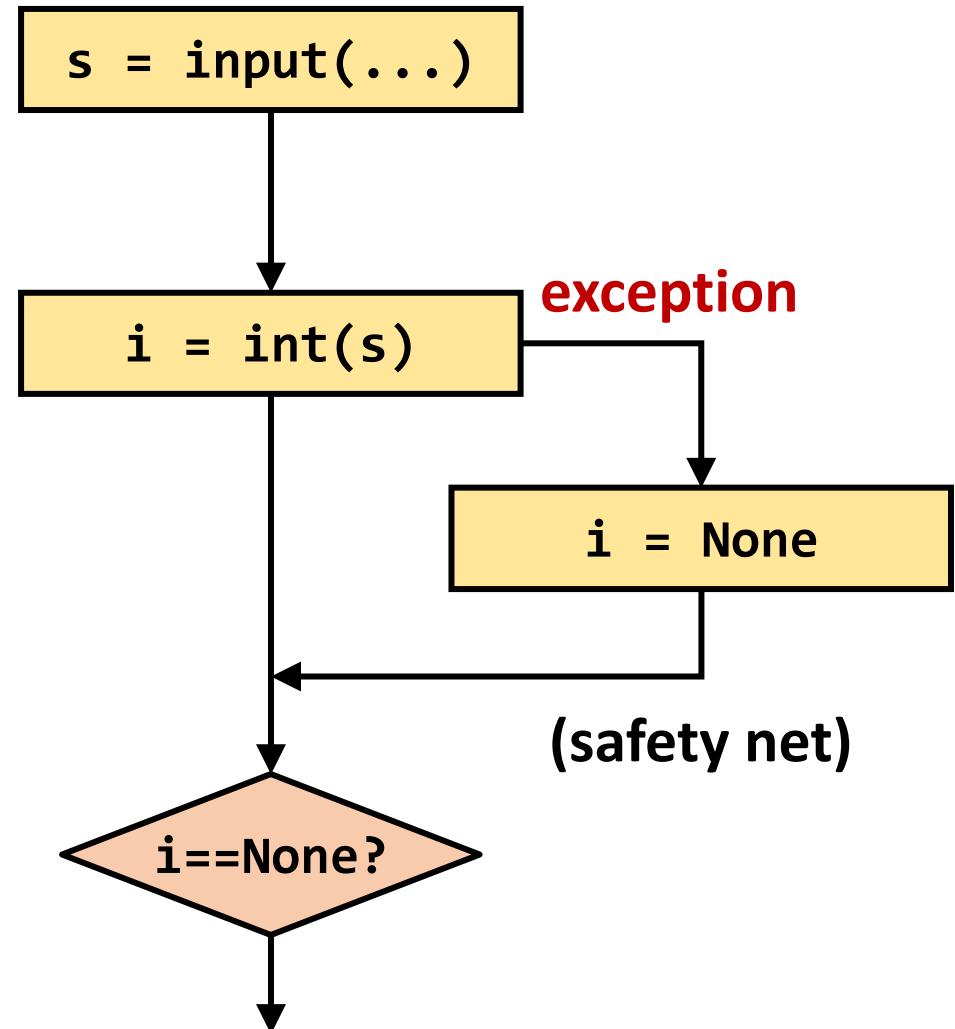
```
x = int(input('Enter a number: '))
x1 = x + 1
print(x, '+ 1 =', x1)
```

```
while True:
    try:
        x = int(input('Enter a number: '))
        break
    except:      # catch all exceptions
        print('Oops, try again...')
x1 = x + 1
print(x, '+ 1 =', x1)
```

Example

```
s = input('Enter a number: ')
try:
    i = int(s)
except:
    i = None

if i is None:
    print('Not a number')
else:
    print('Nice work')
```



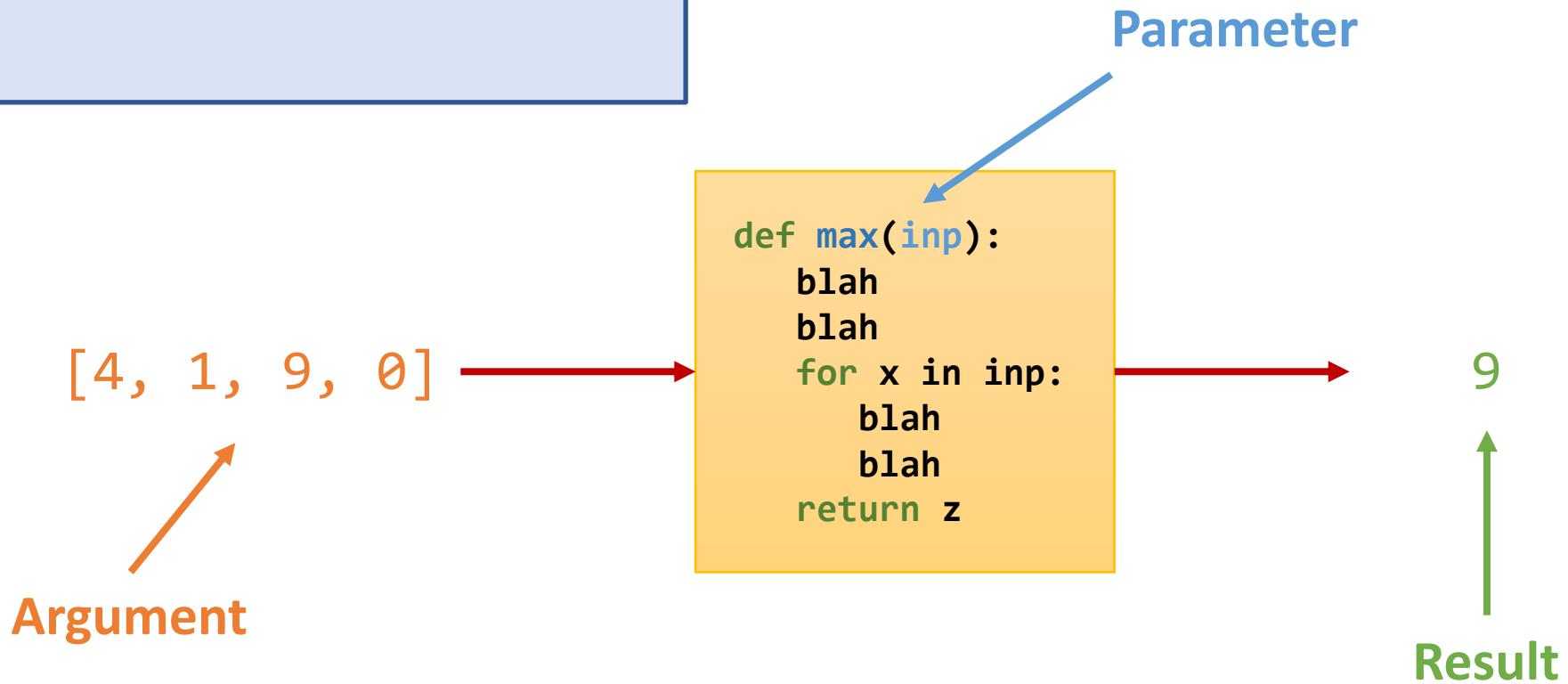
Functions

Python Functions

- A **function** is some reusable code that takes argument(s) as input, does some computation, and then returns a result or results.
- Built-in functions
 - Provided as part of Python
 - `print()`, `input()`, `type()`, `float()`, `int()`, `max()`, ...
- User-defined functions
 - Functions that we define ourselves and then use
 - A function can be defined using the **def** reserved word
 - A function is called (or invoked) by using the function name, parentheses, and arguments in an expression

Arguments, Parameters, and Results

```
>>> big = max([4, 1, 9, 0])  
>>> print(big)  
9
```



Parameters

- A **parameter** is a variable which we use in the function **definition**.
- It is a "handle" that allows the code in the function to access the **arguments** for a particular function invocation.

```
>>> def greet(lang):  
...     if lang == 'kr':  
...         print('안녕하세요')  
...     elif lang == 'fr':  
...         print('Bonjour')  
...     elif lang == 'es':  
...         print('Hola')  
...     else:  
...         print('Hello')  
  
>>> greet('en')  
Hello  
>>> greet('es')  
Hola  
>>> greet('kr')  
안녕하세요  
>>>
```

Return Value

- A "fruitful" function is one that produces a **result** (or **return value**)
- The **return** statement ends the function execution and "sends back" the **result** of the function

```
>>> def greet(lang):  
...     if lang == 'kr':  
...         return '안녕하세요'  
...     elif lang == 'fr':  
...         return 'Bonjour'  
...     elif lang == 'es':  
...         return 'Hola'  
...     else:  
...         return 'Hello'  
  
>>> print(greet('en'), 'Jack')  
Hello Jack  
>>> print(greet('es'), 'Sally')  
Hola Sally  
>>> print(greet('kr'), '홍길동')  
안녕하세요 홍길동  
>>>
```

Multiple Parameters / Arguments

- We can define more than one **parameter** in the function definition
- We simply add more **arguments** when we call the function
- We match the number and order of arguments and parameters

```
def mymax(a, b):  
    if a > b:  
        return a  
    else:  
        return b  
  
x = mymax(3, 5)  
print(x)  
5
```

Default and Keyword Arguments

- Default arguments
 - You can specify default values for arguments that aren't passed
- Keyword arguments
 - Callers can specify which argument in the function to receive a value by using the argument's name in the call

```
def student(name, id='00000', dept='CSE'):  
    print(name, id, dept)  
  
student('John')  
student('John', '00001')  
student(name='John')  
student(id='20191', dept='EE', name='Jack')
```

Arbitrary Arguments

- For functions that take any number of arguments
- Zero or more normal arguments may appear before the variable number of arguments,
- All the arbitrary arguments are collected and transferred using a tuple

```
def food(name, *likes):
    print(name, 'likes ', end=' ')
    for d in likes:
        print(d, end=' ')
    print()

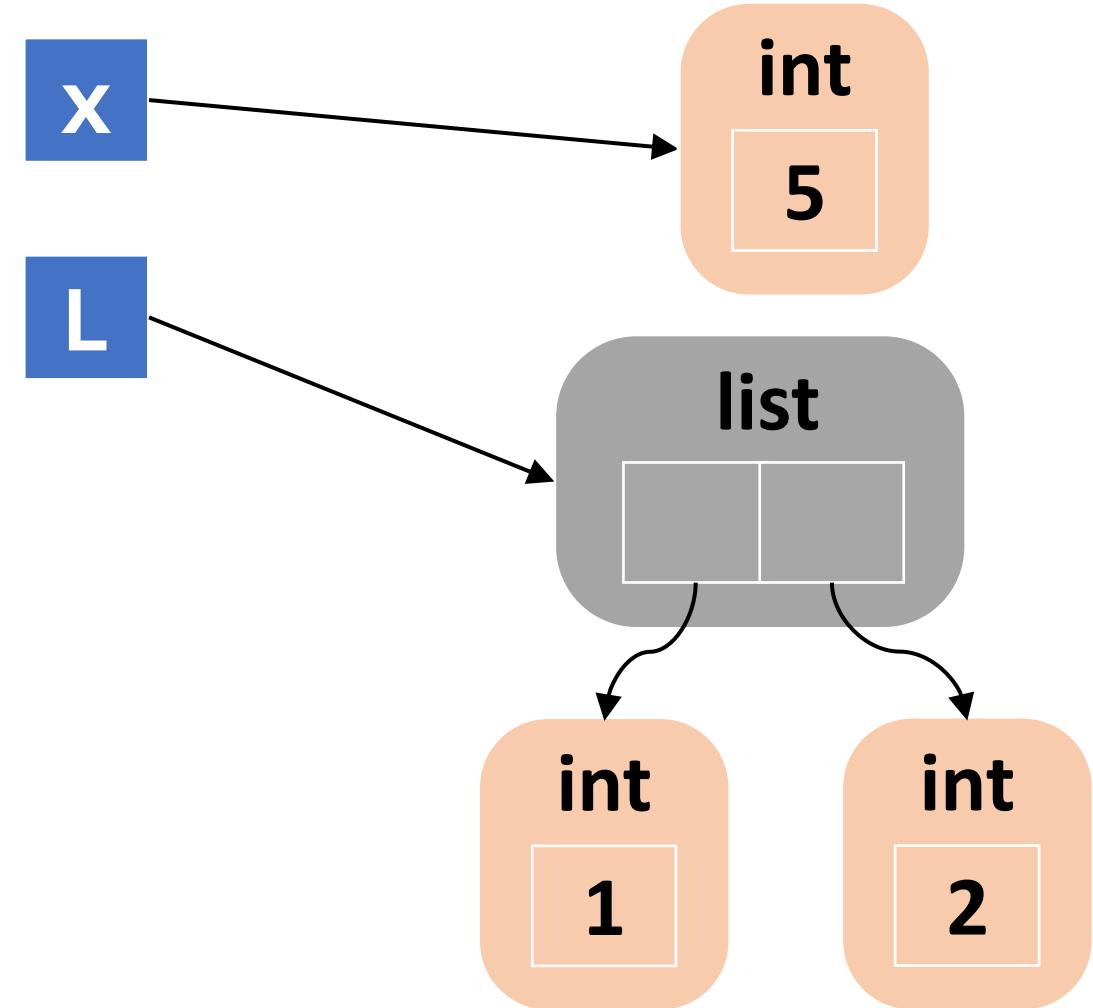
food('John', 'spam', 'egg', 'bacon')
```

Passing Arguments

```
>>> def f(a, b):  
...     a = 9  
...     b[0] = 'spam'  
  
>>> x = 5  
>>> L = [1, 2]  
>>> f(x, L)  
>>> print(x)  
5  
>>> print(L)  
['spam', 2]
```

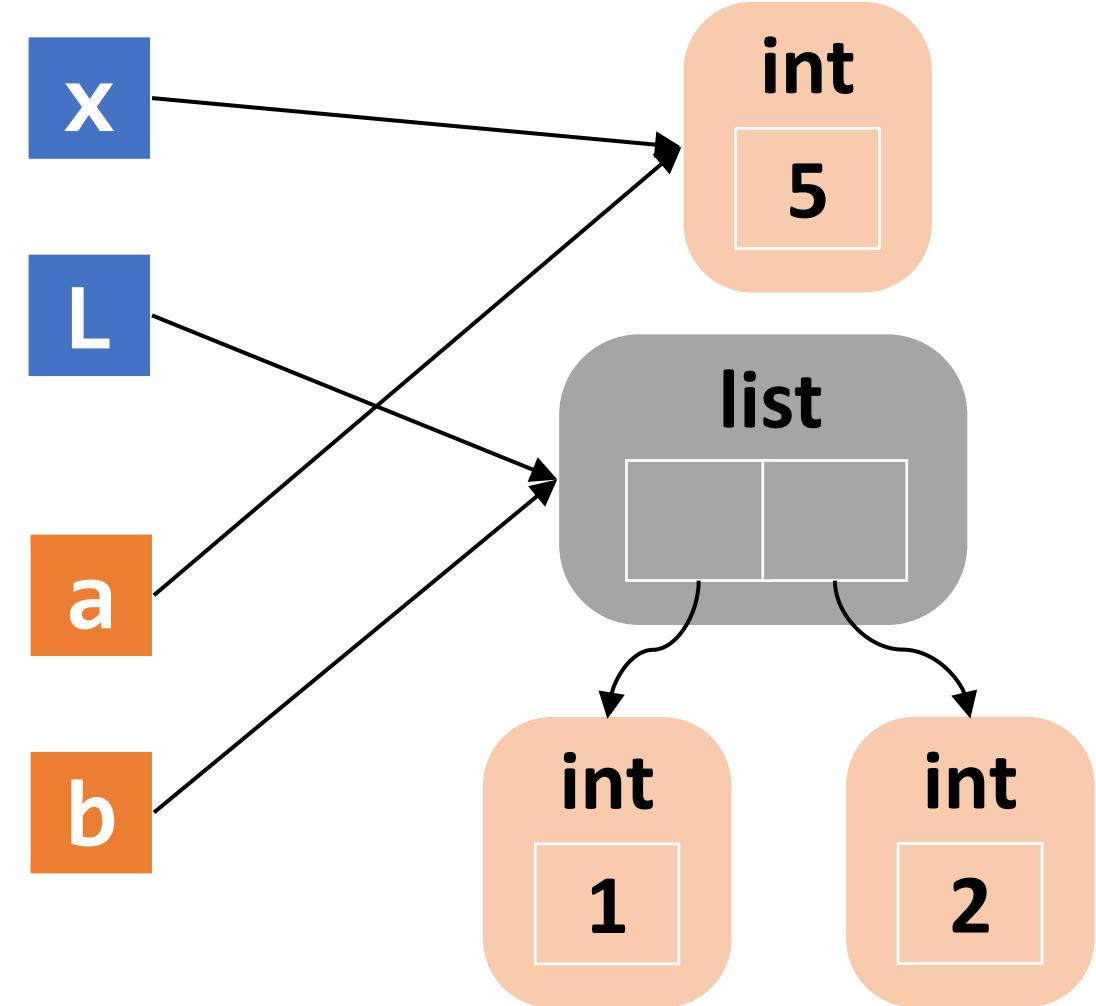
Passing Arguments

```
>>> def f(a, b):  
...     a = 9  
...     b[0] = 'spam'  
  
>>> x = 5  
>>> L = [1, 2]
```



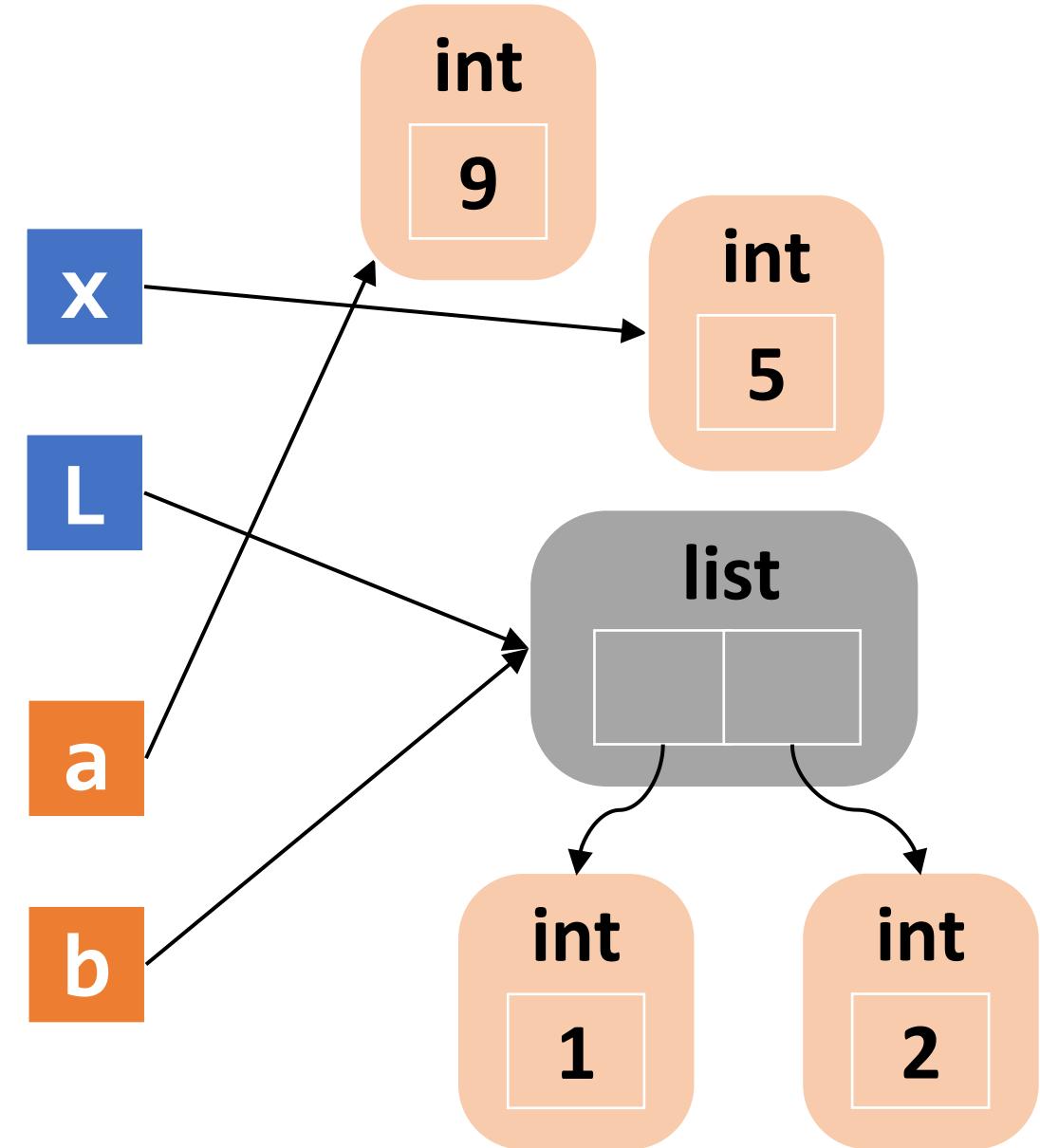
Passing Arguments

```
>>> def f(a, b):  
...     a = 9  
...     b[0] = 'spam'  
  
>>> x = 5  
>>> L = [1, 2]  
>>> f(x, L)  
  
a = x  
b = L
```



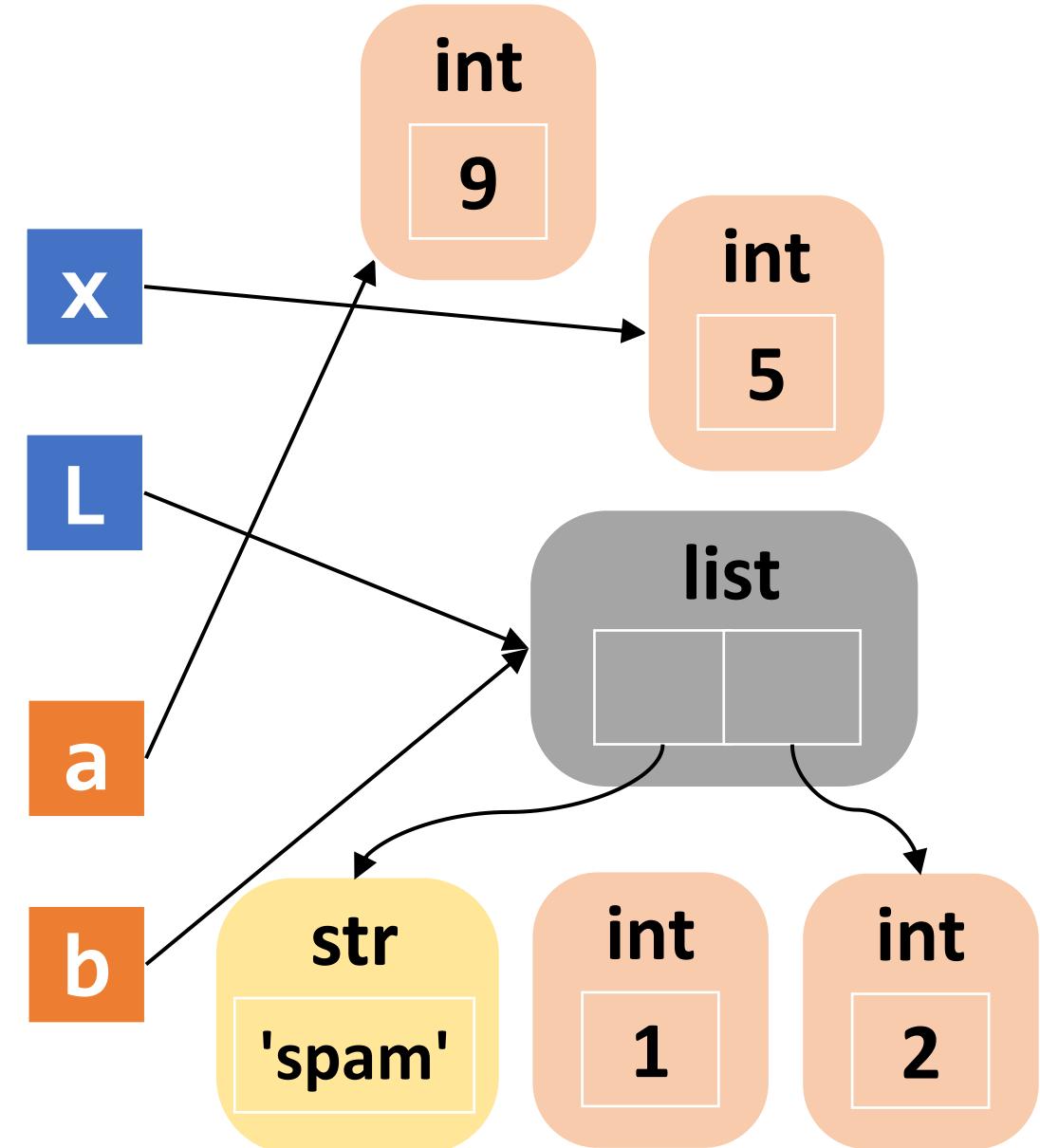
Passing Arguments

```
>>> def f(a, b):  
...     a = 9  
...     b[0] = 'spam'  
  
>>> x = 5  
>>> L = [1, 2]  
>>> f(x, L)
```



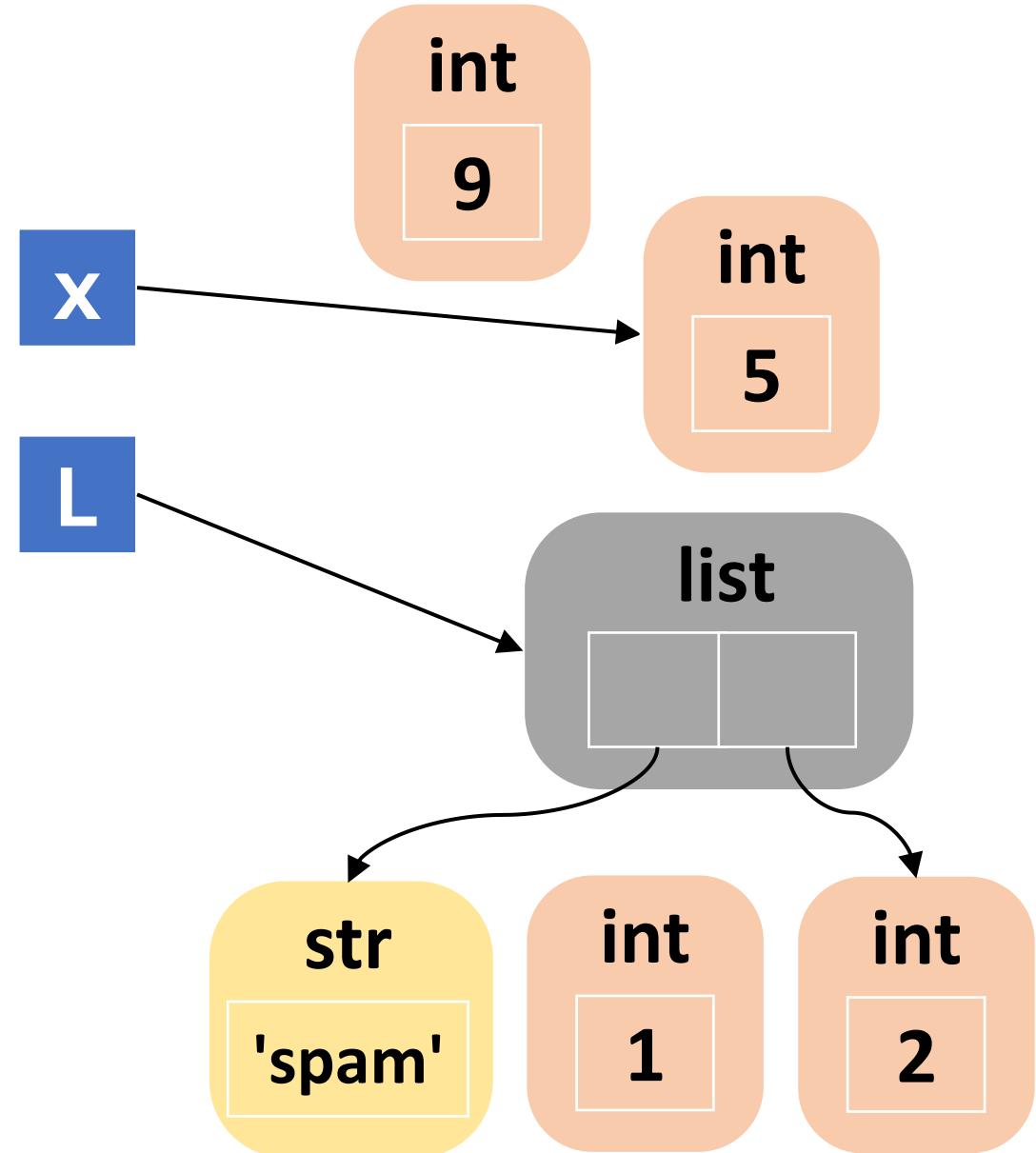
Passing Arguments

```
>>> def f(a, b):  
...     a = 9  
...     b[0] = 'spam'  
  
>>> x = 5  
>>> L = [1, 2]  
>>> f(x, L)
```



Passing Arguments

```
>>> def f(a, b):  
...     a = 9  
...     b[0] = 'spam'  
  
>>> x = 5  
>>> L = [1, 2]  
>>> f(x, L)  
>>> print(x)  
5  
>>> print(L)  
['spam', 2]
```



Recursive Function

- Functions that call themselves either directly or indirectly

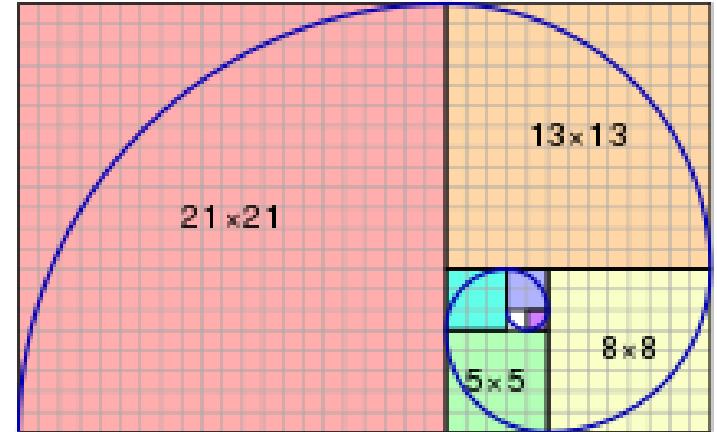
$$f(n) = \begin{cases} n \times f(n - 1) & n > 1 \\ 1 & n \leq 1 \end{cases}$$

```
def f(n):
    if n <= 1:
        return 1
    else:
        return n * f(n-1)
```

Fibonacci Numbers

$$f(n) = f(n - 1) + f(n - 2) \quad n \geq 2$$

$$f(0) = 0, f(1) = 1$$

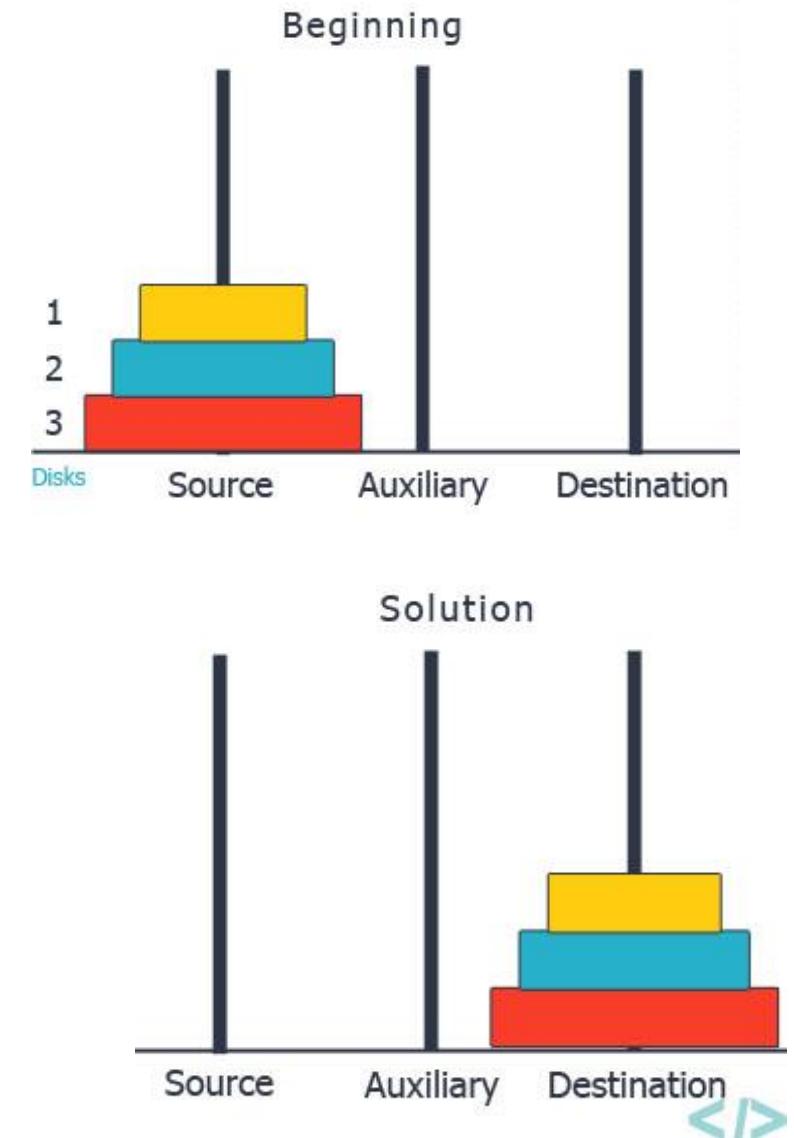


```
def fib(n):
```

Tower of Hanoi

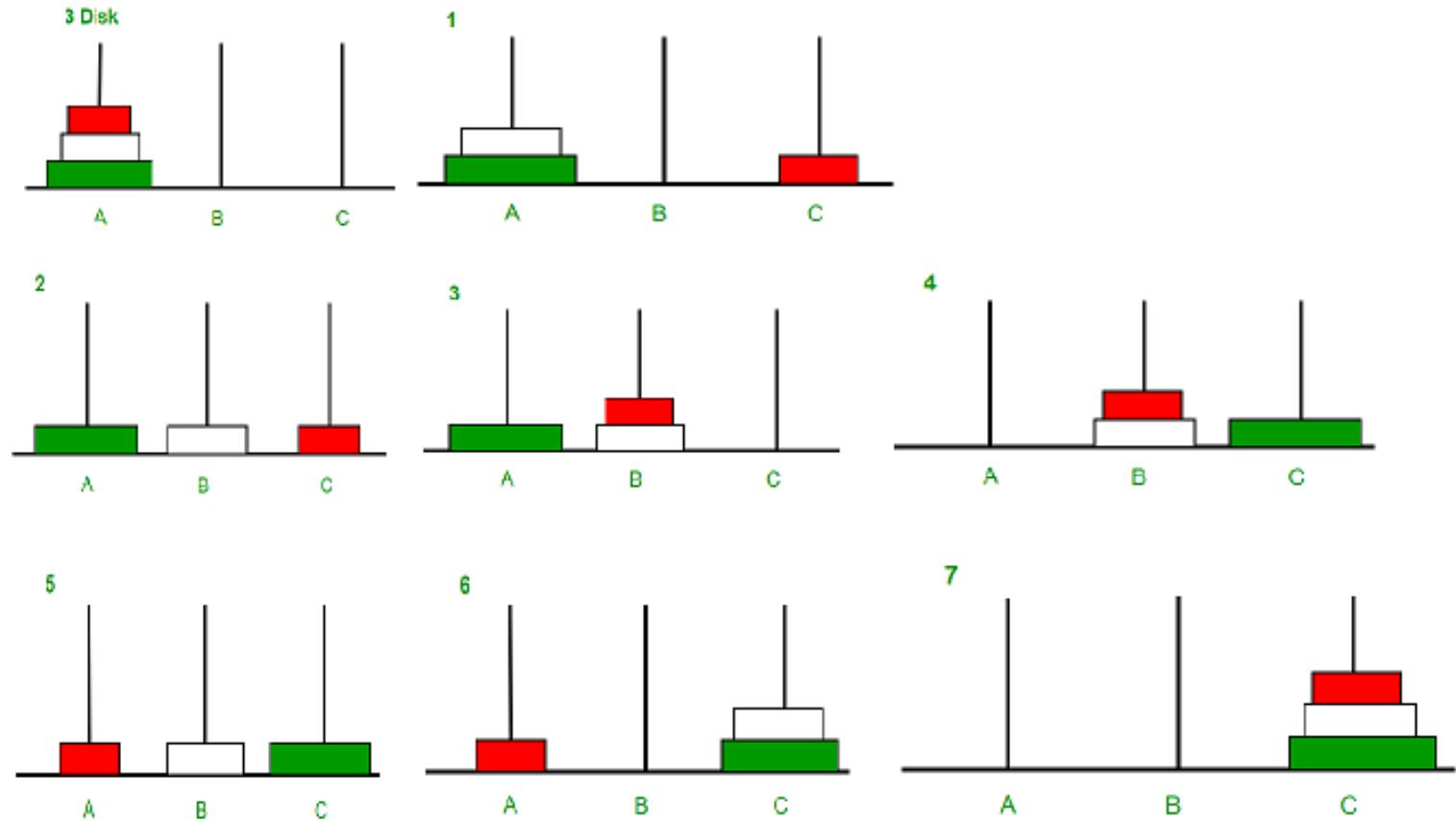
```
def hanoi(n, source, dest, aux):
    if n > 0:
        hanoi(n-1, source, aux, dest)
        print('Move disk', n, 'from',
              source, 'to', dest)
        hanoi(n-1, aux, dest, source)
```

```
hanoi(3, 'A', 'C', 'B')
```



Example: Tower of Hanoi

- Move disk 1 from A to C
- Move disk 2 from A to B
- Move disk 1 from C to B
- Move disk 3 from A to C
- Move disk 1 from B to A
- Move disk 2 from B to C
- Move disk 1 from A to C



Lambda Expressions

- A lambda expression is an anonymous function
- Allow us to define a function much more easily

```
>>> f = lambda x: x * x
>>> print(f(4))

>>> L = ['hello', 'World', 'hi', 'Bye']
>>> sorted(L)
>>> sorted(L, key=str.lower)
>>> sorted(L, key=len)
>>> sorted(L, key=lambda x: x[-1])
```

Why Functions?

- Make the program modular and readable
- Can be reused later
- You can even package them as a library (or a module)

Jin-Soo Kim
jinsoo.kim@snu.ac.kr

Systems Software &
Architecture Lab.
Seoul National University

Jan. 3 – 14, 2022



Python for Data Analytics

Python Sequence and Collections

Sequence vs. Collection

- Both can hold a bunch of values in a single "variable"
- Sequence
 - Sequence has a deterministic ordering
 - Index their entries based on the position
 - Strings, Lists, Tuples
- Collection
 - No ordering
 - Sets, Dictionaries

Lists

Lists

- Ordered collections of arbitrary objects (arrays of object references)
- Accessed by offset (items not sorted)
- Variable-length, heterogeneous, mutable, and arbitrarily nestable
- The most versatile and popular data type in Python

```
prime = [2, 3, 5, 7, 11]
a = [2, 'three', 3.0, 5, 'seven', 11.0]
b = [1, 3, 3, 3, 2, 2]
c = [ 1, [8, 9], 12]
emptylist = []
```

Concatenating/Replicating Lists

- *list1 + list2* : create a new list by adding two existing lists together
- *list * n*: create a new list by replicating the original list *n* times

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> d = a*3
>>> print(d)
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Referencing a List Element

- Just like strings, use an index specified in square brackets

Emma	Olivia	Ava	Isabella	Sophia
0	1	2	3	4

```
>>> names = ['Emma', 'Olivia', 'Ava', 'Isabella', 'Sophia']
>>> print(names[0])
Emma
>>> print(names[2])
Ava
```

Slicing a List

- $list[start : end : step]$
 - *start*: the starting index of the list
 - If omitted, the beginning of the list if *step* > 0 or the end of the list if *step* < 0
 - *end*: the ending index of the list (up to but not including)
 - If omitted, the end of the list if *step* > 0 or the beginning of the list if *step* < 0
 - *step*: the number of elements to skip + 1 (1 if omitted)
 - *start* and *stop* can be a negative number, which means it counts from the end of the array

Emma	Olivia	Ava	Isabella	Sophia
0	1	2	3	4
-5	-4	-3	-2	-1

```
names = ['Emma', 'Olivia', 'Ava', 'Isabella', 'Sophia']
```

Slicing a List: Example

Emma	Olivia	Ava	Isabella	Sophia
0	1	2	3	4
-5	-4	-3	-2	-1

```
>>> names = ['Emma', 'Olivia', 'Ava', 'Isabella', 'Sophia']
>>> print(names[::2])
['Emma', 'Ava', 'Sophia']
>>> print(names[3:])
['Isabella', 'Sophia']
>>> print(names[-3:])
['Ava', 'Isabella', 'Sophia']
>> print(names[::-1])
['Sophia', 'Isabella', 'Ava', 'Olivia', 'Emma']
```

Slicing a List: Example (cont'd)

Emma	Olivia	Ava	Isabella	Sophia
0	1	2	3	4
-5	-4	-3	-2	-1

```
>>> names = ['Emma', 'Olivia', 'Ava', 'Isabella', 'Sophia']
>>> print(names[])
# WRONG!

>>> print(names[:])

>>> print(names[::])

>>> print(names[-10:10])
```

Getting the List Size

- `len(list)`
 - Return the number of elements in the list
 - Actually, `len()` tells us the number of elements of any sequence or collection (e.g., string, list, tuple, dict, set, ...)

```
>>> greet = 'Hello Spam'  
>>> print(len(greet))  
10  
>>> x = [ 1, 2, 'spam', 99, 'ham' ]  
>>> print(len(x))  
5
```

Lists are Mutable

- $list[i] = x$: change the i -th element of the list to x
- $list[i:j] = list2$: replace the elements from i -th to j -th with the new list

```
>>> names = ['Emma', 'Olivia', 'Ava', 'Isabella', 'Sophia']
>>> names[2] = 'Eve'
>>> print(names)
['Emma', 'Olivia', 'Eve', 'Isabella', 'Sophia']
>>> names[1:4] = [ 'Charlotte', 'Mia', 'Amelia' ]
>>> print(names)
['Emma', 'Charlotte', 'Mia', 'Amelia', 'Sophia']
>>> names[5:] = [ 'Ella', 'Avery' ]
>>> print(names)
['Emma', 'Charlotte', 'Mia', 'Amelia', 'Sophia', 'Ella', 'Avery']
```

Useful Functions on a List

- ***list.count(x)***
 - Return the number of times *x* appears in the list
- ***max(list)***
 - Return the maximum value in the list
- ***min(list)***
 - Return the minimum value in the list

```
>>> numbers = [3, 1, 12, 14, 12, 6, 1, 12]
>>> print(numbers.count(12))
3
>>> print(min(numbers), max(numbers))
1 14
```

Finding an Element

- `list.index(x[, start[, end]])`
 - Return zero-based index in the list of the first item whose value is equal to `x`
 - `IndexError` if there is no such item
 - The optional `start` and `end` arguments are used to limit the search to a particular subsequence of the list

```
>>> names = ['Emma', 'Olivia', 'Ava', 'Emma', 'Isabella']
>>> print(names.index('Emma'))
0
>>> print(names.index('Emma', 1, 4))
3
>>> print(names.index('Isabella', 3))
4
```

Building a List

- ***list.append(x)***

- Add an item *x* to the existing list
- The list stays in order and new elements are appended at the end of the list

```
>>> menu = list()
>>> print(menu)
[]
>>> menu.append('spam')
>>> menu.append('ham')
>>> menu.append('spam')
>>> print(menu)
['spam', 'ham', 'spam']
```

Extending Lists

- ***list.extend(list2)***

- Extend the list by appending all the items from the other list
- Faster than a series of **append()**'s

```
>>> menu = ['spam', 'ham']
>>> menu.extend(['egg', 'sausage'])
>>> print(menu)
['spam', 'ham', 'egg', 'sausage']
>>> menu.extend(['spam']*3)
>>> print(menu)
['spam', 'ham', 'egg', 'sausage', 'spam', 'spam', 'spam']
```

Inserting an Element

- `list.insert(i, x)`
 - Insert an item at a given position i (0 means the front of the list)

```
>>> menu = ['spam', 'ham']
>>> print(menu)
['spam', 'ham']
>>> menu.insert(1, 'egg')
>>> print(menu)
['spam', 'egg', 'ham']
>>> menu.insert(0, 'bacon')
>>> print(menu)
['bacon', 'spam', 'egg', 'ham']
```

Removing Elements

- `list.remove(x)`
 - Remove the first item from the list whose value is equal to `x`
- `del list[i]` or `del list[i:j]`
 - Deletes `i`-th element (or from `i`-th to `j-1`th elements) from the list

```
>>> menu = ['spam', 'ham', 'egg', 'sausage', 'bacon']
>>> menu.remove('ham')
>>> print(menu)
['spam', 'egg', 'sausage', 'bacon']
>>> del menu[1:3]
>>> print(menu)
['spam', 'bacon']
```

Popping an Element

- ***list.pop([i])***

- Remove the item at the given *i*-th position in the list, and return it
- If no index is specified, it removes and returns the **last** item in the list

```
>>> menu = ['spam', 'ham', 'egg', 'sausage', 'bacon']
>>> print(menu.pop(1))
ham
>>> print(menu.pop())
bacon
>>> print(menu.pop())
sausage
>>> print(menu.pop())
egg
```

Reversing the List

- *list.reverse()*

- Reverse the elements of the list *in place*
- cf. *list[::-1]* returns the new list with the elements in reversed order

```
>>> names = ['Emma', 'Olivia', 'Ava', 'Isabella', 'Sophia']
>>> names.reverse()
>>> print(names)
['Sophia', 'Isabella', 'Ava', 'Olivia', 'Emma']
>>> new_names = names[::-1]
>>> print(new_names)
['Emma', 'Olivia', 'Ava', 'Isabella', 'Sophia']
```

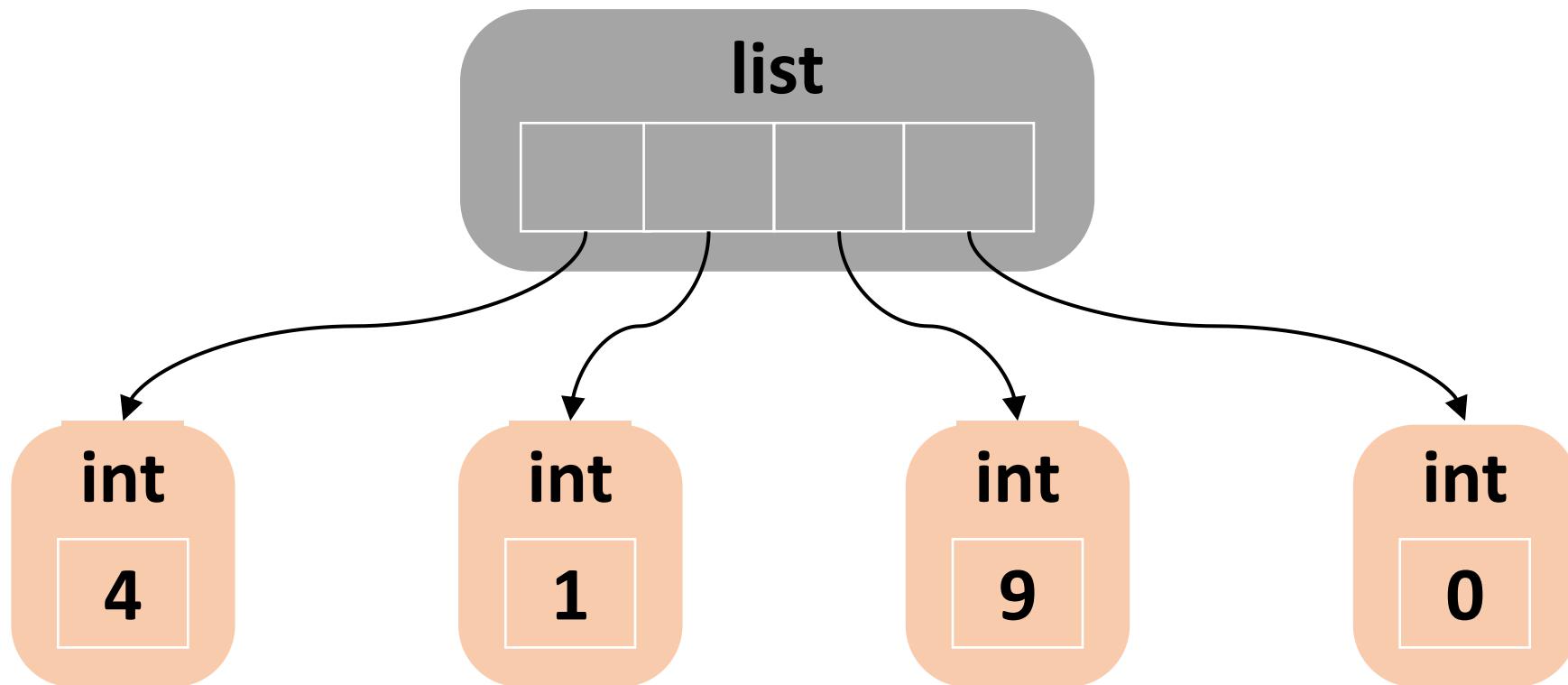
Membership Operators

- **in (not in) operator**
 - Check if an item is in a list or not
 - Returns True or False

```
>>> menu = ['spam', 'ham']
>>> 'spam' in menu
True
>>> 'ham' not in menu
False
>>> 'egg' in menu
False
```

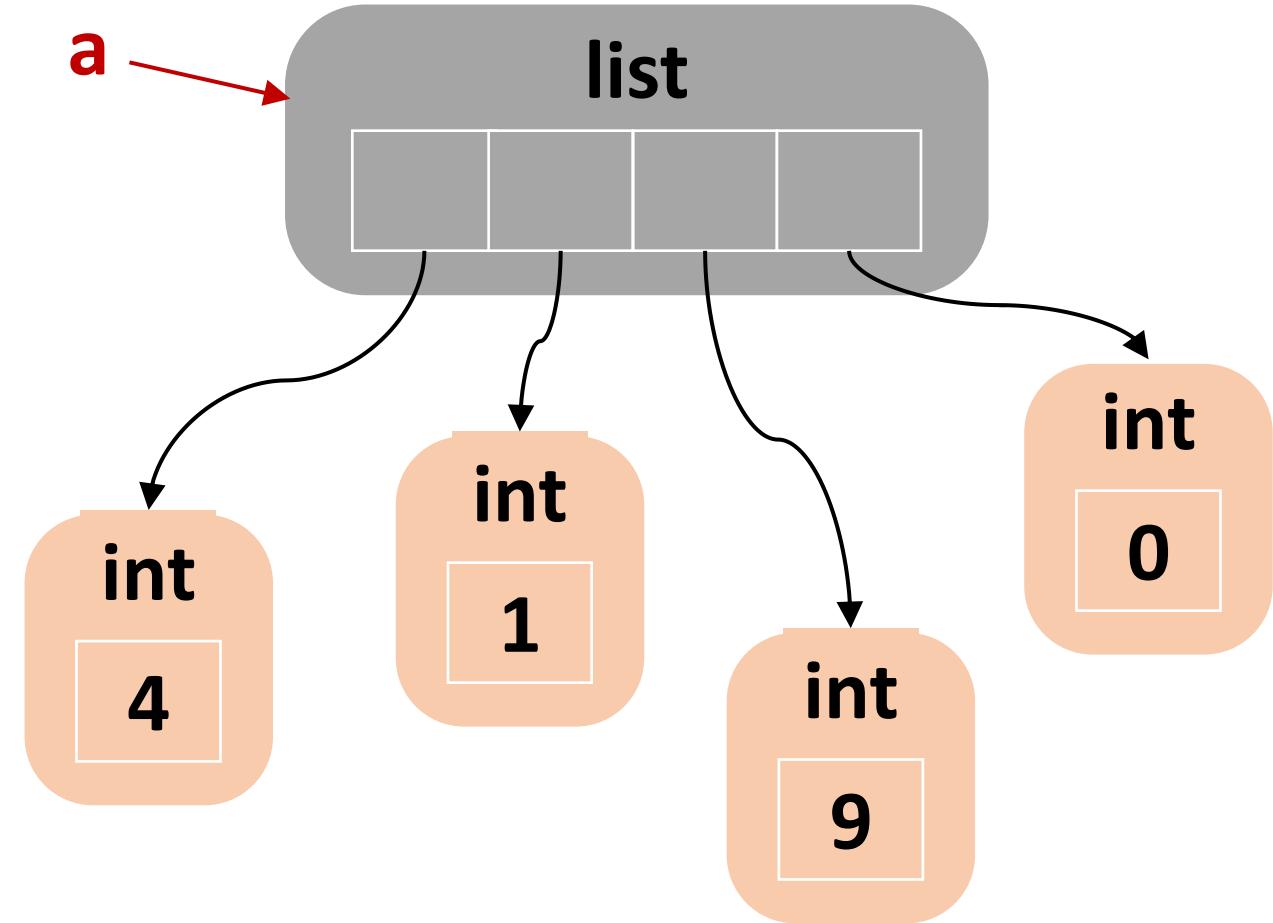
Implementing Lists

- A list has an array of references to other objects



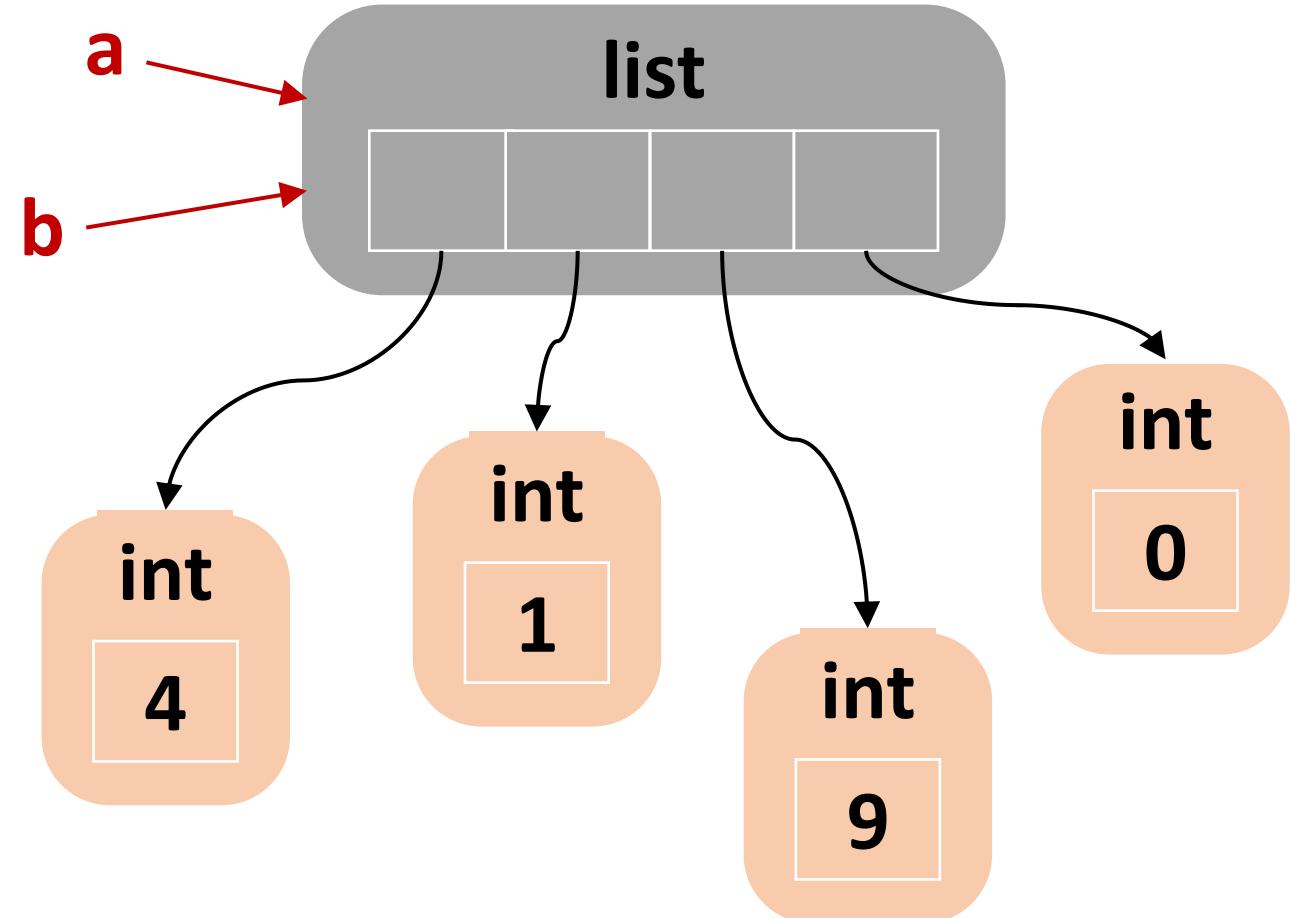
Assigning a List

```
>>> a = [4, 1, 9, 0]
```



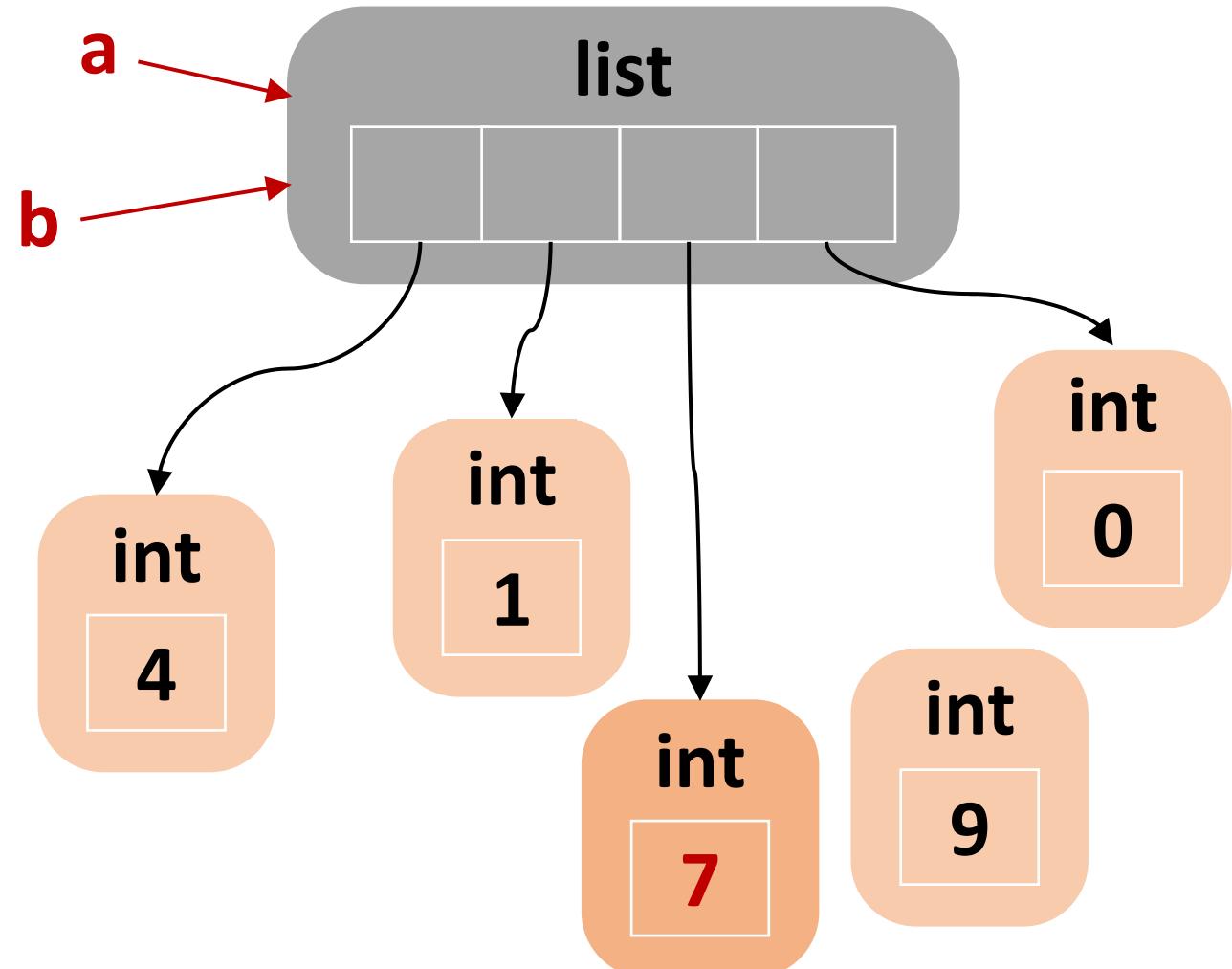
Assigning a List

```
>>> a = [4, 1, 9, 0]  
>>> b = a
```



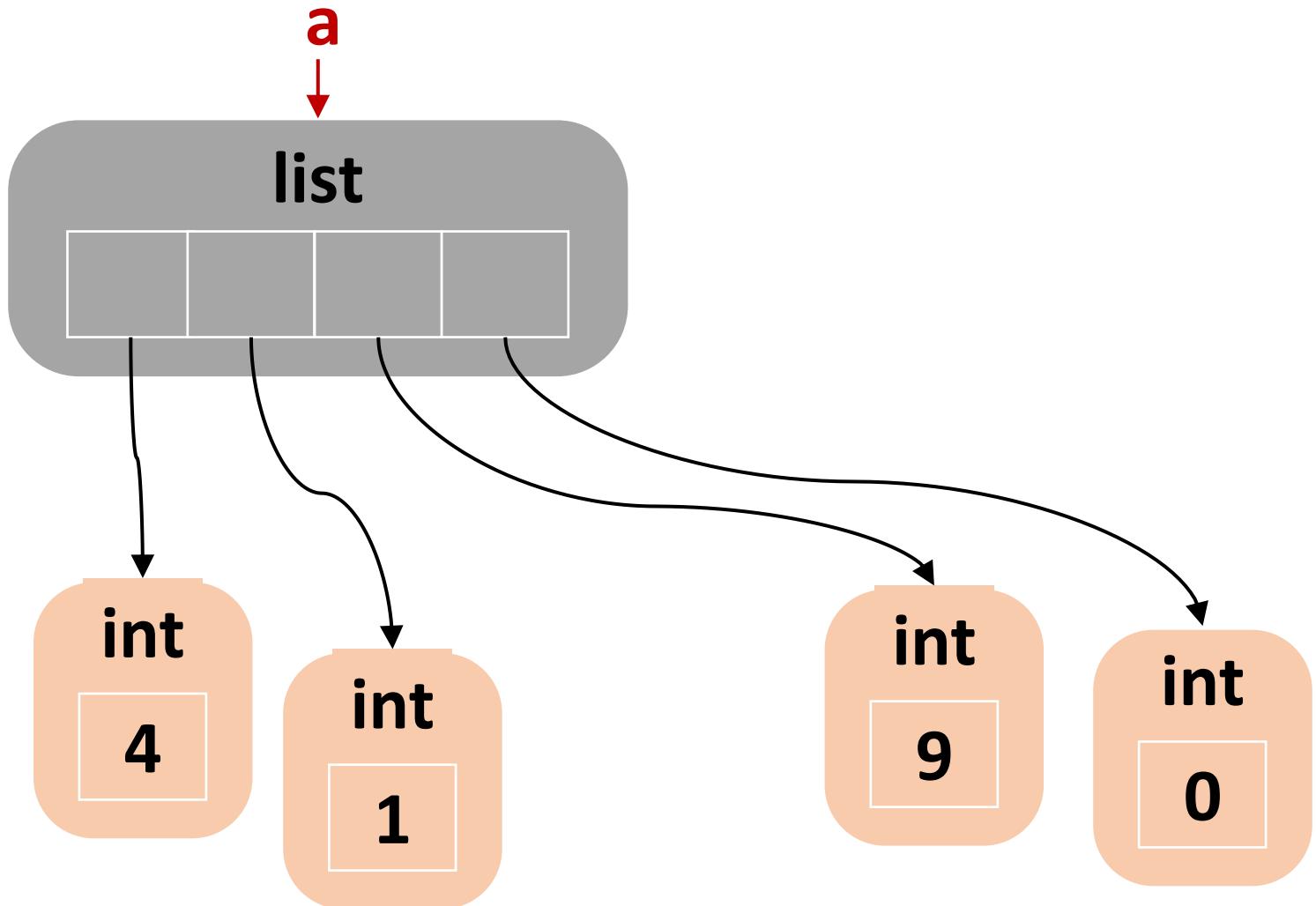
Changing a List Element

```
>>> a = [4, 1, 9, 0]
>>> b = a
>>> a[2] = 7
>>> print(a)
[4, 1, 7, 0]
>>> print(b)
[4, 1, 7, 0]
```



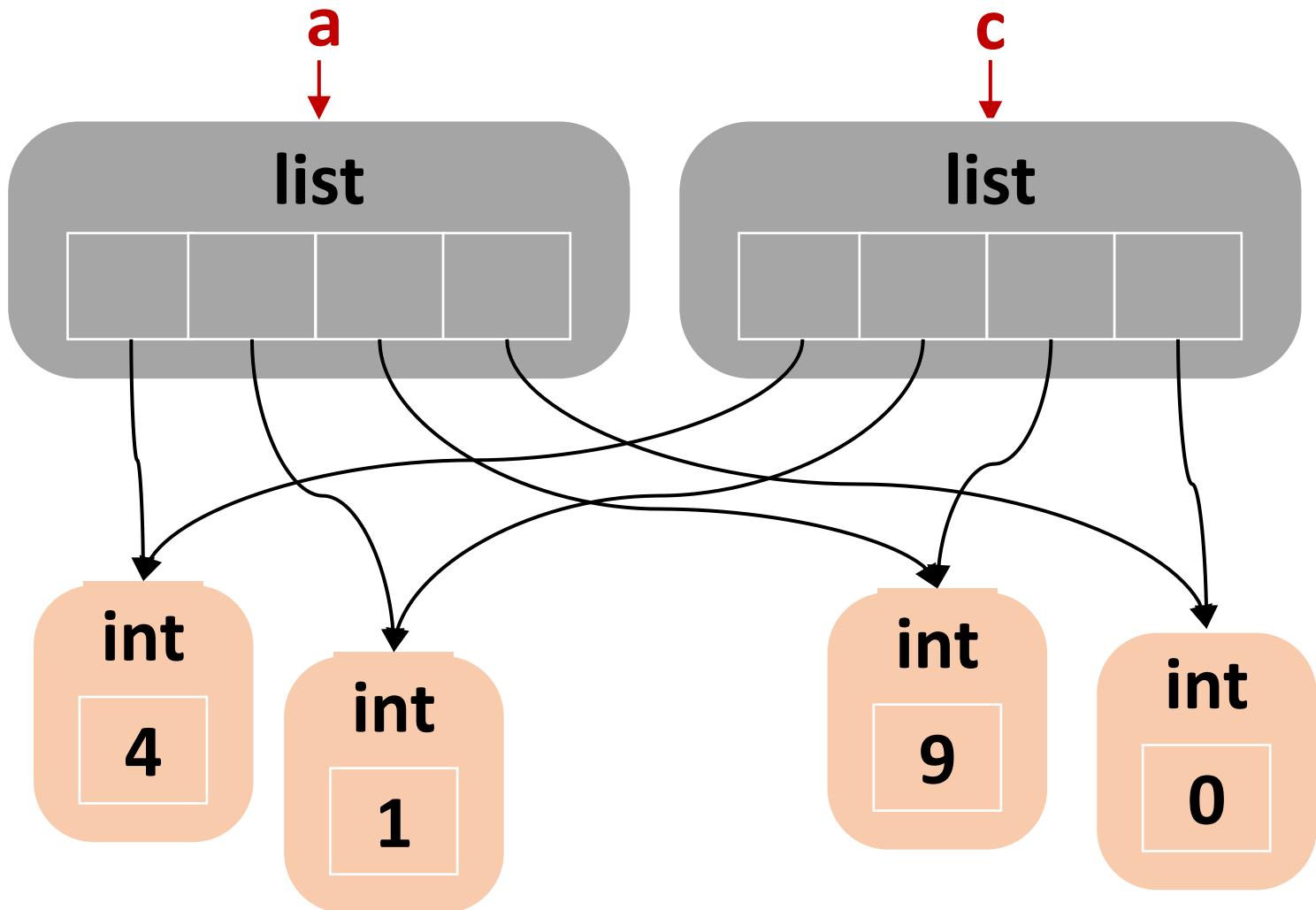
Copying a List

```
>>> a = [4, 1, 9, 0]
```



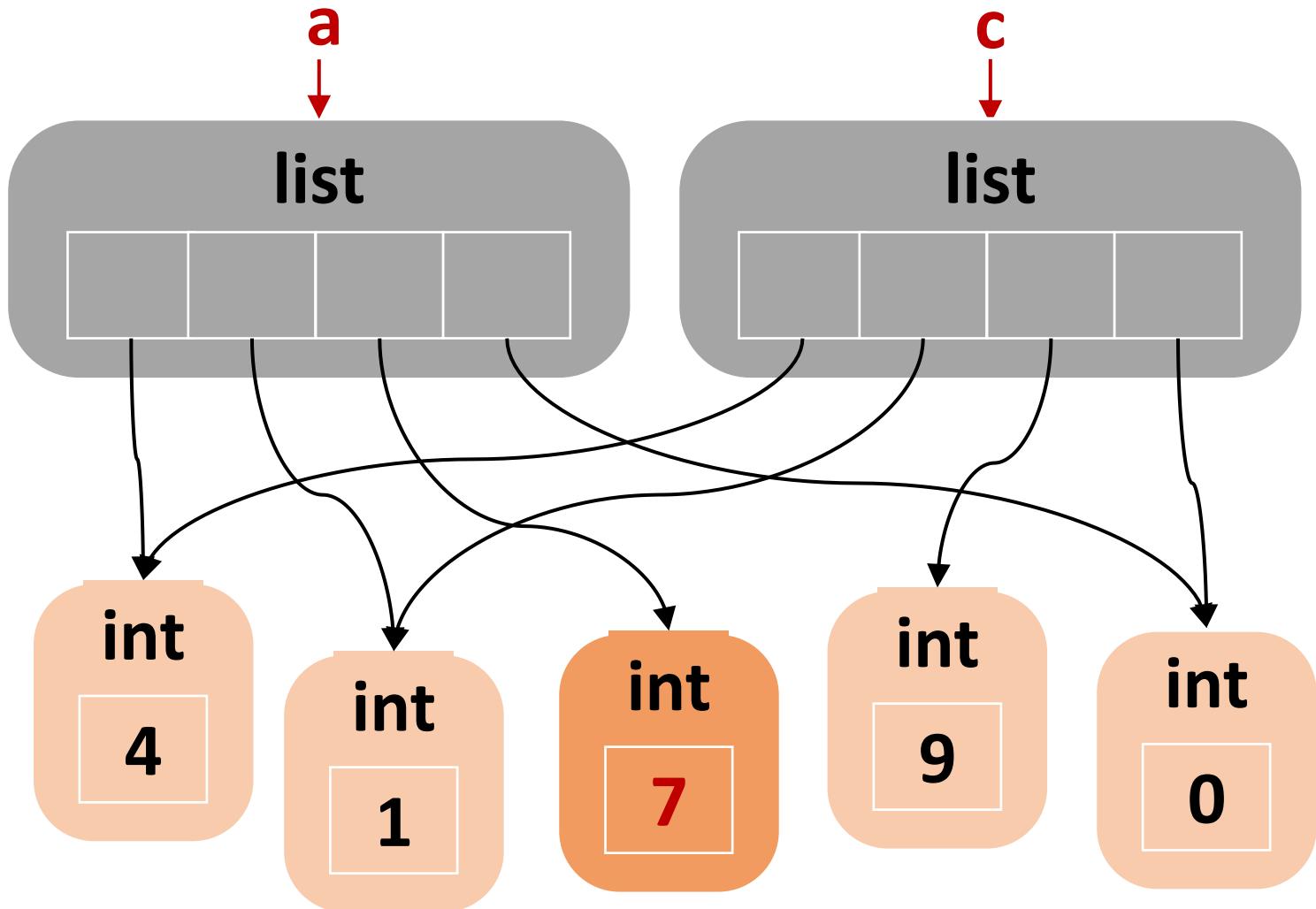
Copying a List

```
>>> a = [4, 1, 9, 0]  
>>> c = a[:]
```



Copying a List

```
>>> a = [4, 1, 9, 0]
>>> c = a[:]
>>> a[2] = 7
>>> print(a)
[4, 1, 7, 0]
>>> print(c)
[4, 1, 9, 0]
```



Copying a List: Other Ways

- Using list slicing

```
>>> a = [1, 2, 3, 4]  
>>> b = a[:]
```

- Using *

```
>>> a = [1, 2, 3, 4]  
>>> b = a*1
```

- Using list()

```
>>> a = [1, 2, 3, 4]  
>>> b = list(a)
```

- Using copy module

```
>>> import copy  
>>> a = [1, 2, 3, 4]  
>>> b = copy.copy(a)
```

Sorting Elements in a List (I)

- `list.sort([key], [reverse])`
 - Sort the elements in the list
 - `key`: a function with a single argument (used to extract a comparison key)
 - `reverse`: if `True`, the list elements are sorted in the reverse order
 - `list.sort()` changes the list `in place`, but don't return the list as a result

```
>>> names = ['Emma', 'Olivia', 'Ava', 'Isabella', 'Sophia']
>>> names.sort()
>>> print(names)
['Ava', 'Emma', 'Isabella', 'Olivia', 'Sophia']
>>> names.sort(reverse=True)
['Sophia', 'Olivia', 'Isabella', 'Emma', 'Ava']
```

Sorting Elements in a List (2)

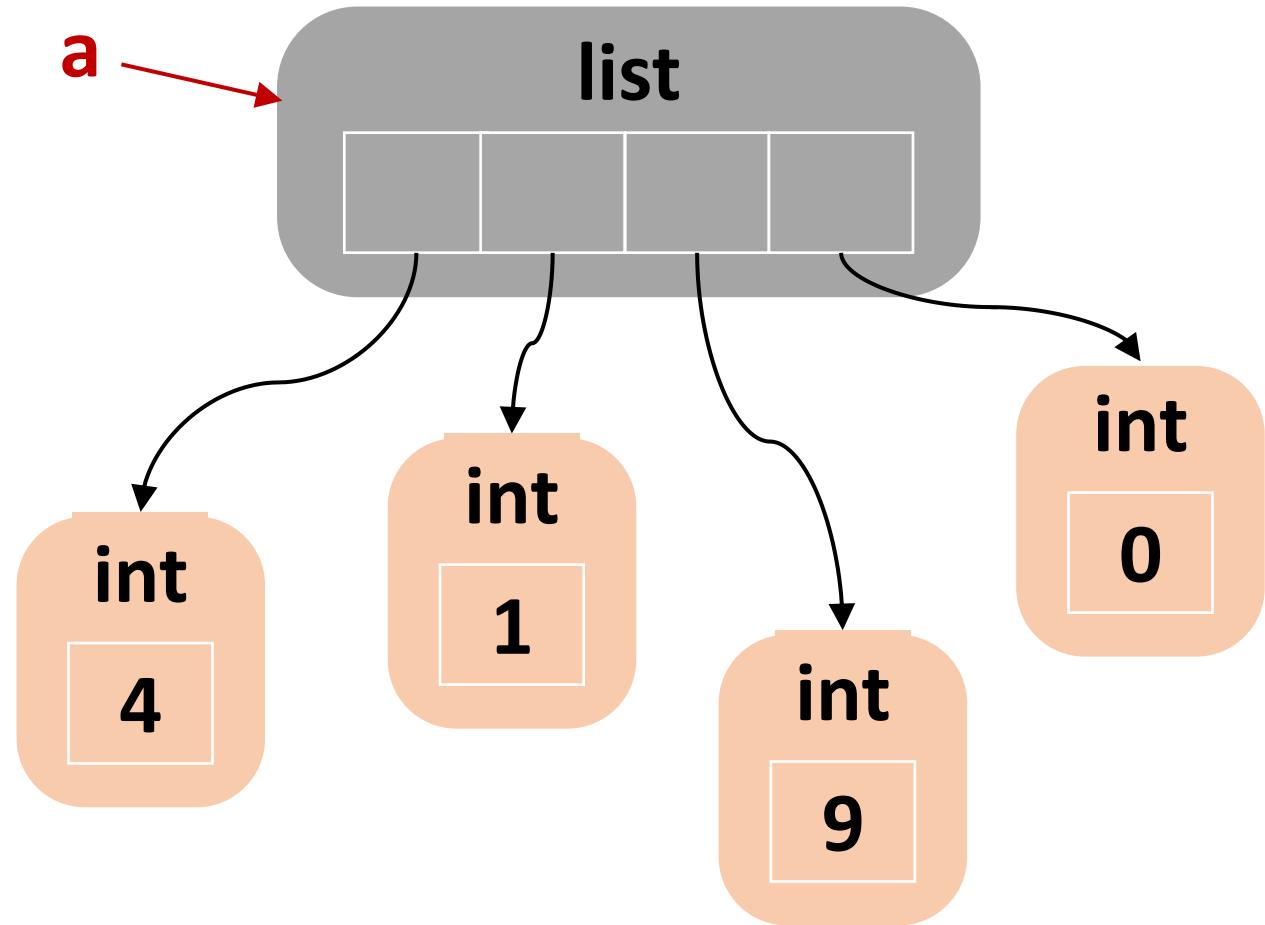
- **sorted(*iterable*, [*key*], [*reverse*])**

- Sort the elements in the list
- *key*: a function with a single argument (used to extract a comparison key from each element)
- *reverse*: if **True**, the list elements are sorted in the reverse order
- **sorted()** returns a new list!

```
>>> names = ['Emma', 'Olivia', 'Ava', 'Isabella', 'Sophia']
>>> sorted_names = sorted(names)
>>> print(sorted_names)
['Ava', 'Emma', 'Isabella', 'Olivia', 'Sophia']
```

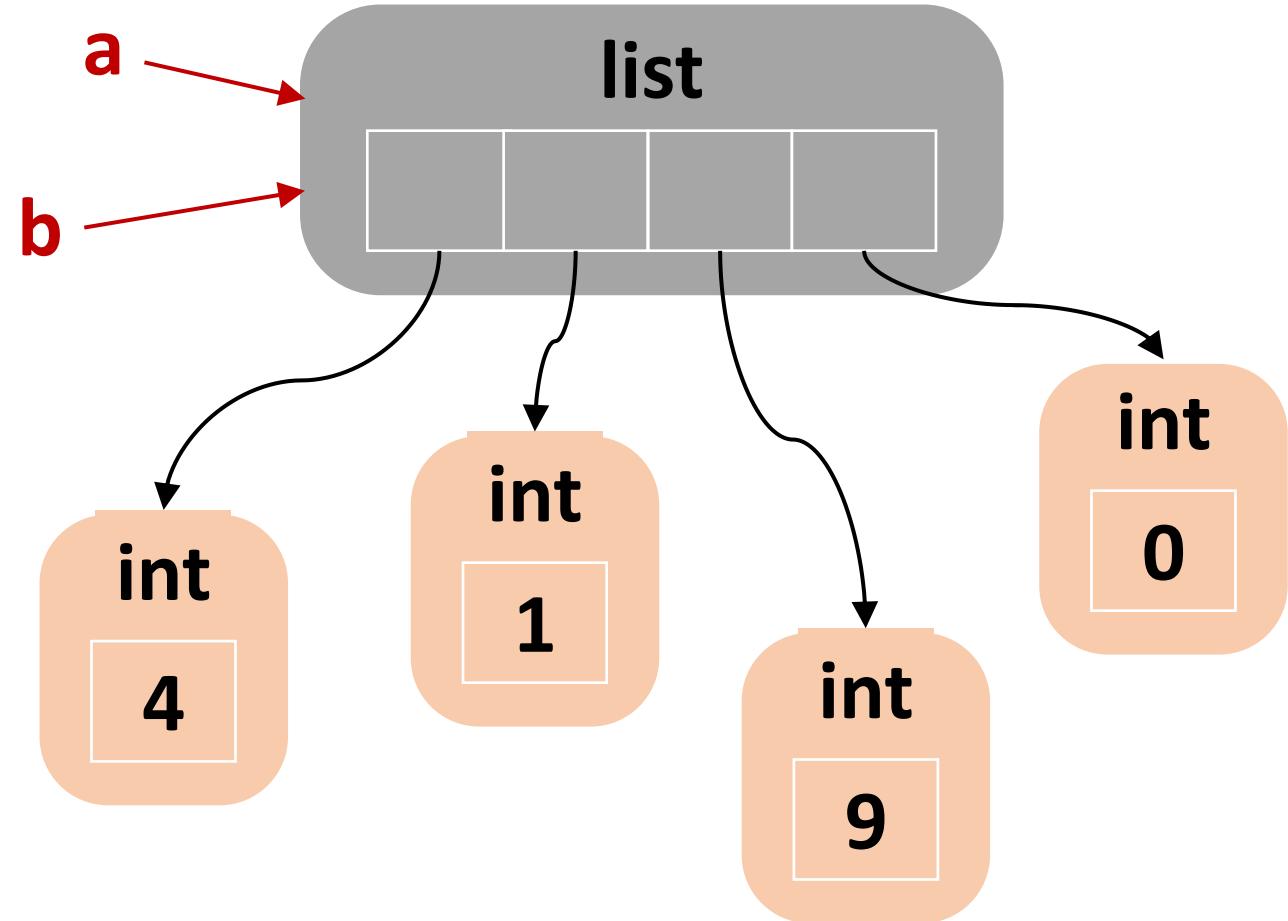
list.sort()

```
>>> a = [4, 1, 9, 0]
```



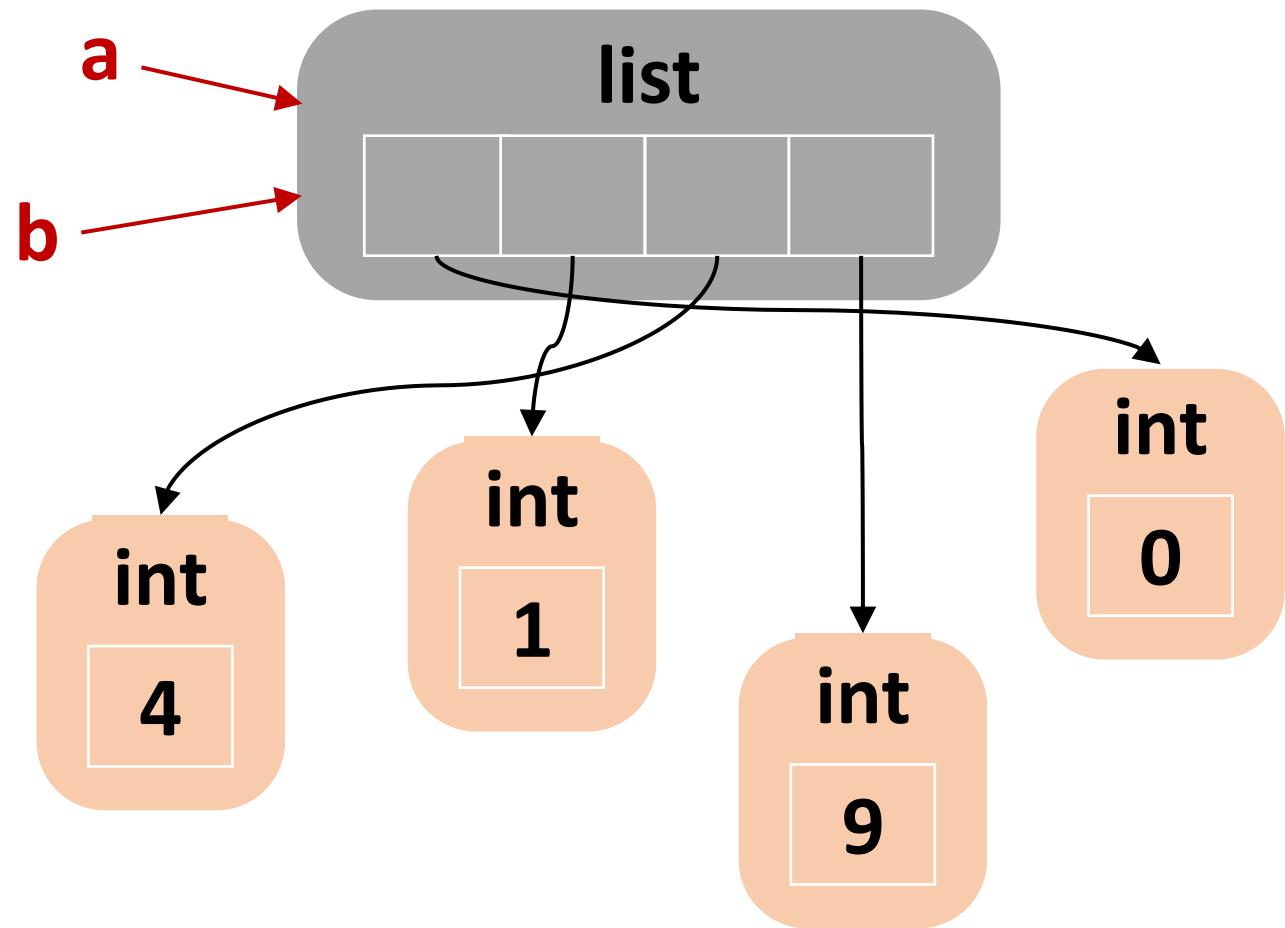
list.sort()

```
>>> a = [4, 1, 9, 0]  
>>> b = a
```



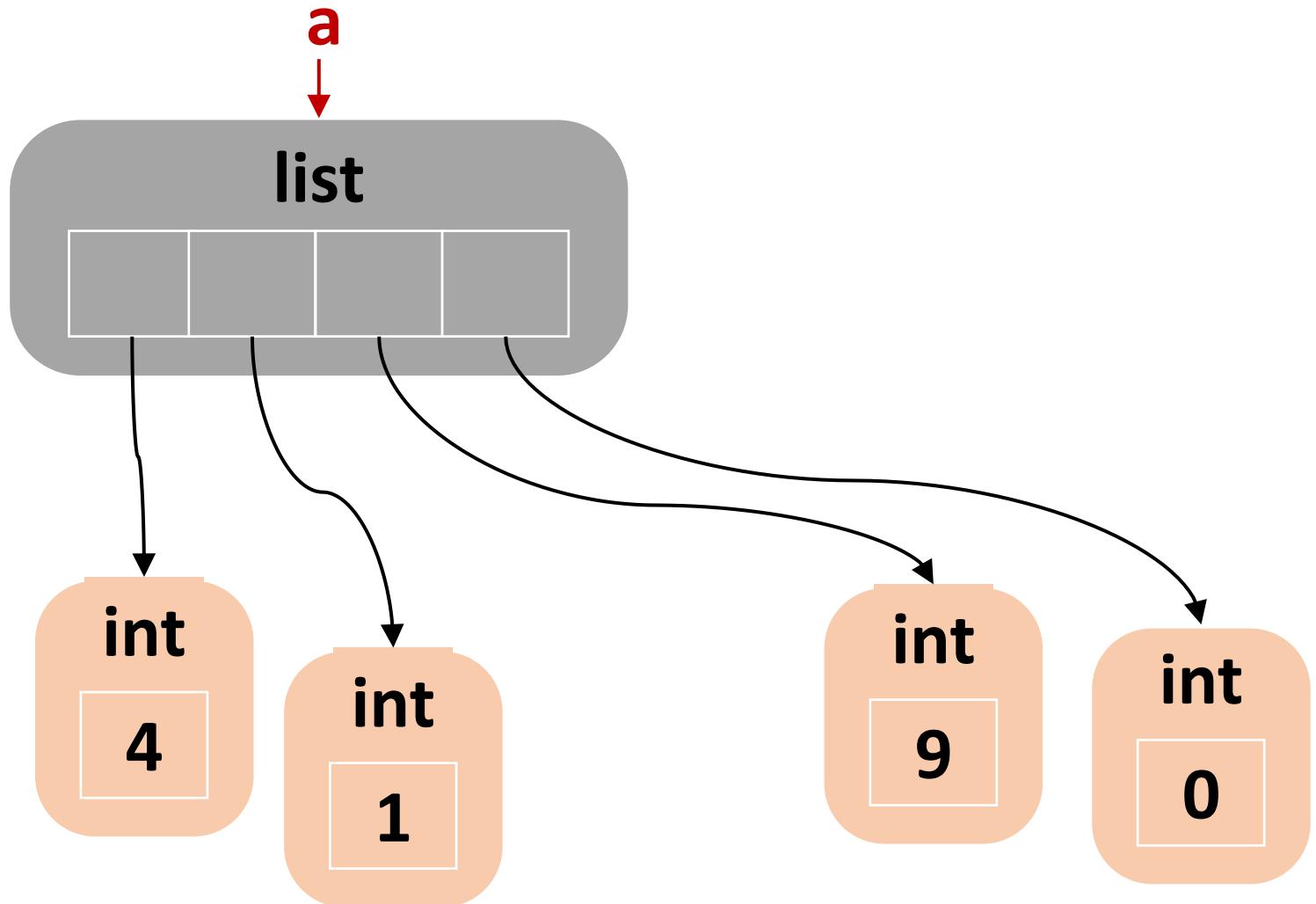
list.sort()

```
>>> a = [4, 1, 9, 0]
>>> b = a
>>> b.sort()
>>> print(b)
[0, 1, 4, 9]
>>> print(a)
[0, 1, 4, 9]
```



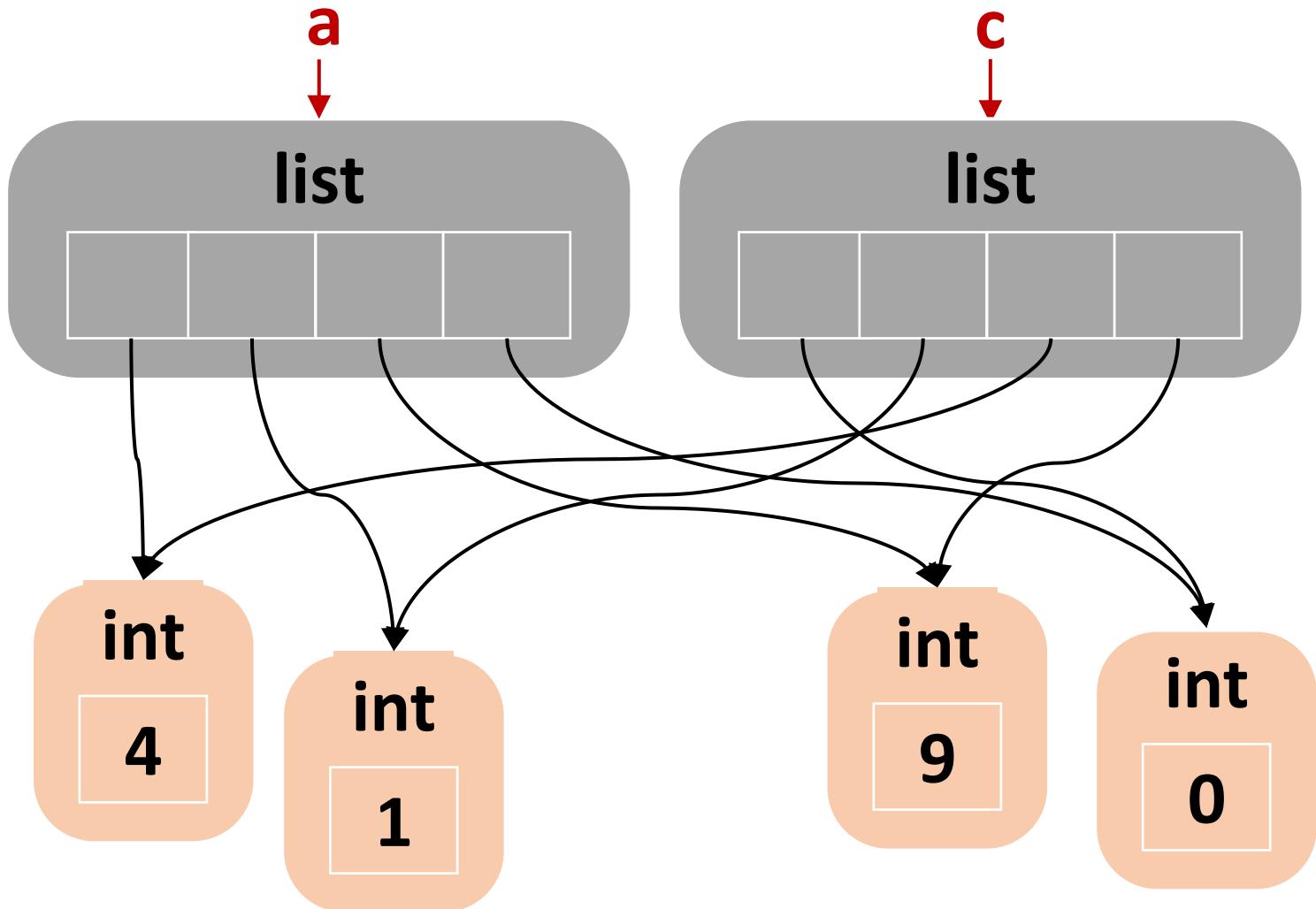
sorted(list)

```
>>> a = [4, 1, 9, 0]
```



sorted(list)

```
>>> a = [4, 1, 9, 0]
>>> c = sorted(a)
>>> print(c)
[0, 1, 4, 9]
>>> print(a)
[4, 1, 9, 0]
```



Multi-dimensional Lists

- Lists within lists

```
>>> M = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
>>> M[1]  
[4, 5, 6]  
>>> M[1][2]  
6  
>>> M[1:]  
[[4, 5, 6], [7, 8, 9]]  
>>> M[2] = [0, 0, 0]  
>>> M  
[[1, 2, 3], [4, 5, 6], [0, 0, 0]]
```

M x N Matrix using Lists

- M x N Matrix: M rows and N columns

```
>>> M = [[1, 2, 3], [4, 5, 6]]      # 2 x 3 matrix
>>> len(M)                          # number of rows
2
>>> len(M[0])                      # number of columns
3
>>> M[1][0]                         # element M(1,0)
4
>>> M[0][0]+M[0][1]+M[0][2]        # sum of first row
6
>>> M[0][1]+M[1][1]                 # sum of second column
7
```

Accessing Multi-dimensional Lists

```
M = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]  
  
for row in M:  
    print(row)  
  
for i in range(len(M)):  
    for j in range(len(M[i])):  
        print(M[i][j], end=' ')  
  
for row in M:  
    for col in row:  
        print(col, end=' ')
```

List Comprehension

- Provides a concise way to create lists

```
new_list = list()
for i in range(10):
    if i % 2 == 0:
        new_list.append(i*i)
```

```
new_list = [ i*i for i in range(10) if i % 2 == 0 ]
```

List Comprehension: General Form

```
new_list = [ expression(i) for i in sequence if filter(i) ]
```

```
new_list = list()
for i in sequence:
    if filter(i):
        new_list.append(expression(i))
```

Simple Lists

```
>>> [0] * 10
[0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> [ i for i in range(10, 20) ]          # list(range(10, 20))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> [ x**2 for x in range(10) ]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> [ i for i in range(100) if i % 3 == 0 and i % 5 == 1 ]
[6, 21, 36, 51, 66, 81, 96]
>>> [ random.randint(0, 99) for _ in range(10) ]    # import random
[50, 15, 22, 3, 88, 50, 71, 63, 40, 62]
```

Lists From Lists

```
>>> [ item*3 for item in [2, 3, 5] ]  
[6, 9, 15]  
  
>>> [ i if i > 0 else 0 for i in [-2, 5, 4, -7] ]  
[0, 5, 4, 0]  
  
>>> [ word[0] for word in ['hello', 'world', 'spam'] ]  
['h', 'w', 's']  
  
>>> [ x.upper() for x in ['spam', 'ham', 'egg'] ]  
['SPAM', 'HAM', 'EGG']  
  
>>> [ x + y for x in [10, 30, 50] for y in [20, 40, 60] ]  
[30, 50, 70, 50, 70, 90, 70, 90, 110]
```

Nested Lists

```
>>> [ [0]*4 for _ in range(3) ]                      # [[0]*4]*3 ??  
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]  
>>> [ [i for i in range(4)] for _ in range(3) ]  
[[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]]  
>>> [ [x, y] for x in [1, 2, 3] for y in [7, 8, 9] ]  
[[1, 7], [1, 8], [1, 9], [2, 7], [2, 8], [2, 9], [3, 7], [3, 8], [3, 9]]  
>>> [ [[x, y] for x in [1, 2, 3]] for y in [7, 8] ]  
[[[1, 7], [2, 7], [3, 7]], [[1, 8], [2, 8], [3, 8]]]
```

Nested Lists (2)

```
>>> matrix = [ [i for i in range(j, j+4)] for j in range(0, 12, 4)]  
>>> matrix  
[[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]  
>>> [ e for row in matrix for e in row ]  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]  
>>> for row in matrix:  
...     print('\t'.join([str(e) for e in row]))  
0      1      2      3  
4      5      6      7  
8      9      10     11  
>>> print('\n'.join(['\t'.join([str(e) for e in row]) for row in matrix]))
```

zip()

- **zip(*iterables)**

- Make an iterator that aggregates elements from each of the *iterables*
- The * operator can be used to unzip a list

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> xy = list(zip(x,y))
>>> xy
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*xy)
>>> x2
(1, 2, 3)
>>> list(y2)
[4, 5, 6]
```

map()

- **map(func, *iterables)**

- Returns a map object (which is an iterator) of the results after applying the given function *func* to each item of a given *iterables*

```
>>> def double(x):
...     return x + x
>>> n = [1, 2, 3, 4]
>>> print(list(map(double, n)))
[2, 4, 6, 8]
>>> print(list(map(lambda x: x + x, n)))
[2, 4, 6, 8]
>>> n2 = [10, 20, 30, 40]
>>> print(list(map(lambda x, y: x + y, n, n2)))
[11, 22, 33, 44]
```

Tuples

Tuples

- Ordered collection of arbitrary objects
- Accessed by offset
- Immutable sequence
- Fixed-length, heterogeneous, and arbitrarily nestable

```
menu = (1, 2, 5, 9)
a = 1, 2, 5, 9
b = (0, 'ham', 3.14, 99)
c = ('a', ('x', 'y'), 'z')
emptytuple = () # tuple()
```

Tuples are like Lists

- Another kind of "sequence"
- Elements are indexed starting at 0

```
>>> num = (4, 1, 9)  
>>> print(num[2])  
9  
>>> print(len(num))  
4  
>>> print(max(num))  
9  
>>> print(min(num))  
1
```

```
>>> for i in num:  
...     print(i)  
4  
1  
9  
>>> print(num + ('a', 'b'))  
(4, 1, 9, 'a', 'b')  
>>> print(num * 2)  
(4, 1, 9, 4, 1, 9)
```

Tuples are Immutable

- Unlike a list, once you create a tuple, you **cannot alter** its contents
- Similar to a string

Lists

```
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print(x)
[9, 8, 6]
```

Strings

```
>>> s = 'ABC'
>>> s[2] = 'D'
Traceback (most recent
call last):
  File "<stdin>", line 1,
in <module>
TypeError: 'str' object
does not support item
assignment
```

Tuples

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback (most recent
call last):
  File "<stdin>", line 1,
in <module>
TypeError: 'tuple' object
does not support item
assignment
```

Things not to do with Tuples

```
>>> x = (4, 1, 9, 0)
>>> x.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
>>> x.append(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> x.reverse()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'reverse'
```

Tuples and Assignments

- We can also put a tuple on the left-hand side of an assignment statement
- We can even omit the parentheses
- Can be used to return multiple values in a function

```
>>> (x, y) = (4, 'spam')
>>> print(x)
4
>>> y = 1
>>> x, y = y, x
>>> print(x, y)
1 4
```

```
def ret2(a):
    return min(a), max(a)

a, b = ret2([4, 1, 9, 0])
```

Tuples are Comparable

- The comparison operators work with tuples and other sequences
- If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ('Jones', 'Sally') < ('Jones', 'Sam')
True
>>> ('Jones', 'Sally') > ('Adams', 'Sam')
True
```

Sets

Sets

- Unordered collection of unique immutable objects
- Not ordered (cannot be accessed by offset)
- Items can be added or removed
- Variable-length and heterogeneous, but not nestable
- Support operations corresponding to mathematical set theory

```
choice = {1, 2, 5, 9}
s = {'a', 'b', 'c', 'c'} # ???
t = {0, 'ham', 3.14, 99}
emptyset = set()          # wrong: s = {}
```

Set Manipulation Operations

- `s.pop()` remove and return an arbitrary element from s
- `s.clear()` remove all elements from set s
- `s.add(x)` add element x to set s
- `s.remove(x)` remove x from set s
raise `KeyError` if not present
- `s.discard(x)` remove x from set s if present

Mathematical Set Operations

- `s.issubset(t)` True if $s \subset t$ (or $s \leq t$)
- `s.issuperset(t)` True if $s \supset t$ (or $s \geq t$)

- `s.union(t)` return $s \cup t$ (or $s \mid t$)
- `s.intersection(t)` return $s \cap t$ (or $s \& t$)
- `s.difference(t)` return $s - t$
- `s.symmetric_difference(t)` return $t - s$

Set Membership Check is Fast!

```
import time

N = 10000
a = set(range(0, N, 2))
count = 0
start = time.time()
for x in range(N):
    if x in a:
        count += 1
end = time.time()
print(f'elapsed time: {end-start:.6f} sec')
```

Dictionaries

Dictionaries

- Unordered (Ordered since 3.7) collections of arbitrary objects
- Store key-value pairs: accessed by key, not offset
- Variable-length, heterogeneous, and arbitrarily nestable
- Python's most powerful data structure

```
menu = {'spam':9.99, 'egg':0.99}  
a = {1:'a', 1:'b', 2:'a'}  
b = {'food':{'ham':1, 'egg':2}}  
c = {'food':['spam', 'ham', 'egg']}  
emptydict = {} # dict()
```

Lists vs. Dictionaries

- Dictionaries are like lists except that they use **keys** instead of numbers to look up values

```
>>> lst = list()  
>>> lst.append(21)  
>>> lst.append(1)  
>>> print(lst)  
[21, 1]  
>>> lst[0] = 23  
>>> print(lst)  
[23, 1]
```

```
>>> dct = dict()  
>>> dct['age'] = 21  
>>> dct['course'] = 1  
>>> print(dct)  
{'course': 1, 'age': 21}  
>>> dct['age'] = 23  
>>> print(dct)  
{'course': 1, 'age': 23}
```

Counters with a Dictionary

- One common use of dictionaries is **counting** how often we see something

```
>>> lastname = dict()  
>>> lastname['kim'] = 1  
>>> lastname['lee'] = 1  
>>> print(lastname)  
{'kim': 1, 'lee': 1}  
>>> lastname['kim'] = lastname['kim'] + 1  
>>> print(lastname)  
{'kim': 2, 'lee': 1}
```

Dictionary Tracebacks

- It is an error to reference a key which is not in the dictionary
- We can use the **in** operator to see if a key is in the dictionary

```
>>> lastname = dict()  
>>> lastname['kim'] = 1  
>>> print(lastname['park'])  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
KeyError: 'park'  
>>> 'kim' in lastname  
True  
>>> 'park' in lastname  
False
```

Counting with the `in` Operator

- When we encounter a new name, we need to add a new entry in the dictionary
- If this is the second or later time we have seen the name, we simply add one to the count in the dictionary under that name

```
counts = dict()
names = ['kim', 'lee', 'park', 'kim', 'park', 'jang']
for name in names:
    if name not in counts:
        counts[name] = 1
    else:
        counts[name] = counts[name] + 1
print(counts)
```

Counting with `get()`

- `dict.get(key, [default])`

- Return the value of `key` if `key` is in the dictionary, else `default`
- If `default` is not given, it defaults to `None`
- Never raises a `KeyError`

```
counts = dict()
names = ['kim', 'lee', 'park', 'kim', 'park', 'jang']
for name in names:
    counts[name] = counts.get(name, 0) + 1
print(counts)
```

```
{'kim': 2, 'lee': 1, 'park': 2, 'jang': 1}
```

Counting Pattern

- Split the line into words
- Loop through the words
- Use a dictionary to track the count of each word independently

```
counts = dict()
line = input('Enter a line: ')

words = line.split()

print('Words:', words)
print('Counting...')
for word in words:
    counts[word] = counts.get(word, 0) + 1
print(counts)
```

Counting Pattern: Example

```
$ python wordcount.py
```

```
Enter a line: the clown ran after the car and the car ran into  
the tent and the tent fell down on the clown and the car
```

```
Words: ['the', 'clown', 'ran', 'after', 'the', 'car', 'and',  
'the', 'car', 'ran', 'into', 'the', 'tent', 'and', 'the',  
'tent', 'fell', 'down', 'on', 'the', 'clown', 'and', 'the',  
'car']
```

```
Counting...
```

```
{'the': 7, 'clown': 2, 'ran': 2, 'after': 1, 'car': 3, 'and':  
3, 'into': 1, 'tent': 2, 'fell': 1, 'down': 1, 'on': 1}
```

Loops over Dictionaries

- Even though dictionaries are not stored in order, we can write a `for` loop that goes through all the entries in a dictionary
- Actually it goes through all of the `keys` in the dictionary

```
>>> counts = {'kim': 2, 'lee': 1, 'park': 2, 'jang': 1}
>>> for key in counts:
...     print(key, counts[key])
kim 2
lee 1
park 2
jang 1
```

Retrieving Lists of Keys and Values

- Use `dict.keys()`, `dict.values()`, and `dict.items()`
- You can loop over them!

```
>>> counts = {'kim': 2, 'lee': 1, 'park': 2, 'jang': 1}
>>> print(counts.keys())
dict_keys(['kim', 'lee', 'park', 'jang'])
>>> print(counts.values())
dict_values([2, 1, 2, 1])
>>> print(counts.items())
dict_items([('kim', 2), ('lee', 1), ('park', 2), ('jang', 1)])
>>> total_count = 0
>>> for count in counts.values():
...     total_count += count
```

Looping over `dict.items()`

- Loop through the key-value pairs using **two** iteration variables
- The first variable is the **key** and the second is the corresponding **value**

```
>>> counts = {'kim': 2, 'lee': 1, 'park': 2, 'jang': 1}
>>> total_count = 0
>>> for k, v in counts.items():
...     print(k, v)
...     total_count += v
kim 2
lee 1
park 2
jang 1
>>> print(total_count)
6
```

Sorting a Dictionary by Keys

```
>>> d = {'s':4, 'e':1, 'o':9, 'u':0, 'l':3}

>>> for k in sorted(d):
...     print(k, d[k])

>>> for k, v in sorted(d.items()):
...     print(k, v)
```

Sorting a Dictionary by Values

```
>>> d = {'s':4, 'e':1, 'o':9, 'u':0, 'l':3}

>>> for k in sorted(d, key=d.get):
...     print(k, d[k])

>>> for k, v in sorted(d.items(), key=lambda x: x[1]):
...     print(k, v)

>>> for v, k in sorted([(v, k) for k, v in d.items()]):
...     print(k, v)
```

Dictionary Comprehension (I)

```
>>> d = { chr(ord('a')+k):k for k in range(0,5) }

>>> d

{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4 }

>>> { k*2:v**2 for k, v in d.items() }

{'aa': 0, 'bb': 1, 'cc': 4, 'dd': 9, 'ee': 16}

>>> { i:f'{i**0.5:.2f}' for i in range(1,10) if i % 2 == 0 }

{2: '1.41', 4: '2.00', 6: '2.45', 8: '2.83'}

>>> tempf = { 'seoul': 20, 'nyc': 10 }

>>> tempc = { k:(5.0 / 9.0)*(v-32) for k, v in tempf.items() }

>>> tempc

{'seoul': -6.666666666666667, 'nyc': -12.22222222222223 }
```

Dictionary Comprehension (2)

```
>>> months = [ 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun' ]  
>>> days = [ 31, 28, 31, 30, 31, 30 ]  
>>> d2 = { m:d for m, d in zip(months, days) } # dict(zip(months, days))  
>>> d2  
{'Jan': 31, 'Feb': 28, 'Mar': 31, 'Apr': 30, 'May': 31, 'Jun': 30}  
>>> { m:d for m, d in d2.items() if d > 30 }  
{'Jan': 31, 'Mar': 31, 'May': 31}  
>>> { i:m for i, m in enumerate(months) }  
{0: 'Jan', 1: 'Feb', 2: 'Mar', 3: 'Apr', 4: 'May', 5: 'Jun'}  
>>> { i:j for i, j in zip(list('ABCDE'), range(5)) }  
{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4}
```

Summary

	String	List	Tuple	Dictionary	Set
Initialization	<code>r = str()</code> <code>r = ''</code>	<code>l = list()</code> <code>l = []</code>	<code>t = tuple()</code> <code>t = ()</code>	<code>d = dict()</code> <code>d = {}</code>	<code>s = set()</code>
Example	<code>r = '123'</code>	<code>l = [1, 2, 3]</code>	<code>t = (1, 2, 3)</code>	<code>d = {1:'a', 2:'b'}</code>	<code>s = {1, 2, 3}</code>
Category	Sequence	Sequence	Sequence	Collection	Collection
Mutable?	No	Yes	No	Yes	Yes
Items ordered?	Yes	Yes	Yes	No (now Yes)	No
Indexing/slicing	Yes	Yes	Yes	No	No
Duplicate items?	Yes	Yes	Yes	No (unique keys)	No
Items sorted?	No	No	No	No	No
<code>in</code> operator	Yes	Yes	Yes	Yes	Yes

File I/O

A Text File

- A text file can be thought of as a sequence of lines

The First Book of Moses: Called Genesis

1:1 In the beginning God created the heaven and the earth.

1:2 And the earth was without form, and void; and darkness was upon the face of the deep.
And the Spirit of God moved upon the face of the waters.

1:3 And God said, Let there be light: and there was light.

1:4 And God saw the light, that it was good: and God divided the light from the darkness.

1:5 And God called the light Day, and the darkness he called Night. And the evening and the morning were the first day.

1:6 And God said, Let there be a firmament in the midst of the waters, and let it divide the waters from the waters.

1:7 And God made the firmament, and divided the waters which were under the firmament from the waters which were above the firmament: and it was so.

1:8 And God called the firmament Heaven. And the evening and the morning were the second day.

1:9 And God said, Let the waters under the heaven be gathered together unto one place, and let the dry land appear: and it was so.

The Newline Character

- We use a special character called the "newline" to indicate when a line ends
- We represent it as `\n` in strings
- Newline is still one character – not two

```
>>> msg = 'Hello\nWorld!'  
>>> msg  
'Hello\nWorld!'  
>>> print(msg)  
Hello  
World!  
  
>>> msg = 'X\nY'  
>>> print(msg)  
X  
Y  
  
>>> len(msg)  
3
```

File Processing

- A text file has **newlines** at the end of each line

The First Book of Moses: Called Genesis\n

\n

1:1 In the beginning God created the heaven and the earth.

1:2 And the earth was without form, and void; and darkness was upon the face of the deep.
And the Spirit of God moved upon the face of the waters.\n

1:3 And God said, Let there be light: and there was light.\n

1:4 And God saw the light, that it was good: and God divided the light from the darkness.\n

1:5 And God called the light Day, and the darkness he called Night. And the evening and the morning were the first day.\n

1:6 And God said, Let there be a firmament in the midst of the waters, and let it divide the waters from the waters.\n

1:7 And God made the firmament, and divided the waters which were under the firmament from the waters which were above the firmament: and it was so.\n

1:8 And God called the firmament Heaven. And the evening and the morning were the second day.\n

1:9 And God said, Let the waters under the heaven be gathered together unto one place, and let the dry land appear: and it was so.\n

Opening a File

- Before we can read the contents of the file, we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the `open()` function
- `open()` returns a "file handle" – a variable used to perform operations on the file

Using open()

■ `open(filename, mode)`

- Creates a Python file object, which serves as a link to a file residing on your machine
- You can read or write file by calling the returned file object's methods
- *Filename* is a string (pathname)
- *mode* is optional: '`r`' to open for text input (default), '`w`' to create and open for text output, '`a`' to open for appending text to the end

```
>>> f = open('genesis.txt')
>>> print(type(f))
<class '_io.TextIOWrapper'>
```

Using with Statement

- File should be closed after use – what if an exception occurs?

```
f = open('genesis.txt', 'w')
f.write('hello, world\n')
f.close()
```

- "with" simplifies file management

- When you open a file using "with", the file is automatically closed
- The file is properly closed even if an exception is raised at some point

```
with open('genesis.txt', 'w') as f:
    f.write('hello, world\n')
```

File Handle as a Sequence

- A file handle open for read can be treated as a sequence of strings
- Each line in the file is a string in the sequence
- We can use the **for** statement to iterate through a sequence

```
with open('genesis.txt') as f:  
    for line in f:  
        print(line)
```

Counting Lines in a File

- Open a file read-only
- Use a for loop to read each line
- Count the lines and print out the number of lines

```
# open.py
count = 0
with open('genesis.txt') as f:
    for line in f:
        count += 1
print('Line count:', count)
```

```
$ python open.py
Line count: 1530
```

Other Ways of Reading Line(s)

- **f.readline(size=-1)**

- Read and return one line
- *size*: if specified, at most *size* bytes are read

```
with open('genesis.txt') as f:  
    while True:  
        line = f.readline()  
        if not line:  
            break  
        print(line)
```

- **f.readlines(*hint*=-1)**

- Read and return a list of lines
- *hint*: control the number of lines to read

Reading the Whole File

- `f.read(size=-1)`
 - Read and return at most `size` characters as a single string
 - If `size` is negative or `None`, read the whole file until EOF

```
>>> f = open('genesis.txt')
>>> contents = f.read()
>>> print(len(contents))
206951
>>> print(contents[:20])
The First Book of Mo
>>> print(contents[-20:])
a coffin in Egypt.
```

Writing to a File

- **f.write(s)**

- Write the string **s** to the file and return the number of characters written
- The file should be open with '**w**' or '**a**'

```
>>> f = open('new.txt', 'w')
>>> f.write('hello, world\n')
13
>>> f.write('Happy New Year %d' % 2021)
20
>>> f.close()
```

When Files are Missing

```
>>> f = open('nofile')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or
directory: 'nofile'
```

Handling Bad File Names

```
fn = input('Enter a file name: ')
try:
    with open(fn) as f:
        count = 0
        for line in f:
            count += 1
except:
    print('File not found:', fn)
    quit()

print(f'Total {count} lines')
```

```
fn = input('Enter a file name: ')
try:
    f = open(fn)
except:
    print('File not found:', fn)
    quit()

with f:
    count = 0
    for line in f:
        count += 1
print(f'Total {count} lines')
```

Searching Through a File

- `str.startswith()`
 - Put an if statement in our for loop to only print lines that meet some criteria

```
with open('genesis.txt') as f:  
    for line in f:  
        if line.startswith('1:'):   
            print(line)
```

Blank Lines?

- Each line from the file has a **newline** at the end
- The **print** statement adds a **newline** to each line

```
1:1 In the beginning God created the heaven and the earth.\n\n1:2 And the earth was without form, and void; and darkness was upon the face of the\ndeep. And the Spirit of God moved upon the face of the waters.\n\n1:3 And God said, Let there be light: and there was light.\n\n1:4 And God saw the light, that it was good: and God divided the light from the\ndarkness.\n\n
```

Removing the Trailing Newline

- **str.rstrip()**

- Strip the whitespace from the right-hand side of the string
- Whitespace: blank(' '), tab('\t'), newline('\n'), etc.

```
with open('genesis.txt') as f:  
    for line in f:  
        if line.startswith('1:'):   
            line = line.rstrip()  
            print(line)
```

Skipping with Continue

- Skip a line by using the `continue` statement
- `str.isdigit()`
 - Return `True` if all characters in the string are digits

```
with open('genesis.txt') as f:  
    for line in f:  
        if not line[0].isdigit():  
            continue  
        line = line.rstrip()  
        print(line)
```

Using `in` to Select Lines

- Use an `in` operator to look for a certain substring in a line

```
with open('genesis.txt') as f:  
    for line in f:  
        if not line[0].isdigit():  
            continue  
        if not line.startswith('1:'):   
            continue  
        if 'heaven' in line:  
            line = line.rstrip()  
            print(line)
```

Extracting Words

- Use `str.split()`

```
with open('genesis.txt') as f:  
    for line in f:  
        if not line.startswith('1:'):   
            continue  
        words = line.strip().split()  
        print(words)
```

Finding Top 10 Words

```
filename = input('Enter file: ')

with open(filename) as f:
    counts = dict()
    for line in f:
        words = line.strip().lower().split()
        for word in words:
            counts[word] = counts.get(word, 0) + 1

for v, k in sorted([(v,k) for k,v in counts.items()], reverse=True)[:10]:
    print(v, k)
```

Jin-Soo Kim
[\(jinsoo.kim@snu.ac.kr\)](mailto:jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022



Python for Data Analytics

NumPy I

Outline

- What is NumPy?
- Creating Arrays
- Manipulating Arrays
- Universal Functions
- Statistical Operations
- Matrix Operations

What is "NumPy" Module?

- The "**NumPy**" (**Numeric Python**) package provides basic routines for manipulating **large arrays and matrices of numeric data**
- The "**SciPy**" (**Scientific Python**) package extends the functionality of NumPy with a substantial collection of useful algorithms
 - Minimization, Fourier transformation, regression, and other applied math techniques
- NumPy and SciPy are **open source add-on modules** (**not Python Standard Library**)
- More than functionalities of commercial packages like **MatLab**
- Catching up functionalities of **R**
- **>>> import numpy as np**

NumPy History

- Numeric (ancestor of Numpy)
 - Released in 1995 by Jim Hugunin by generalizing Jim Fulton's matrix package
- Numarray
 - A more flexible replacement for Numeric
 - Faster for large arrays, slower than Numeric on small arrays
- SciPy module
 - Created by Travis Oliphant et al. in 2001
 - Provides scientific and technical operations
 - NumPy 1.0 released (as part of SciPy) in 2006 by porting Numarray's features to Numeric
- NumPy module separated from SciPy as a stand-alone package

numpy.ndarray: The N-dimensional Array

- A multidimensional container of items of the same type and size
 - `shape`: a tuple of N non-negative integers that specify the sizes of each dimension
 - `data-type object (dtype)`: the type of items in the array
- 1D numpy.ndarray object
 - Example: `array([3,6])` `array([3.5,6.4,7.2])`
- 2D numpy.ndarray object
 - Example: `array([[1,0,2], [3,5,2]])`
- 3D numpy.ndarray object
 - Example: `array([[[0,0,1],[1,2,3]], [[1,0,2],[2,3,4]], [[3,5,2],[1,1,1]]])`

Array Example

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3], [4, 5, 6]], int)
>>> print(x)
[[1 2 3]
 [4 5 6]]
>>> type(x)
<class 'numpy.ndarray'>
>>> x.shape
(2, 3)                                     # (number of rows, number of columns)
>>> x.dtype
dtype('int64')
```

List vs. Array.array vs. Numpy.ndarray

■ Lists

- Simple
- Can't constrain the type of elements stored in a list

```
>>> a = [[0]*3 for _ in range(3)]  
>>> a  
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]  
>>> b = [1, 3.5, 'hello']
```

■ Array.array

- All elements of the array must be of the same numeric type
- May be used to interface with C code

```
>>> import array  
>>> a = array.array('i', [1, 2, 3])  
>>> a  
array('i', [1, 2, 3])
```

■ Numpy.ndarray

- Supports various **computations** on arrays and matrices

```
>>> import numpy  
>>> a = numpy.array([1, 2, 3], float)  
>>> a  
array([1., 2., 3.])
```

Creating Arrays

array()

- `np.array(object, dtype=None, ...)`
 - `object`: usually a list
 - `dtype`: the desired data type
 - If omitted, the type will be determined as the minimum type required to hold the objects

```
>>> l = [1, 2, 3, 4, 5]
>>> np.array(l)
>>> np.array(l, int)
>>> np.array(l, dtype='i')
>>> np.array(l, dtype=np.uint8)
>>> np.array(l, dtype='f')
>>> np.array(l, float)
```

Data types		Type code
Boolean	bool	?
Integers	int8	i1, b
	int16	i2, h
	int32	i4, i
	int64 (int)	i8, l, q
Unsigned integers	uint8	u1, B
	uint16	u2, H
	uint32	u4, I
	uint64	u8, L, Q,
Floating points	float16	f2
	float32	f4, f
	float64 (float)	f8, d
	float128	f16, g
Complex	complex64	c8, F
	complex128 (complex)	c16, D
	complex256	c32, G
Unicode string	unicode	U

np.inf and np.nan

■ *np.inf* (infinity)

- Too large to be represented
- e.g., $n / 0$, $\text{np.inf} * \text{np.inf}$,
 $\text{np.inf} + \text{np.inf}$, ...

```
>>> np.inf  
inf  
>>> a = array([3, 2, 5])  
>>> a / 0  
array([inf, inf, inf])
```

■ *np.nan* (not-a-number)

- A value that is undefined or unrepresentable
- e.g., $0 / 0$, $\text{np.inf} / \text{np.inf}$,
 $\text{np.inf} * 0$, $\text{np.inf} - \text{np.inf}$, ...

```
>>> np.nan  
nan  
>>> np.log(-1)  
nan  
>>> np.log([-1, 1, 2])  
array([-nan, 0.0, 0.69314718])
```

full() and empty()

- **np.full(shape, value[, dtype][, order])**
 - Return a new array of given shape and type, filled with `value`
- **np.full_like(a, value, ...)**
- **np.empty(shape[, dtype][, order])**
 - Return a new array of given shape and type, without initializing entries
 - Faster than others
- **np.empty_like(a, ...)**

```
>>> np.full(5, 2)
array([2, 2, 2, 2, 2])
>>> np.full((2,3), -1, float)
array([[-1., -1., -1.],
       [-1., -1., -1.]])
```

```
>>> np.empty((2,4))
array([[6.91542951e-310,
       6.91542951e-310, 1.24120911e-316,
       1.24120911e-316],
       [6.91542951e-310,
       6.91542951e-310, 0.0000000e+000,
       0.0000000e+000]])
```

zeros() and ones()

■ `np.zeros(shape[, dtype][, order])`

- Return a new array of given shape and type, filled with zeros
- `order`: 'C' = row-major (C-style),
'F' = column-major (Fortran-style)

```
>>> np.zeros(5)
array([0., 0., 0., 0., 0.])
>>> np.zeros((2,3))
array([[0., 0., 0.],
       [0., 0., 0.]])
```

■ `np.ones(shape, [, dtype][, order])`

- Return a new array of given shape and type, filled with ones

```
>>> np.ones(4)
array([1., 1., 1., 1.])
>>> np.ones((3,3))
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

`zeros_like()` and `ones_like()`

- `np.zeros_like(a[, dtype], ...)`
 - Return an array of zeros with the same shape and type as a given array
- `np.ones_like(a[, dtype], ...)`
 - Return an array of ones with the same shape and type as a given array

```
>>> a = np.full((2,3), -1, float)
array([[-1., -1., -1.],
       [-1., -1., -1.]])
>>> np.zeros_like(a)
array([[0, 0, 0],
       [0, 0, 0]])
```

```
>>> a = np.full((3,4), 10)
>>> np.ones_like(a)
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

identity() and eye()

■ `np.identity(n[, dtype])`

- Return the identity array (a square array with ones on the main diagonal)
- `n`: number of rows (and columns)

```
>>> np.identity(4)
array([[1.,  0.,  0.,  0.],
       [0.,  1.,  0.,  0.],
       [0.,  0.,  1.,  0.],
       [0.,  0.,  0.,  1.]])
```

■ `np.eye(N[, M][, k][, dtype][, order])`

- Return a 2D array with ones on the diagonal and zeros elsewhere
- `M`: number of columns (default `N`)
- `k`: index of the diagonal (default 0)

```
>>> np.eye(2)
array([[1.,  0.],
       [0.,  1.]])
>>> np.eye(3,4,1)
array([[0.,  1.,  0.,  0.],
       [0.,  0.,  1.,  0.],
       [0.,  0.,  0.,  1.]])
```

arange()

- `np.arange([start,]stop[, step][, dtype])`
 - Return an array with evenly spaced values within a given interval: `[start, stop)`
 - When using a non-integer step, it may produce unexpected results
→ Use `np.linspace()` instead

```
>>> np.arange(10, 30, 5)
array([10 15 20 25])
>>> np.arange(0, 2, 0.3)
array([0.  0.3 0.6 0.9 1.2 1.5 1.8])
>>> np.arange(0, -1, -0.1)
array([ 0. , -0.1, -0.2, -0.3, -0.4, -0.5, -0.6, -0.7, -0.8, -0.9])
>>> np.arange(8.0, 8.4, 0.05)
array([8.  , 8.05, 8.1 , 8.15, 8.2 , 8.25, 8.3 , 8.35, 8.4 ])
```

linspace()

- **np.linspace(start, stop[, num][, endpoint]...)**

- Return an array with evenly spaced numbers over a specified interval: `[start, stop]`
- `num`: the number of evenly spaced samples (default 50)
- `endpoint`: if `False`, the endpoint of the interval is excluded (default `True`)

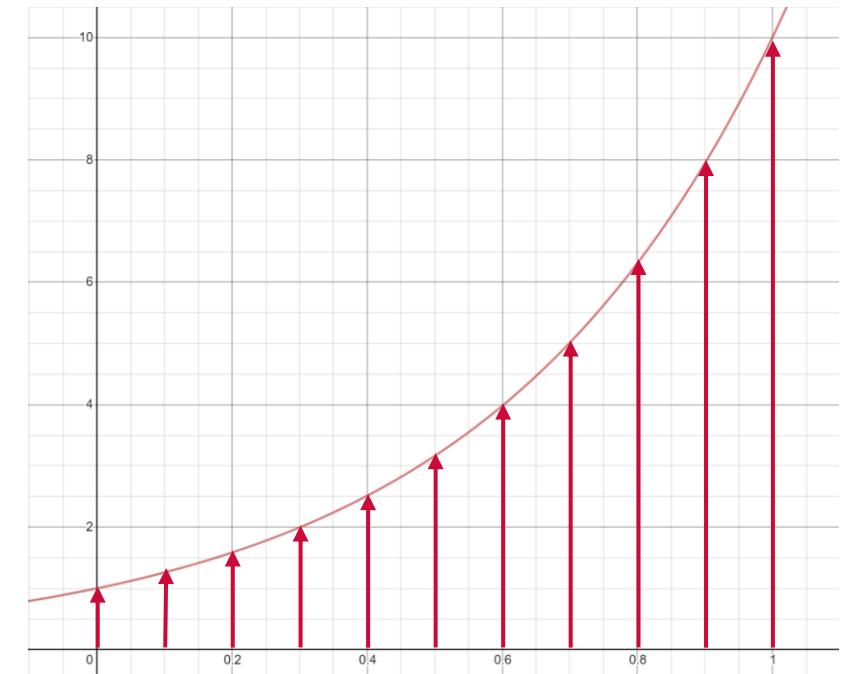
```
>>> np.linspace(0, 100, 5)
array([ 0.,  25.,  50.,  75., 100.])
>>> np.linspace(0, 100, 5, endpoint=False)
array([ 0., 20., 40., 60., 80.])
>>> np.linspace(0, 4, 4)
array([0.          , 1.33333333, 2.66666667, 4.          ])
>>> np.linspace(8.0, 8.4, 8, False)
array([8.  , 8.05, 8.1 , 8.15, 8.2 , 8.25, 8.3 , 8.35])
```

logspace()

- `np.logspace(start, stop[, num][, endpoint][, base], ...)`

- Return `num` numbers spaced evenly on a log scale
- In linear space, the sequence starts at $base^{start}$ and ends with $base^{stop}$
- `base`: the base of the log space (default: 10.0)

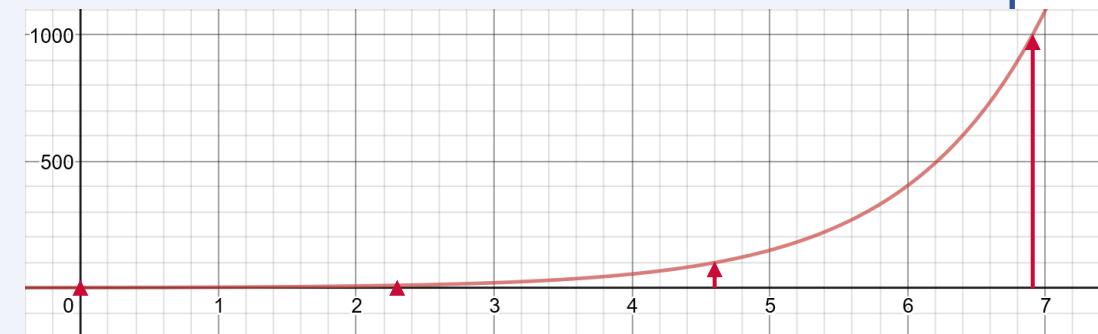
```
>>> np.logspace(0, 1, 11)
array([ 1.          ,  1.25892541,  1.58489319,
       1.99526231,  2.51188643,  3.16227766,
       3.98107171,  5.01187234,  6.30957344,
       7.94328235, 10.        ])
>>> [ 10**x for x in np.linspace(0, 1, 11) ]
```



geomspace()

- **np.geomspace(*start, stop[, num][, endpoint]*, ...)**
 - Return *num* numbers spaced evenly on a log scale (a geometric progression)
 - Each output sample is a constant multiple of the previous

```
>>> import math  
>>> [ np.exp(i) for i in np.linspace(np.log(1), np.log(1000), 4) ]  
[1.0, 9.99999999999998, 99.9999999999996, 999.999999999998]  
  
>>> np.geomspace(1, 1000, 4)  
array([ 1., 10., 100., 1000.])  
  
>>> np.geomspace(-1000, -1, num=4)  
array([-1000., -100., -10., -1.])  
  
>>> np.geomspace(1, 256, 9)  
array([ 1., 2., 4., 8., 16., 32., 64., 128., 256.])
```



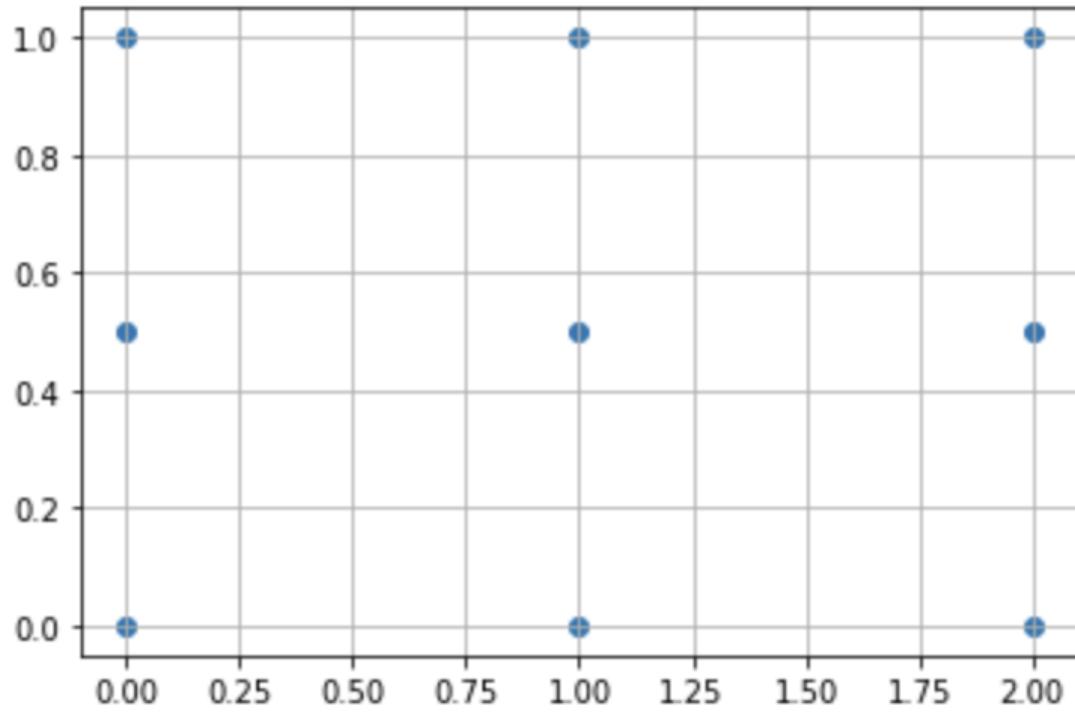
meshgrid()

- `np.meshgrid(x1, x2, ..., xn, ...)`
 - Return coordinate matrices from coordinate vectors

```
>>> x = np.linspace(0, 2, 3)
>>> x
array([0., 1., 2.])
>>> y = np.linspace(0, 1, 3)
>>> y
array([0. , 0.5, 1. ])
>>> xv, yv = np.meshgrid(x, y)
```

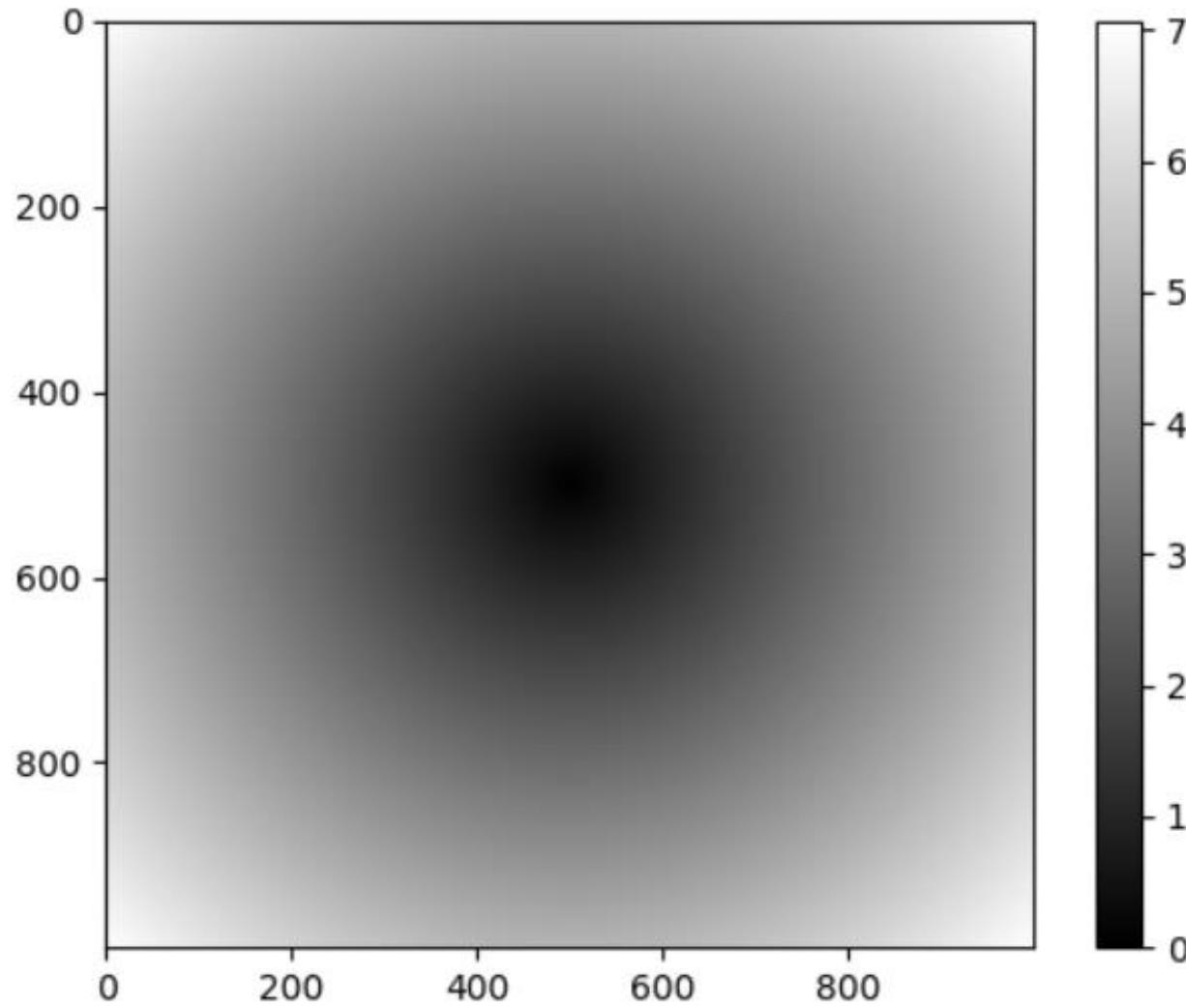
```
>>> xv
array([[0., 1., 2.],
       [0., 1., 2.],
       [0., 1., 2.]])
>>> yv
array([[0. , 0. , 0. ],
       [0.5, 0.5, 0.5],
       [1. , 1. , 1. ]])
```

meshgrid()



```
>>> import matplotlib.pyplot as plt  
>>> plt.scatter(xv, yv)  
>>> plt.grid(True)  
>>> plt.show()  
  
>>> mg = [list(zip(x,y)) for x, y \  
         in zip(xv, yv)]  
>>> mg  
[[ (0.0, 0.0), (1.0, 0.0), (2.0, 0.0)],  
 [(0.0, 0.5), (1.0, 0.5), (2.0, 0.5)],  
 [(0.0, 1.0), (1.0, 1.0), (2.0, 1.0)]]
```

meshgrid() Example



```
import numpy as np  
import matplotlib.pyplot as plt  
  
pts = np.arange(-5, 5, 0.01)  
x, y = np.meshgrid(pts, pts)  
z = np.sqrt(x**2 + y**2)  
plt.imshow(z, cmap=plt.cm.gray)  
plt.colorbar()  
plt.show()
```

random.random() and random.randint()

- numpy.random submodule provides various random number generators
- **np.random.random(size)**
 - Return random floats in the interval [0.0, 1.0]
 - *size*: integer or tuple of integers for output array shape
- **np.random.randint(*low*[, *high*], [*size*],...)**
 - Return random integers in the interval [*low*, *high*)
 - If *high* is omitted, results are from [0, *low*)

```
>>> np.random.random((3,2))
array([[0.44325748, 0.61687924],
       [0.68575248, 0.60672728],
       [0.82738475, 0.38333312]])
```

```
>>> np.random.randint(100)
91
>>> np.random.randint(10, size=5)
array([6, 5, 9, 5, 1])
```

random.rand() and random.randn()

■ np.random.rand(d_0, d_1, \dots, d_n)

- Return random floats in the interval [0.0, 1.0)
- d_0, d_1, \dots, d_n : the dimensions of the output array (not tuple)

```
>>> np.random.rand(3,2)
array([[0.01250554, 0.43358273],
       [0.3730851 , 0.66585267],
       [0.03250322, 0.6765952 ]])
```

■ np.random.randn(d_0, d_1, \dots, d_n)

- Return samples from the "standard normal" distribution $N(0, 1)$
- For random samples from $N(\mu, \sigma^2)$, use $\sigma * np.random.randn() + \mu$

```
>>> np.random.randn()
0.5288740495934477
>>> np.random.randn(3,2)
array([-0.64964129,  1.26874147],
      [ 0.88909375,  0.51902268],
      [ 2.1553506 , -1.73896019]])
```

random.uniform()

- `np.random.uniform([low=0.0], [high=1.0], [size])`

- Draw samples from a uniform distribution
- Samples are uniformly distributed over the interval `[low, high)`

```
>>> np.random.uniform(1.0, 2.0)
1.6937903416817646
>>> np.random.uniform(1.0, 2.0, 5)
array([1.1922301 , 1.72618062, 1.82763685, 1.32765954, 1.45356649])
>>> import math
>>> np.random.uniform(0, math.pi, (3,4))
array([[0.53309063, 1.56158409, 2.34588318, 2.40615273],
       [0.28675017, 0.37922173, 1.24792002, 0.05974539],
       [2.02903176, 0.98991193, 0.61068395, 2.40537881]])
```

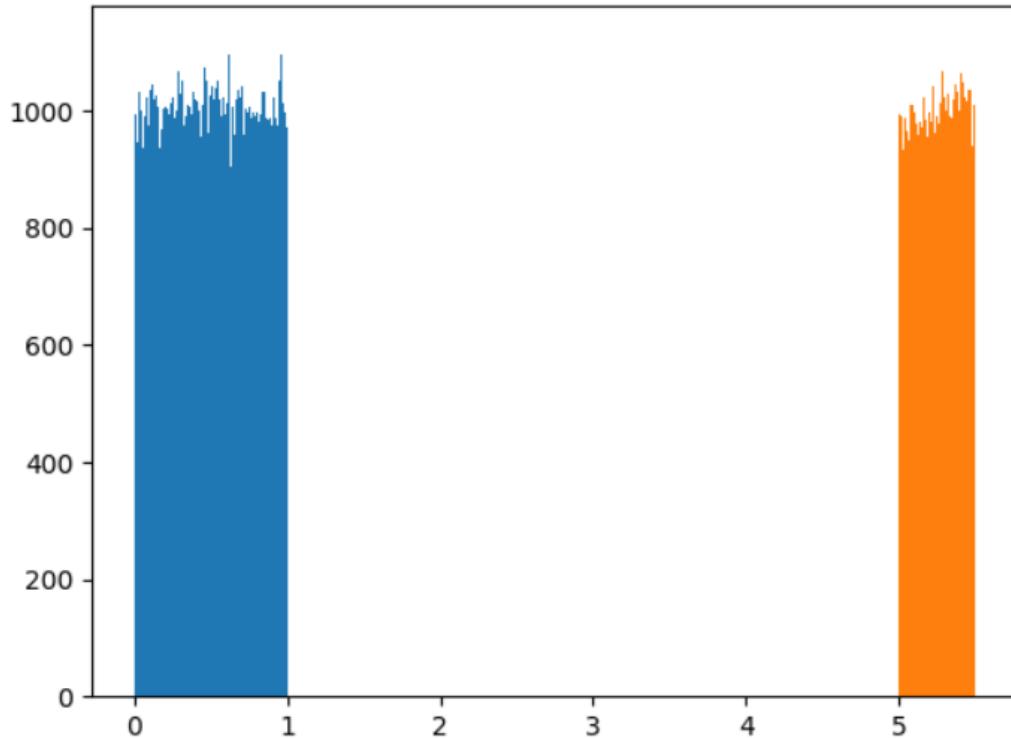
random.normal()

- `np.random.normal([loc=0.0], [scale=1.0], [size])`
 - Draw samples from a normal (Gaussian) distribution
 - `loc`: mean of the distribution, `scale`: standard deviation of the distribution
 - `random.normal(loc, scale) == loc + scale*random.normal()`

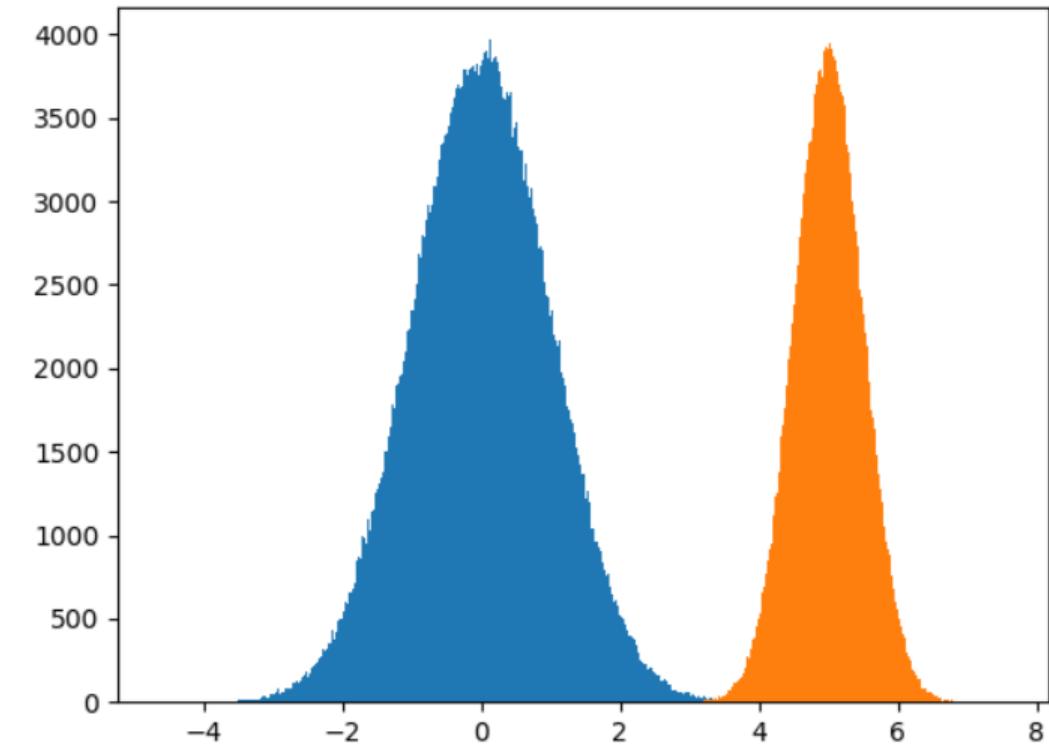
```
>>> np.random.normal()
-0.6901865834995796
>>> np.random.normal(size=5)
array([-0.1249383 ,  1.10623467, -0.38662773, -0.60593157,  0.71653932])
>>> np.random.normal(size=(2,3))
array([[ 0.2093123 ,  0.40961339,  0.64944229],
       [ 1.06565541,  1.0441453 , -0.81924546]])
```

Uniform vs. Normal

```
values = np.random.uniform(size=1000000)
plt.hist(values, 1000)
plt.hist(5+0.5*values, 1000)
plt.show()
```



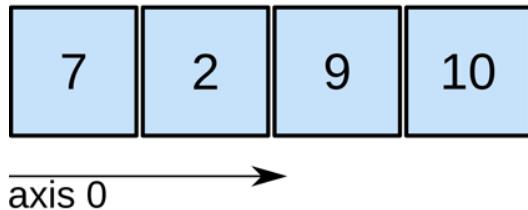
```
values = np.random.normal(size=1000000)
plt.hist(values, 1000)
plt.hist(5+0.5*values, 1000)
plt.show()
```



Manipulating Arrays

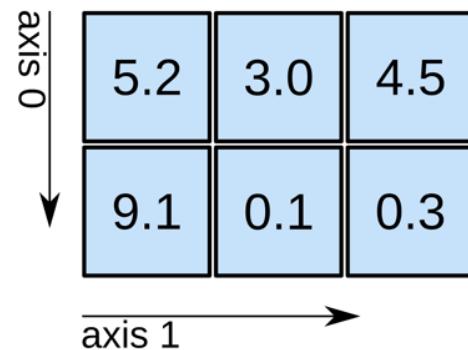
Array Shape

1D array



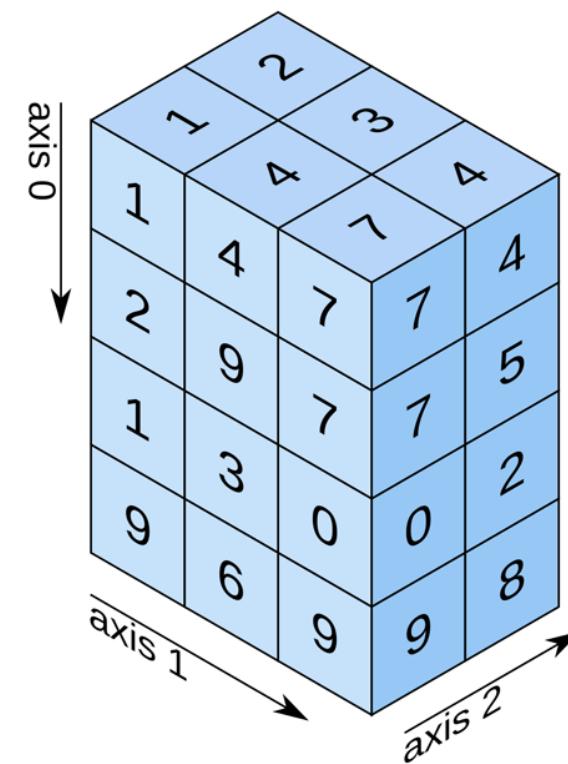
shape: (4,)

2D array



shape: (2, 3)

3D array



shape: (4, 3, 2)

Type Casting

- `a.tolist()` and `a.astype()`

```
>>> a = np.random.randint(0, 10, (3,4))

>>> a
array([[7, 2, 0, 9],
       [5, 1, 3, 4],
       [9, 3, 2, 0]])

>>> a.tolist()
[[7, 2, 0, 9], [5, 1, 3, 4], [9, 3, 2, 0]]

>>> a.astype(np.float)
array([[7., 2., 0., 9.],
       [5., 1., 3., 4.],
       [9., 3., 2., 0.]])
```

Reshaping

- **a.reshape(shape)**

- Return an array containing the same data with a new shape

- **a.resize(shape)**

- Change shape and size of array in-place
 - Same as:
a.shape = shape

- **a.flatten()**

- Return a flattened array

```
>>> a=np.arange(6)
>>> a.shape
(6,)
>>> a.reshape(2,3)
array([[0, 1, 2],
       [3, 4, 5]])
>>> a.resize(2,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5]])
>>> a.flatten()
array([0, 1, 2, 3, 4, 5])
>>> a.shape = (1, 6)      # ???
```

Reshaping (cont'd)

- One dimension can be -1 in `a.reshape()`
 - The value is inferred from the length of the array and remaining dimensions

```
>>> a = np.arange(12)
>>> a.reshape(3, -1)
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> a.reshape(-1, 3)
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

```
>>> a.reshape(2, 2, -1)
array([[[ 0,  1,  2],
         [ 3,  4,  5]],
        [[ 6,  7,  8],
         [ 9, 10, 11]]])
>>> a.reshape(-1, 2, 3)
array([[[ 0,  1,  2],
         [ 3,  4,  5]],
        [[ 6,  7,  8],
         [ 9, 10, 11]]])
```

Iteration

- Use the **for** loop

```
>>> a = np.arange(5)  
>>> a  
array([0, 1, 2, 3, 4])  
>>> for i in a:  
...     print(i)  
  
0  
1  
2  
3  
4
```

```
>>> b = np.arange(6).reshape(2,3)  
>>> b  
array([[0, 1, 2],  
       [3, 4, 5]])  
>>> for x in b:  
...     print(x)  
  
[0 1 2]  
[3 4 5]
```

Iteration (cont'd)

- One "for" loop for each dimension

```
>>> a =  
np.arange(18).reshape(3,2,-1)  
>>> a  
array([[[ 0,  1,  2],  
       [ 3,  4,  5]],  
  
      [[ 6,  7,  8],  
       [ 9, 10, 11]],  
  
      [[12, 13, 14],  
       [15, 16, 17]]])
```

```
>>> for x in a:  
...     print(x)  
...     print('-'*10)  
[[0 1 2]  
 [3 4 5]]  
-----  
[[ 6  7  8]  
 [ 9 10 11]]  
-----  
[[12 13 14]  
 [15 16 17]]  
-----
```

```
>>> for x in a:  
...     for y in x:  
...         print(y)  
...         print(' -'*10)  
[0 1 2]  
-----  
[3 4 5]  
-----  
[6 7 8]  
-----  
[ 9 10 11]  
-----  
[12 13 14]  
-----  
[15 16 17]  
-----
```

Indexing

■ Single element indexing

- Similar to Python lists
- Negative indices for indexing from the end of the array

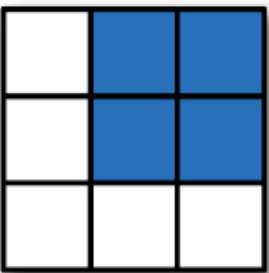
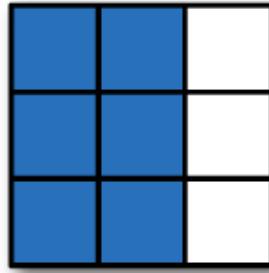
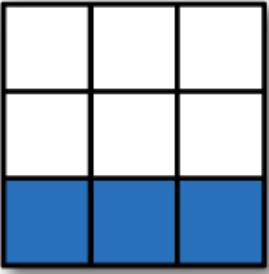
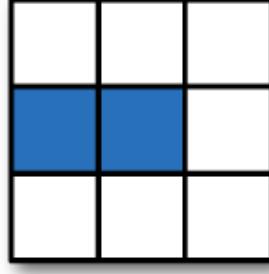
```
>>> x = np.arange(10)  
>>> x[2]  
2  
>>> x[-2]  
8
```

■ Multidimensional indexing

- Used for multidimensional arrays
- If you use fewer indices than dimensions, you get a subdimensional array
- $x[0, 2] == x[0][2]$: $x[0][2]$ is more inefficient as a new temporary array is created

```
>>> x.shape = (2, 5)  
>>> x[0]  
array([0, 1, 2, 3, 4])  
>>> x[1, 3]  
8  
>>> x[0][2]  
2
```

Slicing

Expression	Shape	Expression	Shape
	<code>arr[:2, 1:]</code> (2, 2)		<code>arr[:, :2]</code> (3, 2)
	<code>arr[2]</code> (3,) <code>arr[2, :]</code> (3,) <code>arr[2:, :]</code> (1, 3)		<code>arr[1, :2]</code> (2,) <code>arr[1:2, :2]</code> (1, 2)

Slicing Example

```
>>> x = np.arange(10)
>>> x[2:5]
array([2, 3, 4])
>>> x[:-7]
array([0, 1, 2])
>>> x[1:7:2]
array([1, 3, 5])
>>> x[::-1]
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

```
>>> y = np.arange(35).reshape(5,7)
>>> y[2,:]
array([14, 15, 16, 17, 18, 19, 20])
>>> y[:,2]
array([ 2,  9, 16, 23, 30])
>>> y[1:5:2,::3]
array([[ 7, 10, 13],
       [21, 24, 27]])
>>> y[-1:,-2:]
array([[33, 34]])
```

Views

- Slices of arrays only produce new "views" of the original data
- Use `a.copy()` to copy the internal array data

```
>>> x = np.arange(6).reshape(2, 3)
>>> y = x[:,1]
>>> y
array([1, 4])
>>> y[:] = 9
>>> y
array([9, 9])
>>> x
array([[0, 9, 2],
       [3, 9, 5]])
```

```
>>> x = np.arange(6).reshape(2, 3)
>>> y = x[:,1].copy()
>>> y
array([1, 4])
>>> y[:] = 9
>>> y
array([9, 9])
>>> x
array([[0, 1, 2],
       [3, 4, 5]])
```

Transposing

■ `a.transpose(axes)`

- Return a **view** of the array with axes transposed
- For a 1-D array, no effect
- For a 2-D array, this is a standard matrix transpose
- For an n-D array and `axes` are given, their order indicates how the axes are permuted. Otherwise, shapes are reversed
 - `x.shape = (1, 2, 3)`
→ `x.transpose(1, 2, 0).shape = (2, 3, 1)`

```
>>> a = np.arange(6)
>>> a.transpose()
array([0, 1, 2, 3, 4, 5])
>>> b = a.reshape(3,2)
>>> b
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> b.transpose()
array([[0, 2, 4],
       [1, 3, 5]])
>>> c = np.arange(6).reshape(1,2,3)
>>> c.transpose().shape
(3, 2, 1)
```

Index Arrays

- NumPy arrays can be indexed with other arrays or lists
- Returns a new array (not a view)

```
>>> x = np.arange(8, 0, -1)
>>> x
array([8, 7, 6, 5, 4, 3, 2, 1])
>>> x[np.array([3, 3, 1, 6])]
array([5, 5, 7, 2])
>>> x[[3, 3, -1, 6]]
array([5, 5, 1, 2])
>>> x[[[1,1],[2,3]]]
array([[7, 7],
       [6, 5]])
```

```
array([[ 0,  1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12, 13],
       [14, 15, 16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25, 26, 27],
       [28, 29, 30, 31, 32, 33, 34]])
```



```
>>> y = np.arange(35).reshape(5,7)
>>> y[[0,2,4], [0,1,2]]
array([ 0, 15, 30])
>>> y[[0,2,4], 1]
array([ 1, 15, 29])
>>> y[[0,2,4]]      # some rows
array([[ 0,  1,  2,  3,  4,  5,  6],
       [14, 15, 16, 17, 18, 19, 20],
       [28, 29, 30, 31, 32, 33, 34]])
>>> y[:, [0, 2]]    # ???
```

Boolean Index Arrays

& (and)
| (or)
~ (not)

- Only choose the elements that satisfy the Boolean expression
- Returns a new array (not a view)

```
>>> a = np.arange(1,7)
>>> a
array([1, 2, 3, 4, 5, 6])
>>> b = [True, True, False, False,
         True, False]
>>> a[b]
array([1, 2, 5])
>>> c = [1, 1, 0, 0, 1, 0]
>>> a[c]
array([2, 2, 1, 1, 2, 1])
```

```
>>> x = np.arange(9).reshape(3,3)
>>> y = (x % 2 == 0) | (x % 3 == 0)
>>> y
array([[ True, False,  True],
       [ True,  True, False],
       [ True, False,  True]])
>>> x[y]
array([0, 2, 3, 4, 6, 8])
>>> x[(x % 2 == 0) | (x % 3 == 0)]
array([0, 2, 3, 4, 6, 8])
```

Concatenating

- **`np.concatenate((a1, a2, ..., an), axis)`**
 - Join a sequence of arrays along an existing axis
 - `a1, a2, ..., an`: sequence of arrays
 - `axis`: the axis along which the arrays will be joined. If axis is `None`, arrays are flattened before use. (default 0)

```
>>> a = np.array([1, 2, 3])
>>> b = np.array([4, 5, 6])
>>> np.concatenate((a, b))
array([1, 2, 3, 4, 5, 6])
```

```
>>> x = np.array([[1, 2], [3, 4]])
>>> y = np.array([[5, 6]])
>>> np.concatenate((x, y), axis=0)
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Stacking

■ `np.hstack(tup)`

- Stack arrays in sequence horizontally (column wise)
- *tup*: tuple of arrays

$$\begin{array}{c} \text{[blue box]} \\ + \end{array} \quad \begin{array}{c} \text{[blue box]} \\ = \end{array} \quad \begin{array}{c} \text{[blue box]} \mid \text{[blue box]} \end{array}$$

■ `np.vstack(tup)`

- Stack arrays in sequence vertically (row wise)
- *tup*: tuple of arrays

$$\begin{array}{c} \text{[blue box]} \\ + \end{array} \quad \begin{array}{c} \text{[blue box]} \\ = \end{array} \quad \begin{array}{c} \text{[blue box]} \\ \text{[blue box]} \end{array}$$

```
>>> a=np.arange(6).reshape(2,3)
>>> b=np.arange(6,12).reshape(2,3)
>>> np.hstack((a,b))
array([[ 0,  1,  2,  6,  7,  8],
       [ 3,  4,  5,  9, 10, 11]])
>>> np.vstack((a,b))
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

np.r_[]

■ np.r_[axes, ...]

- Stack arrays along their first axis in the string (*default: '0'*)

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

>>> np.r_[a, b]
array([0, 1, 2, 3, 4, 5])

>>> np.r_[[a], [b]]
array([[0, 1, 2],
       [3, 4, 5]])

>>> np.r_['1', [a], [b]]  
axis to stack
array([[0, 1, 2,
       3, 4, 5]])
```

```
>>> np.r_['0,2', a, b]
array([[0, 1, 2],
       [3, 4, 5]])

>>> np.r_['1,2', a, b]
array([[0, 1, 2, 3, 4, 5]])

>>> np.r_['1,2,0', a, b]
array([[0, 3],
       [1, 4],
       [2, 5]]), 1, 2, 3, 4, 5])]

# np.r_['1', a.reshape(3,1), b.reshape(3,1)]  
force to 2D shape (1, 3)  
if necessary
```

np.c_[]

- *np.c_[]*

- Short-hand for np.r_['-1,2,0', ...]

stack on last axis

```
>>> np.c_[[0,1,2], [3,4,5]]  
array([[0, 3],  
       [1, 4],  
       [2, 5]])  
  
>>> np.c_[[[0,1,2]], [[3,4,5]]]  
array([[0, 1, 2, 3, 4, 5]])
```

```
# Already in 3D  
# Just stack on last axis  
  
>>> np.c_[[[0,1,2]]], [[[3,4,5]]]  
array([[[0, 1, 2, 3, 4, 5]]])
```

Summary: For 2D Arrays

- Stacking horizontally

```
array([[ 0,  1,  2, 10, 11, 12],  
       [ 3,  4,  5, 13, 14, 15]])  
  
>>> a = np.arange(6).reshape(2,3)  
>>> b = np.arange(10,16).reshape(2,3)  
  
>>> np.concatenate((a, b), axis=1)  
  
>>> np.hstack((a, b))  
  
>>> np.c_[a, b]  
  
>>> np.r_['1', a, b]
```

- Stacking vertically

```
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [10, 11, 12],  
       [13, 14, 15]])  
  
>>> np.concatenate((a, b))  
  
>>> np.concatenate((a, b), axis=0)  
  
>>> np.vstack((a, b))  
  
>>> np.r_[a, b]  
  
>>> np.r_['0', a, b]
```

Tiling

- **np.tile(A, reps)**

- Construct an array by repeating *A* the number of times given by *reps*

```
>>> a = np.array([0, 1, 2])
>>> np.tile(a, 2)
array([0, 1, 2, 0, 1, 2])
>>> np.tile(a, (2, 2))
array([[0, 1, 2, 0, 1, 2],
       [0, 1, 2, 0, 1, 2]])
>>> np.tile(a, (3,1,2))
array([[[0, 1, 2, 0, 1, 2]],
       [[0, 1, 2, 0, 1, 2]],
       [[0, 1, 2, 0, 1, 2]]])
```

```
>>> b = np.array([[1, 2], [3, 4]])
>>> np.tile(b, 2)
array([[1, 2, 1, 2],
       [3, 4, 3, 4]])
>>> np.tile(b, (2, 1))
array([[1, 2],
       [3, 4],
       [1, 2],
       [3, 4]])
```

Jin-Soo Kim
[\(jinsoo.kim@snu.ac.kr\)](mailto:jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022



Python for Data Analytics

NumPy II

Outline

- What is NumPy?
- Creating Arrays
- Manipulating Arrays
- Universal Functions
- Statistical Operations
- Matrix Operations

Universal Functions

Arithmetic Operations

- Shape of both operands must be same!
- One operand can be a constant

```
>>> a = np.array([1, 2, 3], float)
>>> b = np.array([4, 5, 6], float)
>>> a + b
array([5., 7., 9.])
>>> a - b
array([-3., -3., -3.])
>>> a * b
array([ 4., 10., 18.])
>>> b / a
array([4. , 2.5, 2. ])
```

```
>>> b % a
array([0., 1., 0.])
>>> b**a
array([ 4., 25., 216.])
>>> a * 0.5
array([0.5, 1. , 1.5])
>>> b > 5
array([False, False, True])
>>> a + b == 5
array([ True, False, False])
```

Operations: Python List vs. NumPy Array

■ Operator *

- List * n: repetition of the whole list
- Array * n: multiply n to every element in the array

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

■ Operator +

- List1 + List2: concatenation of two lists
- Array1 + Array2: Element-wise addition

```
>>> L + L
[1, 2, 3, 1, 2, 3]
>>> A + A
array([2, 4, 6])
```

Universal Functions (ufuncs)

- A function that operates on ndarrays in an element-by-element fashion
 - A "vectorized" wrapper for a function that takes a fixed number of specific inputs and produces a fixed number of specific outputs
 - Supports array broadcasting and type casting
- Available ufuncs
 - Math operations (add, subtract, multiply, divide, power, sqrt, exp, log,...)
 - Trigonometric functions (sin, cos, tan, arcsin, arccos, sinh, cosh, tanh,...)
 - Bit-twiddling functions (bitwise_and, invert, left_shift, right_shift,...)
 - Comparison functions (greater, less, equal, logical_and, maximum, fmax,...)
 - Floating functions (fabs, modf, floor, ceil, isnan, ifinf,...)

Unary Universal Functions

Function	Description
abs, fabs	Compute the absolute value element-wise for integer, floating-point, or complex values
sqrt	Compute the square root of each element (equivalent to <code>arr ** 0.5</code>)
square	Compute the square of each element (equivalent to <code>arr ** 2</code>)
exp	Compute the exponent e^x of each element
log, log10, log2, log1p	Natural logarithm (base e), log base 10, log base 2, and $\log(1 + x)$, respectively
sign	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
ceil	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
floor	Compute the floor of each element (i.e., the largest integer less than or equal to each element)
rint	Round elements to the nearest integer, preserving the <code>dtype</code>
modf	Return fractional and integral parts of array as a separate array
isnan	Return boolean array indicating whether each value is NaN (Not a Number)
isfinite, isinf	Return boolean array indicating whether each element is finite (non- \inf , non-NaN) or infinite, respectively
cos, cosh, sin, sinh, tan, tanh	Regular and hyperbolic trigonometric functions
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Inverse trigonometric functions
logical_not	Compute truth value of <code>not x</code> element-wise (equivalent to <code>~arr</code>).

Binary Universal Functions

Function	Description
add	Add corresponding elements in arrays
subtract	Subtract elements in second array from first array
multiply	Multiply array elements
divide, floor_divide	Divide or floor divide (truncating the remainder)
power	Raise elements in first array to powers indicated in second array
maximum, fmax	Element-wise maximum; fmax ignores NaN
minimum, fmin	Element-wise minimum; fmin ignores NaN
mod	Element-wise modulus (remainder of division)
copysign	Copy sign of values in second argument to values in first argument
greater, greater_equal, less, less_equal, equal, not_equal	Perform element-wise comparison, yielding boolean array (equivalent to infix operators <code>></code> , <code>>=</code> , <code><</code> , <code><=</code> , <code>==</code> , <code>!=</code>)
logical_and, logical_or, logical_xor	Compute element-wise truth value of logical operation (equivalent to infix operators <code>&</code> , <code>^</code>)

ufuncs: Examples

```
>>> a = np.arange(1, 5)
>>> a
array([1, 2, 3, 4])
>>> b = np.sqrt(a)
>>> b
array([1.          , 1.41421356, 1.73205081, 2.          ])
>>> np.ceil(b)
array([1., 2., 2., 2.])
>>> np.maximum(a/2, b)
array([1.          , 1.41421356, 1.73205081, 2.          ])
>>> f, i = np.modf(b)
>>> f
array([0.          , 0.41421356, 0.73205081, 0.          ])
>>> i
array([1., 1., 1., 2.])
```

Array Broadcasting

- Allows arithmetic operations on arrays with different shapes
- The smaller array is "broadcast" across the larger array so that they have compatible shapes

$$\begin{array}{c} \text{np.arange(3)} + 5 \\ \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline 5 & 5 & 5 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 5 & 6 & 7 \\ \hline \end{array} \end{array}$$
$$\begin{array}{c} \text{np.ones((3, 3))} + \text{np.arange(3)} \\ \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 0 & 1 & 2 \\ \hline 0 & 1 & 2 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 1 & 2 & 3 \\ \hline 1 & 2 & 3 \\ \hline \end{array} \end{array}$$
$$\begin{array}{c} \text{np.arange(3).reshape((3, 1))} + \text{np.arange(3)} \\ \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 1 & 1 & 1 \\ \hline 2 & 2 & 2 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 0 & 1 & 2 \\ \hline 0 & 1 & 2 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 1 & 2 & 3 \\ \hline 2 & 3 & 4 \\ \hline \end{array} \end{array}$$

Broadcasting Rule

- The size of the trailing axes for both arrays in an operation must either be the same size or one of them must be one

Image	(3d array)	256 x	256 x	3
Scale	(1d array)			3
Result	(3d array)	256 x	256 x	3

A	(4d array)	8 x	1 x	6 x	1
B	(3d array)		7 x	1 x	5
Result	(4d array)	8 x	7 x	6 x	5

Broadcasting Example (I)

```
>>> a = np.array([1, 2, 3])
>>> b = 2
>>> a * b
array([2, 4, 6])
>>> a = np.array([[ 0.0,  0.0,  0.0],
...                 [10.0, 10.0, 10.0],
...                 [20.0, 20.0, 20.0],
...                 [30.0, 30.0, 30.0]])
>>> b = array([1.0, 2.0, 3.0])
>>> a + b
array([[ 1.,   2.,   3.],
       [ 11.,  12.,  13.],
       [ 21.,  22.,  23.],
       [ 31.,  32.,  33.]])
```

a:	3
b:	1
result:	3
a:	4 x 3
b:	3
result:	4 x 3

Broadcasting Example (2)

```
>>> a = np.array([0.0, 10.0, 20.0, 30.0])
>>> b = np.array([1.0, 2.0, 3.0])
>>> a[:, np.newaxis] + b
array([[ 1.,  2.,  3.],
       [ 11., 12., 13.],
       [ 21., 22., 23.],
       [ 31., 32., 33.]])
```

```
>>> a = np.arange(12).reshape(4, 3)
```

```
>>> b = np.array([1, 2, 3, 4])
>>> a + b
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: operands could not be broadcast
together with shapes (4,3) (4,)

*Increase a
dimension:
(4,) → (4, 1)*

a: 4 x 1
b: 3
result: 4 x 3

~~a: 4 x 3
b: 4
result:~~

Conditional Operation

- **np.where(condition, [x, y])**

- When only *condition* is provided, return the indices of the elements that the condition is met
- Otherwise return elements chosen from *x* or *y* depending on condition

```
>>> a = np.array([[1,2], [4,5]])
>>> np.where(a % 2 == 0)
(array([0, 1]), array([1, 0]))
>>> a[np.where(a % 2 == 0)]
array([2, 4])
>>> a[a % 2 == 0]
array([2, 4])
```

```
>>> b = np.array([[9,8], [7,6]])
>>> np.where(a % 2 == 0, a, b)
array([[9, 2],
       [4, 6]])
>>> np.where(a>2, a+1, 0)
array([[0, 0],
       [5, 6]])
```

Statistical Operations

Statistical Operations

- Mean

$$\text{Mean}(x) = \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

- Variance

$$\text{Var}(x) = \sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

- Median

- The value in the middle
- If n is an even number, take the average of the two middle values

$$\text{Median}(x) = \frac{x_{\lfloor(n+1)/2\rfloor} + x_{\lceil(n+1)/2\rceil}}{2}$$

- Standard deviation

$$\text{Std}(x) = \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

sum() and prod()

- **a.sum([axis], [dtype], ...)**

- Return the **sum** of the array elements over the given *axis*
- If *axis* is not given, the sum of all the elements is computed
- Similar to *np.sum(a, ...)*

- **a.prod([axis], [dtype], ...)**

- Return the **product** of the array elements over a given *axis*
- If *axis* is not given, the product of all the elements is computed
- Similar to *np.prod(a, ...)*

```
>>> a = np.arange(1, 11)
>>> a.sum()
55
>>> a.prod()
3628800
>>> x = np.arange(1,13).reshape(3,4)
>>> x
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
>>> x.sum(axis=0) # along the rows
array([15, 18, 21, 24])
>>> x.sum(axis=1) # along the cols
array([ 10,  26,  42])
```

nansum() and nanprod()

- **np.nansum(a, [axis], [dtype], ...)**
 - Return the **sum** of the array elements over the *axis* treating NaNs as zero
 - If *axis* is not given, the sum of all the elements is computed
 - No *a.nansum(...)* form available!
- **np.nanprod(a, [axis], [dtype], ...)**
 - Return the **product** of the array elements treating NaNs as ones
 - If *axis* is not given, the product of all the elements is computed
 - No *a.nanprod(...)* form available!

```
>>> a = np.array([[1.0, 2.0, np.nan],  
[4.0, np.nan, 3.0]])  
>>> a  
array([[ 1.,  2., nan],  
       [ 4., nan,  3.]])  
>>> np.nansum(a)  
10.0  
>>> np.nansum(a, axis=1)  
array([3., 7.])  
>>> np.nanprod(a)  
24.0  
>>> np.nanprod(a, axis=0)  
array([4., 2., 3.])
```

mean() and var()

- **a.mean([axis], [dtype], ...)**

- Return the **average** of the array elements over the given *axis*
- If *axis* is not given, compute the mean of the flattened array
- Similar to *np.mean(a, ...)*

- **a.var([axis], [dtype], ...)**

- Return the **variance** of the array elements over a given *axis*
- If *axis* is not given, compute the variance of the flattened array
- Similar to *np.var(a, ...)*

```
>>> a = np.arange(1, 11)
>>> a.mean()
5.5
>>> a.var()
8.25
>>> x = np.arange(1,13).reshape(3,4)
>>> x
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
>>> x.mean()
6.5
>>> x.var()
11.916666666666666
```

std() and median()

- **a.std([axis], [dtype], ...)**
 - Return the **standard deviation** of the array elements over the given *axis*
 - If *axis* is not given, compute the standard deviation of the flattened array
 - Similar to **np.std(a, ...)**
- **np.median(a, [axis], [dtype], ...)**
 - Return the **median** along the given *axis*
 - If *axis* is not given, compute the median of the flattened array
 - No **a.median(...)** form available!

```
>>> a = np.arange(1, 11)
>>> a.std()
2.8722813232690143
>>> a.median()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'numpy.ndarray' object
has no attribute 'median'
>>> np.median(a)
5.5
x = np.arange(1,13).reshape(3,4)
>>> x.std()
3.452052529534663
>>> np.median(x)
6.5
```

Why NumPy?

- NumPy statistics is much easier than nested list statistics
- Python List vs. Numpy.array's sum() of 2D data

```
>>> a = np.arange(1,10).reshape(3,3)
>>> a
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> a.sum()
45
```

```
def nested_sum(L):
    sum = 0
    for i in L:
        if isinstance(i, list):
            sum += nested_sum(i)
        else:
            sum += i
    return sum;
>>> b = [[1,2,3], [4,5,6], [7,8,9]]
>>> nested_sum(b)
```

45

Covariance

■ Covariance (공분산)

- A measure of the joint variability of two random variables

$$Cov(X, Y) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

- If the greater values of one variable mainly correspond with the greater values of the other variable, and the same holds for the lesser values (i.e., the variables tend to show similar behavior), the covariance is positive
- In the opposite case, the covariance is negative
- The sign of the covariance therefore shows the tendency in the linear relationship between the variables

Correlation Coefficient

- Pearson (product-moment) correlation coefficient (상관계수), r

- A measure of correlation between two variables X and Y

- $-1 \leq r \leq 1$ where

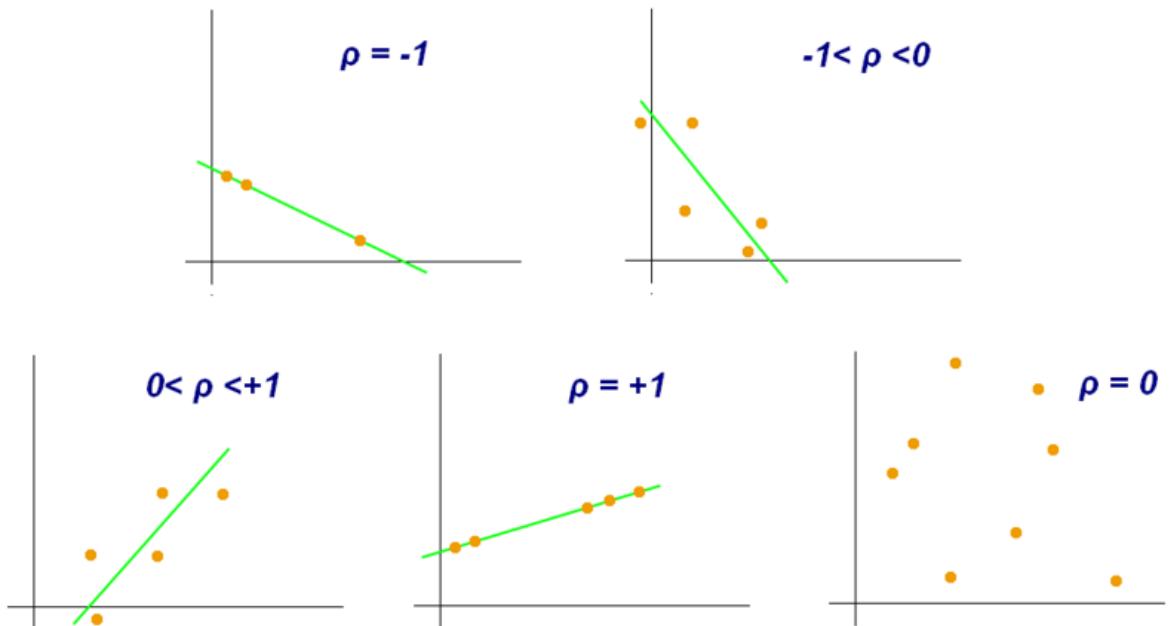
- 0: no linear correlation,

- 1: total positive correlation,

- 1: total negative correlation

$$r = \frac{Cov(X, Y)}{\sigma_X \sigma_Y}$$

$$= \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$



cov() and corrcoef()

■ `np.cov(m, [bias], ...)`

- Estimate a covariance matrix, given data and weights
- Covariance indicates the level to which two variables vary together
- For N-dimensional samples, $X = [x_1, x_2, \dots, x_n]^T$, the covariance matrix element C_{ij} is the covariance of x_i and x_j . (C_{ii} is the variance of x_i)
- *m*: A 1-D or 2-D array containing multiple variables and observations
- *bias*: If False, normalization is by $(N - 1)$, otherwise by N (default: False)

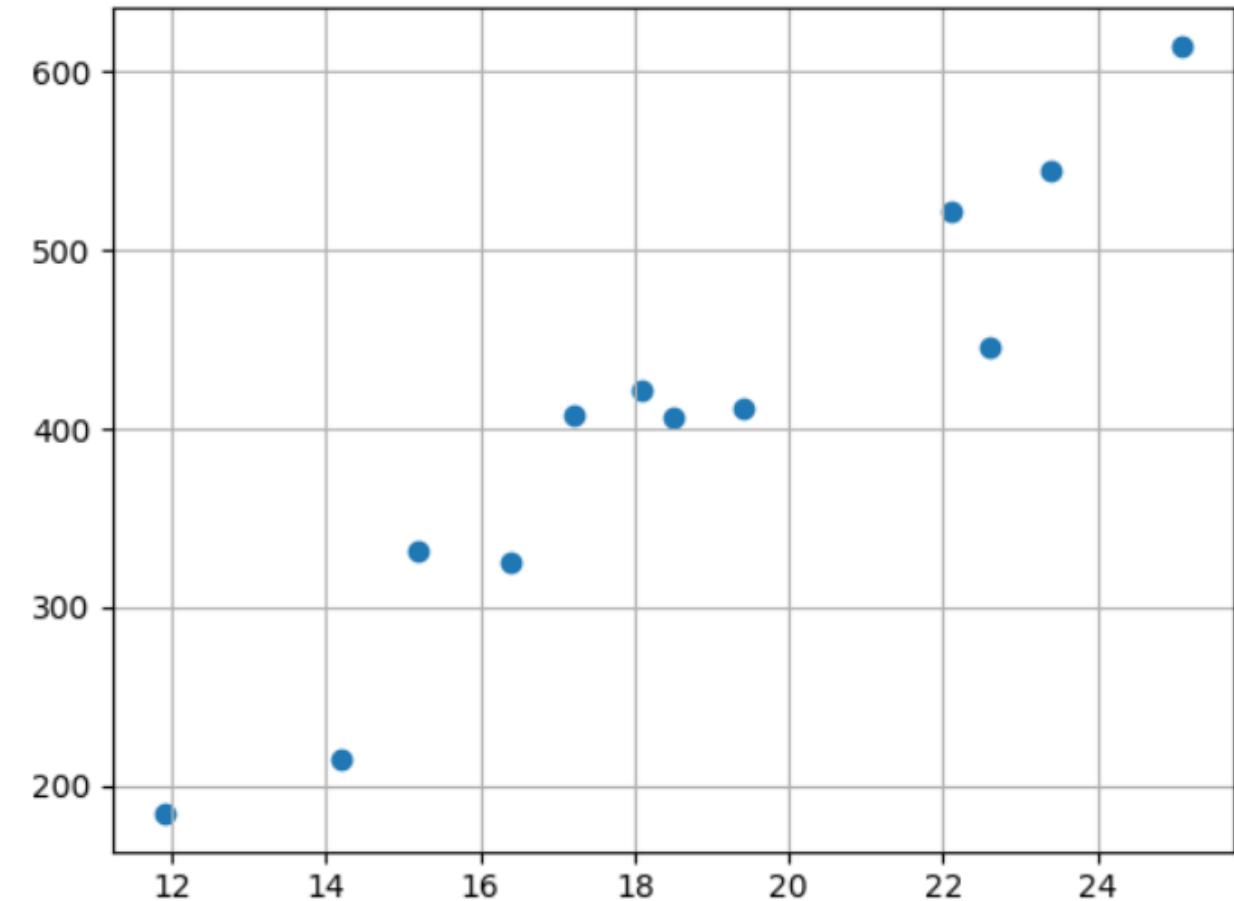
■ `a.corrcoef(x, ...)`

- Return Pearson product-moment correlation coefficients
- The relationship between the correlation coefficient matrix, R , and the covariance matrix, C , is:

$$R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} * C_{jj}}}$$

Example: Temperature vs. Ice Cream Sales

Temperature (°C)	Ice Cream Sales
14.2	\$215
16.4	\$325
11.9	\$185
15.2	\$332
18.5	\$406
22.1	\$522
19.4	\$412
25.1	\$614
23.4	\$544
18.1	\$421
22.6	\$445
17.2	\$408



Example: Temperature vs. Ice Cream Sales

```
t = np.array([14.2, 16.4, 11.9, 15.2,  
             18.5, 22.1, 19.4, 25.1, 23.4,  
             18.1, 22.6, 17.2])  
  
s = np.array([215, 325, 185, 332, 406,  
             522, 412, 614, 544, 421, 445, 408])  
  
C = np.cov([t, s])  
  
R = np.corrcoef([t, s])  
  
print(C)  
  
print(R)  
  
# import matplotlib.pyplot as plt  
# plt.scatter(t, s)  
# plt.grid(True)  
# plt.show()
```

```
[[16.08931818 484.09318182]  
 [484.09318182 15886.81060606]]  
  
[[1.          0.95750662]  
 [0.95750662 1.          ]]
```

Matrix Operations

Array vs. Matrix

- Numpy matrices are strictly 2-dimensional, while numpy arrays (ndarrays) are N-dimensional
- Matrix objects are a subclass of ndarray, so they inherit all the attributes and methods of ndarrays
- Numpy matrices provide a convenient notation for matrix multiplication
 - If a and b are matrices, then a*b is their matrix product

```
>>> a = np.array([[4, 3], [2, 1]])
>>> b = np.array([[1, 2], [3, 4]])
>>> a * b
array([[4, 6],
       [6, 4]])
```

```
>>> ma = np.mat(a)
>>> mb = np.mat(b)
>>> ma * mb
matrix([[13, 20],
        [ 5,  8]])
```

Creating Matrices

- **`np.matrix(data[, dtype][, copy])`**

- Return a matrix from an array-like object, or from a string of data
- If `data` is a string, it is interpreted as a matrix with commas or spaces separating columns, and semicolons separating rows

- **`np.mat(data[, dtype])`**

- Interpret the input as a matrix
- Unlike `matrix()`, `mat()` does not make a copy if the input is already a matrix or an ndarray

```
>>> m = np.matrix('1 2; 3 4')
>>> m
matrix([[1, 2],
        [3, 4]])
>>> np.matrix([[1, 2], [3, 4]])
matrix([[1, 2],
        [3, 4]])
>>> y = np.array([[1,2], [3,4]])
>>> my = np.mat(y)
>>> my
matrix([[1, 2],
        [3, 4]])
>>> np.asarray(my)
array([[1, 2],
        [3, 4]])
```

Array vs. Matrix: Comparison

To Be Deprecated

	array	matrix
Dimensions	Number of dimensions can be larger than 2	Exactly two dimensions
Operator *	Element-wise multiplication	Matrix multiplication
Operator @	Matrix multiplication	Matrix multiplication
np.multiply()	Element-wise multiplication	Element-wise multiplication
np.dot()	Matrix multiplication	Matrix multiplication
Handling vectors	1-dimensional	2-dimensional with 1xN (row vector) or Nx1 (column vector) shape
Attributes	.T (transpose)	.T (transpose), .A (asarray()), .H (conjugate transpose), .I (inverse)
Initialization	Can use Python sequences e.g., <code>array([[1,2,3], [4,5,6]])</code>	Additionally, can use a convenient string initializer e.g., <code>np.matrix('1 2 3; 4 5 6')</code>

Product Operations in Vector/Matrix

■ For vectors

- Inner product (벡터내적) Supported by `np.dot()` or `np.inner()`
- Outer product (벡터외적) Supported by `np.outer()`
- Cross product (벡터곱) Supported by `np.cross()`

■ For matrices

- Matrix multiplication (행렬곱) Supported by `np.dot()` or `np.matmul()`
- Inner product (행렬내적) Supported by `np.inner()`

Vector Inner Product (벡터내적)

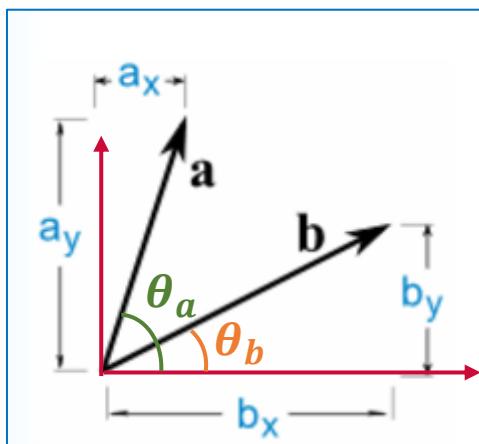
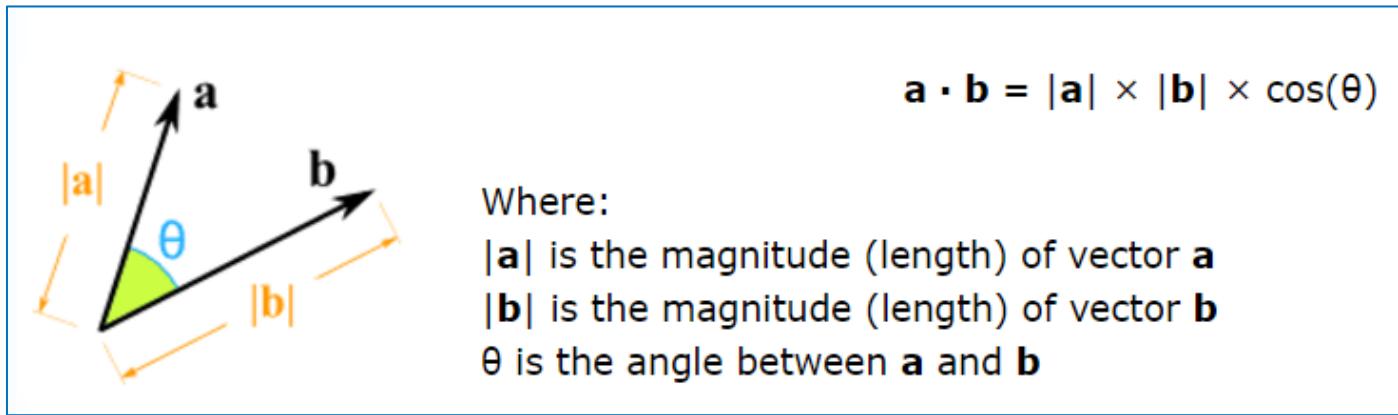
- Vector · Vector → Scalar
- The inner product of two vectors in matrix form:

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= \mathbf{a}^T \mathbf{b} \\ &= (\mathbf{a}_1 \ \mathbf{a}_2 \ \cdots \ \mathbf{a}_n) \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_n \end{pmatrix} \\ &= \mathbf{a}_1 \mathbf{b}_1 + \mathbf{a}_2 \mathbf{b}_2 + \cdots + \mathbf{a}_n \mathbf{b}_n \\ &= \sum_{i=1}^n \mathbf{a}_i \mathbf{b}_i \end{aligned}$$

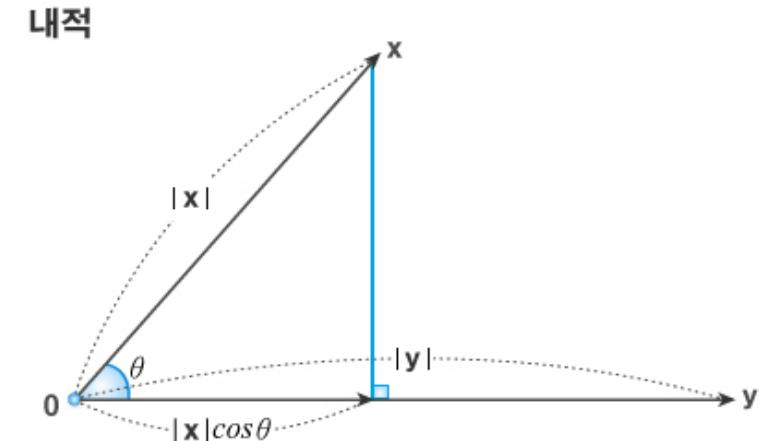
$$(\mathbf{a} \ \mathbf{b} \ \mathbf{c}) \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} = \mathbf{a} + 4\mathbf{b} + 7\mathbf{c}$$

Application of Vector Inner Product

- Compute the angle between two vectors



$$\begin{aligned} a_x &= |\mathbf{a}| \cos \theta_a & b_x &= |\mathbf{b}| \cos \theta_b \\ a_y &= |\mathbf{a}| \sin \theta_a & b_y &= |\mathbf{b}| \sin \theta_b \\ \mathbf{a} \cdot \mathbf{b} &= |\mathbf{a}| |\mathbf{b}| \cos(\theta_a - \theta_b) \\ &= |\mathbf{a}| |\mathbf{b}| (\cos \theta_a \cos \theta_b + \sin \theta_a \sin \theta_b) \\ &= |\mathbf{a}| \cos \theta_a |\mathbf{b}| \cos \theta_b + |\mathbf{a}| \sin \theta_a |\mathbf{b}| \sin \theta_b \\ &= a_x b_x + a_y b_y \end{aligned}$$



$$\cos \theta = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|}$$

$$\theta = \arccos \left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} \right)$$

dot() and inner()

■ *np.dot(a, b, ...)*

- If both *a* and *b* are 1-D arrays, return the inner product of vectors
- Otherwise, return different results depending on the input dimensions

■ *np.inner(a, b, ...)*

- Return the inner product of vectors for 1-D arrays
- In higher dimensions, return the sum product over the last axes
- == `sum(a*b)`

```
>>> a = np.array([1, 2, 3], float)
>>> b = np.array([0, 1, 1], float)
>>> np.dot(a, b)
5.0
>> np.dot(a, 2)      # scalar
array([2., 4., 6.])
>>> np.inner(a, b)
5.0
>>> np.inner(3, b)    # scalar
array([0., 3., 3.])
>>> sum(a*b)
5.0
```

Vector Outer Product (벡터외적)

- Vector · Vector → Matrix
- The outer product of two vectors in matrix form:

$$a \otimes b = ab^T$$

$$= \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} (b_1 \ b_2 \ \cdots \ b_n)$$

$$\begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} (a \ b) = \begin{pmatrix} 1a & 1b \\ 4a & 4b \\ 7a & 7b \end{pmatrix}$$

$$= \begin{pmatrix} a_1b_1 & a_1b_2 & \cdots & a_1b_n \\ a_2b_1 & a_2b_2 & \cdots & a_2b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_nb_1 & a_nb_2 & \cdots & a_nb_n \end{pmatrix}$$

Application of Vector Outer Product

- Matrix multiplication can be implemented using vector outer product

$$\begin{aligned}\mathbf{AB} &= (\bar{\mathbf{a}}_1 \quad \bar{\mathbf{a}}_2 \quad \cdots \quad \bar{\mathbf{a}}_m) \begin{pmatrix} \bar{\mathbf{b}}_1 \\ \bar{\mathbf{b}}_2 \\ \vdots \\ \bar{\mathbf{b}}_m \end{pmatrix} \\ &= \bar{\mathbf{a}}_1 \otimes \bar{\mathbf{b}}_1 + \bar{\mathbf{a}}_2 \otimes \bar{\mathbf{b}}_2 + \cdots + \bar{\mathbf{a}}_m \otimes \bar{\mathbf{b}}_m \\ &= \sum_{i=1}^m \bar{\mathbf{a}}_i \otimes \bar{\mathbf{b}}_i\end{aligned}$$

where this time

$$\bar{\mathbf{a}}_i = \begin{pmatrix} A_{1i} \\ A_{2i} \\ \vdots \\ A_{ni} \end{pmatrix}, \quad \bar{\mathbf{b}}_i = (B_{i1} \quad B_{i2} \quad \cdots \quad B_{ip}).$$

$$\begin{aligned}\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix} &= \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} \otimes (a \quad d) + \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix} \otimes (b \quad e) + \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix} \otimes (c \quad f) \\ &= \begin{pmatrix} 1a & 1d \\ 4a & 4d \\ 7a & 7d \end{pmatrix} + \begin{pmatrix} 2b & 2e \\ 5b & 5e \\ 8b & 8e \end{pmatrix} + \begin{pmatrix} 3c & 3f \\ 6c & 6f \\ 9c & 9f \end{pmatrix} \\ &= \begin{pmatrix} 1a + 2b + 3c & 1d + 2e + 3f \\ 4a + 5b + 6c & 4d + 5e + 6f \\ 7a + 8b + 9c & 7d + 8e + 9f \end{pmatrix}.\end{aligned}$$

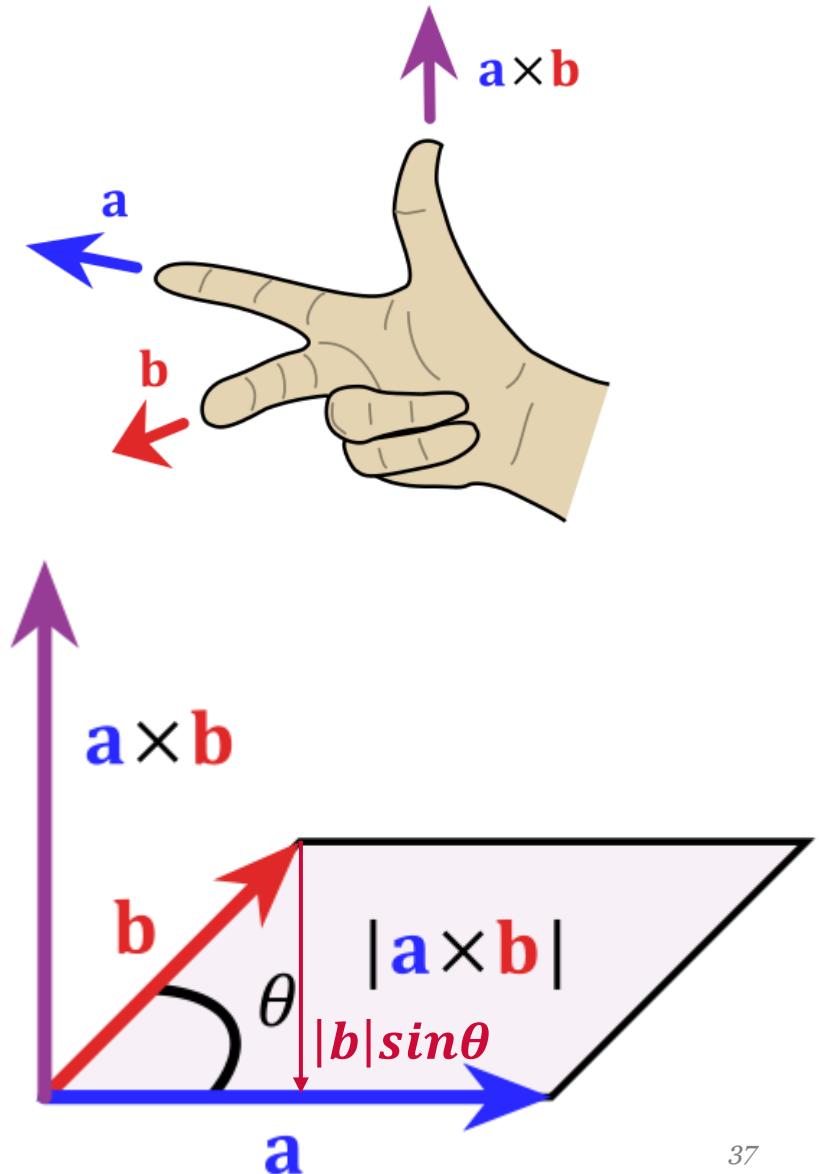
Vector Cross Product (벡터곱)

- Vector \times Vector \rightarrow Vector
- The cross product is defined by the formula

$$\mathbf{a} \times \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \sin\theta \mathbf{n}$$

where θ is the angle between \mathbf{a} and \mathbf{b} , and \mathbf{n} is a unit vector perpendicular to the plane containing \mathbf{a} and \mathbf{b} with a magnitude equal to the area of the parallelogram that the vectors span

$$|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}| |\mathbf{b}| \sin\theta$$



outer() and cross()

■ **np.outer(a, b, ...)**

- Compute the outer product of two vectors

```
>>> x = np.outer(np.ones((5,)),  
np.linspace(-2, 2, 5))  
  
>>> x  
array([[ -2., -1.,  0.,  1.,  2.],  
       [-2., -1.,  0.,  1.,  2.],  
       [-2., -1.,  0.,  1.,  2.],  
       [-2., -1.,  0.,  1.,  2.],  
       [-2., -1.,  0.,  1.,  2.]])
```

■ **np.cross(a, b, ...)**

- Return the cross product of two vectors

```
>>> x = np.array([1,4,0])  
>>> y = np.array([2,2,1])  
  
>>> np.cross(x,y)  
array([ 4, -1, -6])
```

Matrix Multiplication (행렬곱)

$$\mathbf{A} = \begin{pmatrix} a & b & c \\ x & y & z \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} \alpha & \rho \\ \beta & \sigma \\ \gamma & \tau \end{pmatrix},$$

their matrix products are:

$$\mathbf{AB} = \begin{pmatrix} a & b & c \\ x & y & z \end{pmatrix} \begin{pmatrix} \alpha & \rho \\ \beta & \sigma \\ \gamma & \tau \end{pmatrix} = \begin{pmatrix} a\alpha + b\beta + c\gamma & a\rho + b\sigma + c\tau \\ x\alpha + y\beta + z\gamma & x\rho + y\sigma + z\tau \end{pmatrix},$$

and

$$\mathbf{BA} = \begin{pmatrix} \alpha & \rho \\ \beta & \sigma \\ \gamma & \tau \end{pmatrix} \begin{pmatrix} a & b & c \\ x & y & z \end{pmatrix} = \begin{pmatrix} \alpha a + \rho x & \alpha b + \rho y & \alpha c + \rho z \\ \beta a + \sigma x & \beta b + \sigma y & \beta c + \sigma z \\ \gamma a + \tau x & \gamma b + \tau y & \gamma c + \tau z \end{pmatrix}.$$

dot() and matmul()

■ `np.dot(a, b, ...)`

- If both `a` and `b` are 2-D arrays, return the result of matrix multiplication
- The use of `matmul()` or `a @ b` is preferred

■ `np.matmul(a, b, ...)`

- Return the matrix product of two arrays

```
>>> a = np.array([[0, 1], [2, 3]])
>>> b = np.array([2, 3])
>>> c = np.array([[1, 1], [4, 0]])
>>> np.dot(b, a)
array([ 6, 11])
>>> np.dot(a, b)
array([ 3, 13])
>>> np.matmul(a, c)
array([[ 4,  0],
       [14,  2]])
>>> c @ a
array([[2, 4],
       [0, 4]])
```

$$\begin{aligned}b \bullet a &\rightarrow [2 \ 3] \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \\a \bullet b &\rightarrow \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} [2] \\a \bullet c &\rightarrow \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 4 & 0 \end{bmatrix} \\c \bullet a &\rightarrow [1 \ 1] \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}\end{aligned}$$

Summary

- NumPy Core

- Array creation, Array manipulation, Binary operations, String operations, ...

- Submodules

- `numpy.rec`: Creating record arrays
- `numpy.char`: Creating character arrays
- `numpy.ctypeslib`: C-types Foreign Function Interface
- `numpy.dual`: Optionally Scipy-accelerated routines
- `numpy.emath`: Mathematical functions with automatic domain
- `numpy.fft`: Discrete Fourier Transform
- `numpy.linalg`: Linear Algebra
- `numpy.matlib`: Matrix Library
- `numpy.random`: Random sampling
- `numpy.testing`: Test support

File I/O

- **np.savetxt(fname, A[, delimiter], ...)**
 - Save an array to a text file named *fname*

```
a = np.array([[1, 2, 3], [4, 5, 6]])
np.savetxt(r'C:\Users\jinsoo\Desktop\spam.csv', a, delimiter=',')
np.savetxt(r'C:\Users\jinsoo\Desktop\spam.txt', a, delimiter=' ')
```

A screenshot of Microsoft Excel showing a 2x3 matrix of numbers. The matrix is defined by the code above. The first row contains 1.000E+00, 2.000E+00, and 3.000E+00. The second row contains 4.000E+00, 5.000E+00, and 6.000E+00. The spreadsheet has columns labeled A, B, C and rows labeled 1, 2, 3. The active cell is A1.

1.000E+00	2.000E+00	3.000E+00
4.000E+00	5.000E+00	6.000E+00
3		

A screenshot of Windows Notepad showing two files. The left window is titled "spam.csv - Windows 메모장" and contains the following text:
1.000000000000000e+00 2.000000000000000e+00 3.000000000000000e+00
4.000000000000000e+00 5.000000000000000e+00 6.000000000000000e+00

The right window is titled "spam.txt - Windows 메모장" and contains the same text.

```
1.000000000000000e+00 2.000000000000000e+00 3.000000000000000e+00
4.000000000000000e+00 5.000000000000000e+00 6.000000000000000e+00
```

File I/O (cont'd)

- **np.loadtxt(fname[, dtype][, delimiter], ...)**

- Load data from a text file named *fname*

```
b = np.loadtxt(r'C:\Users\jinsoo\Desktop\spam.csv', delimiter=',')
print(b)
c = np.loadtxt(r'C:\Users\jinsoo\Desktop\spam.txt', dtype=int,
delimiter=' ')
print(c)
```

```
[[1., 2., 3.]
 [4., 5., 6.]]
[[1, 2, 3]
 [4, 5, 6]]
```

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022



Python for Data Analytics

Pandas I

Outline

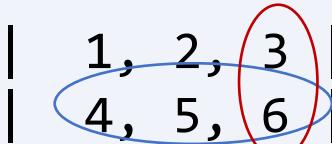
- Why Pandas?
- Pandas Series
- Pandas DataFrame
 - Creating DataFrame
 - Manipulating Columns
 - Manipulating Rows
 - Arithmetic operations
 - Group Aggregation
 - Hierarchical Indexing
 - Combining and Merging
 - Time Series Data

Why Pandas?

Limitations in NumPy

- Remember? Array slicing in NumPy

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])  
>>> a[1, :]  
array([[4, 5, 6]])  
>>> a[:, 2]  
array([3, 6])  
>>> a[-1:, -2:]  
array([[5, 6]])
```



How about?

	AAPL_High	AAPL_Low
Date		
2010-01-04	214.499996	212.380001
2010-01-05	215.589994	213.249994
2010-01-06	215.230000	210.750004
2010-01-07	212.000006	209.050005
2010-01-08	212.000006	209.060005

2010-01-06 ~ 2010-01-07
사이에 발생한 data 추출?

2010년에 월별로 발생한
data를 grouping?

Limitations in NumPy (cont'd)

How about?

	AAPL_High	AAPL_Low
Date		
2010-01-04	214.499996	212.380001
2010-01-05	215.589994	213.249994
2010-01-06	215.230000	210.750004
2010-01-07	212.000006	209.050005
2010-01-08	212.000006	209.060005

	GOOG_High	GOOG_Low
Date		
2010-01-04	629.511067	624.241073
2010-01-05	627.841071	621.541045
2010-01-06	625.861078	606.361042
2010-01-07	610.001045	592.651008
2010-01-08	603.251034	589.110988



	AAPL_High	AAPL_Low	GOOG_High	GOOG_Low
Date				
2010-01-04	214.499996	212.380001	629.511067	624.241073
2010-01-05	215.589994	213.249994	627.841071	621.541045
2010-01-06	215.230000	210.750004	625.861078	606.361042
2010-01-07	212.000006	209.050005	610.001045	592.651008
2010-01-08	212.000006	209.060005	603.251034	589.110988

두 테이블의 join?

SQL and Tables (I)

- Find all instructors in Comp. Sci. dept. with salary > 80000

```
select name  
from instructor  
where dept_name = 'Comp. Sci.' and salary > 80000;
```

Instructor relation			
ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

ID	name	dept_name	salary
83821	Brandt	Comp. Sci.	92000

SQL and Tables (2)

- For all instructors who have taught courses, find their names and the course ID of the courses they taught

```
select name, course_id  
from instructor, teaches  
where instructor.ID = teaches.ID;
```

instructor	ID	name	dept_name	salary
	10101	Srinivasan	Comp. Sci.	65000
	12121	Wu	Finance	90000
	15151	Mozart	Music	40000
	22222	Einstein	Physics	95000
	32343	El Said	History	60000
	45565	Katz	Math	75000

teaches	ID	course_id	sec_id	semester	year
	10101	CS-101	1	Fall	2009
	10101	CS-315	1	Spring	2010
	10101	CS-347	1	Fall	2009
	12121	FIN-201	1	Spring	2010
	15151	MU-199	1	Spring	2010
	22222	PHY-101	1	Fall	2009

```
select *  
from instructor natural join teaches;
```

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

SQL and Tables (3)

- Group instructors in each department

```
select *  
from instructor  
group by dept_name;
```

Instructor relation			
ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

The diagram illustrates the grouping process. A blue arrow points from the original 'Instructor relation' table to a second table, indicating that the rows are being categorized by department. Another blue arrow points from the second table to a third table, further refining the groupings.

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

What is "Pandas" Module?

- panel data analysis or Python data analysis
- For building and manipulating "relational" or "tabular" data both easy and intuitive
- Built on top of NumPy (2005)
- Open source
 - Original author: Wes McKinney
 - Now part of the PyData project focused on improving Python data libraries
 - <http://pandas.pydata.org>
- `>>> import pandas as pd`

Pandas History

- Developer [Wes McKinney](#) started working on Pandas in 2008 while at AQR Capital Management (global investment management firm)
- Need for a [high performance, flexible analysis tool for quantitative analysis on financial data](#)
- Before leaving AQR, he was able to convince management to allow him to open source the library
- Another AQR employee, Chang She, joined the effort in 2012 as the second major contributor to the library
- In 2015, Pandas signed on as a sponsored project of NumFOCUS, a non-profit charity in United States

Pandas Module

- Primary data structures
 - **Series** (1-dimensional)
 - **DataFrame** (2-dimensional) -- similar to `data.frame` in R
 - **Panel** (3-dimensional or more)
- Things that pandas does well
 - Easy handling of [missing data](#)
 - [Size mutability](#): columns can be inserted and deleted ([Add & drop columns](#))
 - [Powerful, flexible group by](#) functionality: [Groupby & aggregation](#)
 - [Intelligent label-based slicing, fancy indexing, and subsetting](#) of large data sets
 - [Intuitive merging and joining](#) data sets: [Join \(merge\) two data](#)
 - [Robust I/O tools](#) for loading data from CSV & Excel files, database, and web sources

Series

Series

- A **one-dimensional array-like object containing a sequence of values and an associated array of data labels, called its index**

Index (Integer)	Data
0	1.0
1	2.1
2	1.5
3	4.7
4	3.2
5	1.9

Index (Label)	Data
'Sun'	0
'Mon'	8
'Tue'	9
'Wed'	8
'Thu'	10
'Fri'	6
'Sat'	4

Creating Series (I)

- From a Python list

```
import numpy as np
import pandas as pd
s = pd.Series([1,3,np.nan,6,8])
s
```

```
0    1.0
1    3.0
2    NaN
3    6.0
4    8.0
dtype: float64
```

automatic indexing
(record id/key)

- From a NumPy ndarray

```
import numpy as np
import pandas as pd
a = np.array([1,3,np.nan,6,8])
s = pd.Series(a)
s
```

```
0    1.0
1    3.0
2    NaN
3    6.0
4    8.0
dtype: float64
```



Creating Series (2)

- From a Python dictionary

```
d = {'spam':5.99, 'egg':0.99, 'ham':3.99}  
s = pd.Series(d)  
s
```

```
spam      5.99  
egg       0.99  
ham       3.99  
dtype: float64
```

- Each element doesn't have to be the same type

```
l = [ 'pi', 3.14, 'ABC', 100, True ]  
s = pd.Series(l)  
s
```

```
0      pi  
1    3.14  
2     ABC  
3    100  
4    True  
dtype: object
```

Index Labels

- Index labels can be specified when Series is created
- Index labels can be changed in-place

```
data = [100, 200, 300, 400]
labels = ['a','b','c','d']
s = pd.Series(data, index=labels)
s
```

```
a    100
b    200
c    300
d    400
dtype: int64
```

```
s.index = ['W','X','Y','Z']
s
```

```
W    100
X    200
Y    300
Z    400
dtype: int64
```

pandas.Series()

- `pd.Series([data], [index], [dtype], ...)`
 - One-dimensional ndarray with axis labels (including time series)
 - `data`: contains data stored in Series
 - `index`: values must be hashable and have the same length as `data`
(default: `np.arange(len(data))`)
 - Non-unique index values are allowed

```
a = [2, 4, 5, 8]
b = ['a', 'b', 'c', 'c']
s = pd.Series(a)
s
```

```
0    2
1    4
2    5
3    8
dtype: int64
```

```
s2 = pd.Series(a, b)
s2
```

```
a    2
b    4
c    5 ←
c    8 ←
dtype: int64
```

Handling Missing Entries

- Series creation from dictionary
- Extracting data from another dictionary

```
data = {'Ohio':35000, 'Texas':71000,  
        'Oregon':16000, 'Utah':5000}  
s = pd.Series(data)  
s
```

```
Ohio      35000  
Texas     71000  
Oregon    16000  
Utah      5000  
dtype: int64
```

```
states = ['California', 'Ohio',  
          'Oregon', 'Texas']  
s2 = pd.Series(data, index=states)  
s2
```

California	NaN	← value unknown
Ohio	35000.0	
Oregon	16000.0	
Texas	71000.0	
dtype:	float64	

Checking Null Values

- `pd.isnull(obj)`

- `pd.isna(obj)`

- Return an array of Boolean indicating whether the corresponding element is missing
- Same as `obj.isnull()`

```
pd.isnull(s2)
```

California	True
Ohio	False
Oregon	False
Texas	False
<code>dtype: bool</code>	

```
s2.notnull()
```

California	False
Ohio	True
Oregon	True
Texas	True
<code>dtype: bool</code>	

Unique Values and Value Counts

- `series.unique()`

- Return unique values of Series object

```
s = pd.Series(np.random.randint(0,3,5),  
              index=list('ABCDE'))
```

```
s  
  
A    1  
B    2  
C    0  
D    2  
E    1  
dtype: int32
```

```
s.unique()
```

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

- `series.value_counts(normalize=False, sort=True, ascending=False, ...)`

- Return a Series containing counts of unique values

```
s.value_counts()
```

```
2    2  
1    2  
0    1  
dtype: int64
```

```
s.value_counts(normalize=True)
```

```
2    0.4  
1    0.4  
0    0.2  
dtype: float64
```

Selecting Elements

```
d = {'a':1, 'b':2, 'c':3, 'd':4, 'e':5}  
s = pd.Series(d)  
s
```

```
a    1  
b    2  
c    3  
d    4  
e    5  
dtype: int64
```

```
s[2]
```

```
3
```

```
s['c']
```

```
3
```

```
s[2:4] exclusive!
```

```
c    3  
d    4  
dtype: int64
```

```
s['c':'e'] inclusive!
```

```
c    3  
d    4  
e    5  
dtype: int64
```

```
s[s > 3]
```

```
d    4  
e    5  
dtype: int64
```

```
s[[1,2,4]]
```

```
b    2  
c    3  
e    5  
dtype: int64
```

```
s[['c','e','a']]
```

```
c    3  
e    5  
a    1  
dtype: int64
```

Manipulating Series

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

```
a    2  
b    4  
c    6  
dtype: int64
```

```
2**s  
  
a    2  
b    4  
c    8  
dtype: int64
```

```
np.exp(s)
```

```
a    2.718282  
b    7.389056  
c   20.085537  
dtype: float64
```

```
t = pd.Series({'a': 4, 'c':5, 'x':1, 'y':7})  
s + t
```

```
a    5.0  
b    NaN  
c    8.0  
x    NaN  
y    NaN  
dtype: float64
```

String Functions and `to_list()`

```
s
```

```
0           action,sf
1   drama,comic,romance
2           fantasy
dtype: object
```

```
s.str.replace(',', ' ')
```

```
0           [action, sf]
1   [drama, comic, romance]
2           [fantasy]
dtype: object
```

```
s.to_list()
```

```
['action,sf', 'drama,comic,romance', 'fantasy']
```

```
s.str.replace(',', ' ')
```

```
0           action sf
1   drama comic romance
2           fantasy
dtype: object
```

```
s.str.upper()
```

```
0           ACTION,SF
1   DRAMA,COMIC,ROMANCE
2           FANTASY
dtype: object
```

Creating DataFrame

DataFrame

- A 2-D array of indexed data
 - Similar to a spreadsheet or SQL table
- DataFrame is the most commonly used pandas object

	기간	자치구	세대	인구	인구.1	인구.2	인구.3	인구.4	인구.5	인구.6	인구.7	인구.8	세대당인구	65세이상고령자
0	기간	자치구	세대	합계	합계	합계	한국인	한국인	한국인	등록외국인	등록외국인	등록외국인	세대당인구	65세이상고령자
1	기간	자치구	세대	계	남자	여자	계	남자	여자	계	남자	여자	세대당인구	65세이상고령자
2	2020.3/4	합계	4,405,833	9,953,009	4,840,912	5,112,097	9,699,232	4,719,170	4,980,062	253,777	121,742	132,035	2.2	1,552,356
3	2020.3/4	종로구	74,861	159,842	77,391	82,451	149,952	73,024	76,928	9,890	4,367	5,523	2	28,396
4	2020.3/4	중구	63,594	135,321	66,193	69,128	125,800	61,526	64,274	9,521	4,667	4,854	1.98	24,265
5	2020.3/4	용산구	112,451	244,953	119,074	125,879	229,786	110,604	119,182	15,167	8,470	6,697	2.04	39,995
6	2020.3/4	성동구	136,096	302,695	147,582	155,113	295,591	144,444	151,147	7,104	3,138	3,966	2.17	45,372
7	2020.3/4	광진구	166,857	361,923	174,077	187,846	348,064	168,095	179,969	13,859	5,982	7,877	2.09	50,047

pandas.DataFrame()

- `pd.DataFrame([data], [index], [columns], [dtype], ...)`
 - The primary pandas data structure
 - Two-dimensional size-mutable, potentially heterogenous tabular data structure with labeled axes (rows and columns)
 - `data`: ndarray, list, dictionary, or dataframe
 - `index`: index to use for resulting frame.
(default: `np.arange(len(data))`)
 - `columns`: column labels to use for resulting frame

```
a = {'c0':[2, 3, 5, 8],  
      'c1':[12, 76, 32, 29]}  
b = ['a','b','c','d']  
s = pd.DataFrame(a)  
s
```

	c0	c1
0	2	12
1	3	76
2	5	32
3	8	29

```
s2 = pd.DataFrame(a, b)  
s2
```

	c0	c1
a	2	12
b	3	76
c	5	32
d	8	29

Dict → DataFrame

- From a Python dictionary of equal-length lists

```
d = { 'C1': ['A', 'B', 'C', 'D', 'E'],
      'C2': [10, 20, 30, 40, 50],
      'C3': [1.5, 2.7, 0.5, 3.2, 1.1] }
df = pd.DataFrame(d)
df
```

	C1	C2	C3
0	A	10	1.5
1	B	20	2.7
2	C	30	0.5
3	D	40	3.2
4	E	50	1.1

```
d = { 'C1': ['A', 'B', 'C', 'D', 'E'],
      'C2': [10, 20, 30, 40, 50],
      'C3': [1.5, 2.7, 0.5, 3.2, 1.1] }
df = pd.DataFrame(d, index=['R1','R2','R3','R4','R5'])
df
```

	C1	C2	C3
R1	A	10	1.5
R2	B	20	2.7
R3	C	30	0.5
R4	D	40	3.2
R5	E	50	1.1

Arranging Columns

- Part of columns can be selected
- Order of columns can be changed
- New columns can be added
 - Missing values are set to NaN

```
d = { 'C1': ['A', 'B', 'C', 'D', 'E'],
      'C2': [10, 20, 30, 40, 50],
      'C3': [1.5, 2.7, 0.5, 3.2, 1.1],
      }
df = pd.DataFrame(d, columns=['C3', 'C1', 'C4'],
                  index=['R'+str(i) for i in range(5)])
df
```

	C3	C1	C4
R0	1.5	A	NaN
R1	2.7	B	NaN
R2	0.5	C	NaN
R3	3.2	D	NaN
R4	1.1	E	NaN

Index and Column Names

- `df.index.name` and `df.columns.name`

```
d = { 'C1': [ 'A', 'B', 'C', 'D', 'E'],
      'C2': [ 10, 20, 30, 40, 50],
      'C3': [ 1.5, 2.7, 0.5, 3.2, 1.1] }
df = pd.DataFrame(d, index=['R1','R2','R3','R4','R5'])
df.index.name="My index"
df.columns.name="My columns"
df
```

My columns	C1	C2	C3	
My index	R1	A	10	1.5
	R2	B	20	2.7
	R3	C	30	0.5
	R4	D	40	3.2
	R5	E	50	1.1

NumPy ndarray → DataFrame

- From a NumPy ndarray

```
df = pd.DataFrame(np.arange(9).reshape((3,3)))  
df
```

	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8

```
df = pd.DataFrame(np.random.randn(5,4),  
                  index=list('ABCDE'),  
                  columns=list('WXYZ'))  
df
```

	W	X	Y	Z
A	-0.021447	-0.345292	0.245482	0.070852
B	0.207992	0.089159	-1.168557	0.666561
C	-0.615616	-0.543517	0.643749	0.297941
D	-0.137813	-0.601318	0.158598	-0.475202
E	0.311148	1.942433	0.075748	-1.851384

NumPy ndarray → DataFrame: Example

```
df = pd.DataFrame({'A': np.linspace(0,10,5),
                   'B': np.random.rand(5),
                   'C': np.random.choice(['Low', 'Medium', 'High'], 5),
                   'D': np.random.normal(100, 10, 5)})
df
```

	A	B	C	D
0	0.0	0.906282	High	94.088962
1	2.5	0.271948	Low	107.275864
2	5.0	0.868661	Medium	120.123371
3	7.5	0.122301	Medium	92.471376
4	10.0	0.123001	High	97.648295

DataFrame → NumPy ndarray

■ `df.values`

- Return a Numpy representation of the DataFrame

```
df = pd.DataFrame(np.random.randn(5,4),  
                  index=list('ABCDE'),  
                  columns=list('WXYZ'))  
df
```

```
df.values
```

	W	X	Y	Z
A	1.778562	0.683962	-0.526035	0.279652
B	0.488745	0.967074	1.558245	-0.723281
C	1.185515	-0.045885	-1.875274	-0.769951
D	-0.828883	-1.734295	-0.405570	-0.231408
E	-2.212240	1.062259	-0.445875	0.179177

```
array([[ 1.77856239,  0.6839622 , -0.52603465,  0.27965186],  
       [ 0.48874492,  0.9670737 ,  1.55824544, -0.72328108],  
       [ 1.18551512, -0.04588526, -1.87527446, -0.76995123],  
       [-0.82888265, -1.73429489, -0.40557033, -0.23140774],  
       [-2.21223955,  1.06225859, -0.44587544,  0.17917668]])
```

CSV File → DataFrame

- From a CSV(Comma-Separated Values) file

 [pokemon.csv](#) -

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Stage	Legendary
1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	FALSE
2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	2	FALSE

```
df = pd.read_csv('pokemon.csv')  
df
```

Getting a Glimpse of Your Data

■ `df.head()` and `df.tail()`

`df.head()`

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Stage	Legendary	
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	2	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	3	False
3	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
4	5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	2	False

`df.tail()`

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Stage	Legendary	
146	147	Dratini	Dragon	NaN	300	41	64	45	50	50	50	1	False
147	148	Dragonair	Dragon	NaN	420	61	84	65	70	70	70	2	False
148	149	Dragonite	Dragon	Flying	600	91	134	95	100	100	80	3	False
149	150	Mewtwo	Psychic	NaN	680	106	110	90	154	90	130	1	True
150	151	Mew	Psychic	NaN	600	100	100	100	100	100	100	1	False

■ `df.shape`

`df.shape`

(151, 13)

DataFrame Info.

- **df.info()**

```
df = pd.read_csv('pokemon.csv')
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 151 entries, 0 to 150
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   #           151 non-null    int64  
 1   Name         151 non-null    object  
 2   Type 1       151 non-null    object  
 3   Type 2       67 non-null    object  
 4   Total         151 non-null    int64  
 5   HP            151 non-null    int64  
 6   Attack        151 non-null    int64  
 7   Defense       151 non-null    int64  
 8   Sp. Atk       151 non-null    int64  
 9   Sp. Def       151 non-null    int64  
 10  Speed          151 non-null    int64  
 11  Stage          151 non-null    int64  
 12  Legendary      151 non-null    bool   
dtypes: bool(1), int64(9), object(3)
memory usage: 14.4+ KB
```

- **df.isnull().sum()**

Name	0
Type 1	0
Type 2	84
Total	0
HP	0
Attack	0
Defense	0
Sp. Atk	0
Sp. Def	0
Speed	0
Stage	0
Legendary	0
dtype: int64	

Descriptive Statistics

■ df.describe()

```
df.describe()
```

	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Stage
count	151.000000	151.000000	151.000000	151.000000	151.000000	151.000000	151.000000	151.000000
mean	407.07947	64.211921	72.549669	68.225166	67.139073	66.019868	68.933775	1.582781
std	99.74384	28.590117	26.596162	26.916704	28.534199	24.197926	26.746880	0.676832
min	195.000000	10.000000	5.000000	5.000000	15.000000	20.000000	15.000000	1.000000
25%	320.000000	45.000000	51.000000	50.000000	45.000000	49.000000	46.500000	1.000000
50%	405.000000	60.000000	70.000000	65.000000	65.000000	65.000000	70.000000	1.000000
75%	490.000000	80.000000	90.000000	84.000000	87.500000	80.000000	90.000000	2.000000
max	680.000000	250.000000	134.000000	180.000000	154.000000	125.000000	140.000000	3.000000

Indexing

- Row id = **key** = label = record id = **index**
- Used for
 - Accessing individual/multiple rows
 - Aligning multiple DataFrames and Series
- **`df.set_index(keys, ...)`**
 - Set the DataFrame index using existing column(s)
 - **Return a new DataFrame** with changed row labels (not in-place update)
 - **keys**: label or array/list of labels
 - e.g., `df = df.set_index(['Day', 'Time'])`
- **`df.reset_index()`** : go back to integer index

Position-based vs. Label-based Index

- Use `df.set_index()` to set the index using existing column(s)

```
data = {'District': ['Gwanak', 'Gangnam', 'Songpa', 'Jung'],
        'Male' : [257638, 260358, 326602, 66193 ],
        'Female' : [256917, 283727, 350071, 69128 ],
        'Household': [275248, 234021, 281417, 63594 ]}
df = pd.DataFrame(data)
df
```

```
dfnew = df.set_index(['District'])
dfnew
```

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
1	Gangnam	260358	283727	234021
2	Songpa	326602	350071	281417
3	Jung	66193	69128	63594

	Male	Female	Household
District			
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594

Renaming Rows/Columns

- **`df.rename([index], [columns], [inplace], ...)`**

- Rename any index, row or column
- A part of rows or columns can be altered
- *index*: dict. for changing row indexes
- *columns*: dict. for changing column indexes
- *inplace*: If True, *df* is updated in place.

Otherwise, return a new *df*
(default: False)

```
newrow = {'Gwanak': '관악구', 'Jung': '중구'}  
df.rename(index=newrow)
```

District	Male	Female	Household
관악구	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
중구	66193	69128	63594

```
newcol = {'Household': '세대수'}  
df.rename(columns=newcol)
```

District	Male	Female	세대수
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594

Reindexing

- **df.reindex([index], [columns], [fill_value], ...)**

- Reindex the dataframe with optional filling logic
- *index*: list for changing row indexes
- *columns*: list for changing column indexes
- *fill_value*: value to use for missing values (default: NaN)

df		Male Female Household		
		District		
	Gwanak	257638	256917	275248
	Gangnam	260358	283727	234021
	Songpa	326602	350071	281417
	Jung	66193	69128	63594

```
newidx = ['Gwanak', 'Gangnam', 'Seocho', 'Jung']
newcol = ['Household', 'Population']
df2 = df.reindex(index=newidx, columns=newcol)
df2
```

District	Household	Population
Gwanak	275248.0	NaN
Gangnam	234021.0	NaN
Seocho	NaN	NaN
Jung	63594.0	NaN

```
df3 = df2.reindex(index=df.index, columns=df.columns)
df3
```

District	Male	Female	Household
Gwanak	NaN	NaN	275248.0
Gangnam	NaN	NaN	234021.0
Songpa	NaN	NaN	NaN
Jung	NaN	NaN	63594.0

Transposing a DataFrame

- `df.transpose()` or `df.T`

```
d = { 'C1': ['A', 'B', 'C', 'D', 'E'],
      'C2': [10, 20, 30, 40, 50],
      'C3': [1.5, 2.7, 0.5, 3.2, 1.1] }
df = pd.DataFrame(d, index=['R'+str(i) for i in range(5)])
df
```

	C1	C2	C3
R0	A	10	1.5
R1	B	20	2.7
R2	C	30	0.5
R3	D	40	3.2
R4	E	50	1.1

```
df.T
```

	R0	R1	R2	R3	R4
C1	A	B	C	D	E
C2	10	20	30	40	50
C3	1.5	2.7	0.5	3.2	1.1

Filling / Dropping NaNs

■ `df.fillna(value, method, axis, ...)`

- `value`: value to use to fill holes
- `method`: 'bfill', 'ffill', or None (default: None)
- `axis`: axis along which to fill missing values

`df`

`df.fillna(0)`

	A	B	C
0	2.1	-1.0	0.8
1	NaN	0.5	NaN
2	NaN	NaN	NaN

	A	B	C
0	2.1	-1.0	0.8
1	0.0	0.5	0.0
2	0.0	0.0	0.0

■ `df.dropna(axis, how, ...)`

- `axis`: drop rows (0) or columns (1) which contain missing values (default: 0)
- `how`: 'any' or 'all'. If any (or all) values are NaN, drop that row or column. (default: 'any')

`df.dropna()`

`df.dropna(how='all')`

	A	B	C
0	2.1	-1.0	0.8

	A	B	C
0	2.1	-1.0	0.8
1	NaN	0.5	NaN

Manipulating Columns

Selecting Columns (I)

```
import numpy as np
import pandas as pd

data = {'District' :['Gwanak', 'Gangnam', 'Songpa', 'Jung'],
        'Male' :[257638, 260358, 326602, 66193],
        'Female' :[256917, 283727, 350071, 69128],
        'Household':[275248, 234021, 281417, 63594]}

df = pd.DataFrame(data)
df
```

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
1	Gangnam	260358	283727	234021
2	Songpa	326602	350071	281417
3	Jung	66193	69128	63594

df['Male']

```
0    257638
1    260358
2    326602
3    66193
Name: Male, dtype: int64
```

df.Female

```
0    256917
1    283727
2    350071
3    69128
Name: Female, dtype: int64
```

df[['Male', 'Female']]

	Male	Female
0	257638	256917
1	260358	283727
2	326602	350071
3	66193	69128

Selecting Columns (2)

```
import numpy as np
import pandas as pd

data = {'District' :['Gwanak', 'Gangnam', 'Songpa', 'Jung'],
        'Male' :[257638, 260358, 326602, 66193],
        'Female' :[256917, 283727, 350071, 69128],
        'Household':[275248, 234021, 281417, 63594]}

df = pd.DataFrame(data)
df
```

```
df.loc[:, 'Male']
```

```
0    257638
1    260358
2    326602
3    66193
Name: Male, dtype: int64
```

```
df.iloc[:, 2]
```

```
0    256917
1    283727
2    350071
3    69128
Name: Female, dtype: int64
```

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
1	Gangnam	260358	283727	234021
2	Songpa	326602	350071	281417
3	Jung	66193	69128	63594

```
df.loc[:, ['Male', 'Female']]
df.loc[:, 'Male':'Female']
df.iloc[:, [1, 2]]
df.iloc[:, 1:3]
```

	Male	Female
0	257638	256917
1	260358	283727
2	326602	350071
3	66193	69128

Adding a Column (I)

- With the same initial value

```
df['NewCol'] = 5      df.NewCol = 1 X  
df
```

	District	Male	Female	Household	NewCol
0	Gwanak	257638	256917	275248	5
1	Gangnam	260358	283727	234021	5
2	Songpa	326602	350071	281417	5
3	Jung	66193	69128	63594	5

- With new values

```
df['Name'] = ['관악구', '강남구', '송파구', '중구']  
df
```

	District	Male	Female	Household	Name
0	Gwanak	257638	256917	275248	관악구
1	Gangnam	260358	283727	234021	강남구
2	Songpa	326602	350071	281417	송파구
3	Jung	66193	69128	63594	중구

Also works for the existing column (e.g., `df.Female = 1`)

Adding a Column (2)

- With the sequential number
- With random numbers

```
df['No'] = np.arange(4.0)  
df
```

```
df['Random'] = np.random.random(size=len(df.index))  
df
```

	District	Male	Female	Household	No
0	Gwanak	257638	256917	275248	0.0
1	Gangnam	260358	283727	234021	1.0
2	Songpa	326602	350071	281417	2.0
3	Jung	66193	69128	63594	3.0

	District	Male	Female	Household	Random
0	Gwanak	257638	256917	275248	0.633339
1	Gangnam	260358	283727	234021	0.450869
2	Songpa	326602	350071	281417	0.015534
3	Jung	66193	69128	63594	0.256491

Adding a Column (3)

■ With expressions

```
df['Population'] = df.Male + df.Female  
df
```

	District	Male	Female	Household	Population
0	Gwanak	257638	256917	275248	514555
1	Gangnam	260358	283727	234021	544085
2	Songpa	326602	350071	281417	676673
3	Jung	66193	69128	63594	135321

```
df['Large'] = df.Household > 100000  
df
```

	District	Male	Female	Household	Large
0	Gwanak	257638	256917	275248	True
1	Gangnam	260358	283727	234021	True
2	Songpa	326602	350071	281417	True
3	Jung	66193	69128	63594	False

Adding a Column (4)

- With a Series
 - Unlike a list, the length can be smaller than the column size
 - Its labels will be realigned exactly to the DataFrame's index, insert missing values in any holes

```
df['Rate'] = pd.Series([3.7, 2.1], index=[3,0])  
df
```

	District	Male	Female	Household	Rate
0	Gwanak	257638	256917	275248	2.1
1	Gangnam	260358	283727	234021	NaN
2	Songpa	326602	350071	281417	NaN
3	Jung	66193	69128	63594	3.7

Deleting Columns

■ Using `del`

- Delete in-place
- Only one column at a time

```
del df['Household']
df
del df.Household X
```

	District	Male	Female
0	Gwanak	257638	256917
1	Gangnam	260358	283727
2	Songpa	326602	350071
3	Jung	66193	69128

■ Using `df.drop()`

- Return a new DataFrame (`inplace=False`)
- Also used to drop rows
or `axis='columns'`

```
dfnew = df.drop(['Male', 'Female'], axis=1)
dfnew
```

axis=1

↓

axis=0

	District	Household
0	Gwanak	275248
1	Gangnam	234021
2	Songpa	281417
3	Jung	63594

Manipulating Rows

Slicing Rows Using []

- $df[start : stop : step]$

```
df[0:3]
```

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417

```
df[0::2]
```

District	Male	Female	Household
Gwanak	257638	256917	275248
Songpa	326602	350071	281417

```
df[-1::-1]
```

District	Male	Female	Household
Jung	66193	69128	63594
Songpa	326602	350071	281417
Gangnam	260358	283727	234021
Gwanak	257638	256917	275248

```
df[-1:]
```

District	Male	Female	Household
Jung	66193	69128	63594

Selecting Rows by Integer Position

- `df.iloc[loc]`

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594

```
df.iloc[1]
```

```
Male          260358
Female        283727
Household    234021
Name: Gangnam, dtype: int64
```

```
df.iloc[[1,3]]
```

District	Male	Female	Household
Gangnam	260358	283727	234021
Jung	66193	69128	63594

```
df.iloc[0:2] exclusive!
```

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021

Selecting Rows by Label

- `df.loc[label]`

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594

```
df.loc['Gangnam']
```

```
Male          260358
Female        283727
Household    234021
Name: Gangnam, dtype: int64
```

```
df.loc[['Gangnam', 'Jung']]
```

District	Male	Female	Household
Gangnam	260358	283727	234021
Jung	66193	69128	63594

```
df.loc['Gwanak':'Songpa']
```

inclusive!

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417

Selecting Rows by Boolean Vector (I)

■ `df[bool_vec]`

```
df[[True, True, True, False]]
```

	Male	Female	Household
District			
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417

```
df.Household > 100000
```

```
District
Gwanak      True
Gangnam     True
Songpa      True
Jung        False
Name: Household, dtype: bool
```

```
df[df.Household > 100000]
```

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417

```
df[(df.Household > 100000) &
    (df.Male > 300000)]
```

and &
or /
not ~
 

District	Male	Female	Household
Songpa	326602	350071	281417

Selecting Rows by Boolean Vector (2)

- Use `isin()` to select all rows whose values contain the specified value(s)

```
df[(df.Household == 275248) | (df.Household == 281417)]
```

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
2	Songpa	326602	350071	281417

```
df[df.Household.isin([275248, 281417])]
```

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
2	Songpa	326602	350071	281417

Changing Rows

	Male	Female	Household
District			
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594

```
df.loc['Jung'] = 0  
df.iloc[1:3] = np.random.random(size=(2,3))  
df.iloc[0] = np.arange(3.0)  
df
```

	Male	Female	Household
District			
Gwanak	0.000000	1.000000	2.000000
Gangnam	0.646357	0.383127	0.648388
Songpa	0.980973	0.034829	0.270816
Jung	0.000000	0.000000	0.000000

Deleting Rows

■ Using df.drop()

- Return a new DataFrame
- Use `inplace=True` for in-place deletion

```
dfnew = df.drop('Gangnam')  
dfnew
```

District	Male	Female	Household
Gwanak	257638	256917	275248
Songpa	326602	350071	281417
Jung	66193	69128	63594

```
df.drop(['Gangnam', 'Songpa'], inplace=True)  
df
```

District	Male	Female	Household
Gwanak	257638	256917	275248
Jung	66193	69128	63594

```
df
```

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594

Adding Rows

■ Use `df.loc[]`

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594

```
df.loc['Jongro'] = [ 77391, 82451, 74861]  
df
```

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594
Jongro	77391	82451	74861

You can also use
`pd.concat()` or
`df.append()`

`df.iloc[4] = [...]` X

```
df.loc[4] = ['Jongro', 77391, 82451, 74861]  
df
```

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
1	Gangnam	260358	283727	234021
2	Songpa	326602	350071	281417
3	Jung	66193	69128	63594

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
1	Gangnam	260358	283727	234021
2	Songpa	326602	350071	281417
3	Jung	66193	69128	63594
4	Jongro	77391	82451	74861

```
df.loc[df.index.max()+1] = {  
    'District': 'Seocho', 'Household': 173483,  
    'Male': 205671, 'Female': 224324  
}  
df
```

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
1	Gangnam	260358	283727	234021
2	Songpa	326602	350071	281417
3	Jung	66193	69128	63594
4	Jongro	77391	82451	74861
5	Seocho	205671	224324	173483

Accessing Values

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
1	Gangnam	260358	283727	234021
2	Songpa	326602	350071	281417
3	Jung	66193	69128	63594

```
df.loc[0, 'Household']
```

275248

```
df.loc[1, ['Male', 'Female']]
```

Male 260358

Female 283727

Name: 1, dtype: object

```
df.loc[[2, 3], 'Household']
```

2 281417

3 63594

Name: Household, dtype: int64

```
df.loc[[2, 3], ['Male', 'Female']]
```

	Male	Female
2	326602	350071
3	66193	69128

Slicing Rows and Columns

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
1	Gangnam	260358	283727	234021
2	Songpa	326602	350071	281417
3	Jung	66193	69128	63594

```
df.iloc[0:3,1:3]
```

	Male	Female
0	257638	256917
1	260358	283727
2	326602	350071

Male Female Household

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
1	Gangnam	260358	283727	234021
2	Songpa	326602	350071	281417
3	Jung	66193	69128	63594

```
df.loc['Gwanak':'Songpa', 'Male':'Female']
```

Male Female

	District	Male	Female
0	Gwanak	257638	256917
1	Gangnam	260358	283727
2	Songpa	326602	350071

Changing Values

	District	Male	Female	Household
0	Gwanak	257638	256917	275248
1	Gangnam	260358	283727	234021
2	Songpa	326602	350071	281417
3	Jung	66193	69128	63594

```
df.loc[0:2, 'Household'] = -1  
df
```

	District	Male	Female	Household
0	Gwanak	257638	256917	-1
1	Gangnam	260358	283727	-1
2	Songpa	326602	350071	-1
3	Jung	66193	69128	63594

```
df.loc[df.Male > 200000, 'Male'] = 200000  
df
```

	District	Male	Female	Household
0	Gwanak	200000	256917	-1
1	Gangnam	200000	283727	-1
2	Songpa	200000	350071	-1
3	Jung	66193	69128	63594

```
df.loc[df.District=='Jung', ['Male', 'Female']] = 0  
df
```

	District	Male	Female	Household
0	Gwanak	200000	256917	-1
1	Gangnam	200000	283727	-1
2	Songpa	200000	350071	-1
3	Jung	0	0	63594

Iteration over Rows

■ `df.iterrows()`

- Iterate over rows of DataFrame as `(index, Series)` pairs

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594

```
for index, row in df.iterrows():
    print(row['Male'], row['Female'])
```

257638 256917

260358 283727

326602 350071

66193 69128

```
for index, row in df.iterrows():
    print(index)
    print(row)
```

Gwanak

Male 257638

Female 256917

Household 275248

Name: Gwanak, dtype: int64

Gangnam

Male 260358

Female 283727

Household 234021

Name: Gangnam, dtype: int64

Songpa

Male 326602

Female 350071

Household 281417

Name: Songpa, dtype: int64

Jung

Male 66193

Female 69128

Household 63594

Name: Jung, dtype: int64

Iteration over Columns

■ `df.items()`

- Iterate over DataFrame columns as `(index, Series)` pairs

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594

```
for index, col in df.items():
    print(col['Gwanak'], col['Gangnam'])
```

257638 260358
256917 283727
275248 234021

```
for index, col in df.items():
    print(index)
    print(col)
```

```
Male
District
Gwanak      257638
Gangnam     260358
Songpa       326602
Jung         66193
Name: Male, dtype: int64

Female
District
Gwanak      256917
Gangnam     283727
Songpa       350071
Jung         69128
Name: Female, dtype: int64

Household
District
Gwanak      275248
Gangnam     234021
Songpa       281417
Jung         63594
Name: Household, dtype: int64
```

Summary

■ Selecting rows or columns

Operation	Syntax	Result
Select columns	<code>df['Col'], df.Col, df[['Col1','Col2']]</code>	Series
Select rows by integer index	<code>df.iloc[1], df.iloc[0:3], df.iloc[[2,4]]</code>	Series or DataFrame
Select rows by label	<code>df.loc['R0'], df.loc['R0':'R3'], df.loc[['R2','R4']]</code>	Series or DataFrame
Select rows by Boolean vector	<code>df[df.Col1 > 10], df[df.Col1.isin([1,2])]</code>	DataFrame
Slice rows	<code>df[0:3], df[0:10:2], df[-5:]</code>	DataFrame
Slice rows and columns	<code>df.iloc[1:2,2:4], df.iloc[[0,3],[1,2]] df.loc['R1':'R3','C1':'C3'], df.loc[['R1','R3'],['C1','C3']]</code>	Series or DataFrame

■ Deleting rows or columns

- `df.drop(0), df.drop([0,2]), df.drop(['Col1','Col2'],axis=1)`

Pandas I/O

I/Os for Pandas DataFrame

- A collection of convenient I/O functions supporting various file formats

```
to_csv()      to_excel()     to_hdf()       to_sql()       to_json()      to_html()  
read_csv()    read_excel()   read_hdf()     read_sql()     read_json()    read_html()
```

- (cf.) HDF (Hierarchical Data Format): Standardized file format for scientific data
- From CSV file to Pandas DataFrame: *pd.read_csv(path)*
- From Pandas DataFrame to CSV file: *df.to_csv(path)*

pandas.read_csv()

- `pd.read_csv(filepath, sep=',', header='infer', names=None, index_col=None, encoding='utf-8', skiprows=0, thousands=None, ...)`
 - Read a comma-separated values (csv) file
 - `filepath`: any valid string path. The string could be a URL.
 - `sep` (or `delimiter`): delimiter to use
 - `header`: row number(s) to use as the column names
 - `names`: list of column names to use
 - `index_col`: column(s) to use as the row labels of the Data Frame
 - `encoding`: encoding to use
 - `skiprows`: line numbers to skip or number of lines to skip at the start of the file
 - `thousands`: thousands separator

Reading a CSV File

```
df = pd.read_csv('pokemon.csv', index_col=1)  
df.head()
```

Name	#	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Stage	Legendary
Bulbasaur	1	Grass	Poison	318	45	49	49	65	65	45	1	False
Ivysaur	2	Grass	Poison	405	60	62	63	80	80	60	2	False
Venusaur	3	Grass	Poison	525	80	82	83	100	100	80	3	False
Charmander	4	Fire	NaN	309	39	52	43	60	50	65	1	False
Charmeleon	5	Fire	NaN	405	58	64	58	80	65	80	2	False

DataFrame.to_csv()

- `df.to_csv(filepath, sep=',', columns=None, header=True, index=True, encoding=None, ...)`
 - Write DataFrame to a comma-separated values (csv) file
 - `filepath`: any valid string path.
 - `sep` (or `delimiter`): delimiter to use
 - `columns`: columns to write
 - `header`: write out the column names
 - `index`: write row names
 - `encoding`: encoding to use (default: 'utf-8')

```
>>> df.to_csv('mydf.csv', sep='\t')
>>> df.to_csv('dataset.csv', sep='\t', encoding='utf-8')
```

Jin-Soo Kim
[\(jinsoo.kim@snu.ac.kr\)](mailto:jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022



Python for Data Analytics

Pandas II

Outline

- Why Pandas?
- Pandas Series
- Pandas DataFrame
 - Creating DataFrame
 - Manipulating Columns
 - Manipulating Rows
 - Arithmetic operations
 - Group Aggregation
 - Hierarchical Indexing
 - Combining and Merging
 - Time Series Data

Arithmetic Operations

Arithmetic (I)

```
df1 = pd.DataFrame(np.random.randint(0,10,size=(4,2)),  
                   index=list('abcd'))
```

```
df2 = pd.DataFrame(np.random.randint(0,10,size=(4,2)),  
                   index=list('abde'))
```

df1 + df2

df1 - df2

df1 * df2

df1 / df2

	0	1
a	17.0	9.0
b	18.0	3.0
c	NaN	NaN
d	11.0	4.0
e	NaN	NaN

	0	1
a	-1.0	-7.0
b	0.0	3.0
c	NaN	NaN
d	7.0	4.0
e	NaN	NaN

	0	1
a	72.0	8.0
b	81.0	0.0
c	NaN	NaN
d	18.0	0.0
e	NaN	NaN

	0	1
a	0.8888889	0.125
b	1.000000	inf
c	NaN	NaN
d	4.500000	inf
e	NaN	NaN

	0	1		0	1
a	8	1	a	9	8
b	9	3	b	9	0
c	1	7	d	2	0
d	9	4	e	4	7
	df1			df2	

df1 / 10

df2 + 0.5

	0	1
a	0.8	0.1
b	0.9	0.3
c	0.1	0.7
d	0.9	0.4
e	4.5	7.5

	0	1
a	9.5	8.5
b	9.5	0.5
c	2.5	0.5
d	4.5	7.5

Arithmetic (2)

- Alignment is performed on both rows and columns
- The result will be all NaNs if no column or row labels are in common

```
df1 = pd.DataFrame(np.arange(9.0).reshape((3,3)),  
                   columns=list('bcd'), index=list('xyz'))  
df2 = pd.DataFrame(np.arange(12.0).reshape((4,3)),  
                   columns=list('bde'), index=list('wxyu'))  
df1 + df2
```

	b	c	d		b	d	e		b	c	d	e		
x	0.0	1.0	2.0		w	0.0	1.0	2.0	u	NaN	NaN	NaN	NaN	
y	3.0	4.0	5.0	+	x	3.0	4.0	5.0	=	w	NaN	NaN	NaN	NaN
z	6.0	7.0	8.0		y	6.0	7.0	8.0		x	3.0	NaN	6.0	NaN
									y	9.0	NaN	12.0	NaN	
									z	NaN	NaN	NaN	NaN	

```
df1 = pd.DataFrame(np.arange(4.0).reshape((2,2)),  
                   columns=list('ab'), index=list('mn'))  
df2 = pd.DataFrame(np.arange(4.0).reshape((2,2)),  
                   columns=list('cd'), index=list('op'))  
df1 + df2
```

	a	b		c	d		a	b	c	d			
m	0.0	1.0		o	0.0	1.0	=	m	NaN	NaN	NaN	NaN	
n	2.0	3.0	+	p	2.0	3.0		n	NaN	NaN	NaN	NaN	
									o	NaN	NaN	NaN	NaN
									p	NaN	NaN	NaN	NaN

Arithmetic Methods

Operation	Meaning
<code>df1.add(df2)</code>	<code>df1 + df2</code>
<code>df1.sub(df2)</code>	<code>df1 - df2</code>
<code>df1.div(df2)</code>	<code>df1 / df2</code>
<code>df1.floordiv(df2)</code>	<code>df1 // df2</code>
<code>df1.mul(df2)</code>	<code>df1 * df2</code>
<code>df1.pow(df2)</code>	<code>df1 ** df2</code>
<code>df1.radd(df2)</code>	<code>df2 + df1</code>
<code>df1.rsub(df2)</code>	<code>df2 - df1</code>
<code>df1.rdiv(df2)</code>	<code>df2 / df1</code>
<code>df1.rfloordiv(df2)</code>	<code>df2 // df1</code>
<code>df1.rmul(df2)</code>	<code>df2 * df1</code>
<code>df1.rpow(df2)</code>	<code>df2 ** df1</code>

- `fill_value` can be specified

- Existing missing values and any new elements is set to this value
- If data in both corresponding DataFrame locations is missing, the result will be NaN

df1.add(df2, fill_value=0)														
	b	c	d		b	d	e		b	c	d	e		
x	0.0	1.0	2.0		w	0.0	1.0	2.0		u	9.0	NaN	10.0	11.0
y	3.0	4.0	5.0	+	x	3.0	4.0	5.0	=	w	0.0	NaN	1.0	2.0
z	6.0	7.0	8.0		y	6.0	7.0	8.0		x	3.0	1.0	6.0	5.0
					u	9.0	10.0	11.0		y	9.0	4.0	12.0	8.0
										z	6.0	7.0	8.0	NaN

Applying NumPy Universal Function

- A universal function (ufunc) operates on ndarrays in an element-by-element fashion
 - `np.fabs()`, `np.exp()`, `np.sqrt()`, `np.log()`, ...

	a	b	c	np.sqrt(df)	df.apply(np.exp)
R0	0.0	1.0	2.0		
R1	3.0	4.0	5.0	R0 0.000000 1.000000 1.414214	R0 1.000000 2.718282 7.389056
R2	6.0	7.0	8.0	R1 1.732051 2.000000 2.236068	R1 20.085537 54.598150 148.413159
R3	9.0	10.0	11.0	R2 2.449490 2.645751 2.828427	R2 403.428793 1096.633158 2980.957987
				R3 3.000000 3.162278 3.316625	R3 8103.083928 22026.465795 59874.141715

Applying Functions

- **`df.apply(func, axis=0, ...)` or `series.apply(func, ...)`**
 - Apply a function along an axis of the DataFrame or on values of Series
 - `axis=0` or `axis='index'`: apply `func` to each column (default)
 - `axis=1` or `axis='columns'`: apply `func` to each row
- **`df.applymap(func, na_action=None, ...)`**
 - Apply a function to a Dataframe elementwise
 - `no_action`: if 'ignore', propagate NaN values, without passing them to `func`
- **`series.map(func, na_action=None, ...)`**
 - Map values of Series according to input correspondence

	DataFrame	Series
<code>apply()</code>	✓	✓
<code>applymap()</code>	✓	✗
<code>map()</code>	✗	✓

Applying Functions: Example

```
df['S'] = df.apply(np.sum, axis=1)  
df
```

```
df.loc['e'] = df.apply(np.average, axis=0)  
df
```

```
df['T'] = df.apply(lambda x: x.max() - x.min(), axis=1)  
df
```

df	X	Y	Z
a	3	1	6
b	0	1	8
c	3	6	2
d	3	0	2

	X	Y	Z	S
a	3	1	6	10
b	0	1	8	9
c	3	6	2	11
d	3	0	2	5

	X	Y	Z	S
a	3.00	1.0	6.0	10.00
b	0.00	1.0	8.0	9.00
c	3.00	6.0	2.0	11.00
d	3.00	0.0	2.0	5.00
e	2.25	2.0	4.5	8.75

	X	Y	Z	S	T
a	3.00	1.0	6.0	10.00	9.00
b	0.00	1.0	8.0	9.00	9.00
c	3.00	6.0	2.0	11.00	9.00
d	3.00	0.0	2.0	5.00	5.00
e	2.25	2.0	4.5	8.75	6.75

```
df['M'] = df['Z'].apply(np.square)  
df['N'] = np.random.choice(['Yes', 'No'], 4)  
df
```

```
df['N'] = df['N'].map({'Yes':1, 'No':0})  
df
```

	X	Y	Z	M	N
a	3	1	6	36	Yes
b	0	1	8	64	No
c	3	6	2	4	No
d	3	0	2	4	Yes

	X	Y	Z	M	N
a	3	1	6	36	1
b	0	1	8	64	0
c	3	6	2	4	0
d	3	0	2	4	1

```
df.applymap(np.sqrt)
```

	X	Y	Z	S	T
a	1.732051	1.000000	2.449490	3.162278	3.000000
b	0.000000	1.000000	2.828427	3.000000	3.000000
c	1.732051	2.449490	1.414214	3.316625	3.000000
d	1.732051	0.000000	1.414214	2.236068	2.236068
e	1.500000	1.414214	2.121320	2.958040	2.598076

Encoding Text to Numbers

- `series.factorize(...)` or `pd.factorize(values, ...)`
 - Encode the object as an enumerated type or categorical variable
 - Useful for obtaining a numerical representation of an array for distinct values
 - Return two arrays: an integer array and the corresponding unique values

	Name	Type
0	Tom	INFJ
1	Bob	ISTJ
2	Andy	INFJ
3	Mary	ENTP
4	Ann	ESFP

```
codes, uniques = df['Type'].factorize()  
codes  
  
array([0, 1, 0, 2, 3], dtype=int64)  
  
uniques  
  
Index(['INFJ', 'ISTJ', 'ENTP', 'ESFP'], dtype='object')  
  
df.Type = df.Type.map({'INFJ':0, 'ISTJ':1,  
                      'ENTP':2, 'ESFP':3})
```

factorize() is faster than map()

```
df['Type'] = codes  
df
```

	Name	Type
0	Tom	0
1	Bob	1
2	Andy	0
3	Mary	2
4	Ann	3

One Hot Encoding

- `pd.get_dummies(df, prefix=None, prefix_sep='_', columns=None, ...)`
 - Convert categorical variable into dummy/indicator variables
 - `prefix`: string to append DataFrame column names
 - `prefix_sep`: If appending prefix, delimiter to use
 - `columns`: Column names in the DataFrame to be encoded. If None, all the columns with object or category dtype will be converted

	value	color	pd.get_dummies(df)				
0	143	red		value	color_blue	color_green	color_red
1	23	blue	0	143	0	0	1
2	46	red	1	23	1	0	0
3	0	green	2	46	0	0	1
4	78	blue	3	0	0	1	0
			4	78	1	0	0

*OneHotEncoder in Scikit-learn
is more recommended
(sklearn.preprocessing.OneHotEncoder)*

Common Statistical Functions

Method	Description
<code>count()</code>	Number of non-null observations
<code>sum()</code>	Sum of values
<code>mean()</code>	Mean of values
<code>median()</code>	Arithmetic median of values
<code>min()</code>	Minimum
<code>max()</code>	Maximum
<code>std()</code>	Bessel-corrected sample standard deviation
<code>var()</code>	Unbiased variance
<code>skew()</code>	Sample skewness (3rd moment)
<code>kurt()</code>	Sample kurtosis (4th moment)
<code>quantile()</code>	Sample quantile (value at %)
<code>mad()</code>	Mean absolute deviation from mean value
<code>cov()</code>	Unbiased covariance
<code>corr()</code>	Correlation

District	Male	Female	Household
<code>Gwanak</code>	257638	256917	275248
<code>Gangnam</code>	260358	283727	234021
<code>Songpa</code>	326602	350071	281417
<code>Jung</code>	66193	69128	63594
<code>df.Household.mean()</code>			
213570.0			
<code>df.Male.max()</code>			
326602			
<code>df.loc['Songpa'].sum()</code>			
958090			
<code>df['Female'].std()</code>			
120431.30086035219			

idxmax() and idxmin()

- `df.idxmax([axis=0], [skipna=True], ...)`
 - Return index of first occurrence of maximum over requested *axis*
- `df.idxmin([axis=0], [skipna=True], ...)`
 - Return index of first occurrence of minimum over requested *axis*

	a	b	c
W	3	9	8
X	9	0	6
Y	4	5	7
Z	8	7	2

```
df.idxmax()
```

```
a      X  
b      W  
c      W  
dtype: object
```

```
df.idxmin(axis=1)
```

```
W      a  
X      b  
Y      a  
Z      c  
dtype: object
```

```
df['a'].idxmax()
```

```
'X'
```

```
df.loc['X'].idxmin()
```

```
'b'
```

Sorting DataFrame by Index

■ `df.sort_index(axis=0, level=None, ascending=True, ...)`

- Sort by labels along an axis
- `axis=0` or `axis='index'`: sort along the rows (default)
- `axis=1` or `axis='columns'`: sort along the columns
- `level`: sort on values in specified index level

```
df.sort_index(axis=0, ascending=False)
```

		Male	Female	Household
District	Name			
Songpa	송파구	326602	350071	281417
Jung	중구	66193	69128	63594
Gwanak	관악구	257638	256917	275248
Gangnam	강남구	260358	283727	234021

```
df.sort_index(axis=0, level=1)
```

		Male	Female	Household
District	Name			
Gangnam	강남구	260358	283727	234021
Gwanak	관악구	257638	256917	275248
Songpa	송파구	326602	350071	281417
Jung	중구	66193	69128	63594

Sorting DataFrame by Values

- `df.sort_values(by, axis=0, ascending=True, ...)`

- Sort by the values along either axis
- `by`: name or list of names to sort by
- `axis`: axis to be sorted

```
df.sort_values('Male')
```

District	Male	Female	Household
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417
Jung	66193	69128	63594

```
df.sort_values('Songpa', axis=1)
```

District	Male	Female	Household
Jung	66193	69128	63594
Gwanak	257638	256917	275248
Gangnam	260358	283727	234021
Songpa	326602	350071	281417

District	Household	Male	Female
Gwanak	275248	257638	256917
Gangnam	234021	260358	283727
Songpa	281417	326602	350071
Jung	63594	66193	69128

Ranking DataFrame

- `df.rank(axis=0, method='average', ascending=True, ...)`

- Compute numerical data ranks (1 through n) along axis
- `axis`: index to direct ranking
- `method`: how to rank the records that have the same value (tie-breaking rule)
 - 'average', 'min', 'max', 'first' or 'dense'

```
df['average'] =  
    df.score.rank(method='average',  
                  ascending=False)
```

	name	score	average	min	max	first	dense
0	kim	100	1.0	1.0	1.0	1.0	1.0
1	lee	80	6.0	5.0	7.0	5.0	4.0
2	park	95	2.5	2.0	3.0	2.0	2.0
3	choi	80	6.0	5.0	7.0	6.0	4.0
4	seo	80	6.0	5.0	7.0	7.0	4.0
5	hong	90	4.0	4.0	4.0	4.0	3.0
6	min	95	2.5	2.0	3.0	3.0	2.0

Group Aggregation

Example DataFrame Object

```
df = pd.DataFrame({'A':['foo','bar','foo','bar','foo','bar','foo','foo'],
                   'B':['one','one','two','three','two','two','one','three'],
                   'C':np.random.randn(8), \
                   'D':np.random.randn(8)})
```

df

Two categorical values: A, B
Two numerical values: C, D

	A	B	C	D
0	foo	one	-1.307901	1.174486
1	bar	one	-0.409657	0.872421
2	foo	two	-1.278086	0.028901
3	bar	three	0.315299	-2.273053
4	foo	two	-0.147411	-0.892378
5	bar	two	0.218316	-0.693276
6	foo	one	-1.024522	-0.358686
7	foo	three	1.996648	-0.273449

df[df.A == 'foo']

df[df.A == 'bar']

	A	B	C	D
0	foo	one	-1.307901	1.174486
2	foo	two	-1.278086	0.028901
4	foo	two	-0.147411	-0.892378
6	foo	one	-1.024522	-0.358686
7	foo	three	1.996648	-0.273449

	A	B	C	D
1	bar	one	-0.409657	0.872421
3	bar	three	0.315299	-2.273053
5	bar	two	0.218316	-0.693276

Groupby and Aggregation

- **df.groupby(by, axis=0, ...)**

- Used to group large amounts of data and compute operations on these groups
- *by*: label, function, a list of labels, ...
(Used to determine the groups)
- *axis*: 0 or 'index' for rows, 1 or 'columns' for columns
(default: 0)

- Aggregation stat functions after grouping

- mean(), sum(), median(), var(), etc.

```
g = df.groupby('A')  
g.mean()
```

	C	D
A		
bar	0.041319	-0.697969
foo	-0.352255	-0.064225
g.corr()		

	C	D
A		
bar	C	1.000000 -0.920023
	D	-0.920023 1.000000
foo	C	1.000000 -0.404475
	D	-0.404475 1.000000

Iterating over Groups

```
g.get_group('bar')
```

	A	B	C	D
1	bar	one	-0.409657	0.872421
3	bar	three	0.315299	-2.273053
5	bar	two	0.218316	-0.693276

```
g.get_group('foo')
```

	A	B	C	D
0	foo	one	-1.307901	1.174486
2	foo	two	-1.278086	0.028901
4	foo	two	-0.147411	-0.892378
6	foo	one	-1.024522	-0.358686
7	foo	three	1.996648	-0.273449

```
for key, items in g:  
    print(f'{key}:')  
    print(g.get_group(key))
```

bar:

	A	B	C	D
1	bar	one	-0.409657	0.872421
3	bar	three	0.315299	-2.273053
5	bar	two	0.218316	-0.693276

foo:

	A	B	C	D
0	foo	one	-1.307901	1.174486
2	foo	two	-1.278086	0.028901
4	foo	two	-0.147411	-0.892378
6	foo	one	-1.024522	-0.358686
7	foo	three	1.996648	-0.273449

Accessing a Group

■ Get a group's contents

```
g.get_group('bar')
```

	A	B	C	D
1	bar	one	-0.409657	0.872421
3	bar	three	0.315299	-2.273053
5	bar	two	0.218316	-0.693276

```
g.get_group('bar')['C'].values
```

```
array([-0.409657,  0.315299,  0.218316])
```

```
df.groupby('A')['C'].mean()
```

```
A  
bar    0.041319  
foo   -0.352254  
Name: C, dtype: float64
```

```
df.groupby('A')[['C']].mean()
```

```
C  
A  
---  
bar  0.041319  
foo -0.352254
```

```
df.groupby('A')[['C', 'D']].mean()
```

```
C      D  
A  
---  
bar  0.041319 -0.697969  
foo -0.352254 -0.064225
```

Describing a Group

```
g.describe()
```

C								
	count	mean	std	min	25%	50%	75%	max
A								
bar	3.0	0.041319	0.393556	-0.409657	-0.095670	0.218316	0.266807	0.315299
foo	5.0	-0.352255	1.394782	-1.307901	-1.278086	-1.024522	-0.147411	1.996648
D								
	count	mean	std	min	25%	50%	75%	max
	3.0	-0.697969	1.572742	-2.273053	-1.483164	-0.693276	0.089573	0.872421
	5.0	-0.064225	0.768016	-0.892378	-0.358686	-0.273449	0.028901	1.174486

Grouping by Multiple Columns

```
gm = df.groupby(['A', 'B'])  
gm.mean()
```

		C	D
A	B		
bar	one	-0.409657	0.872421
	three	0.315299	-2.273053
	two	0.218316	-0.693276
foo	one	-1.166211	0.407900
	three	1.996648	-0.273449
	two	-0.712749	-0.431739

```
gm.mean().unstack()
```

		C			D		
B	A	one	three	two	one	three	two
C							
bar	one	-0.409657	0.315299	0.218316	0.872421	-2.273053	-0.693276
foo	one	-1.166211	1.996648	-0.712749	0.407900	-0.273449	-0.431739

Grouping by List/Dict

```
years = [2019, 2020, 2020, 2018, 2018, 2020, 2020, 2018]
df[['C', 'D']].groupby(years).mean()
```

	C	D
2018	0.721512	-1.146293
2019	-1.307901	1.174486
2020	-0.623487	-0.037660

	A	B	C	D
0	foo	one	-1.307901	1.174486
1	bar	one	-0.409657	0.872421
2	foo	two	-1.278086	0.028901
3	bar	three	0.315299	-2.273053
4	foo	two	-0.147411	-0.892378
5	bar	two	0.218316	-0.693276
6	foo	one	-1.024522	-0.358686
7	foo	three	1.996648	-0.273449

기간	자치구	세대수				
		한국인남자	한국인여자	외국인남자	외국인여자	
2020.1/4	중구	63045	61839	64336	4930	5364
	관악구	270760	250743	248631	8239	9049
	송파구	279301	325859	348236	3199	3589
2020.2/4	중구	63354	61697	64395	4848	5090
	관악구	273715	250829	248911	7911	8667
	송파구	280135	324317	347195	3066	3489
2020.3/4	중구	63594	61526	64274	4667	4854
	관악구	275248	250084	248490	7554	8427
	송파구	281417	323646	346685	2956	3386

```
d = {'세대수': '세대수', '한국인남자': '남자',
      '한국인여자': '여자', '외국인남자': '남자',
      '외국인여자': '여자'}
df.groupby(d, axis=1).sum()
```

		남자	세대수	여자
2020.1/4	중구	66769	63045	69700
	관악구	258982	270760	257680
	송파구	329058	279301	351825
2020.2/4	중구	66545	63354	69485
	관악구	258740	273715	257578
	송파구	327383	280135	350684
2020.3/4	중구	66193	63594	69128
	관악구	257638	275248	256917
	송파구	326602	281417	350071

Grouping by Function

	A	B	C	D
0	foo	one	-1.307901	1.174486
1	bar	one	-0.409657	0.872421
2	foo	two	-1.278086	0.028901
3	bar	three	0.315299	-2.273053
4	foo	two	-0.147411	-0.892378
5	bar	two	0.218316	-0.693276
6	foo	one	-1.024522	-0.358686
7	foo	three	1.996648	-0.273449

```
df.groupby(lambda x: x//2).sum()
```

	C	D
0	-1.717558	2.046907
1	-0.962787	-2.244152
2	0.070905	-1.585654
3	0.972126	-0.632135

```
def onetwo(x):
    return 'onetwo' if x in ['one', 'two'] \
        else 'three'
```

```
df2 = df.set_index('B')
df2.groupby(onetwo).sum()
```

	C	D
onetwo	-3.949261	0.131468
three	2.311947	-2.546502

Getting Results as DataFrame

■ `df.groupby.transform(func, ...)`

- Call the function on each group and return a DataFrame having the same indexes as the original object filled with the transformed values
- func: Can be a string (e.g., 'mean', 'sum', etc.) for built-in aggregate functions

```
df['외국인평균'] = df.groupby('기간')[['외국인']].transform('mean')
df['외국인이많은구'] = df['외국인'] > df['외국인평균']
```

```
df.groupby('A').transform('sum')
```

	A	B		B
0	foo	1	0	5
1	bar	4	1	15
2	bar	3	2	15
3	bar	6	3	15
4	foo	4	4	5
5	bar	2	5	15

기간	자치구	세대수			한국인		외국인	
		2020.1/4	중구	63045	126175	10294	외국인평균	외국인이많은구
2020.2/4	관악구	270760	499374	17288				
	송파구	279301	674095	6788				
	중구	63354	126092	9938				
2020.3/4	관악구	273715	499740	16578				
	송파구	280135	671512	6555				
	중구	63594	125800	9521				
	관악구	275248	498574	15981				
	송파구	281417	670331	6342				

기간	자치구	세대수			한국인		외국인	
		2020.1/4	중구	63045	126175	10294	11456.666667	False
2020.2/4	관악구	270760	499374	17288	11456.666667	True		
	송파구	279301	674095	6788	11456.666667	False		
	중구	63354	126092	9938	11023.666667	False		
2020.3/4	관악구	273715	499740	16578	11023.666667	True		
	송파구	280135	671512	6555	11023.666667	False		
	중구	63594	125800	9521	10614.666667	False		
	관악구	275248	498574	15981	10614.666667	True		
	송파구	281417	670331	6342	10614.666667	False		

Hierarchical Indexing

Reading a Sample Dataset (I)

- `seoul2020.txt`: 서울시 주민등록인구 (구별) 통계 (2020년 1-3분기)
 - <https://data.seoul.go.kr/dataList/419/S/2/datasetView.do>

기간	자치구	세대	인구	인구	인구	인구	인구	인구	인구	인구	인구	인구	세대당인구	65세이상고령자									
기간	자치구	세대	합계	합계	한국인	한국인	한국인	한국인	등록외국인	등록외국인	등록외국인	등록외국인	세대당인구	65세이상고령자									
기간	자치구	세대	계	남자	여자	계	남자	여자	계	남자	여자	계	세대당인구	65세이상고령자									
2020.1/4	합계	4,354,006	10,013,781	4,874,995	5,138,786	9,733,655	4,742,217	4,991,438	280,126	132,778	147,348	2.24	1,518,239										
2020.1/4	종로구	74,151	161,984	78,271	83,713	151,217	73,704	77,513	10,767	4,567	6,200	2.04	28,073										
2020.1/4	중구	63,045	136,469	66,769	69,700	126,175	61,839	64,336	10,294	4,930	5,364	2	23,794										
2020.1/4	용산구	110,895	246,165	119,961	126,204	229,579	110,667	118,912	16,586	9,294	7,292	2.07	39,439										
2020.1/4	성동구	135,643	307,193	149,891	157,302	299,042	146,300	152,742	8,151	3,591	4,560	2.2	44,728										
2020.1/4	광진구	165,287	365,990	176,226	189,764	350,417	169,568	180,849	15,573	6,658	8,915	2.12	48,989										
2020.1/4	동대문구	165,279	362,793	178,202	184,591	346,156	171,896	174,260	16,637	6,306	10,331	2.09	60,367										
2020.1/4	중랑구	182,220	400,678	198,122	202,556	395,619	196,076	199,543	5,059	2,046	3,013	2.17	66,764										
2020.1/4	성북구	193,801	454,532	218,561	235,971	442,494	213,926	228,568	12,038	4,635	7,403	2.28	72,172										
2020.1/4	강북구	144,805	316,750	154,141	162,609	312,985	152,747	160,238	3,765	1,394	2,371	2.16	61,660										
2020.1/4	도봉구	138,595	333,495	162,774	170,721	331,238	161,879	169,359	2,257	895	1,362	2.39	60,023										
2020.1/4	노원구	217,148	535,495	258,696	276,799	531,037	256,726	274,311	4,458	1,970	2,488	2.45	82,682										
2020.1/4	은평구	208,209	482,509	231,953	250,556	478,019	230,147	247,872	4,490	1,806	2,684	2.3	82,245										
2020.1/4	서대문구	142,109	325,875	154,502	171,373	312,642	150,051	162,591	13,233	4,451	8,782	2.2	53,038										
2020.1/4	마포구	176,133	386,086	181,204	204,882	374,570	176,943	197,627	11,516	4,261	7,255	2.13	53,283										
2020.1/4	양천구	177,436	460,532	226,109	234,423	456,339	224,241	232,098	4,193	1,868	2,325	2.57	62,761										
2020.1/4	강서구	263,645	595,703	288,134	307,569	589,302	285,085	304,217	6,401	3,049	3,352	2.24	85,992										
2020.1/4	구로구	177,275	438,308	218,970	219,338	405,837	200,558	205,279	32,471	18,412	14,059	2.29	67,432										
2020.1/4	금천구	111,542	251,370	128,643	122,727	232,583	118,044	114,539	18,787	10,599	8,188	2.09	38,508										
2020.1/4	영등포구	177,743	404,766	202,617	202,149	371,903	184,316	187,587	32,863	18,301	14,562	2.09	59,373										
2020.1/4	동작구	181,761	407,802	196,943	210,859	395,014	191,272	203,742	12,788	5,671	7,117	2.17	63,378										
2020.1/4	관악구	270,760	516,662	258,982	257,680	499,374	250,743	248,631	17,288	8,239	9,049	1.84	76,664										
2020.1/4	서초구	173,580	434,801	207,877	226,924	430,568	205,787	224,781	4,233	2,090	2,143	2.48	58,332										
2020.1/4	강남구	233,624	549,898	263,163	286,735	544,804	260,654	284,150	5,094	2,509	2,585	2.33	72,602										
2020.1/4	송파구	279,301	680,883	329,058	351,825	674,095	325,859	348,236	6,788	3,199	3,589	2.41	89,539										
2020.1/4	강동구	190,019	457,042	225,226	231,816	452,646	223,189	229,457	4,396	2,037	2,359	2.38	66,401										
2020.2/4	합계	4,384,076	9,985,652	4,859,501	5,126,151	9,720,846	4,732,275	4,988,571	264,806	127,226	137,580	2.22	1,534,957										
2020.2/4	종로구	74,497	160,520	77,745	82,775	150,383	73,288	77,095	10,137	4,457	5,680	2.02	28,203										
2020.2/4	중구	63,354	136,030	66,545	69,485	126,092	61,697	64,395	9,938	4,848	5,090	1.99	24,035										
2020.2/4	용산구	111,586	245,362	119,494	125,868	229,431	110,527	118,904	15,931	8,967	6,964	2.06	39,650										

Reading a Sample Dataset (2)

■ What's wrong?

```
import numpy as np
import pandas as pd
df = pd.read_csv('seoul2020.txt', sep='\t', thousands=',')
df.head()
```

	기간	자치구	세대	인구	인구.1	인구.2	인구.3	인구.4	인구.5	인구.6	인구.7	인구.8	인구.9	인구.10	인구.11	인구.12	인구.13	인구.14
0	기간	자치구	세대	합계	합계	합계	한국인	한국인	한국인	등록외국인								
1	기간	자치구	세대	계	남자	여자	계	남자	여자	계	남자	여자	계	남자	여자	계	남자	여자
2	2020.1/4	합계	4,354,006	10,013,781	4,874,995	5,138,786	9,733,655	4,742,217	4,991,438	280,	280,	280,	280,	280,	280,	280,	280,	280,
3	2020.1/4	종로구	74,151	161,984	78,271	83,713	151,217	73,704	77,513	10,	10,	10,	10,	10,	10,	10,	10,	10,
4	2020.1/4	중구	63,045	136,469	66,769	69,700	126,175	61,839	64,336	10,	10,	10,	10,	10,	10,	10,	10,	10,

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80 entries, 0 to 79
Data columns (total 14 columns):
 #   Column      Non-Null Count Dtype  
 --- 
 0   기간       80 non-null    object 
 1   자치구     80 non-null    object 
 2   세대       80 non-null    object 
 3   인구       80 non-null    object 
 4   인구.1     80 non-null    object 
 5   인구.2     80 non-null    object 
 6   인구.3     80 non-null    object 
 7   인구.4     80 non-null    object 
 8   인구.5     80 non-null    object 
 9   인구.6     80 non-null    object 
 10  인구.7     80 non-null    object 
 11  인구.8     80 non-null    object 
 12  세대당인구 80 non-null    object 
 13  65세이상고령자 80 non-null    object 
dtypes: object(14)
memory usage: 8.9+ KB
```

Reading a Sample Dataset (3)

■ Ignoring first two rows

```
df = pd.read_csv('seoul2020.txt', sep='\t', thousands=',', skiprows=2)
df.head()
```

	기간	자치구	세대	계	남자	여자	계.1	남자.1	여자.1	
0	2020.1/4	합계	4354006	10013781	4874995	5138786	9733655	4742217	4991438	28
1	2020.1/4	종로구	74151	161984	78271	83713	151217	73704	77513	1
2	2020.1/4	중구	63045	136469	66769	69700	126175	61839	64336	1
3	2020.1/4	용산구	110895	246165	119961	126204	229579	110667	118912	1
4	2020.1/4	성동구	135643	307193	149891	157302	299042	146300	152742	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9 entries, 4 to 78
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   기간        9 non-null      object 
 1   자치구      9 non-null      object 
 2   세대        9 non-null      int64  
 3   인구        9 non-null      int64  
 4   인구.1      9 non-null      int64  
 5   인구.2      9 non-null      int64  
 6   인구.3      9 non-null      int64  
 7   인구.4      9 non-null      int64  
 8   인구.5      9 non-null      int64  
 9   인구.6      9 non-null      int64  
 10  인구.7      9 non-null      int64  
 11  인구.8      9 non-null      int64  
 12  세대당인구  9 non-null      object 
 13  65세이상고령자 9 non-null      int64  
dtypes: int64(11), object(3)
memory usage: 1.1+ KB
```

<Another way of changing types>

```
df = pd.read_csv('seoul2020.txt', sep='\t')
df = df[df.자치구.isin(['관악구', '송파구', '중구'])]
for col in df.columns[2:]:
    if (col != '세대당인구'):
        df[col] = df[col].apply(lambda x: x.replace(',', '')).astype(np.int64)
```

Using Hierarchical Indexes

```
df = pd.read_csv('seoul2020.txt', thousands=',', sep='\t', skiprows=2)
df = df[df.자치구.isin(['관악구','송파구','중구'])]
df = df.set_index(['기간','자치구'])
df = df[['세대','남자.1','여자.1','남자.2','여자.2']]
df.columns=[['세대수','한국인','한국인','외국인','외국인'],
            ['세대','남자','여자','남자','여자']]
df.columns.names = ['총계', '수']
df
```

기간	총계	세대수	한국인		외국인	
			세대	남자	여자	남자
2020.1/4	중구	63045	61839	64336	4930	5364
	관악구	270760	250743	248631	8239	9049
	송파구	279301	325859	348236	3199	3589
2020.2/4	중구	63354	61697	64395	4848	5090
	관악구	273715	250829	248911	7911	8667
	송파구	280135	324317	347195	3066	3489
2020.3/4	중구	63594	61526	64274	4667	4854
	관악구	275248	250084	248490	7554	8427
	송파구	281417	323646	346685	2956	3386

Using Primary Row Index

```
df['2020.2/4':'2020.3/4']
```

기간	자치구	총계		세대수		한국인		외국인	
		수	세대	남자	여자	남자	여자	남자	여자
2020.2/4	중구	63354	61697	64395	4848	5090			
	관악구	273715	250829	248911	7911	8667			
	송파구	280135	324317	347195	3066	3489			
2020.3/4	중구	63594	61526	64274	4667	4854			
	관악구	275248	250084	248490	7554	8427			
	송파구	281417	323646	346685	2956	3386			

```
df.loc[['2020.1/4','2020.3/4']]
```

기간	자치구	총계		세대수		한국인		외국인	
		수	세대	남자	여자	남자	여자	남자	여자
2020.1/4	중구	63045	61839	64336	4930	5364			
	관악구	270760	250743	248631	8239	9049			
	송파구	279301	325859	348236	3199	3589			
2020.3/4	중구	63594	61526	64274	4667	4854			
	관악구	275248	250084	248490	7554	8427			
	송파구	281417	323646	346685	2956	3386			

Using Secondary Row Index

```
df.loc[[('2020.1/4', '중구'), ('2020.3/4', '관악구')]]
```

기간	자치구	세대수		한국인		외국인	
		세대	남자	여자	남자	여자	남자
2020.1/4	중구	63045	66769	69700	4930	5364	
2020.3/4	관악구	275248	257638	256917	7554	8427	

```
df.loc[(slice(None), '중구'),:]
```

기간	자치구	총계		세대수		한국인		외국인	
		수	세대	남자	여자	남자	여자	남자	여자
2020.1/4	중구	63045	61839	64336	4930	5364			
2020.2/4	중구	63354	61697	64395	4848	5090			
2020.3/4	중구	63594	61526	64274	4667	4854			

Using Column Indexes

```
df.loc[(slice(None), '관악구'),  
       (slice(None), '여자')]
```

	총계	한국인	외국인
	수	여자	여자
기간	자치구		
2020.1/4	관악구	248631	9049
2020.2/4	관악구	248911	8667
2020.3/4	관악구	248490	8427

```
df['한국인']
```

	수	남자	여자	
	기간	자치구		
	2020.1/4	중구	61839	64336
		관악구	250743	248631
		송파구	325859	348236
	2020.2/4	중구	61697	64395
		관악구	250829	248911
		송파구	324317	347195
	2020.3/4	중구	61526	64274
		관악구	250084	248490
		송파구	323646	346685

```
df.loc[:, (slice(None), '여자')]
```

	총계	한국인	외국인	
	수	여자	여자	
기간	자치구			
	2020.1/4	중구	64336	5364
		관악구	248631	9049
		송파구	348236	3589
	2020.2/4	중구	64395	5090
		관악구	248911	8667
		송파구	347195	3489
	2020.3/4	중구	64274	4854
		관악구	248490	8427
		송파구	346685	3386

Unstacking / Stacking

```
df2 = df['한국인']
df2
```

기간	자치구	남자		여자	
		남자	여자	남자	여자
2020.1/4	중구	66769	69700		
	관악구	258982	257680		
	송파구	329058	351825		
2020.2/4	중구	66545	69485		
	관악구	258740	257578		
	송파구	327383	350684		
2020.3/4	중구	66193	69128		
	관악구	257638	256917		
	송파구	326602	350071		

```
df2.unstack()
```

기간	자치구	남자			여자		
		자치구	관악구	송파구	증구	관악구	송파구
기간							
2020.1/4	중구	66769	258982	329058	66769	257680	351825
	관악구	69700	257680				
	송파구						
2020.2/4	중구	66545	258740	327383	66545	257578	350684
	관악구	69485	257578				
	송파구						
2020.3/4	중구	66193	257638	326602	66193	256917	350071
	관악구	69128	256917				
	송파구						

```
df2.unstack().stack()
```

기간	자치구	남자		여자	
		남자	여자	남자	여자
2020.1/4	관악구	258982	257680		
	송파구	329058	351825		
	중구	66769	69700		
2020.2/4	관악구	258740	257578		
	송파구	327383	350684		
	중구	66545	69485		
2020.3/4	관악구	257638	256917		
	송파구	326602	350071		
	중구	66193	69128		

Pivoting

■ `df.pivot(index=None, columns=None, values=None)`

- Return reshaped DataFrame organized by given index/column values
- `index`: column(s) to use to make new frame's index
- `columns`: column(s) to use to make new frame's columns
- `values`: column(s) to use for populating new frame's values. If not specified, all remaining columns will be used and the result will have hierarchically indexed columns

	기간	자치구	세대수	한국인	외국인
2	2020.1/4	중구	63045	126175	10294
21	2020.1/4	관악구	270760	499374	17288
24	2020.1/4	송파구	279301	674095	6788
28	2020.2/4	중구	63354	126092	9938
47	2020.2/4	관악구	273715	499740	16578
50	2020.2/4	송파구	280135	671512	6555
54	2020.3/4	중구	63594	125800	9521
73	2020.3/4	관악구	275248	498574	15981
76	2020.3/4	송파구	281417	670331	6342

Pivoting Example

```
df.pivot('기간', '자치구', '세대수')
```

자치구 관악구 송파구 중구

기간

2020.1/4	270760	279301	63045
2020.2/4	273715	280135	63354
2020.3/4	275248	281417	63594

```
df.pivot('자치구', '기간', '세대수')
```

기간 2020.1/4 2020.2/4 2020.3/4

자치구

관악구	270760	273715	275248
-----	--------	--------	--------

송파구	279301	280135	281417
-----	--------	--------	--------

중구	63045	63354	63594
----	-------	-------	-------

```
df.pivot('기간', '자치구', ['한국인', '외국인'])
```

한국인 외국인

자치구	관악구	송파구	중구	관악구	송파구	중구
-----	-----	-----	----	-----	-----	----

기간

2020.1/4	499374	674095	126175	17288	6788	10294
----------	--------	--------	--------	-------	------	-------

2020.2/4	499740	671512	126092	16578	6555	9938
----------	--------	--------	--------	-------	------	------

2020.3/4	498574	670331	125800	15981	6342	9521
----------	--------	--------	--------	-------	------	------

Swapping and Sorting

```
df = df.swaplevel('기간', '자치구')  
df
```

자치구	기간	세대수		한국인		외국인	
		세대	남자	여자	남자	여자	남자
중구	2020.1/4	63045	66769	69700	4930	5364	
관악구	2020.1/4	270760	258982	257680	8239	9049	
송파구	2020.1/4	279301	329058	351825	3199	3589	
중구	2020.2/4	63354	66545	69485	4848	5090	
관악구	2020.2/4	273715	258740	257578	7911	8667	
송파구	2020.2/4	280135	327383	350684	3066	3489	
중구	2020.3/4	63594	66193	69128	4667	4854	
관악구	2020.3/4	275248	257638	256917	7554	8427	
송파구	2020.3/4	281417	326602	350071	2956	3386	

```
df.sort_index(level=0)
```

자치구	기간	세대수		한국인		외국인	
		세대	남자	여자	남자	여자	남자
관악구	2020.1/4	270760	258982	257680	8239	9049	
	2020.2/4	273715	258740	257578	7911	8667	
	2020.3/4	275248	257638	256917	7554	8427	
송파구	2020.1/4	279301	329058	351825	3199	3589	
	2020.2/4	280135	327383	350684	3066	3489	
	2020.3/4	281417	326602	350071	2956	3386	
중구	2020.1/4	63045	66769	69700	4930	5364	
	2020.2/4	63354	66545	69485	4848	5090	
	2020.3/4	63594	66193	69128	4667	4854	

Applying Functions

```
df.sum(level='자치구')
```

수	세대수	한국인		외국인	
총계	세대	남자	여자	남자	여자
자치구					
중구	189993	199507	208313	14445	15308
관악구	819723	775360	772175	23704	26143
송파구	840853	983043	1052580	9221	10464

자치구	기간	수	세대수	한국인		외국인	
		총계	세대	남자	여자	남자	여자
중구	2020.1/4	63045	66769	69700	4930	5364	5364
관악구	2020.1/4	270760	258982	257680	8239	9049	9049
송파구	2020.1/4	279301	329058	351825	3199	3589	3589
중구	2020.2/4	63354	66545	69485	4848	5090	5090
관악구	2020.2/4	273715	258740	257578	7911	8667	8667
송파구	2020.2/4	280135	327383	350684	3066	3489	3489
중구	2020.3/4	63594	66193	69128	4667	4854	4854
관악구	2020.3/4	275248	257638	256917	7554	8427	8427
송파구	2020.3/4	281417	326602	350071	2956	3386	3386

```
df.sum(level='수', axis=1)
```

자치구	기간	수	세대수	한국인	외국인
중구	2020.1/4	63045	136469	10294	
관악구	2020.1/4	270760	516662	17288	
송파구	2020.1/4	279301	680883	6788	
중구	2020.2/4	63354	136030	9938	
관악구	2020.2/4	273715	516318	16578	
송파구	2020.2/4	280135	678067	6555	
중구	2020.3/4	63594	135321	9521	
관악구	2020.3/4	275248	514555	15981	
송파구	2020.3/4	281417	676673	6342	

Combining and Merging

Appending DataFrames (I)

- `df.append(other, ignore_index=False, ...)`

- Append rows of `other` to the end of caller, returning a new object
- `left`: DataFrame or Series, or list of these
- `ignore_index`: If True, the resulting axis will be labeled 0, 1, ..., n-1

df1			
	id	name	country
0	1	Alice	Korea
1	2	Bob	US
2	9	Charlie	UK
3	5	Emily	France

df2			
	id	name	age
0	4	Judy	15
1	7	David	19
2	8	Bill	11

df1.append(df2)				
	id	name	country	age
0	1	Alice	Korea	NaN
1	2	Bob	US	NaN
2	9	Charlie	UK	NaN
3	5	Emily	France	NaN
0	4	Judy	NaN	15.0
1	7	David	NaN	19.0
2	8	Bill	NaN	11.0

```
df1.append([df2, df2],  
           ignore_index=True)
```

	id	name	country	age
0	1	Alice	Korea	NaN
1	2	Bob	US	NaN
2	9	Charlie	UK	NaN
3	5	Emily	France	NaN
4	4	Judy	NaN	15.0
5	7	David	NaN	19.0
6	8	Bill	NaN	11.0
7	4	Judy	NaN	15.0
8	7	David	NaN	19.0
9	8	Bill	NaN	11.0

Appending DataFrames (2)

- Appending rows with a Python a dictionary

```
newdata = [{'id':8, 'name':'Jack', 'country':'Japan'},  
           {'id':6, 'name':'Kate'}]  
df.append(newdata, ignore_index=True)
```

	id	name	country
0	1	Alice	Korea
1	2	Bob	US
2	9	Charlie	UK
3	5	Emily	France
4	8	Jack	Japan
5	6	Kate	NaN

Merging (Joining)

- `pd.merge(left, right, [how], [on], [left_on], [right_on], [left_index], [right_index], ...)`
 - Merge DataFrame objects with database-style join
 - `left`: DataFrame
 - `right`: Object to merge with
 - `how`: join type -- 'left', 'right', 'outer', or 'inner' (default: 'inner')
 - `on`: column to join on (label or list) -- must be found on both DataFrames
 - `left_on` (or `right_on`): column to join on in the left (or right) DataFrame
 - `left_index` (or `right_index`): if True, use the index from the left (or right) DataFrame

```
result = pd.merge(dfx, dfy, on='key')
```

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	B3	K3

	C	D	key
0	C0	D0	K0
1	C1	D1	K1
2	C2	D2	K2
3	C3	D3	K3

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	C2	D2
3	A3	B3	K3	C3	D3

Columns for Merging

- Merging two DataFrames by their own index

```
pd.merge(dfx, dfy,  
        left_index=True, right_index=True)
```

A B		C D		A B C D			
W	A0 B0	W	C0 D0	W	A0 B0	C0	D0
X	A1 B1	X	C1 D1	X	A1 B1	C1	D1
Y	A2 B2	Y	C2 D2	Y	A2 B2	C2	D2
Z	A3 B3	Z	C3 D3	Z	A3 B3	C3	D3

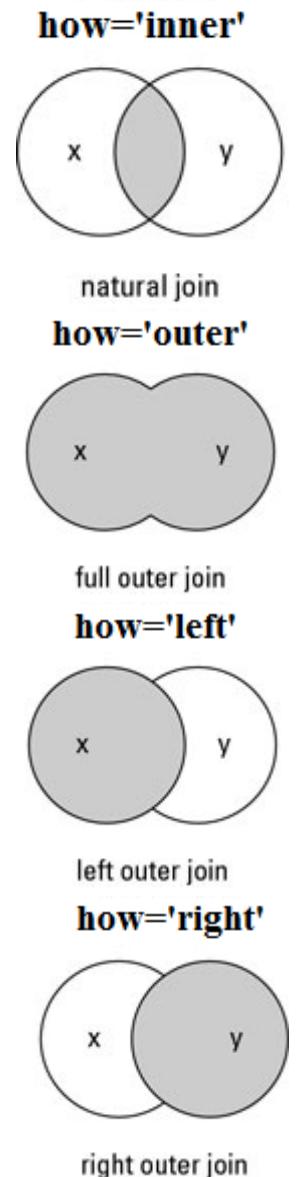
- If no information is given, use overlapping column names as the keys

```
pd.merge(dfx, dfy)
```

A B X			X Y Z			A B X Y Z		
0	A0 B0 X0	0	X0 Y0 Z0	0	A0 B0 X0 Y0 Z0			
1	A1 B1 X1	1	X1 Y1 Z1	1	A1 B1 X1 Y1 Z1			
2	A2 B2 X2	2	X2 Y2 Z2	2	A2 B2 X2 Y2 Z2			
3	A3 B3 X3	3	X3 Y3 Z3	3	A3 B3 X3 Y3 Z3			

Merging (Joining) Types

- Inner join ('inner') -- default
 - Return only the rows in which the left table have matching keys in the right table
- Outer join ('outer')
 - Returns all rows from both tables, join records from the left which have matching keys in the right table.
- Left outer join ('left')
 - Return all rows from the left table, and any rows with matching keys from the right table.
- Right outer join ('right')
 - Return all rows from the right table, and any rows with matching keys from the left table.



Merging (Joining) Example

`pd.merge(df1, df2)`

df1	id	name
0	1	Alice
1	2	Bob
2	3	Charlie
3	4	David
4	5	Emily

inner

	id	name	country
0	2	Bob	Korea
1	4	David	US
2	5	Emily	UK

left

	id	name	country
0	1	Alice	NaN
1	2	Bob	Korea
2	3	Charlie	NaN
3	4	David	US
4	5	Emily	UK

df2	id	country
0	2	Korea
1	4	US
2	5	UK
3	6	Italy

outer

	id	name	country
0	1	Alice	NaN
1	2	Bob	Korea
2	3	Charlie	NaN
3	4	David	US
4	5	Emily	UK
5	6	NaN	Italy

right

	id	name	country
0	2	Bob	Korea
1	4	David	US
2	5	Emily	UK
3	6	NaN	Italy

Many-to-One Join

dfx

	id	name
0	1	Alice
1	2	Bob
2	3	Charlie
3	4	David
4	5	Emily

dfy

	id	country
0	2	Korea
1	2	US
2	4	US
3	5	UK
4	5	France
5	6	Italy

pd.merge(dfx, dfy)

	id	name	country
0	2	Bob	Korea
1	2	Bob	US
2	4	David	US
3	5	Emily	UK
4	5	Emily	France

Concatenating DataFrames (I)

- `pd.concat(objs, axis=0, join='outer', keys=None, ...)`
 - Append rows of *other* to the end of caller, returning a new object
 - *objs*: a sequence of DataFrame or Series
 - *axis*: the axis to concatenate along
 - *join*: how to handle indexes on other axis
 - *keys*: construct hierarchical index using the keys as the outermost level

df1			
	id	name	country
0	1	Alice	Korea
1	2	Bob	US
2	9	Charlie	UK
3	5	Emily	France

df2			
	id	name	age
0	4	Judy	15
1	7	David	19
2	8	Bill	11

```
pd.concat([df1, df2], axis=1)
```

	id	name	country		id	name	age
0	1	Alice	Korea	0.0	4.0	Judy	15.0
1	2	Bob	US	1.0	7.0	David	19.0
2	9	Charlie	UK	2.0	8.0	Bill	11.0
3	5	Emily	France	3.0	NaN	NaN	NaN

```
pd.concat([df1, df2])
```

	id	name	country	age
0	1	Alice	Korea	NaN
1	2	Bob	US	NaN
2	9	Charlie	UK	NaN
3	5	Emily	France	NaN
0	4	Judy	NaN	15.0
1	7	David	NaN	19.0
2	8	Bill	NaN	11.0

Concatenating DataFrames (2)

```
pd.concat([df1, df2],  
          ignore_index=True)
```

	id	name	country	age
0	1	Alice	Korea	NaN
1	2	Bob	US	NaN
2	9	Charlie	UK	NaN
3	5	Emily	France	NaN
4	4	Judy	NaN	15.0
5	7	David	NaN	19.0
6	8	Bill	NaN	11.0

```
pd.concat([df1, df2], join='inner')
```

	id	name
0	1	Alice
1	2	Bob
2	9	Charlie
3	5	Emily
0	4	Judy
1	7	David
2	8	Bill

```
pd.concat([df1, df2],  
          keys=['df1', 'df2'])
```

	id	name	country	age
df1	0	Alice	Korea	NaN
	1	Bob	US	NaN
	2	Charlie	UK	NaN
	3	Emily	France	NaN
df2	0	Judy	NaN	15.0
	1	David	NaN	19.0
	2	Bill	NaN	11.0



Jin-Soo Kim
[\(jinsoo.kim@snu.ac.kr\)](mailto:jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022

Python for Data Analytics

Matplotlib

Outline

- Data Visualization
- Matplotlib
 - Pyplot Basics
 - Scatter Plot
 - Bar Chart
 - Pie Chart
 - Histogram
 - Box Plot
 - Subplots
- Seaborn
- Plotting in Pandas

Data Visualization

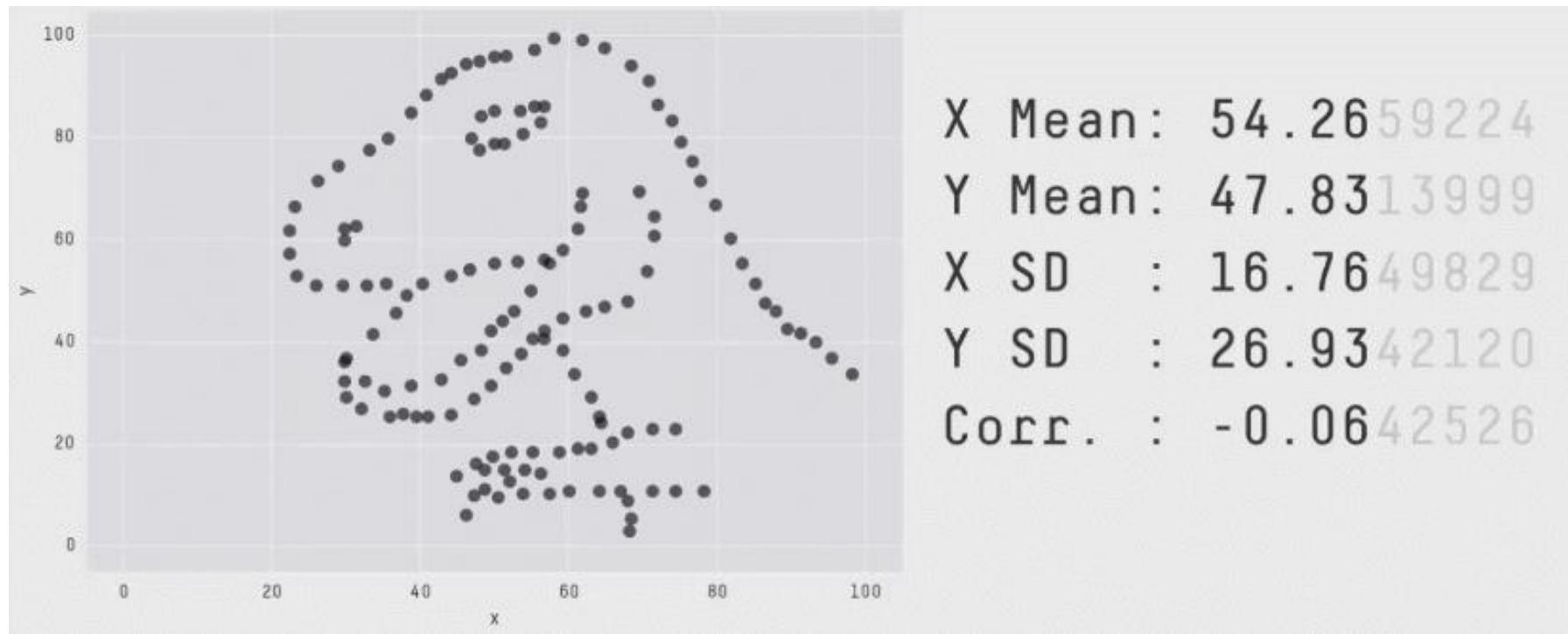
Based on CMU CS15-688 materials

Visualization

- Data exploration visualization: figuring out what is true
- Data presentation visualization: convincing other people it is true
- Before you run any analysis, build any machine learning system, etc., always visualize your data
- If you can't identify a trend or make a prediction for your dataset, neither will an automated algorithm

Visualization vs. Statistics

- Visualization almost always presents a more informative (though less quantitative) view of your data than statistics
 - This is a mathematical property: n data points and m equations to satisfy, with $n > m$



Source: <https://www.autodeskresearch.com/publications/samestats>

Data Types

- Nominal
 - Categorical data
 - No quantitative value, no ordering
 - Example – Pet: {dog, cat, rabbit, ...}
 - Operations: $=, \neq$

- Ordinal
 - Categorical data, with ordering
 - Example – Rating: {1, 2, 3, 4, 5}
 - Operations: $=, \neq, \geq, \leq, >, <$

- Interval
 - Numerical data
 - Zero has no fixed meaning
 - Example – Temperature Celsius
 - Operations: $=, \neq, \geq, \leq, >, <, +, -$

- Ratio
 - Numerical data
 - Zero has special meaning
 - Example – Height, Weight, ...
 - Operations: $=, \neq, \geq, \leq, >, <, +, -, \times, \div$

Visualization Types

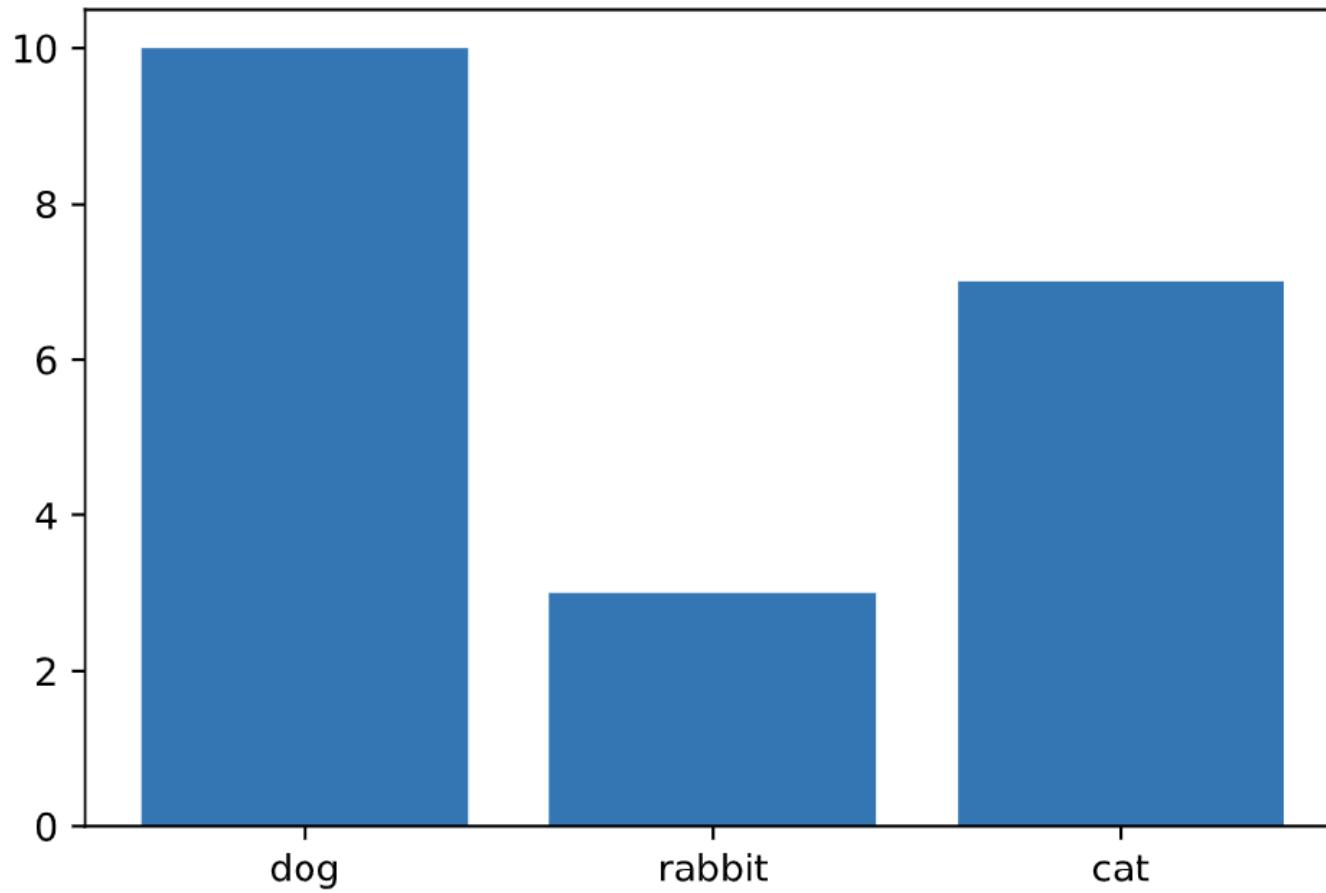
- 1D
 - Bar chart, pie chart, histogram
- 2D
 - Scatter plot, line plot, box and whisker plot, heatmap
- 3D
 - Scatter matrix, bubble chart

1 D: Bar Chart

	Data
Nominal	✓
Ordinal	✓
Interval	✗
Ratio	✗

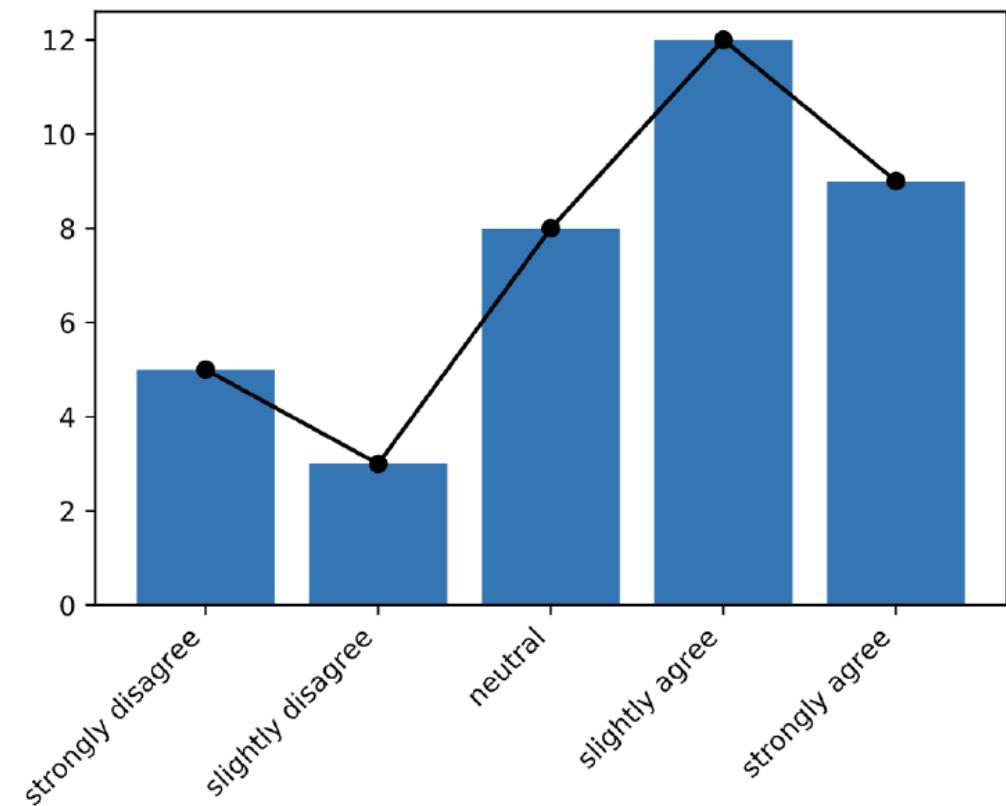
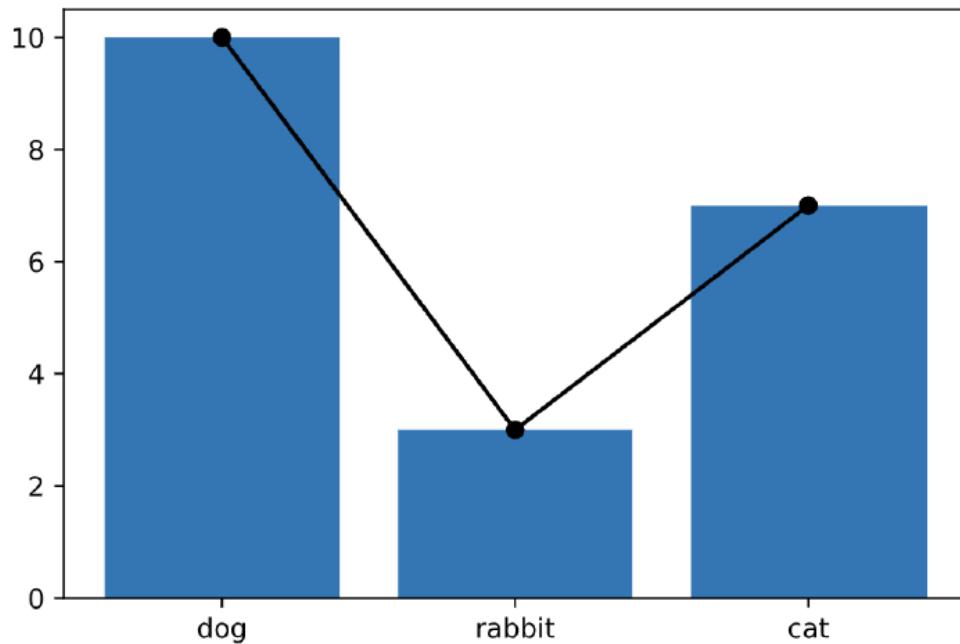


Suggestions, not rules



ID: Bar Chart (bad)

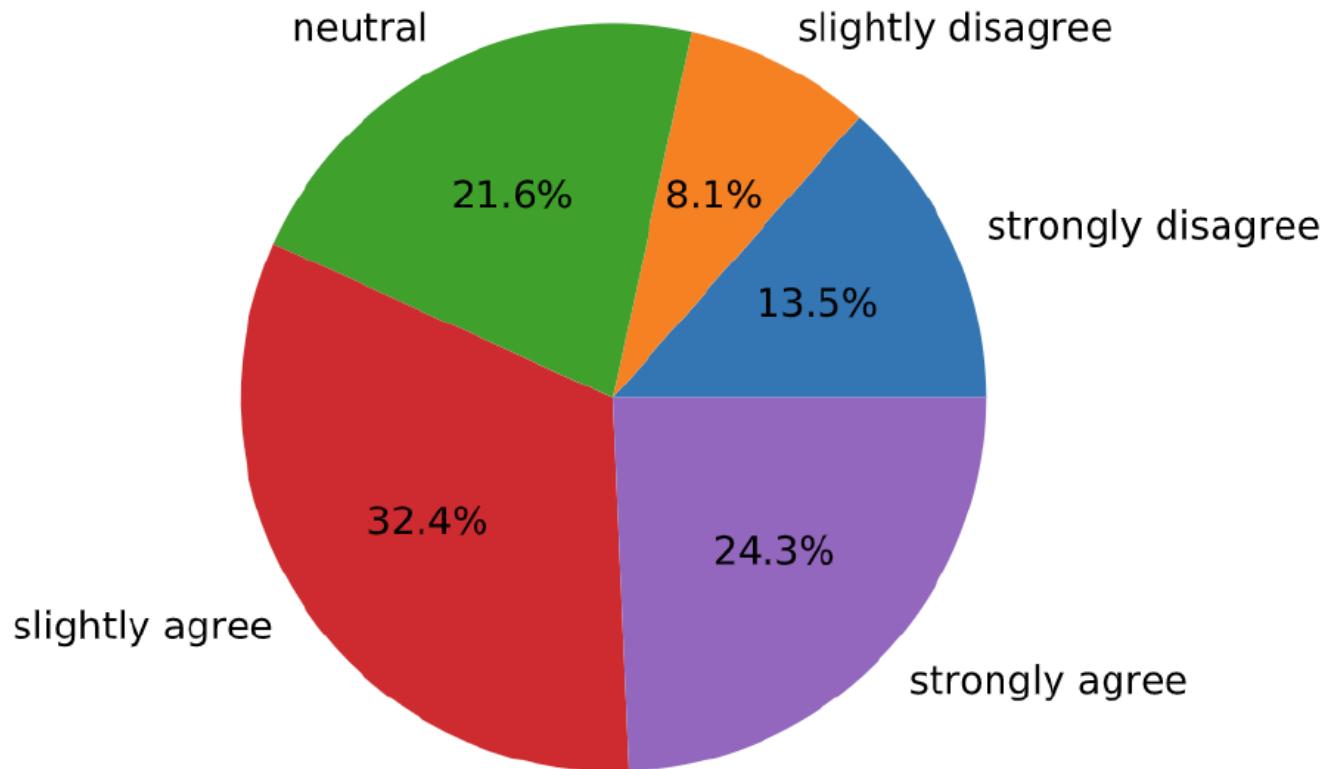
- Don't use lines within a bar chart categorical or ordinal features



ID: Pie Chart

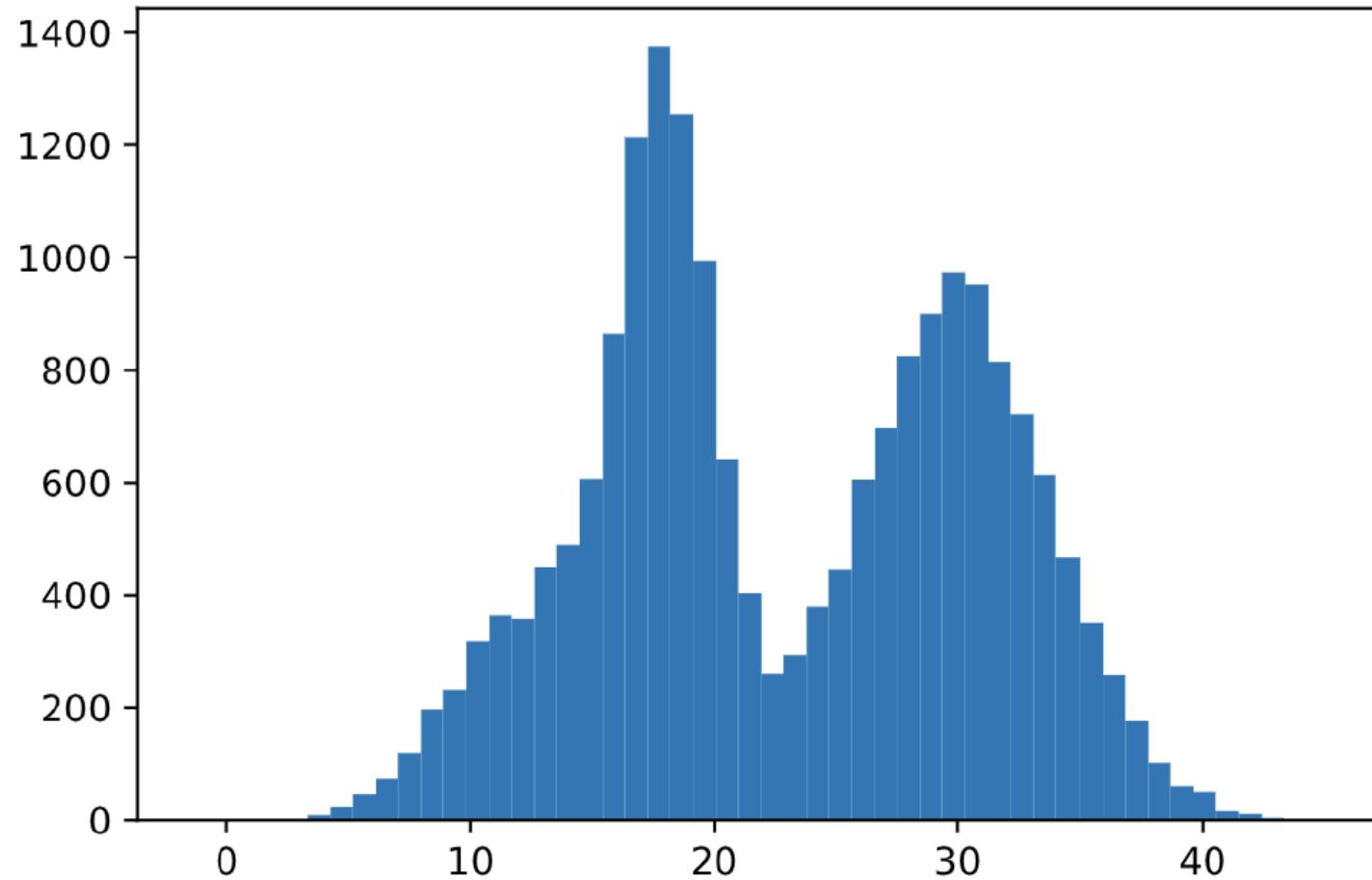
- What's wrong?
 - See "[Why you shouldn't use pie charts](#)"

	Data
Nominal	X
Ordinal	X
Interval	X
Ratio	X



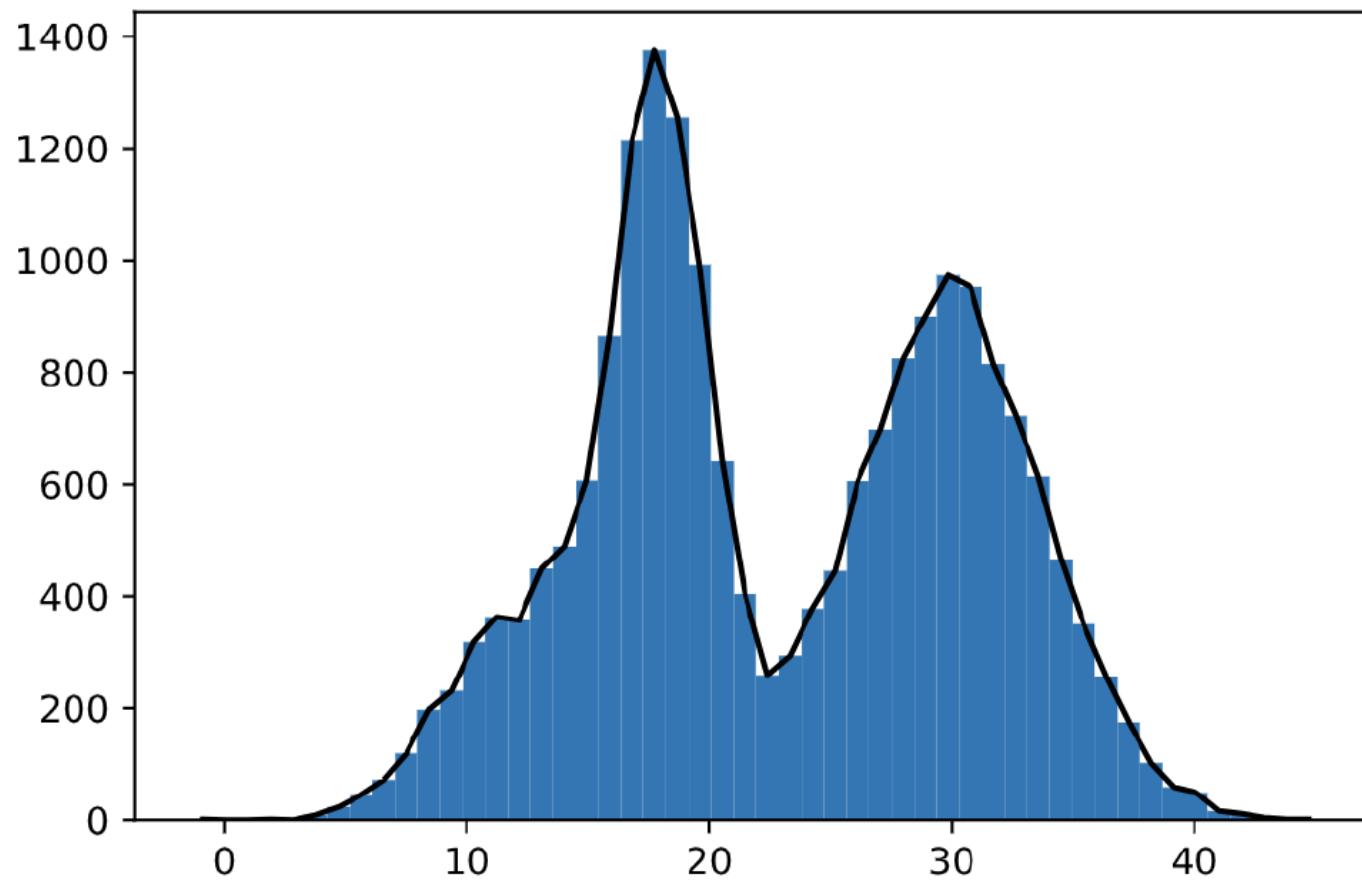
1 D: Histogram

	Data
Nominal	X
Ordinal	X
Interval	✓
Ratio	✓



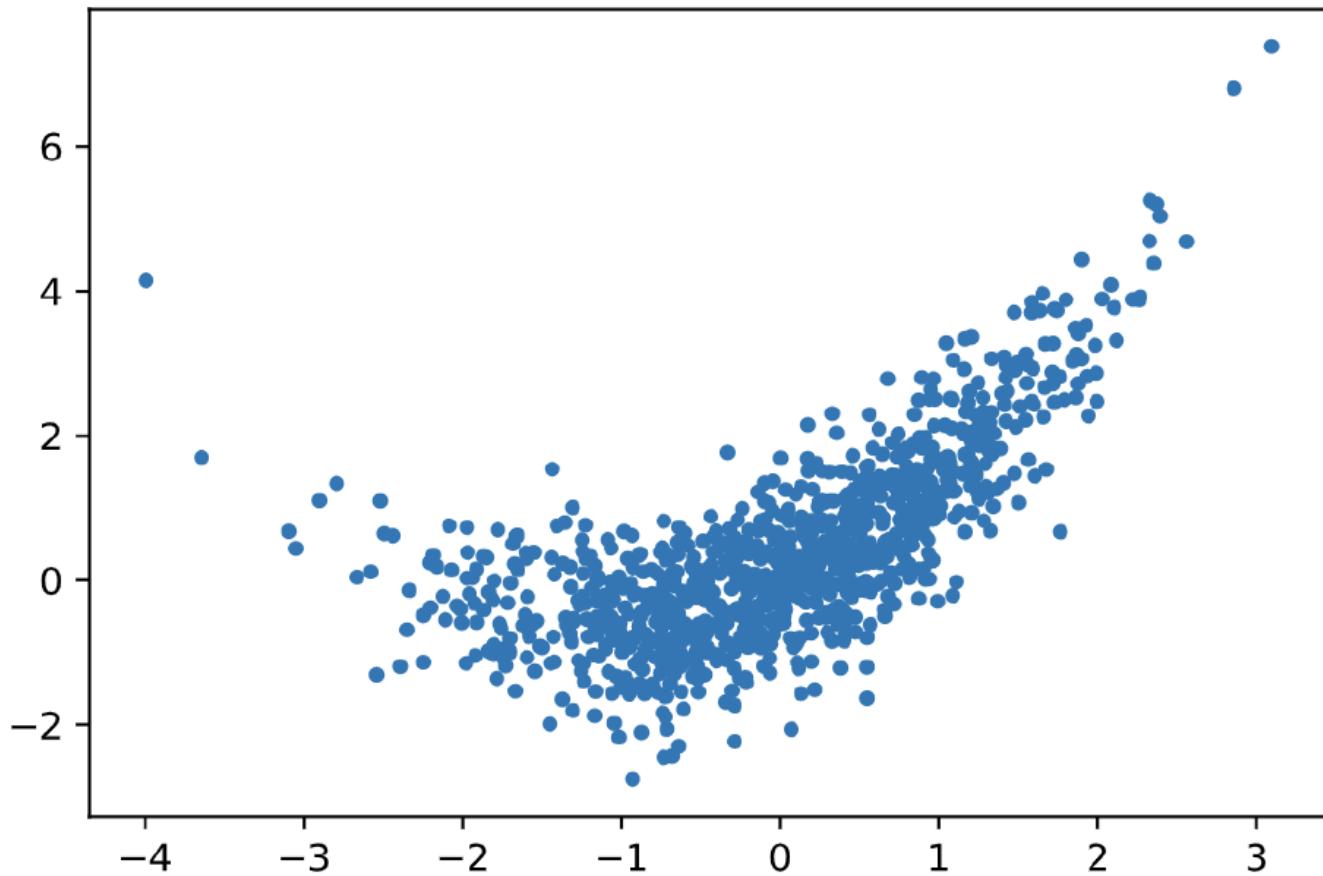
I D: Histogram

- OK to use lines within a histogram (but not very informative)



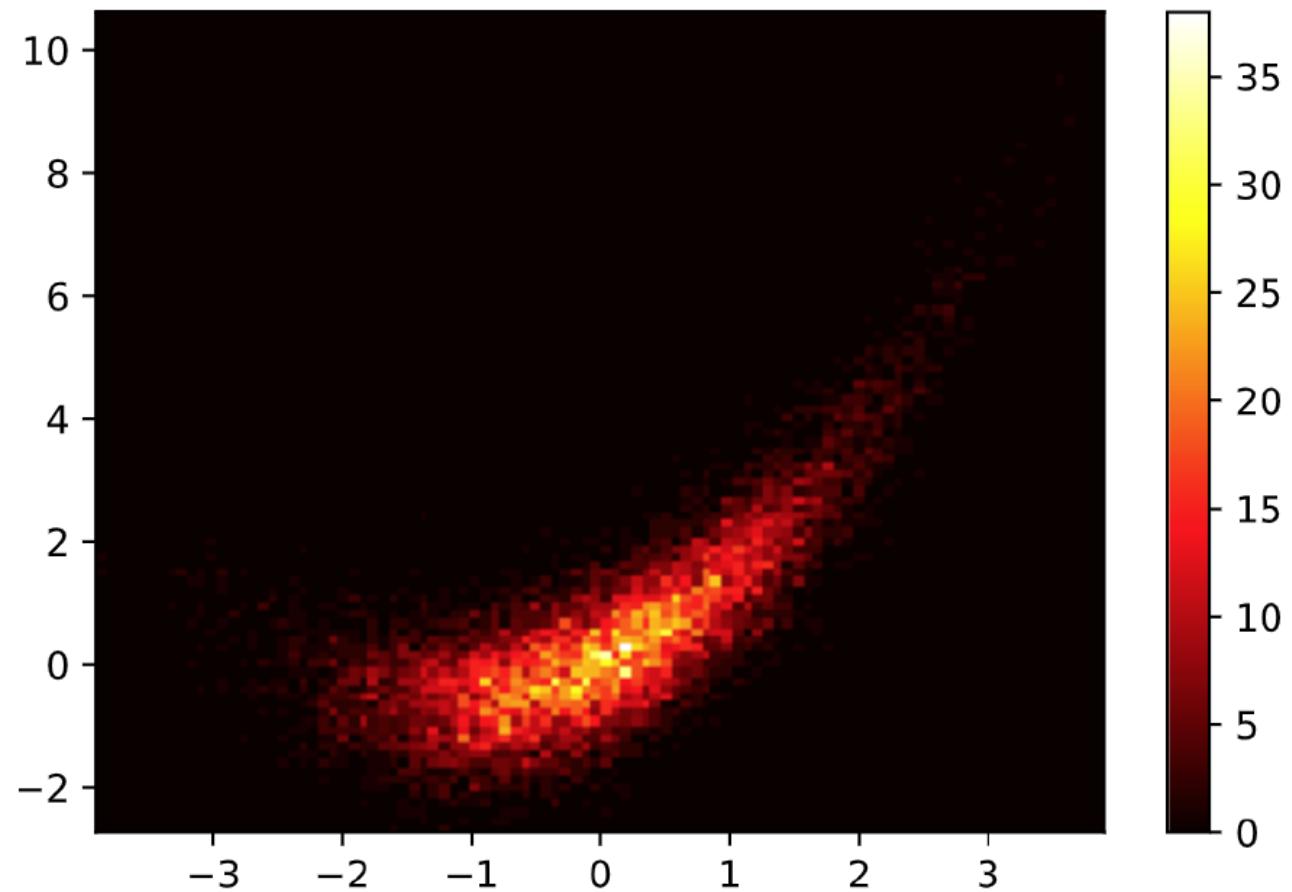
2D: Scatter Plot

	Dim 1	Dim 2
Nominal	X	X
Ordinal	X	X
Interval	✓	✓
Ratio	✓	✓



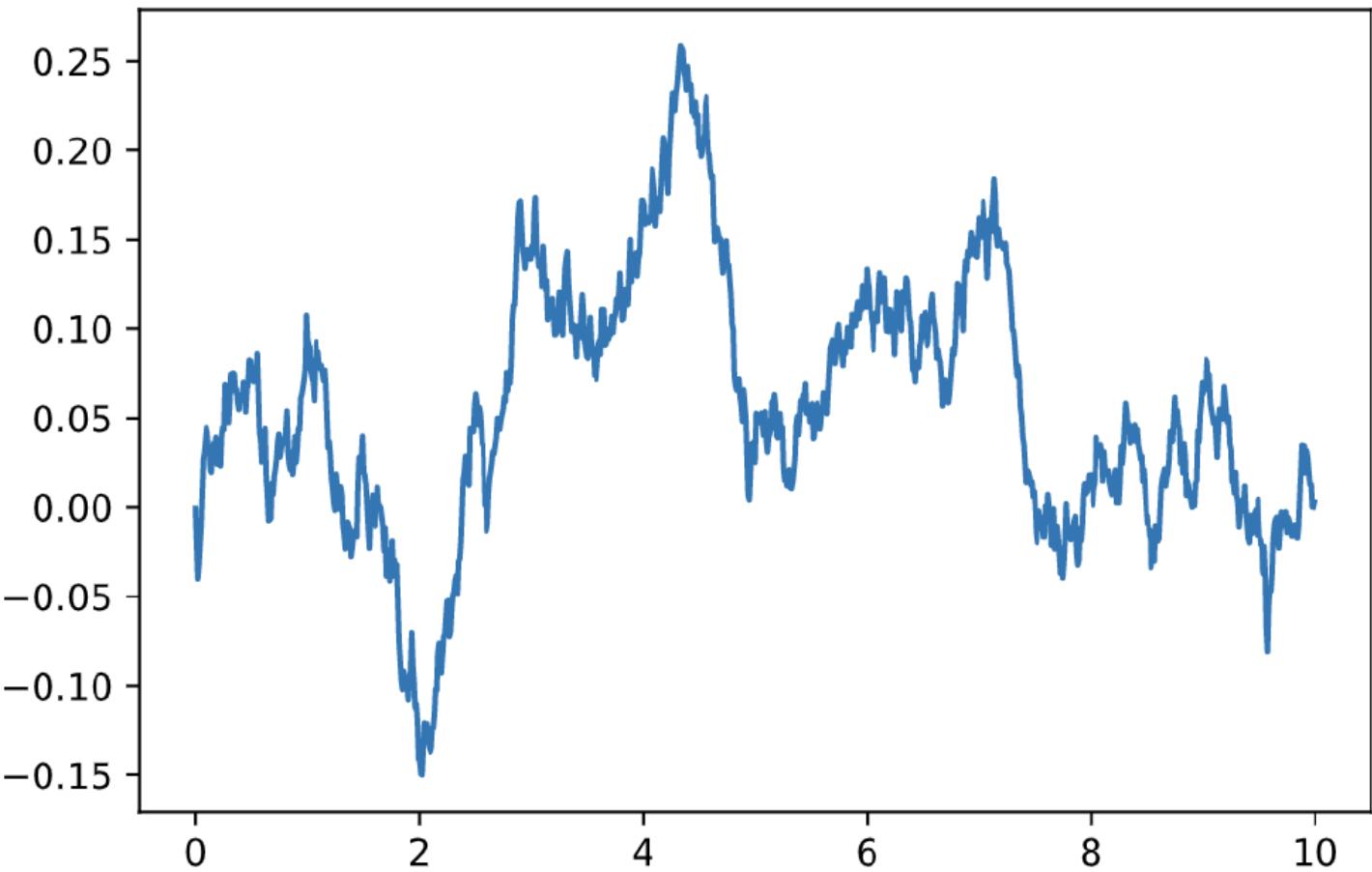
2D: Heatmap (density or 2D histogram)

	Dim 1	Dim 2
Nominal	X	X
Ordinal	X	X
Interval	✓	✓
Ratio	✓	✓



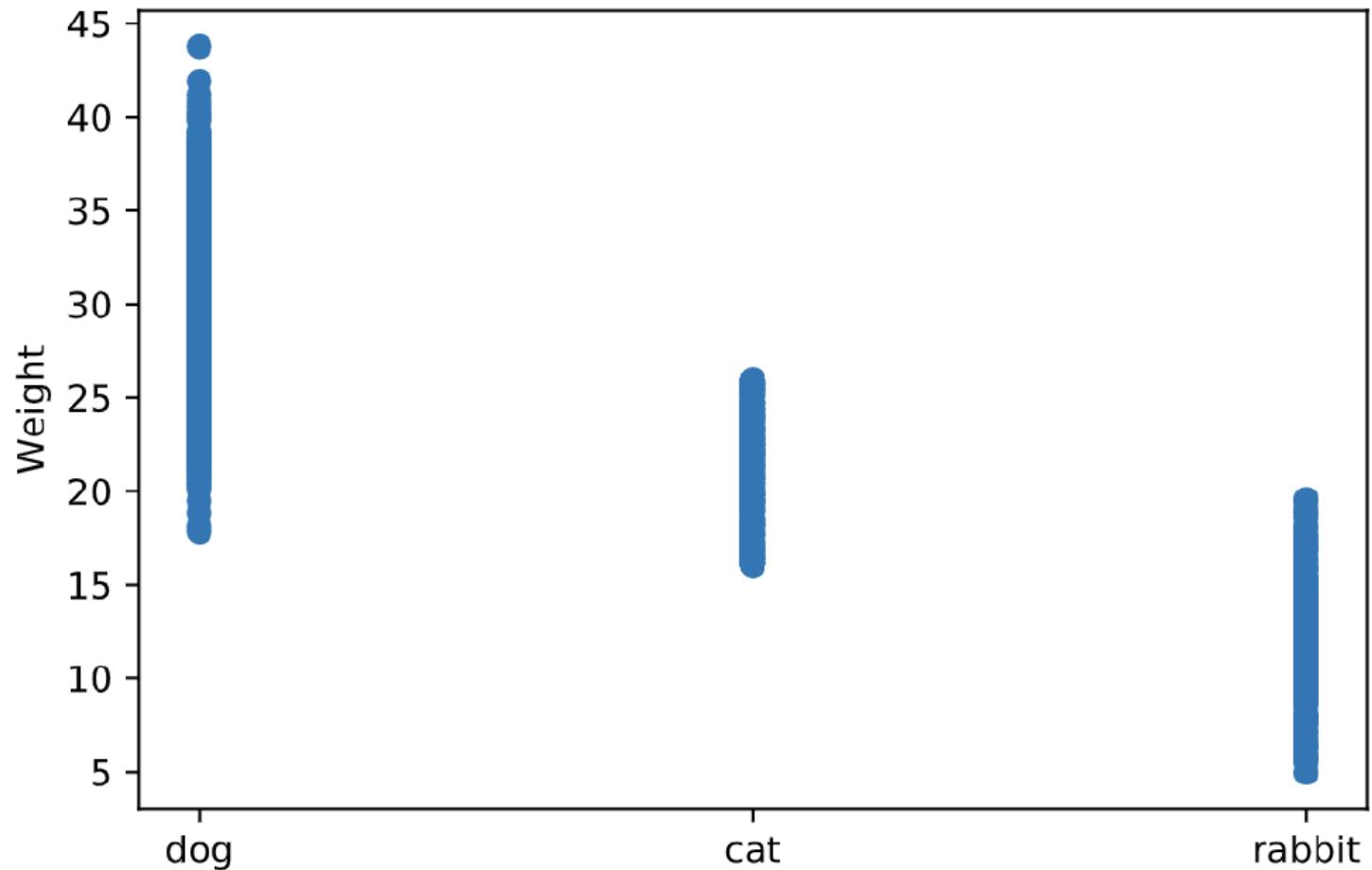
2D: Line Plot

	Dim 1	Dim 2
Nominal	X	X
Ordinal	X	X
Interval	✓	✓
Ratio	✓	✓



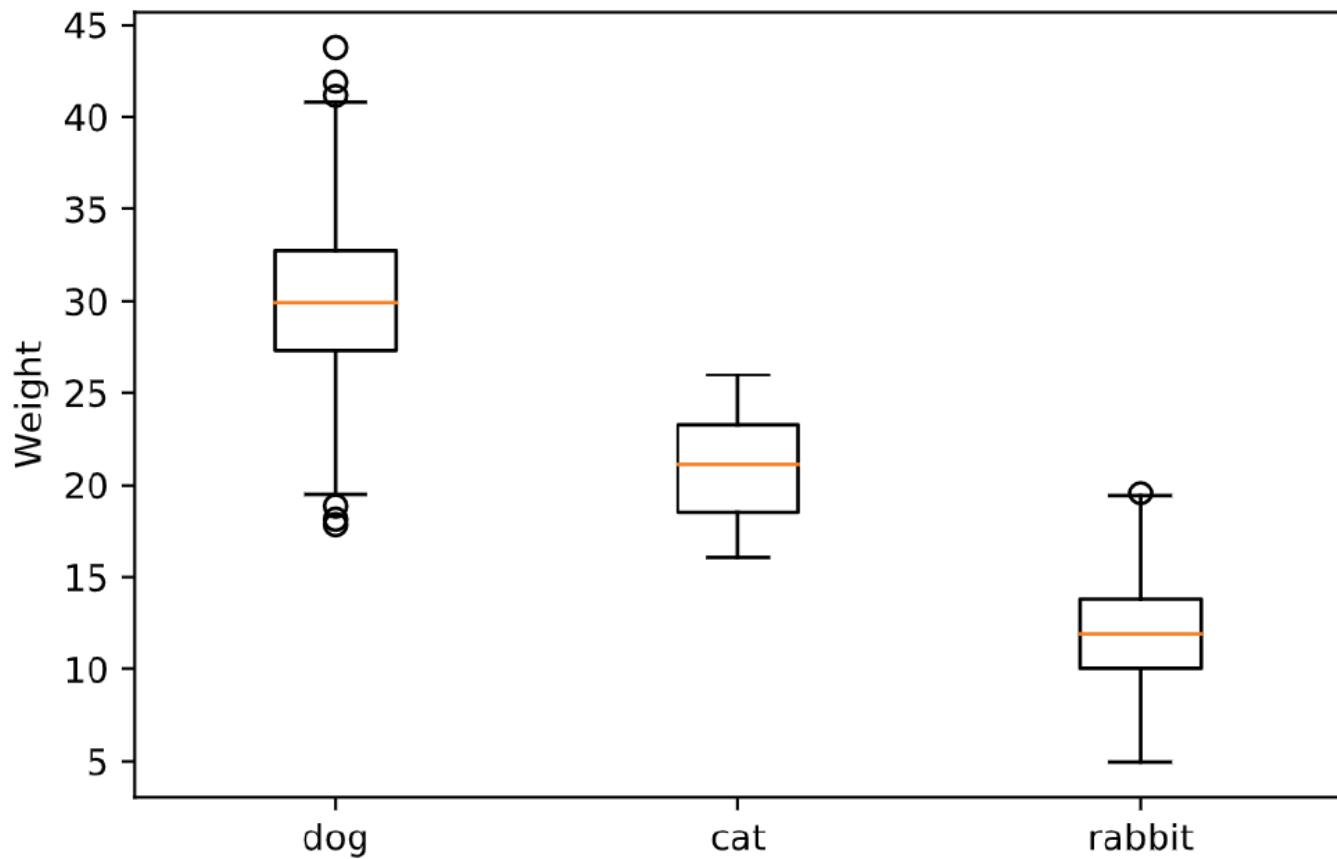
2D: Scatter Plot (bad)

	Dim 1	Dim 2
Nominal	X	X
Ordinal	X	X
Interval	✓	✓
Ratio	✓	✓



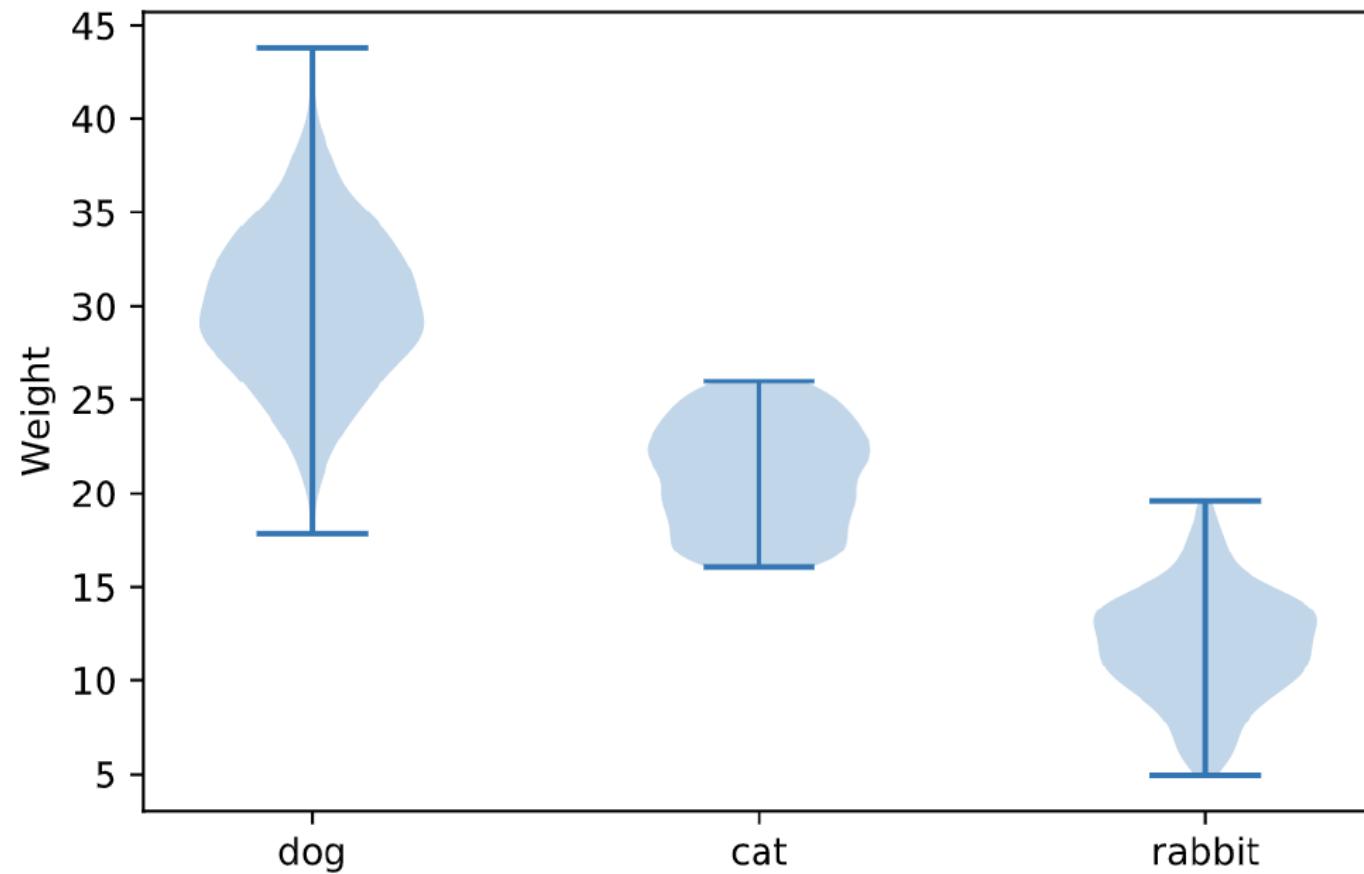
2D: Box and Whiskers

	Dim 1	Dim 2
Nominal	✓	✗
Ordinal	✓	✗
Interval	✗	✓
Ratio	✗	✓



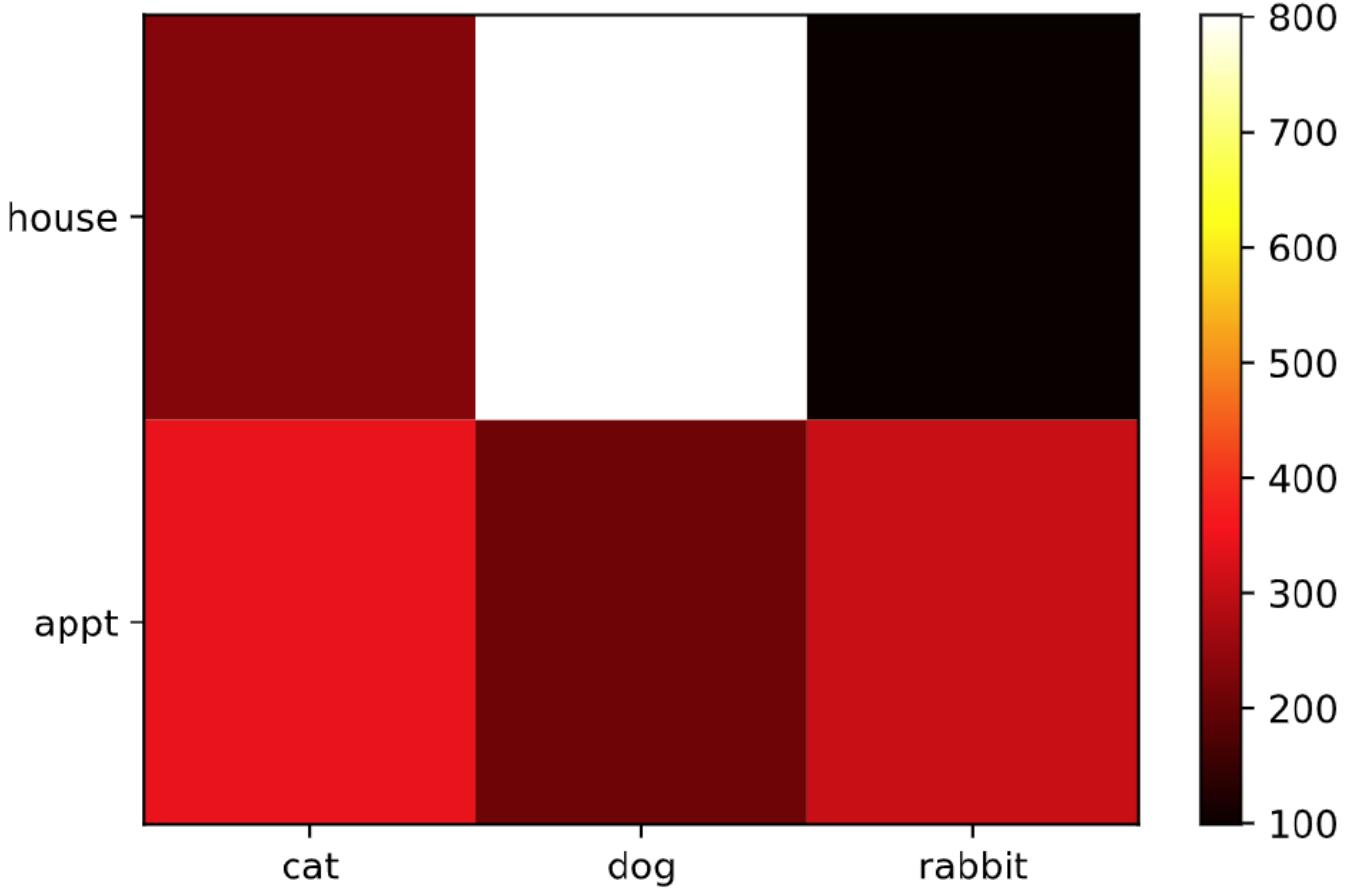
2D:Violin Plot

	Dim 1	Dim 2
Nominal	✓	✗
Ordinal	✓	✗
Interval	✗	✓
Ratio	✗	✓



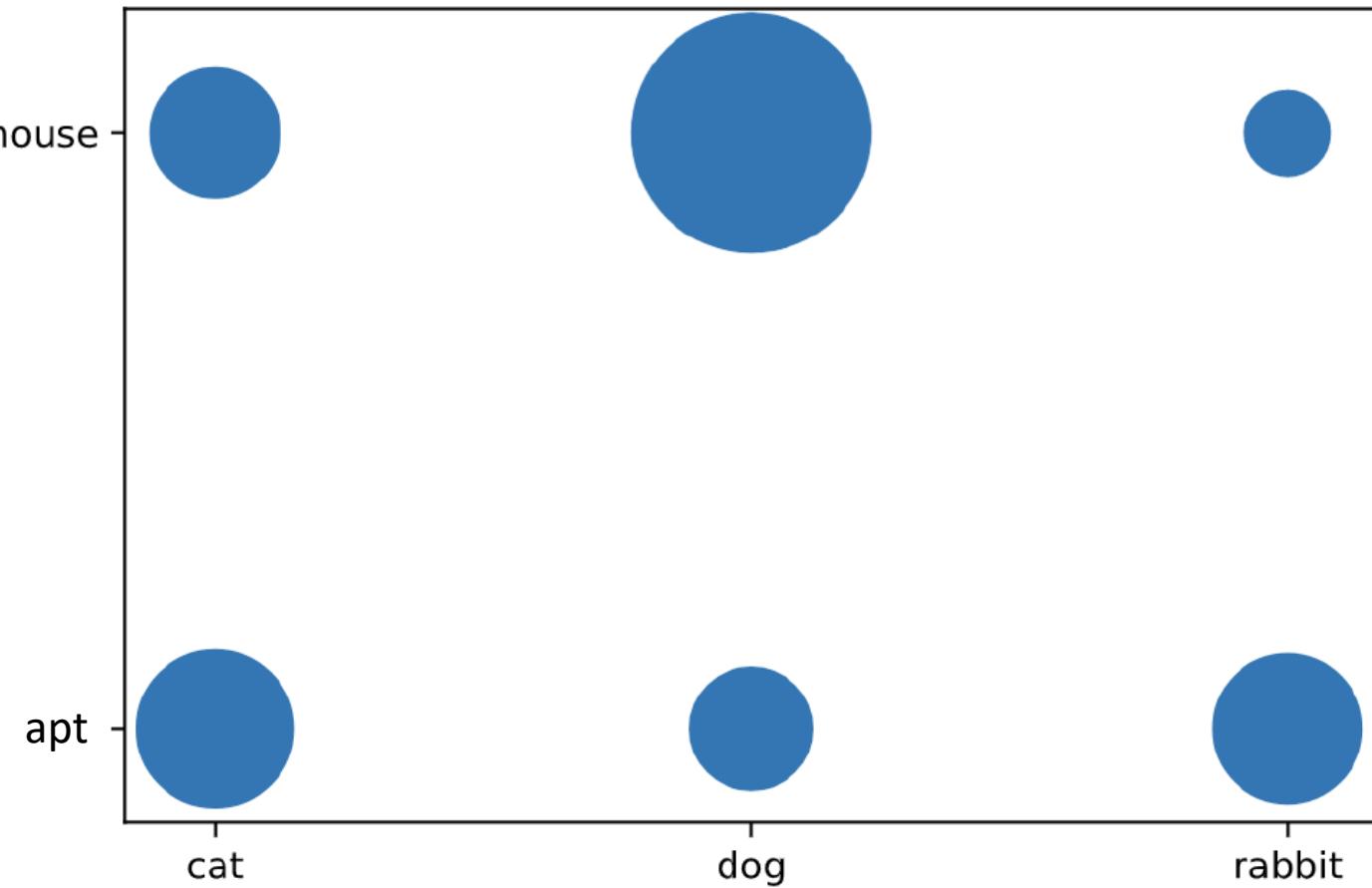
2D: Heatmap (matrix)

	Dim 1	Dim 2
Nominal	✓	✓
Ordinal	✓	✓
Interval	✗	✗
Ratio	✗	✗



2D: Bubble Plot

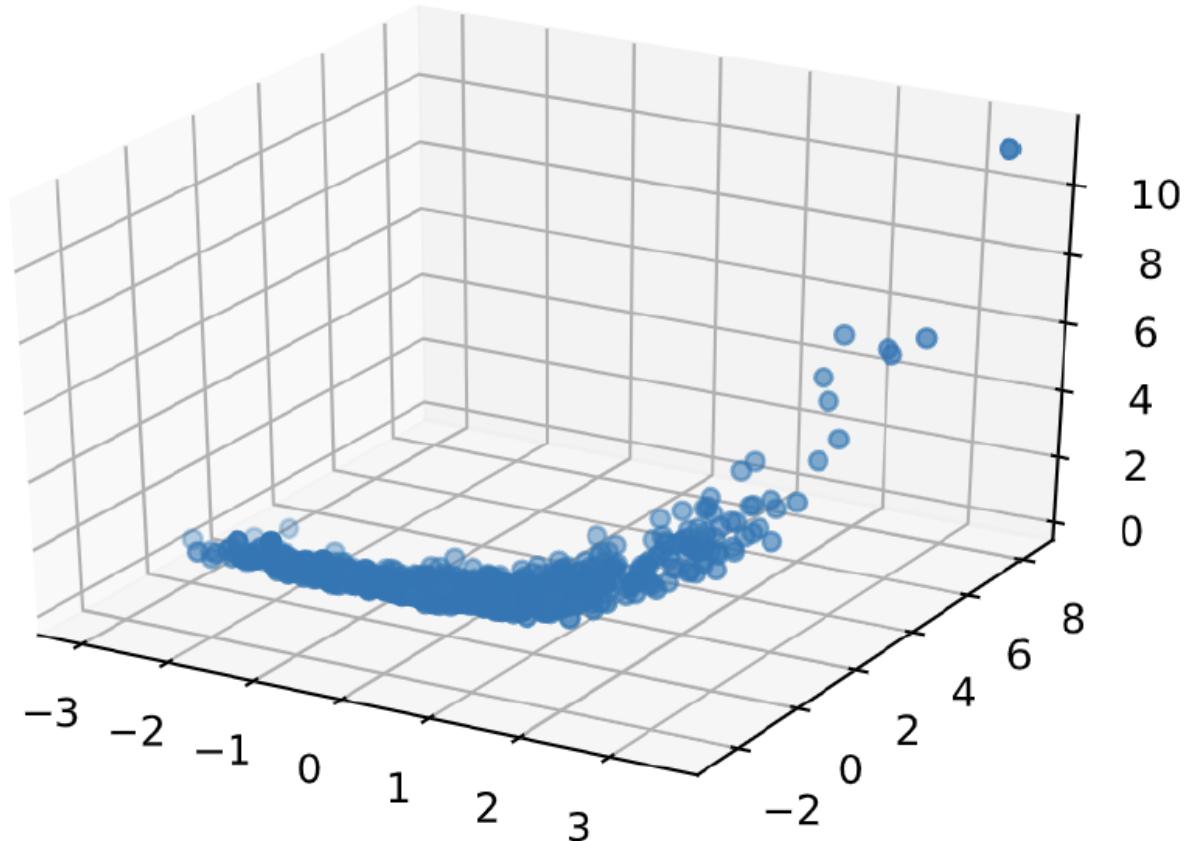
	Dim 1	Dim 2
Nominal		
Ordinal		
Interval	X	X
Ratio	X	X



3D+: 3D Scatter Plot

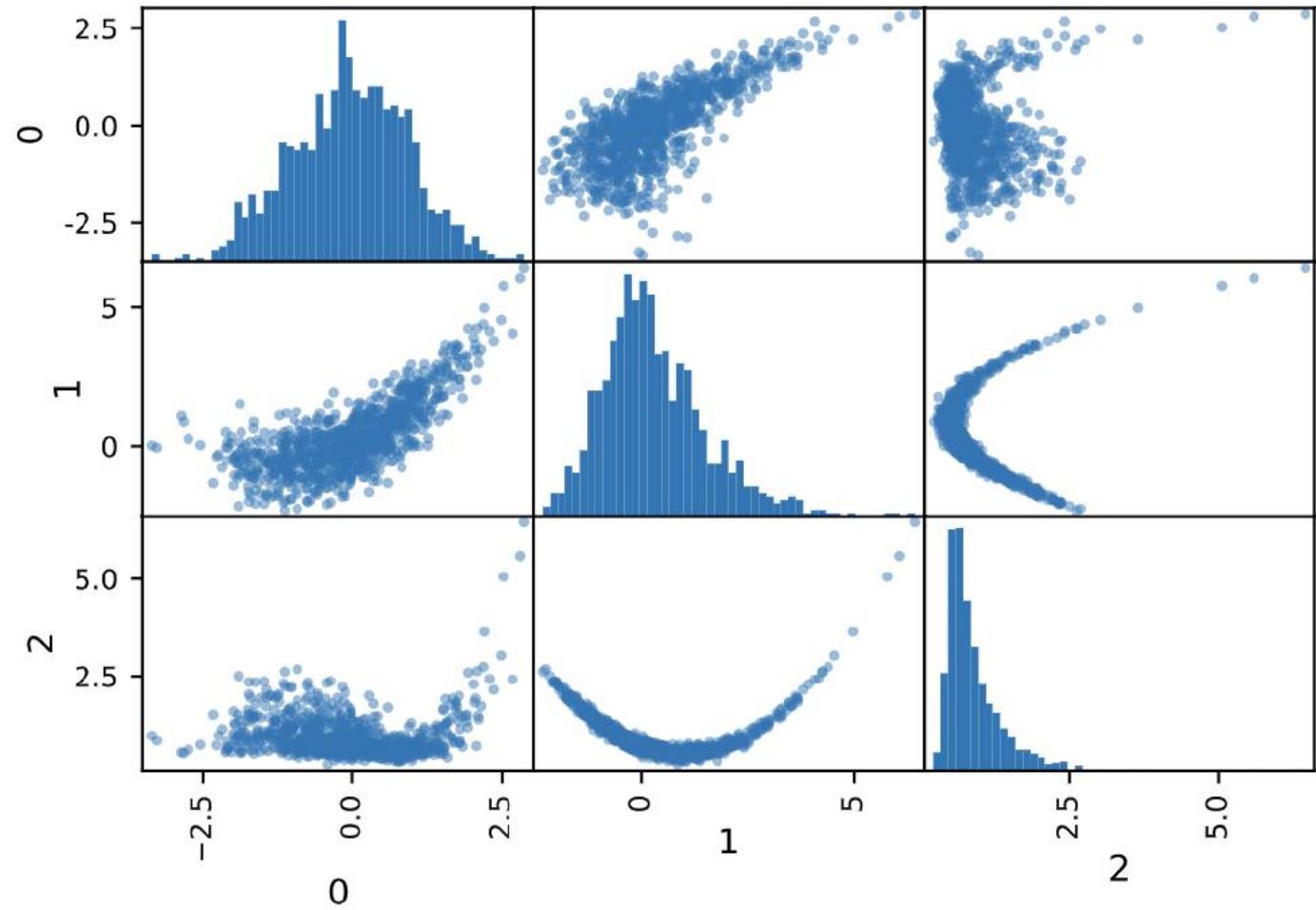
- What's wrong?

	Dim 1	Dim 2	Dim 3
Nominal	X	X	X
Ordinal	X	X	X
Interval	X	X	X
Ratio	X	X	X



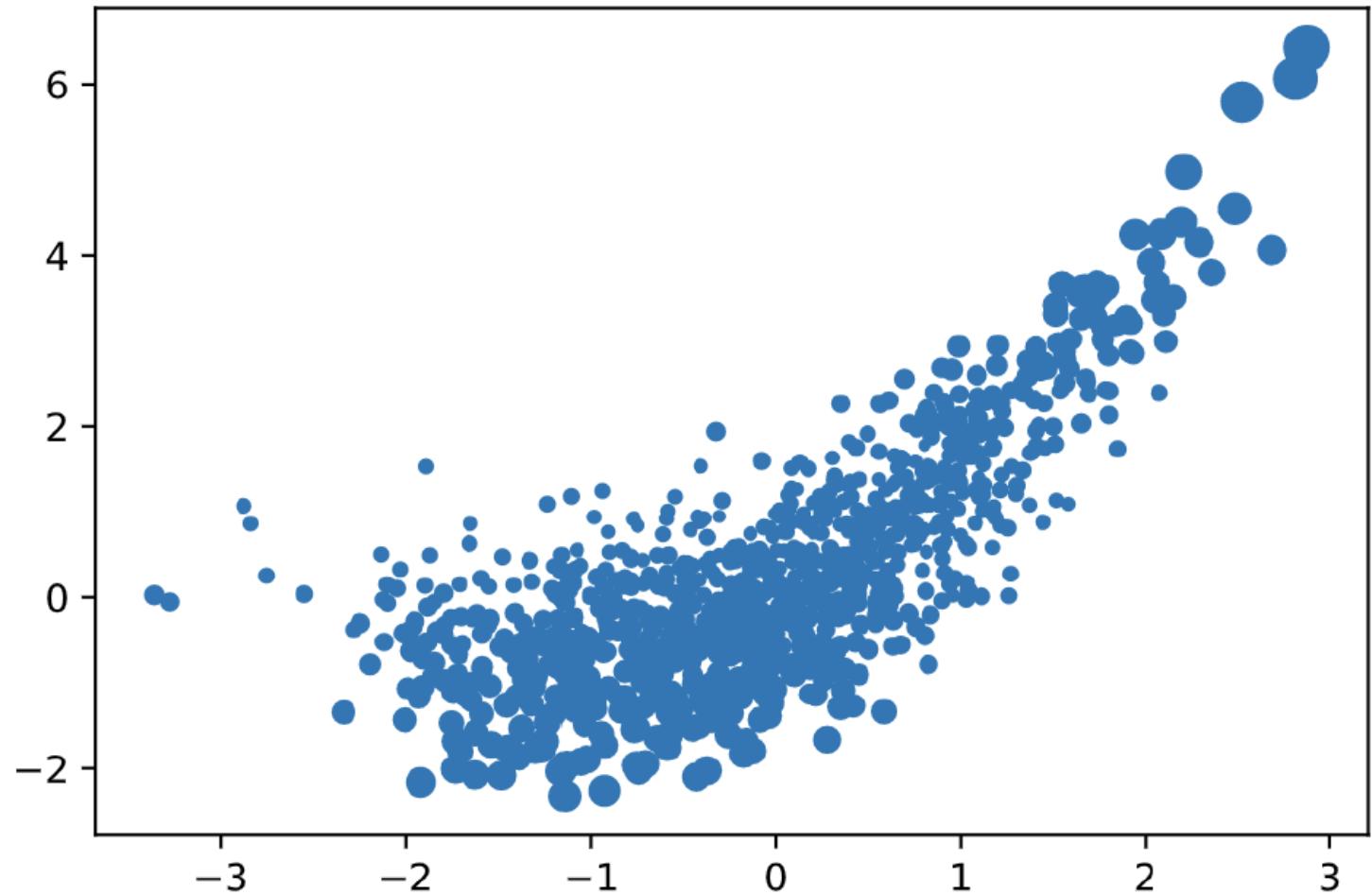
3D+: Scatter Plot Matrix

	Dim 1	Dim 2	Dim 3
Nominal	X	X	X
Ordinal	X	X	X
Interval	✓	✓	✓
Ratio	✓	✓	✓



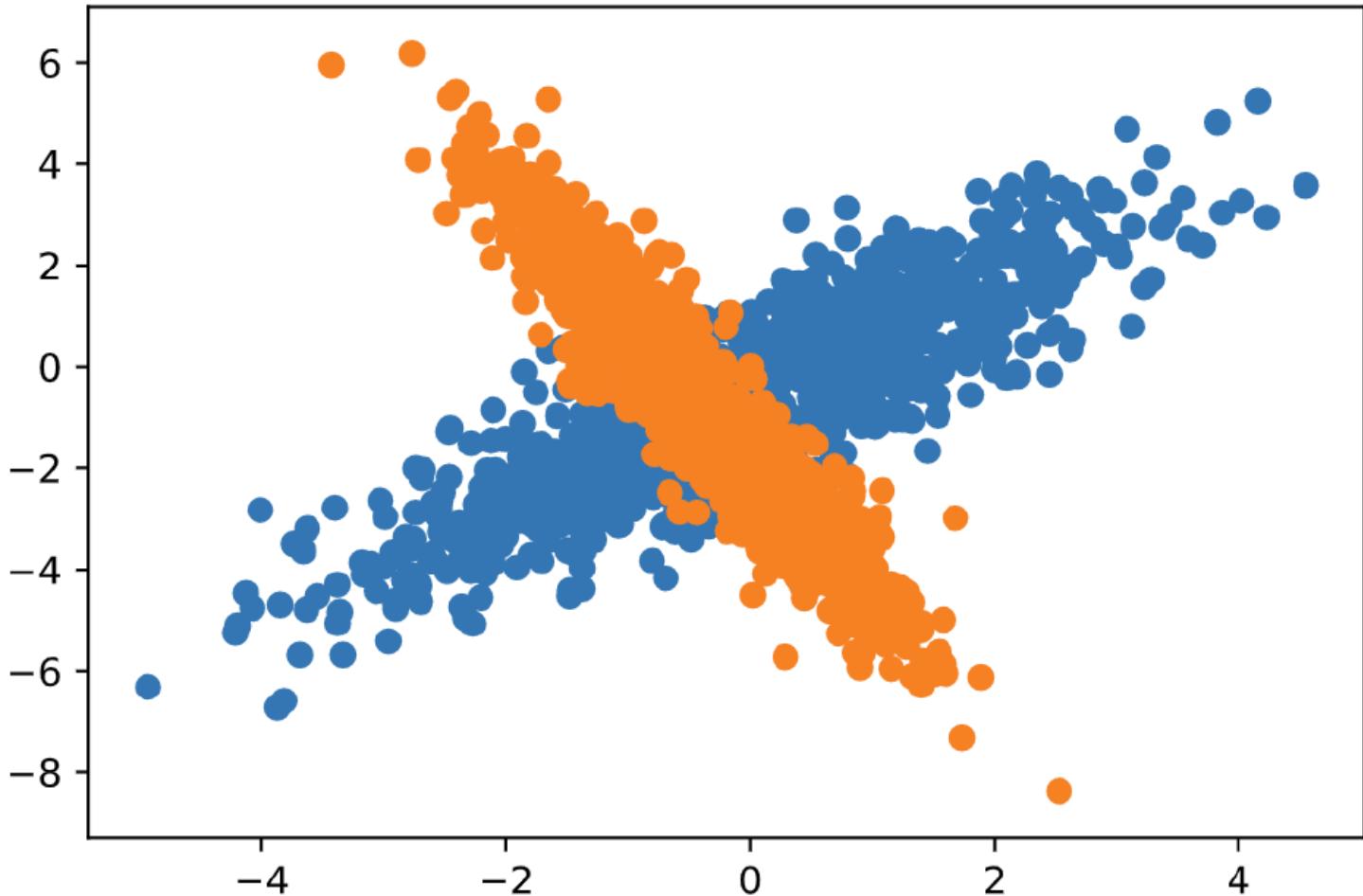
3D+: Bubble Plot

	Dim 1	Dim 2	Dim 3
Nominal	X	X	X
Ordinal	X	X	X
Interval	✓	✓	✓
Ratio	✓	✓	✓



3D+: Color Scatter Plot

	Dim 1	Dim 2	Dim 3
Nominal	X	X	✓
Ordinal	X	X	✓
Interval	✓	✓	X
Ratio	✓	✓	X



Matplotlib

Matplotlib

- The most popular plotting library for Python
 - Based on NumPy
- MATLAB-style plotting interface with hierarchical organized elements
 - pyplot module: provides state-machine environment at the top of hierarchy
- Open source (<http://matplotlib.org>)
 - Original author: John D. Hunter (2003) → Michael Droettboom (2012)
- pyplot tutorials
 - https://matplotlib.org/users/pyplot_tutorial.html

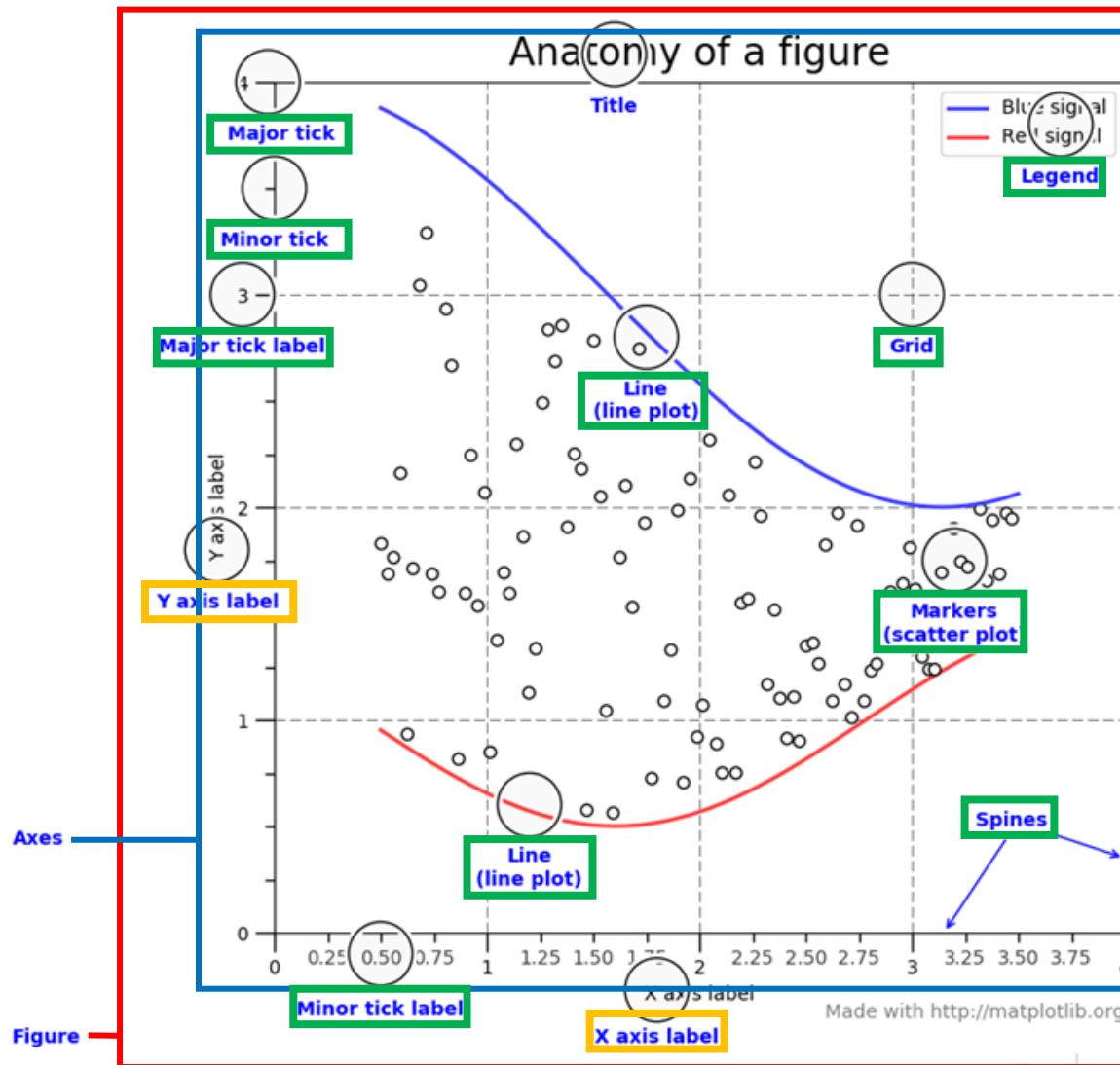
Overview

- **matplotlib submodules**
 - **pyplot**: provides MATLAB-like plotting framework
 - Closely related with actual plotting functionality
 - **image**: image loading, rescaling and display operations
 - **colors**: converting numbers or color arguments to RGB or RGBA
 - **cm(colormap)**: provides function for color mapping functionality
 - **collections**: for efficient drawing of large collections
- We'll focus on **pyplot** module because it is in charge of creating plotting
- Other submodules support better plotting environment

Matplotlib Submodule List

matplotlib.afm	matplotlib.blocking_input	matplotlib.legend_handler	matplotlib.style
matplotlib.animation	matplotlib.category	matplotlib.lines	matplotlib.table
matplotlib.artist	matplotlib.cbook	matplotlib.markers	matplotlib.testing
matplotlib.axes	matplotlib.cm	matplotlib.mathtext	matplotlib.testing.compare
matplotlib.axis	matplotlib.collections	matplotlib.mlab	matplotlib.testing.decorators
matplotlib.backend_bases	matplotlib.colorbar	matplotlib.offsetbox	matplotlib.testing.disable_internet
matplotlib.backend_managers	matplotlib.colors	matplotlib.patches	matplotlib.exceptions
matplotlib.backend_tools	matplotlib.container	matplotlib.path	
matplotlib.backends.backend_agg	matplotlib.contour	matplotlib patheffects	
matplotlib.backends.backend_cairo	matplotlib.dates	matplotlib.projections	
matplotlib.backends.backend_mixed	matplotlib.dviread	matplotlib.projections.polar	
matplotlib.backends.backend_nbagg	matplotlib.figure	matplotlib.pyplot	
matplotlib.backends.backend_pdf	matplotlib.font_manager	matplotlib.rcsetup	
matplotlib.backends.backend_pgf	matplotlib.fontconfig_pattern	matplotlib.sankey	
matplotlib.backends.backend_ps	matplotlib.gridspec	matplotlib.scale	matplotlib.type1font
matplotlib.backends.backend_svg	matplotlib.image	matplotlib.sphinxext.plot_d	matplotlib.units
matplotlib.backends.backend_tkagg	matplotlib.legend	matplotlib.spines	matplotlib.widgets

Pyplot Plotting Example



Matplotlib Classes

- 4 important classes in matplotlib

- **Artist**

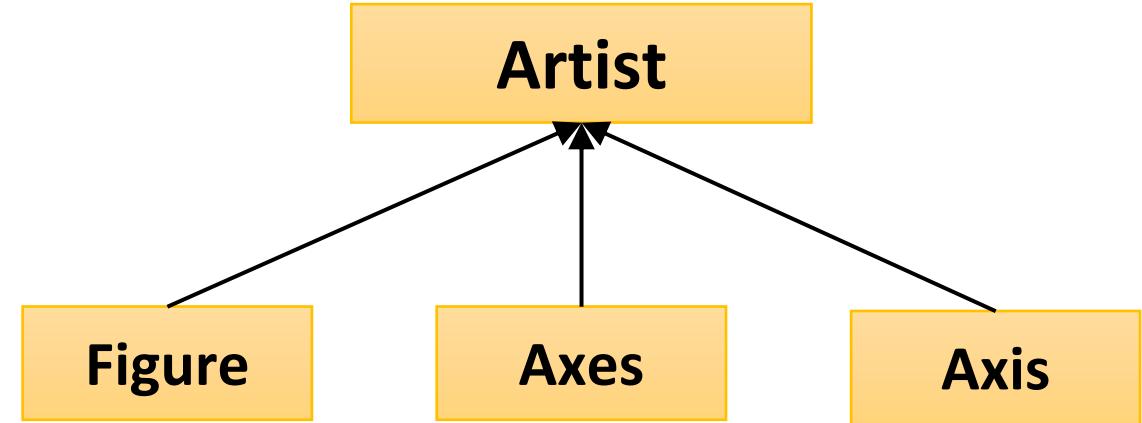
- Basically everything you can see on the figure

- **Figure**

- Consider as a window
 - A figure may contain several child Axes and keep track of them
 - Several figures can be created

- **Axes: region of an image or a plot**

- **Axis: strings labeling ticks**

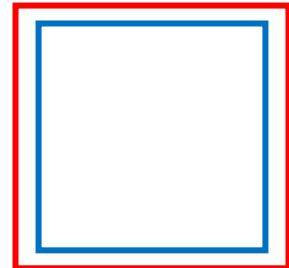


Using Pyplot

- Import library

```
%matplotlib inline  
import matplotlib.pyplot as plt  
%config InlineBackend.figure_format = 'svg'
```

For old jupyter notebook:
plot graphs without show()



- Pyplot uses **1 Figure** with **1 Axes** by default and use it as current Axes
- Each object can be added as needed

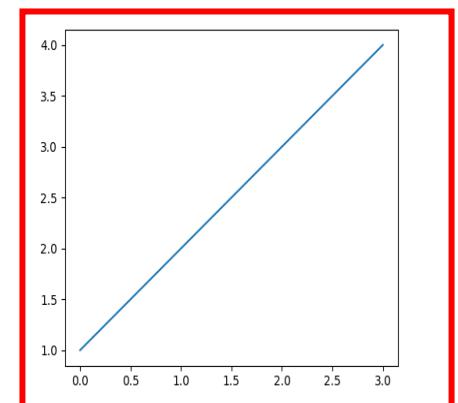
- Plot on current Axes

```
plt.plot([1, 2, 3, 4])
```

- Automatically set the x, y ranges and tick labels

- Display figure

```
plt.show()
```



- For actual display of figures in pyplot, additional statement is necessary

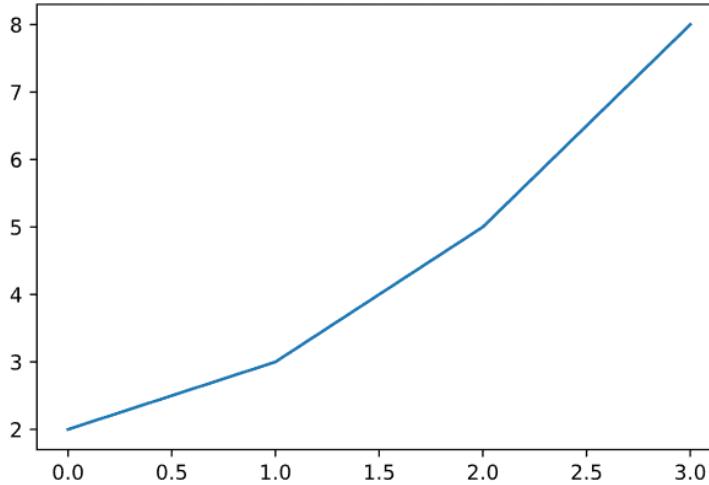
Pyplot Basics

Line and Scatter Plot

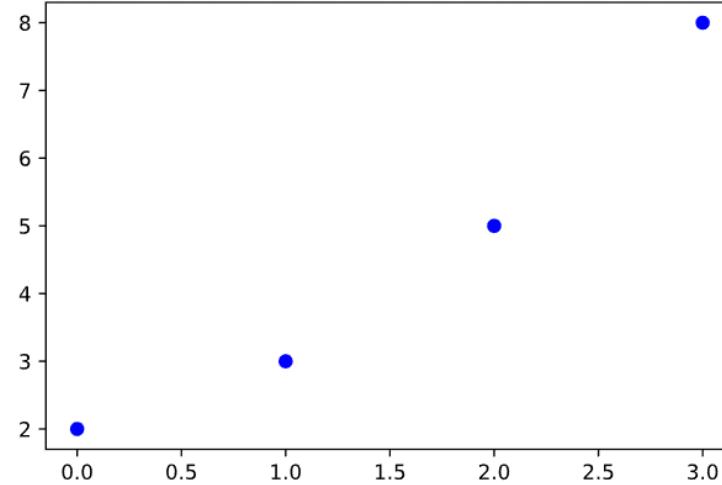
- **`plt.plot([x], y, [fmt], [x2], y2, [fmt2], ...)`**

- The coordinates of the points or line nodes given by `x, y`
- If `x` is omitted, index array `[0, 1, ..., N-1]` is used as `x`
- `fmt`: defines basic formatting like color, marker and linestyle (default: blue line graph)

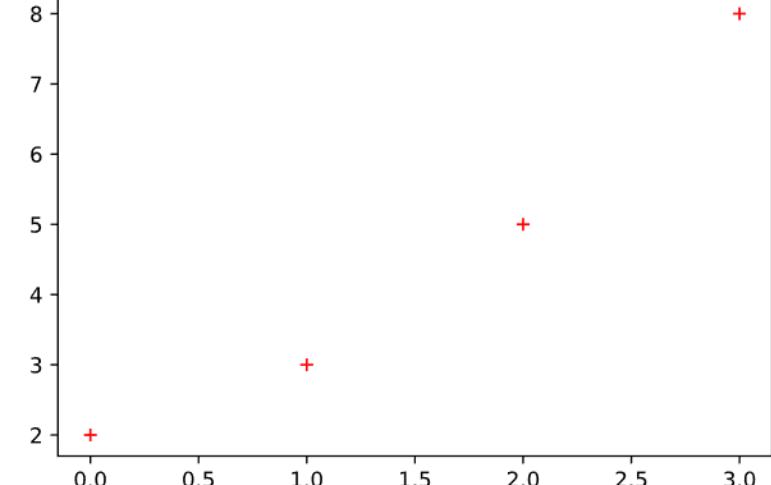
```
plt.plot([2,3,5,8])
```



```
plt.plot([2,3,5,8], 'bo')
```



```
plt.plot([2,3,5,8], 'r+')
```



Markers

'.'	point marker	's'	square marker
', '	pixel marker	'p'	pentagon marker
'o'	circle marker	'*'	star marker
'v'	triangle_down marker	'h'	hexagon1 marker
'^'	triangle_up marker	'H'	hexagon2 marker
'<'	triangle_left marker	'+'	plus marker
'>'	triangle_right marker	'x'	x marker
'1'	tri_down marker	'D'	diamond marker
'2'	tri_up marker	'd'	thin_diamond marker
'3'	tri_left marker	' '	vline marker
'4'	tri_right marker	'_'	hline marker

Line Styles and Colors

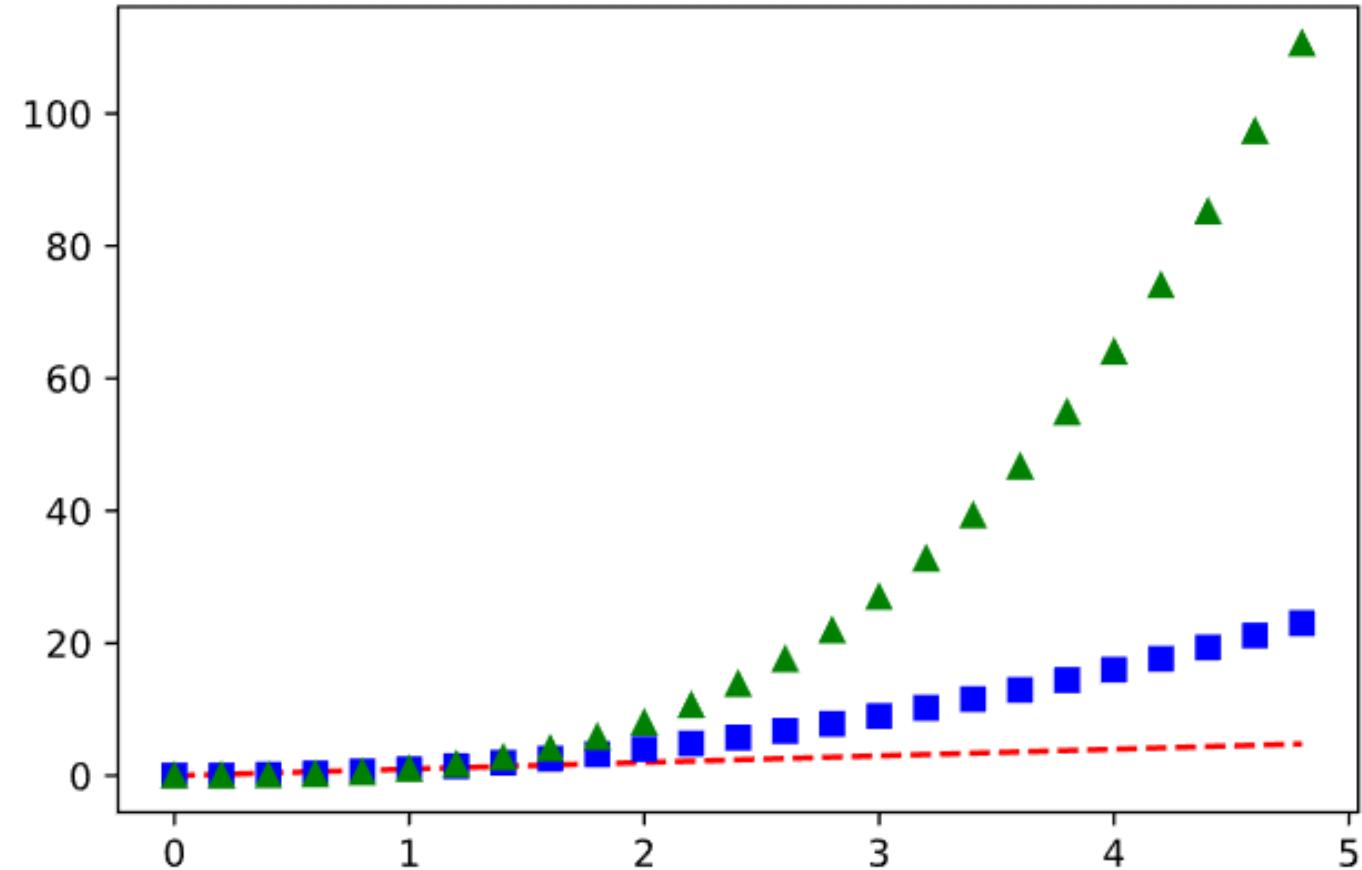
' - '	solid line style	' b '	blue
' -- '	dashed line style	' g '	green
' - . '	dash-dot line style	' r '	red
' : '	dotted line style	' c '	cyan
		' m '	magenta
		' y '	yellow
fmt = '[marker][line][color]'		' k '	black
	(Same color for line and marker)	' w '	white

Plotting Multiple Graphs

- Plotting 3 different data at once

```
t = np.arange(0.0, 5.0, 0.2)
plt.plot(t, t, 'r--',
          t, t**2, 'bs',
          t, t**3, 'g^')
```

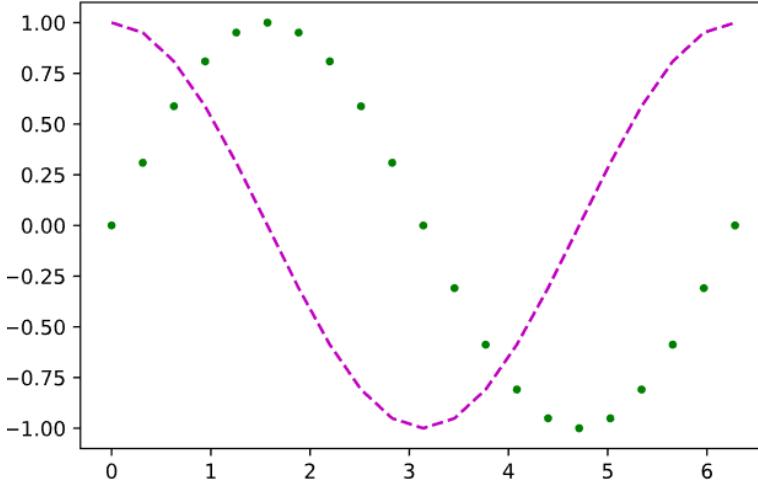
- 'r--': red dashed
- 'bs': blue square
- 'g^': green triangle



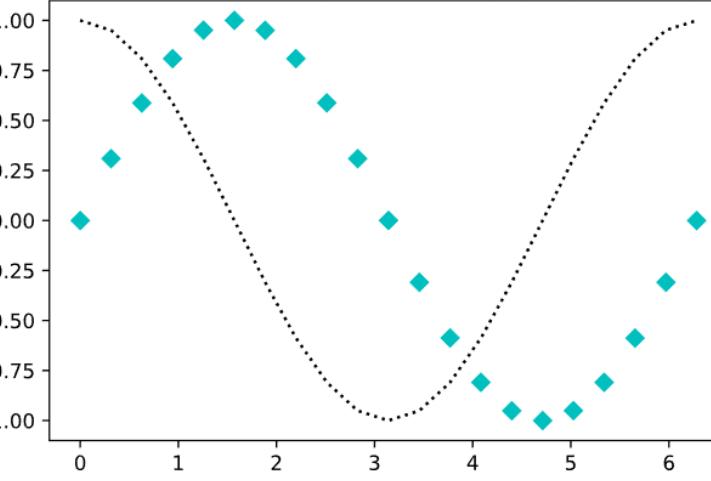
More Examples

```
x = np.linspace(0, 2*np.pi, 21)
y = np.sin(x)
y2 = np.cos(x)
```

```
plt.plot
(x, y, 'g.', x, y2, 'm--')
```

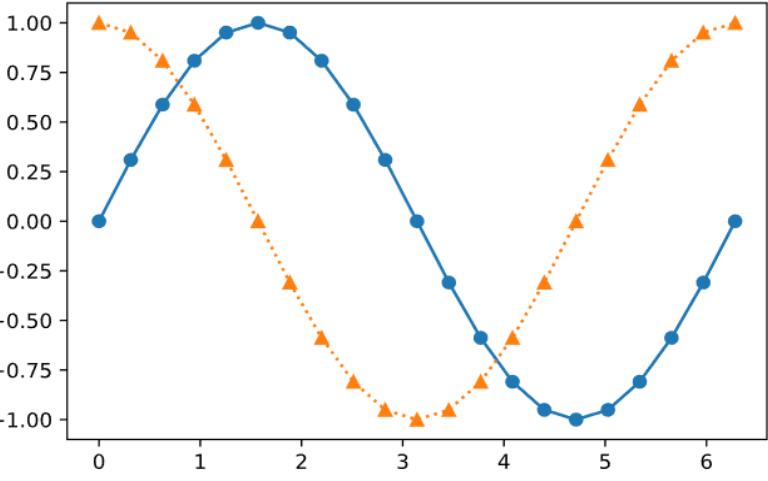


```
plt.plot
(x, y, 'cD', x, y2, 'k-.')
```



Choose colors automatically

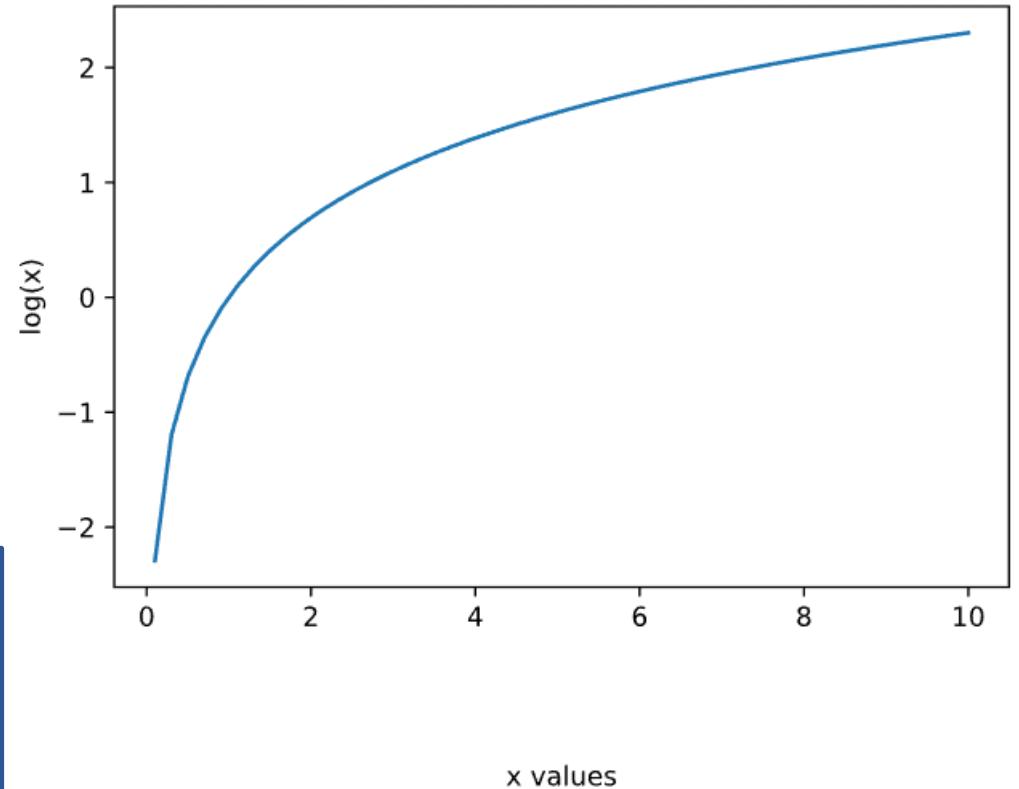
```
plt.plot
(x, y, '-o', x, y2, '^:')
```



Axis Labels

- `plt.xlabel(xlabel, [labelpad], ...)`
- `plt.ylabel(ylabel, [labelpad], ...)`
 - Set the label for the x-axis (or y-axis)
 - `xlabel, ylabel`: string for the label
 - `labelpad`: spacing in points from the axes bounding box including ticks and tick labels

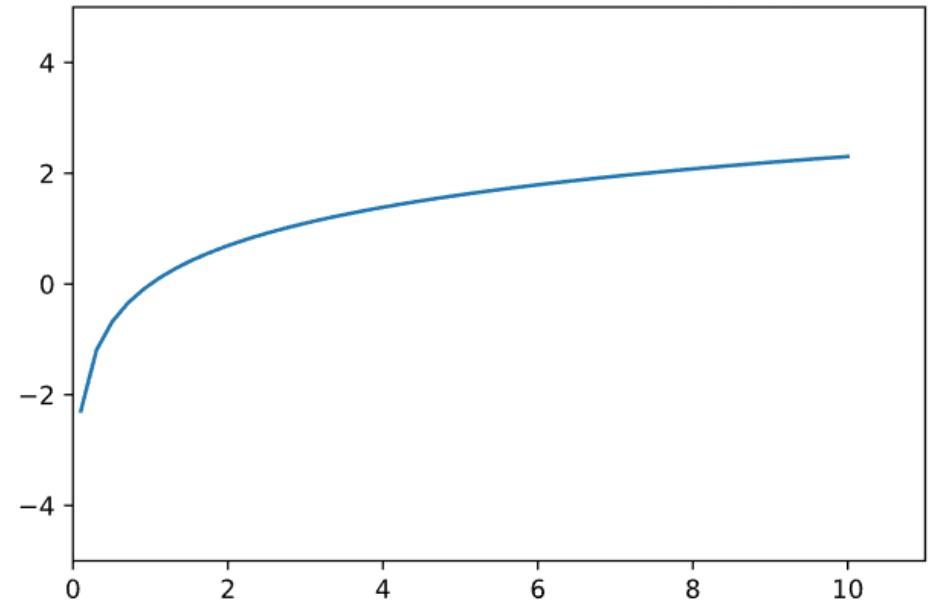
```
x = np.linspace(0.1, 10, 100)
y = np.log(x)
plt.xlabel('x values', labelpad=50)
plt.ylabel('log(x)')
plt.plot(x, y)
```



Axis

- `plt.axis(limits, ...)`
 - Set (or get) some axis properties
 - *limits*: a list with the axis limits to be set.
[xmin, xmax, ymin, ymax]
 - Use 'off' to hide all the axes

```
x = np.linspace(0.1, 10, 100)
y = np.log(x)
plt.axis([0, 11, -5, 5])
# plt.axis('off')
plt.plot(x, y)
```

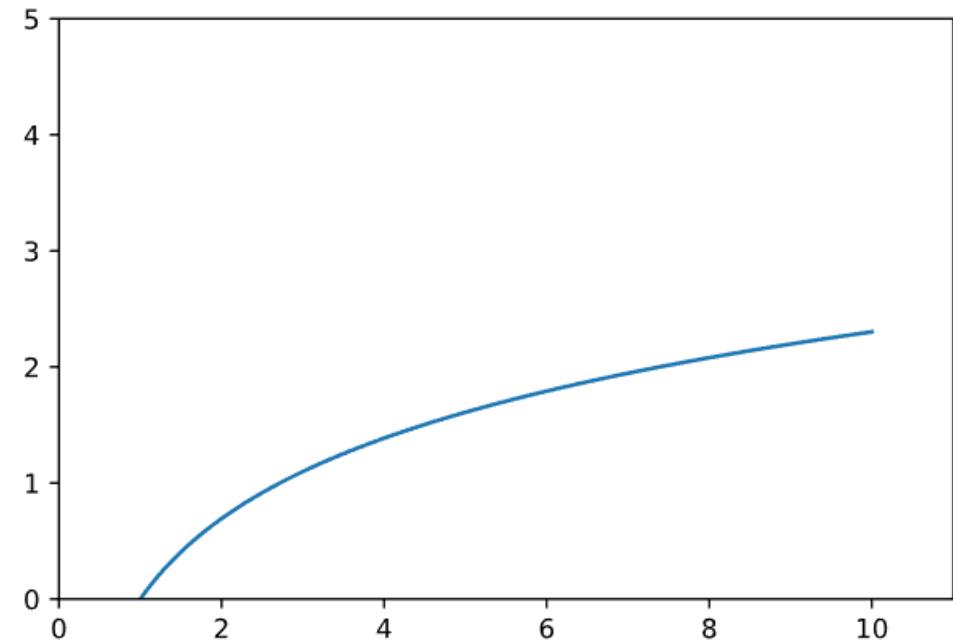


`plt.axis('off')`

Axis Limit

- `plt.xlim(left, right)`
- `plt.ylim(bottom, top)`
 - Set (or get) the x-limits (or y-limits) of the current axes
 - *left, right*: x-limits
 - *bottom, top*: y-limits

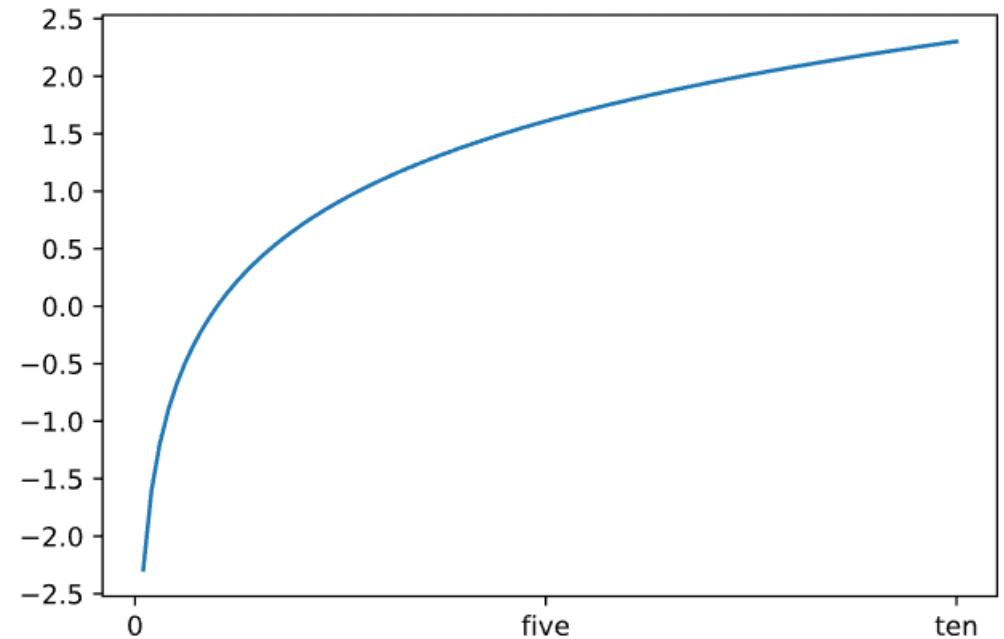
```
x = np.linspace(0.1, 10, 100)
plt.xlim(0, 11)
plt.ylim(top=5)
plt.plot(x, np.log(x))
```



Ticks

- `plt.xticks(ticks, [labels], ...)`
- `plt.yticks(ticks, [labels], ...)`
 - Set the current tick locations and labels of the x-axis (or y-axis)
 - `ticks`: A list of positions ticks should be placed
 - `labels`: A list of explicit labels to place at the given locations

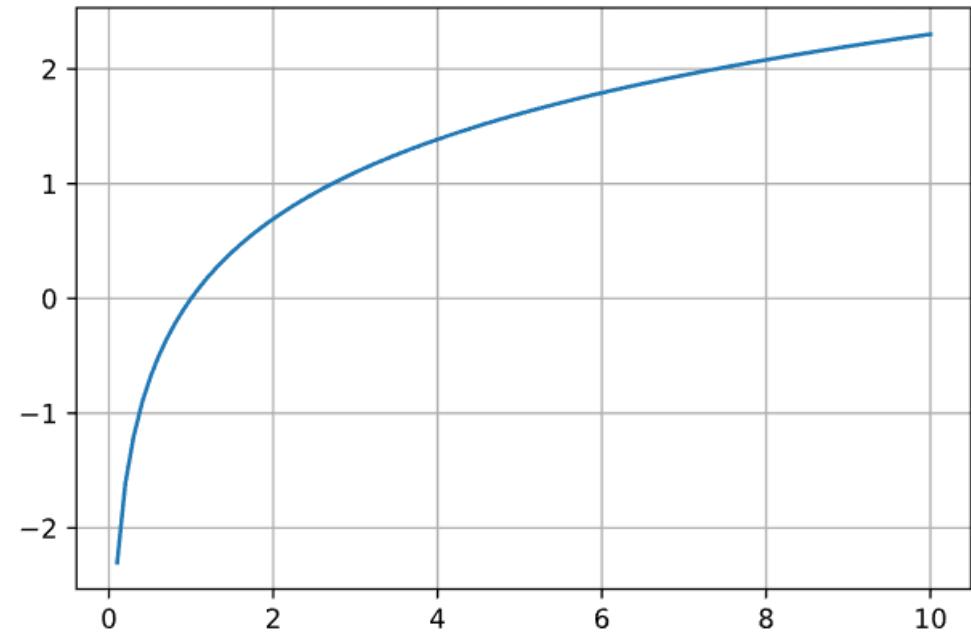
```
x = np.linspace(0.1, 10, 100)
plt.xticks(np.arange(0, 11, 5),
          ('0', 'five', 'ten'))
plt.yticks(np.arange(-3, 3, 0.5))
plt.plot(x, np.log(x))
```



Grids

- `plt.grid([b], [which], [axis], ...)`
 - Configure the grid lines
 - `b`: if `True`, show the grid lines (toggle if no argument given)
 - `which`: the grid lines to apply the changes on ('major', 'minor', or 'both')
 - `axis`: the axis to apply ('x', 'y', or 'both')

```
x = np.linspace(0.1, 10, 100)
plt.grid()
plt.plot(x, np.log(x))
```



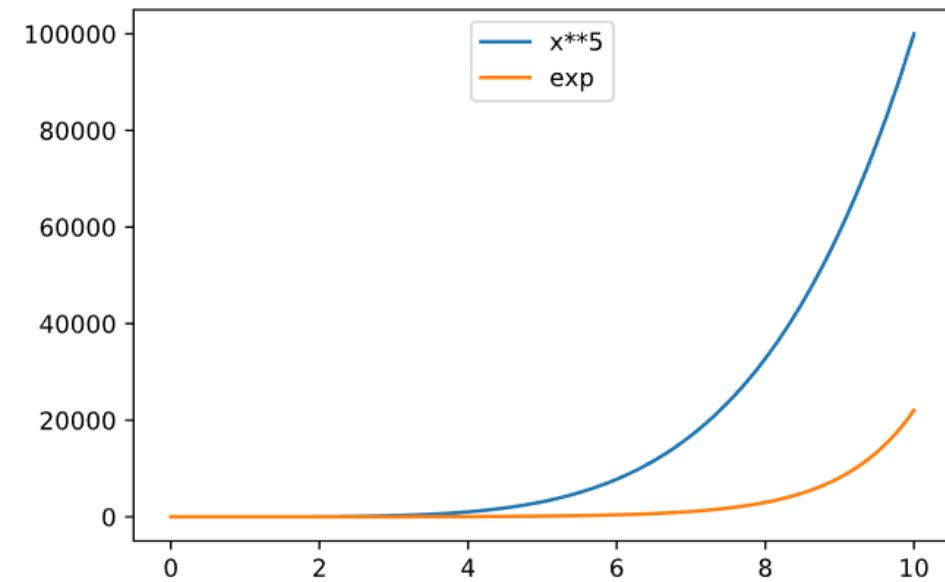
Legend

- ***plt.legend([loc], [ncol], ...)***

- Place a legend on the axes
- If no argument is given, automatically determine and show the legend
- *loc*: the location of the legend (default: 'best')
- *ncol*: the number of columns (default: 1)

```
x = np.linspace(0, 10, 100)
plt.plot(x, x**5, label='x**5')
plt.plot(x, np.exp(x), label='exp')
plt.legend(loc='upper center')
```

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

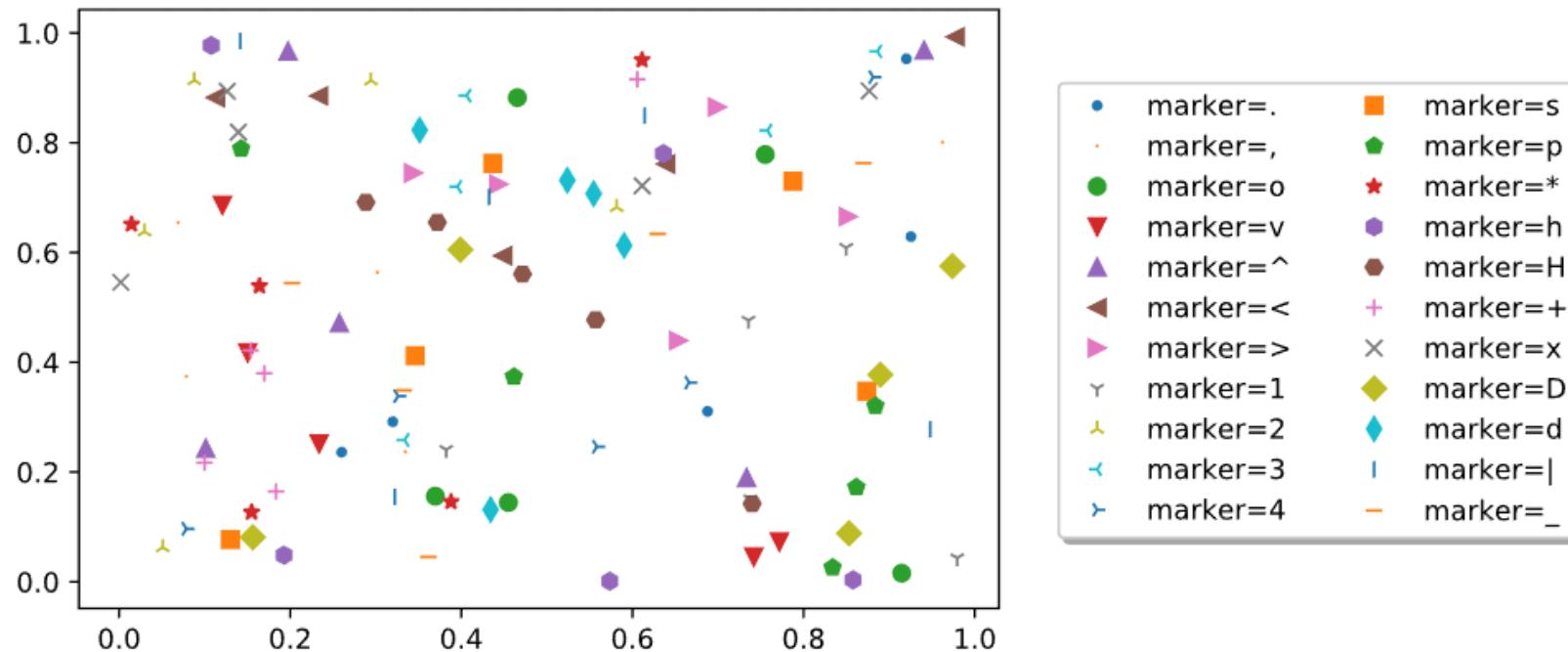


Legend: More Options

- `plt.legend(...)`
 - `bbox_to_anchor`: box that is used to position the legend in the arbitrary location with the form (x, y) or (x, y, width, height)
 - `frameon`: control whether the legend should be drawn on a frame (default:True)
 - `fancybox`: control whether round edges should be enabled (default:True)
 - `shadow`: control whether to draw a shadow behind the legend (default: False)
 - `fontsize`: control the font size of the legend
 - `title`: the legend's title
 - `markerscale`: the relative size of legend markers compared
 - `markerfirst`: if True, legend marker is placed to the left (default:True)
 - ...

Legend: Example

```
markers = '.^ov^<>1234sp*hH+xDd|_'
for m in markers:
    plt.plot(np.random.rand(5), np.random.rand(5), m, label=f'marker={m}')
plt.legend(ncol=2, bbox_to_anchor=(1.05, 0.9), shadow=True)
```



Texts

- **plt.title(label, [loc], [pad], ...)**
 - *label*: text to use for the title
 - *loc*: which title to set ('center', 'left', or 'right')
 - *pad*: the offset from the top of the axes
- **plt.text(x, y, s, [loc], [pad], ...)**
 - *x, y*: the position to place the text
 - *s*: the text to display
- **plt.annotate(s, [xy], [xytext], ...)**
 - *s*: the text to display
 - *xy*: the point to annotate
 - *xytext*: the position to place the text at

```
x = np.linspace(0.1, 10, 100)
plt.plot(x, np.log(x))
plt.title("The log(x) graph")
plt.text(6, -1, 'matplotlib rules!')
plt.annotate('log(1) = 0', xy=(1,0),
             xytext=(2,-0.1),
             arrowprops=dict(facecolor='pink',
                             shrink=0.05))
```

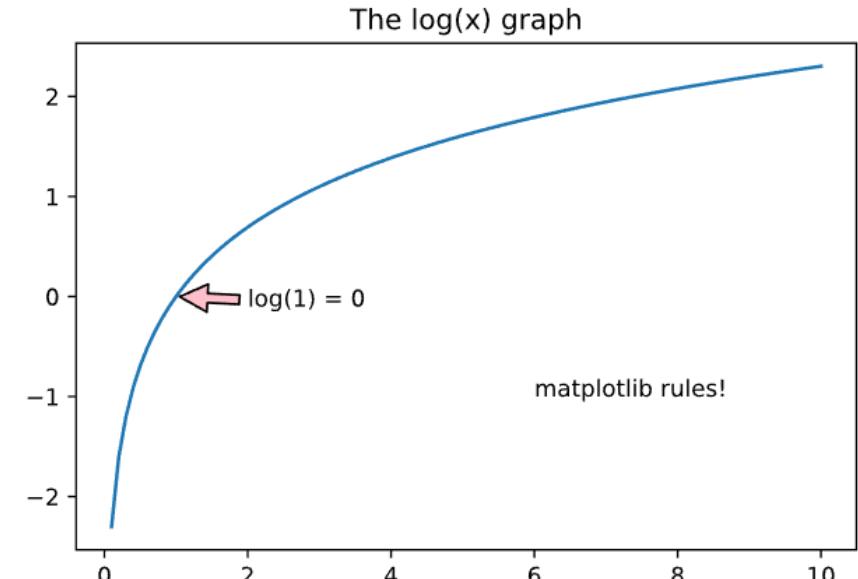
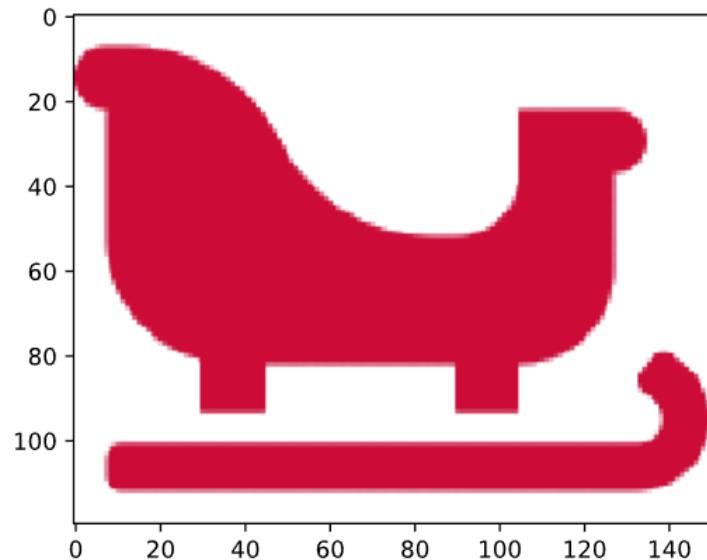
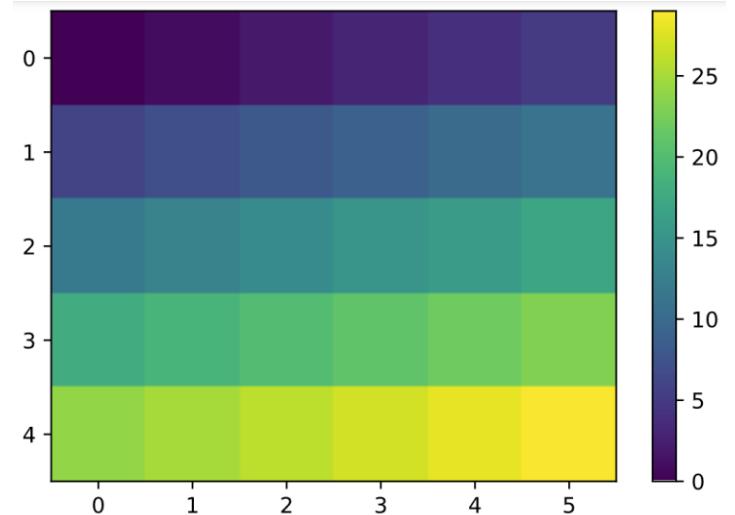


Image Submodule

```
import matplotlib.pyplot as plt  
import matplotlib.image as mpimg  
  
img = mpimg.imread('sleds.png')  
plt.imshow(img)
```



```
a = np.arange(0,30).reshape(5,6)  
plt.imshow(a)  
plt.colorbar()
```



Saving Plots

- `plt.savefig(fname, [format], [dpi], [transparent], ...)`
 - Save the current figure
 - `fname`: file name to save. If format is not given, the output format is inferred from the extension of the fname
 - `format`: the file format (e.g., 'png', 'pdf', 'svg', 'jpg', ...)
 - `dpi`: the resolution in dots per inch
 - `transparent`: if True, make the plot transparent (default: False)

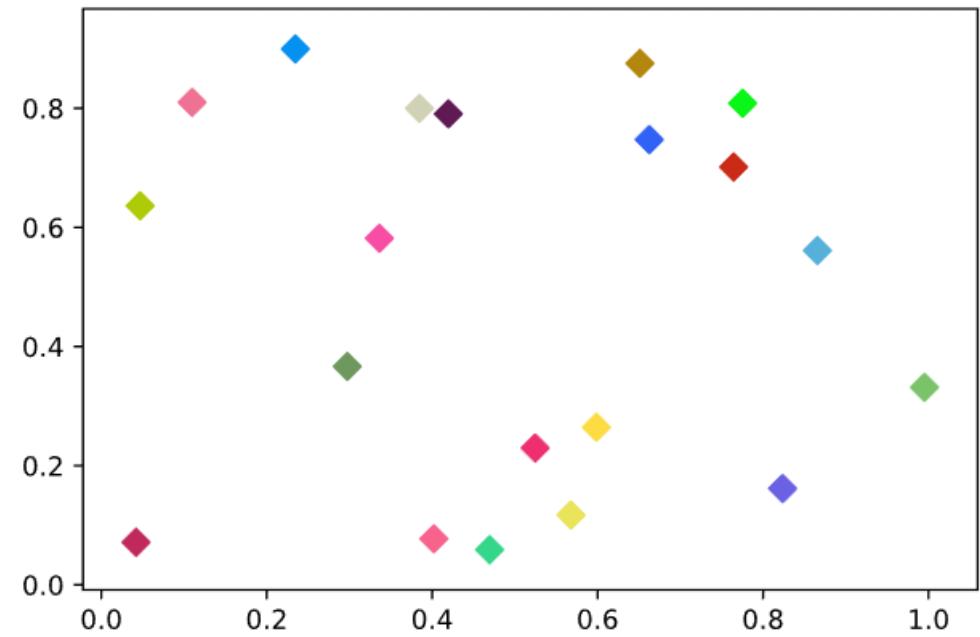
```
x = np.linspace(0.1, 10, 100)
plt.plot(x, np.log(x))
plt.savefig('log.png', dpi=300, transparent=True)
```

Scatter Plot

scatter()

- `plt.scatter(x, y, [s], [c], [marker], [alpha], ...)`
 - A scatter plot of y vs. x with varying marker size and/or color
 - `x, y`: data positions
 - `s, c`: the marker size (in points **2) and its color
 - `marker`: the marker style
 - `alpha`: the alpha blending value
(0: transparent ~ 1: opaque)

```
x = np.random.rand(20)
y = np.random.rand(20)
colors=[f'#{np.random.randint(256**3):06x}'
        for _ in range(20)]
plt.scatter(x, y, s=50, c=colors, marker='D')
```



plot() vs. scatter()

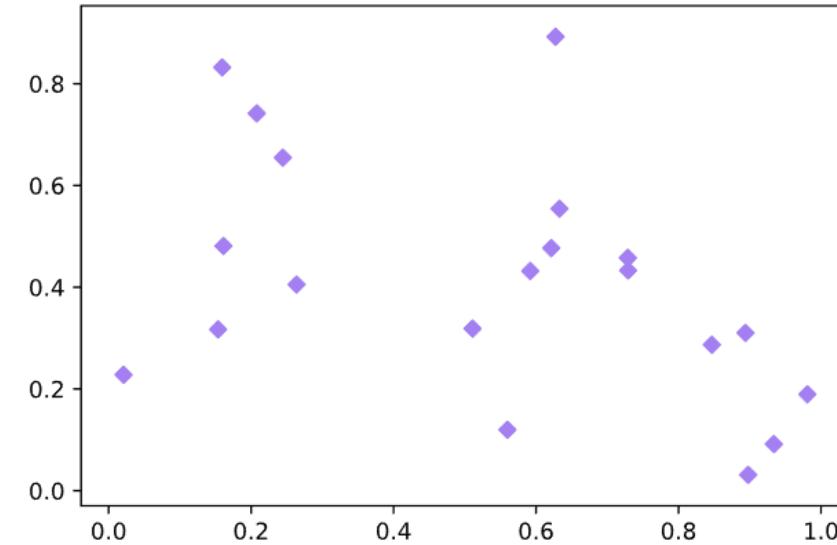
- **plt.plot()**

```
np.random.seed(1010)  
N = 20  
x = np.random.rand(N)  
y = np.random.rand(N)  
plt.plot(x,y,'D',c='#a37ff1',markersize=5)
```

- **plt.scatter()**

```
np.random.seed(1010)  
N = 20  
x = np.random.rand(N)  
y = np.random.rand(N)  
plt.scatter(x,y,s=25,c='#a37ff1',marker='D')
```

- Both can be used for simple cases
→ same result
- **scatter()** is more comfortable to configure each data point
- **plot()** allows to plot the lines connecting data points

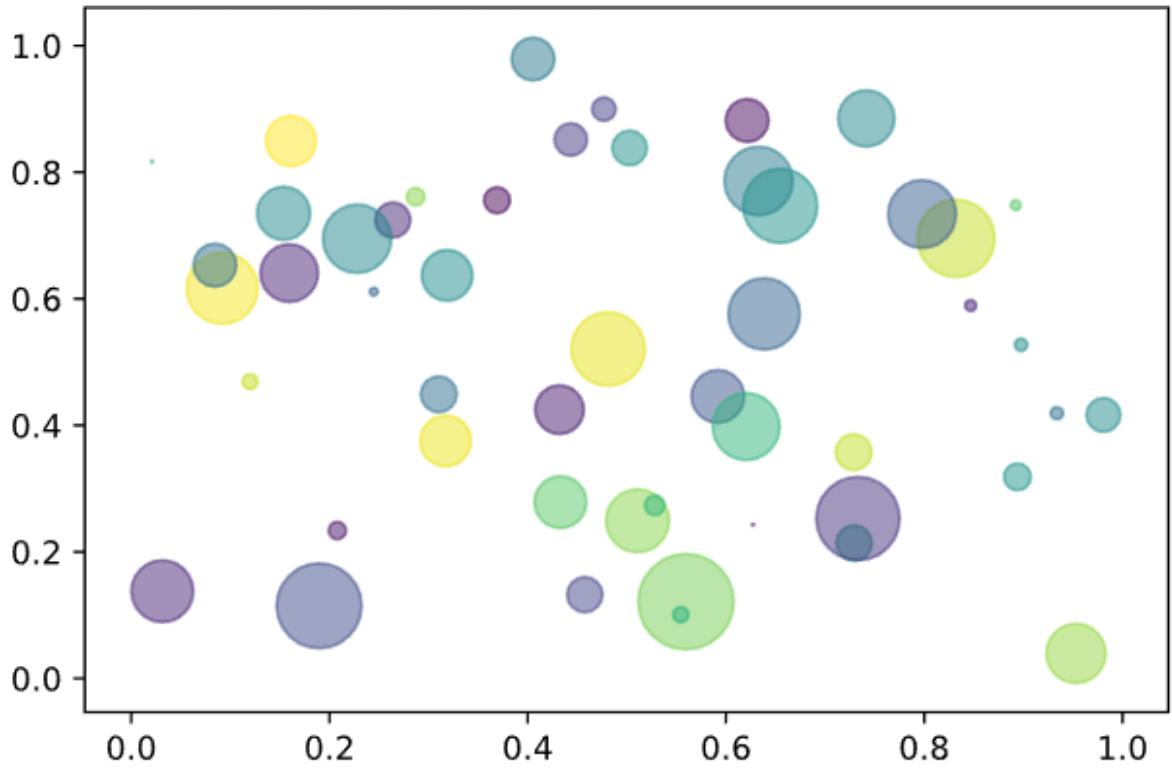


Scatter Plot with Different Marker Sizes

```
np.random.seed(20200101)

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area=(30 * np.random.rand(N))**2

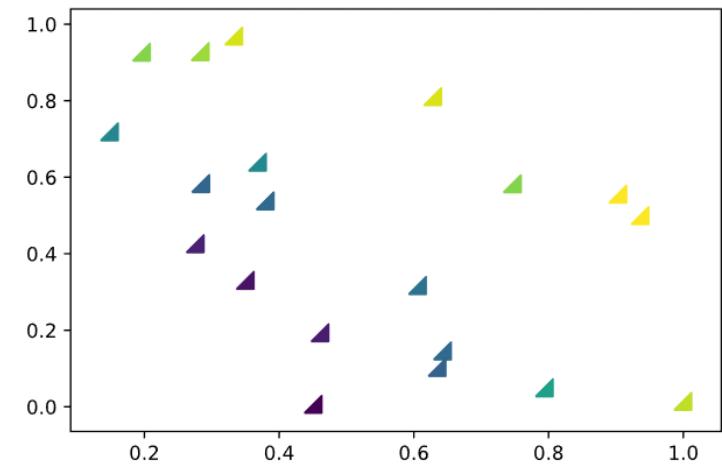
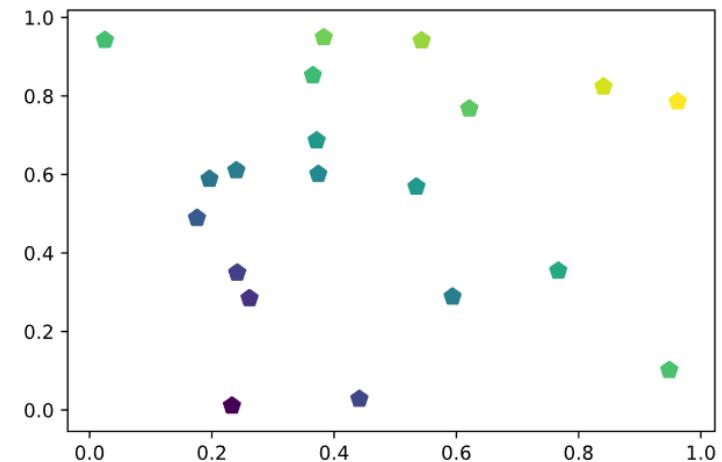
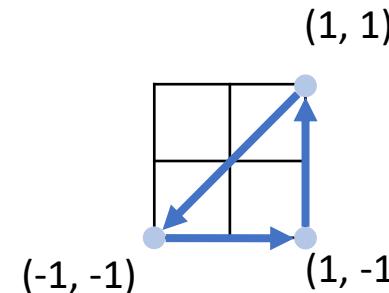
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
```



Scatter Plot with Polygon Symbols

- Marker = (numsides, style, angle)
 - *numsides*: number of sides
 - *style*: regular polygon(0), star-like symbol(1), asterisk(2), circle(3)
 - *angle*: angle of rotation
- User-defined marker:
 - A list of (x, y) describing a symbol with center = (0, 0)

```
x = np.random.rand(20)
y = np.random.rand(20)
z = np.sqrt(x**2 + y**2)
plt.scatter(x, y, s=80, c=z, marker=(5,0))
v = np.array([[-1,-1],[1,-1],[1,1],[-1,-1]])
plt.scatter(x, y, s=80, c=z, marker=v)
```

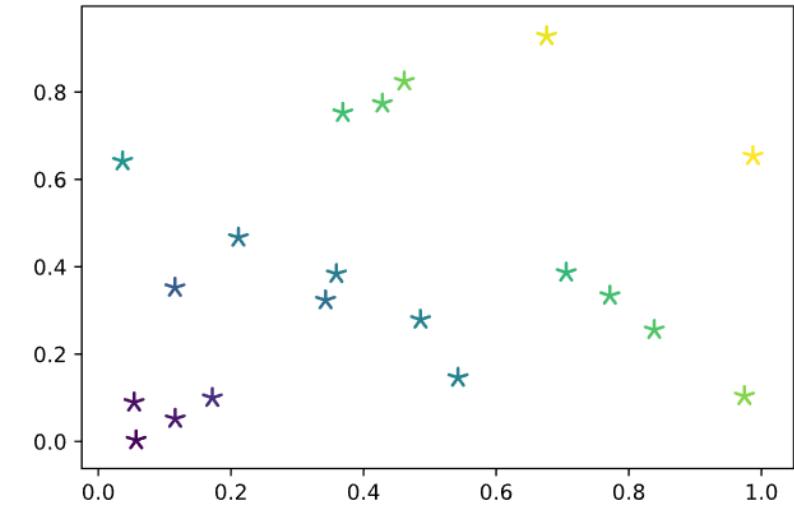
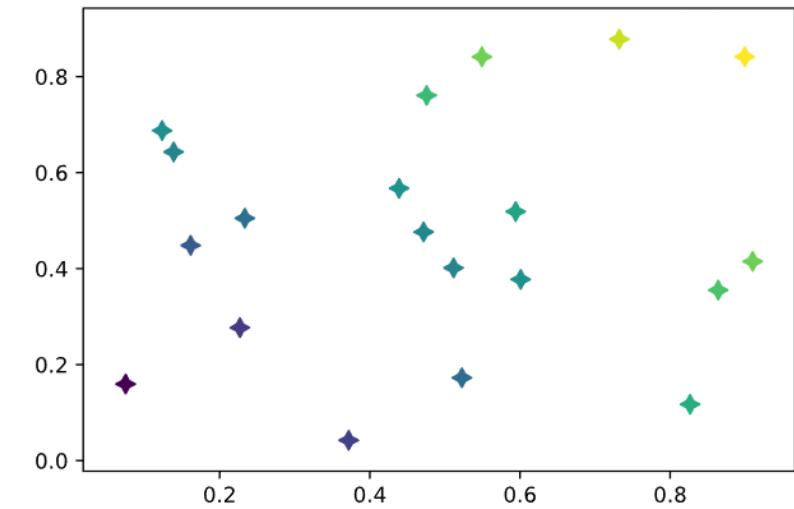


Scatter Plot with Stars and Asterisks

```
x = np.random.rand(20)
y = np.random.rand(20)
z = np.sqrt(x**2 + y**2)

# with 4-side star
plt.scatter(x, y, s=80, c=z, marker=(4,1))

# with 5-side asterisk
plt.scatter(x, y, s=80, c=z, marker=(5,2))
```

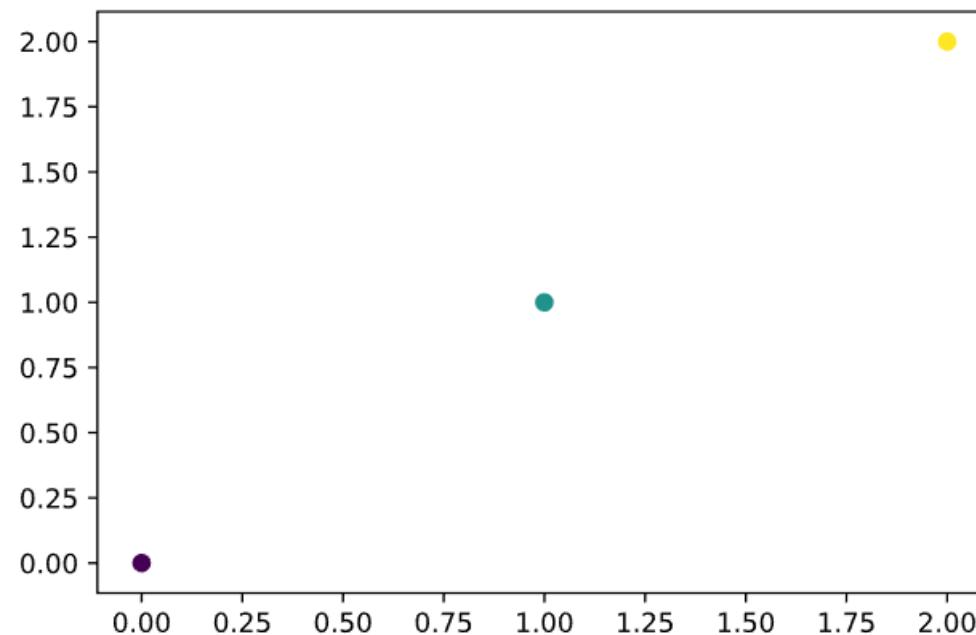


Default Color Map

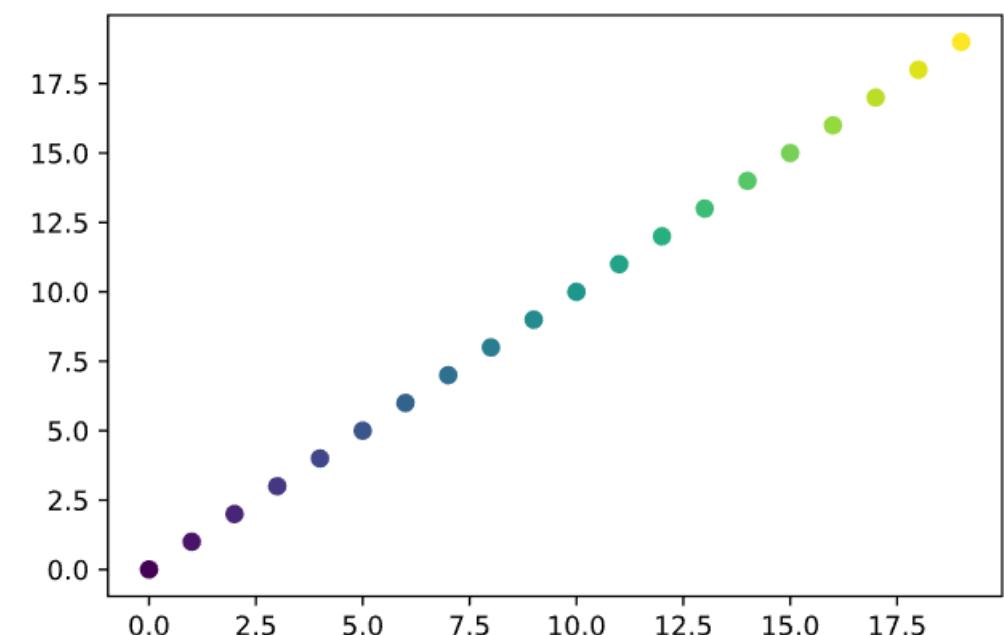
- 'viridis':



```
x = np.arange(3)  
plt.scatter(x, x, c=x)
```

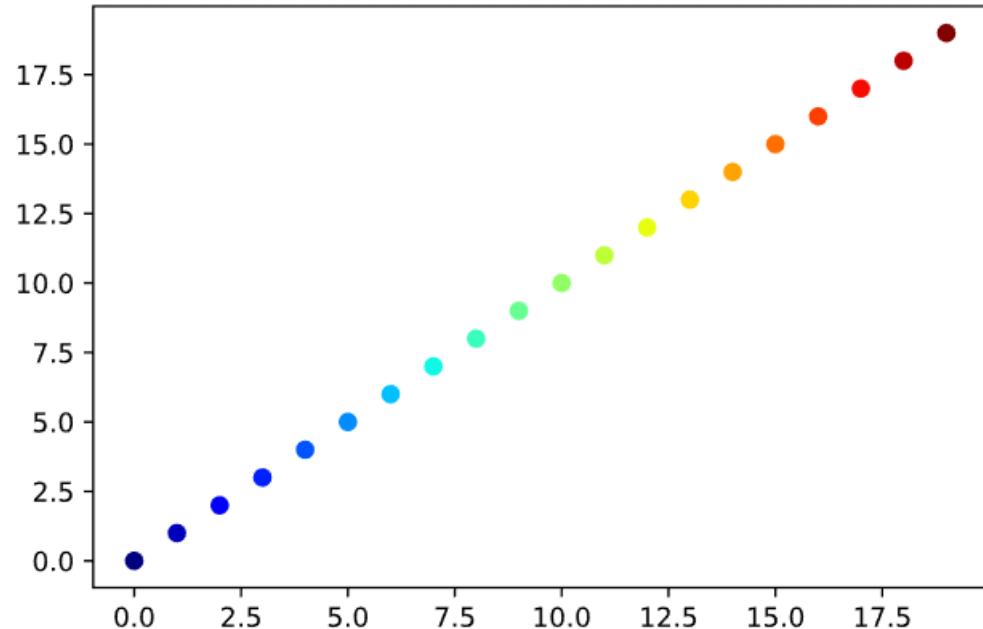


```
x = np.arange(20)  
plt.scatter(x, x, c=x)
```

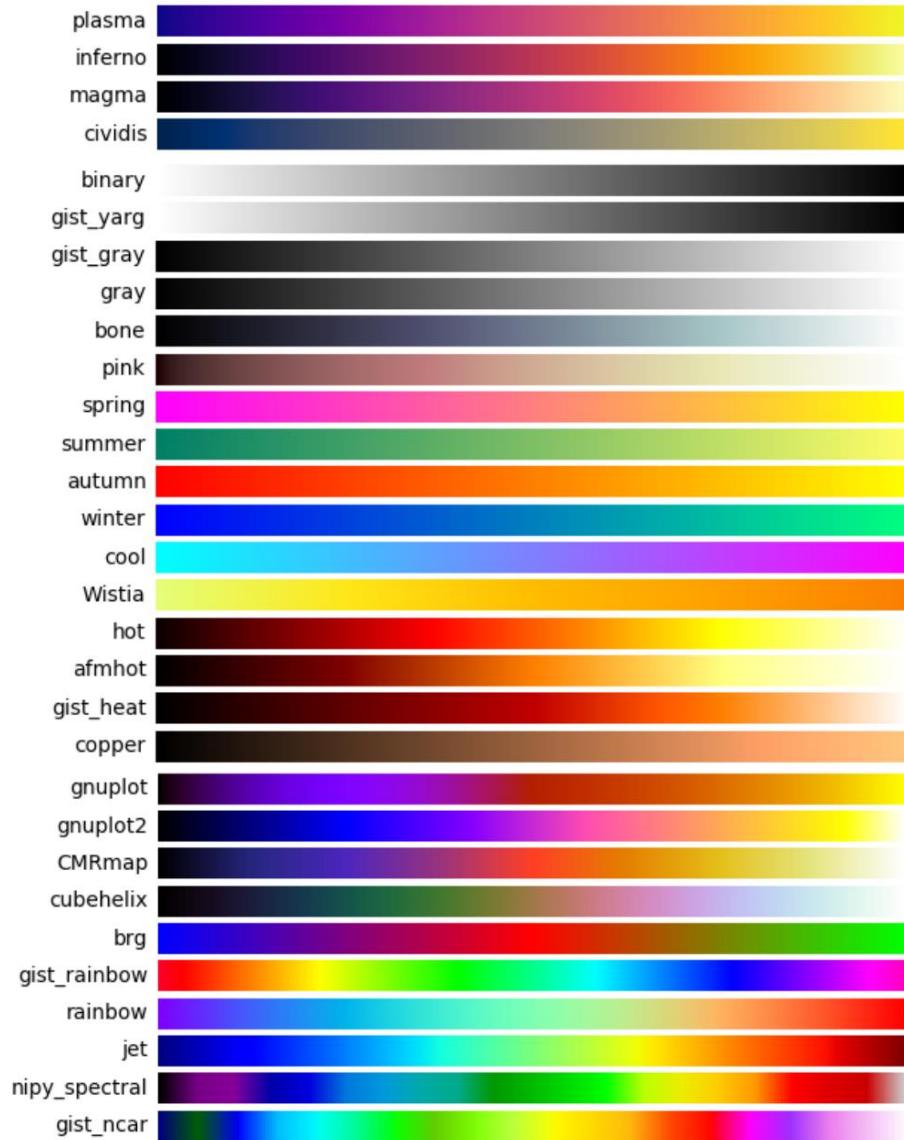


Changing a Color Map

```
plt.rc('image', cmap='jet')
x = np.arange(20)
plt.scatter(x, x, c=x)
```



For other colormaps, visit
<https://matplotlib.org/tutorials/colors/colormaps.html>

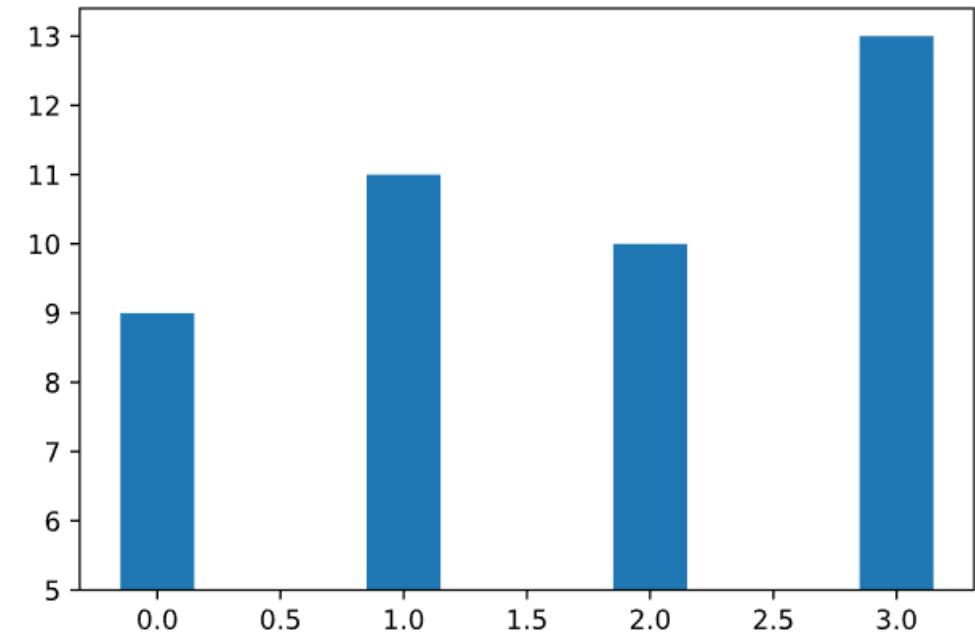


Bar Chart

bar()

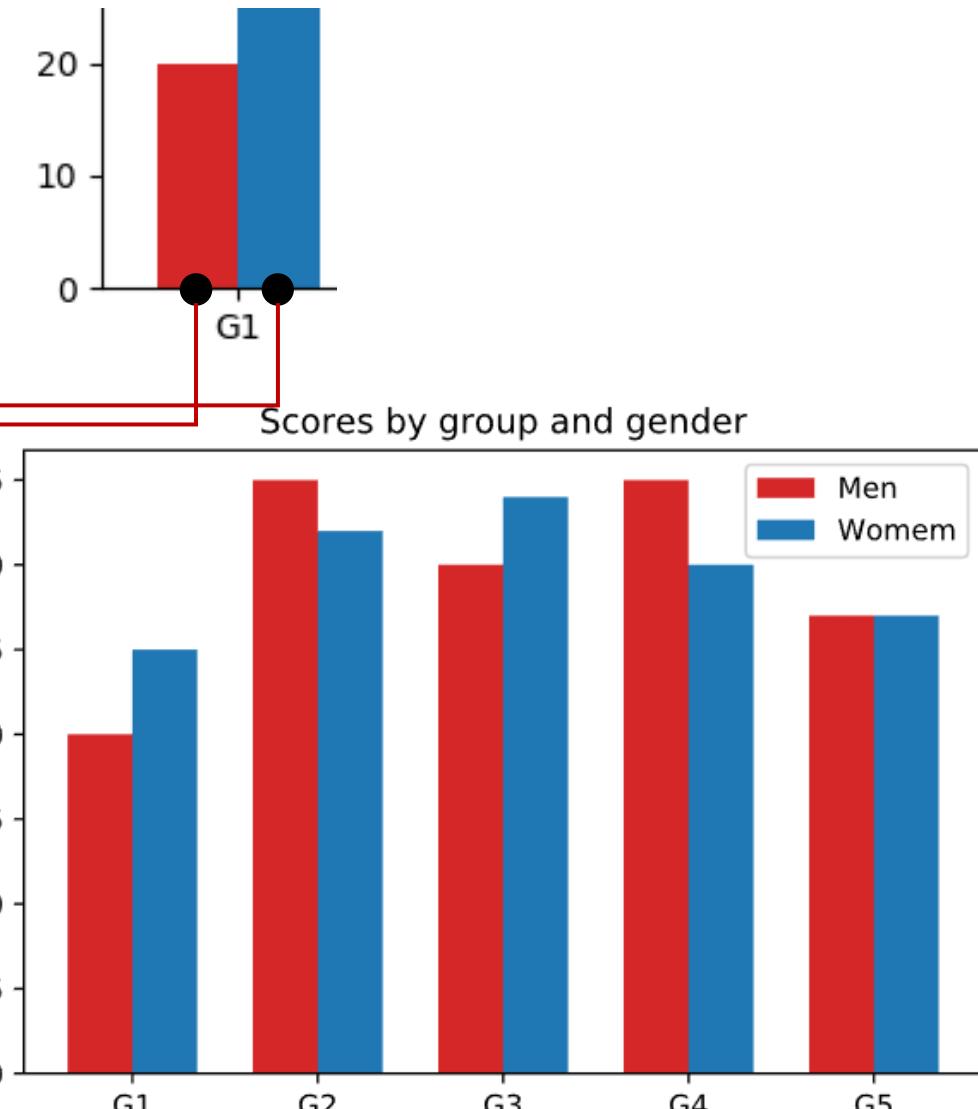
- `plt.bar(x, height, [width], [bottom], [align], ...)`
 - Make a bar plot
 - `x, height`: data points
 - `width`: the width(s) of the bars (default: 0.8)
 - `bottom`: the y coordinate(s) of the bars bases
 - `align`: alignment of the bars to the x coordinates -- 'center' (default) or 'edge'

```
x = np.arange(4)
y = [4, 6, 5, 8]
plt.bar(x, y, width=0.3, bottom=5)
```



Side-by-Side Bar Chart

```
N = 5  
  
m = (20, 35, 30, 35, 27)  
w = (25, 32, 34, 30, 27)  
x = np.arange(N)  
width=0.35  
  
plt.bar(x-width/2, m, width, color='#d62728')  
plt.bar(x+width/2, w, width)  
  
plt.xticks(x, ('G1','G2','G3','G4','G5'))  
plt.ylabel('Scores')  
plt.title('Scores by group and gender')  
plt.legend(('Men', 'Womem'))
```



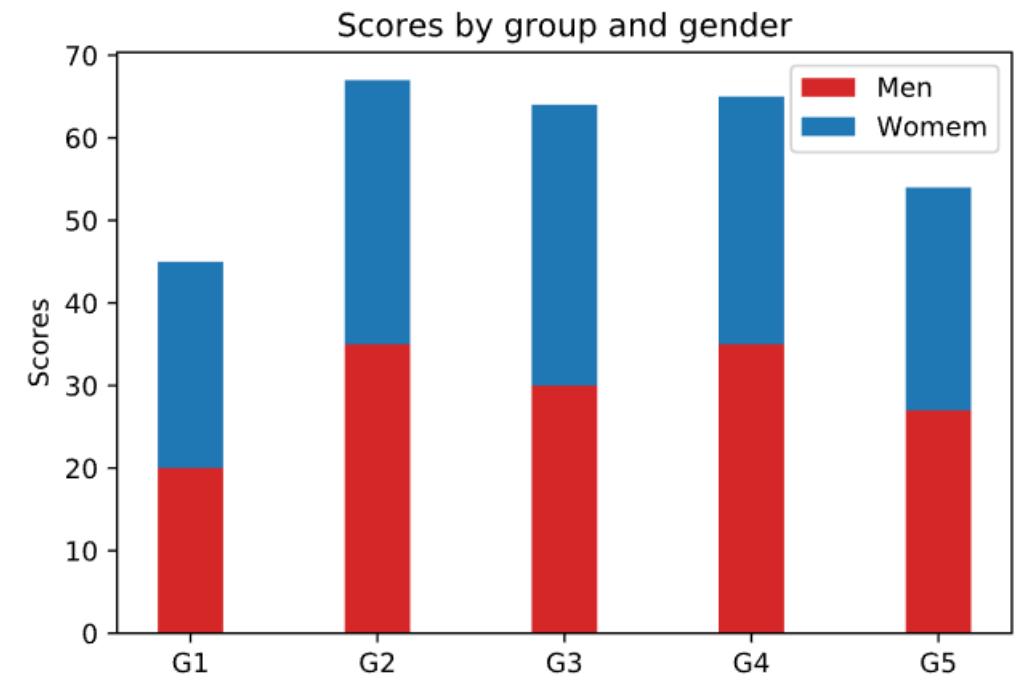
Stacked Bar Chart

```
N = 5

m = (20, 35, 30, 35, 27)
w = (25, 32, 34, 30, 27)
x = np.arange(N)
width=0.35

plt.bar(x, m, width, color='#d62728')
plt.bar(x, w, width, bottom=m)

plt.xticks(x, ('G1','G2','G3','G4','G5'))
plt.ylabel('Scores')
plt.title('Scores by group and gender')
plt.legend(['Men', 'Womem'])
```



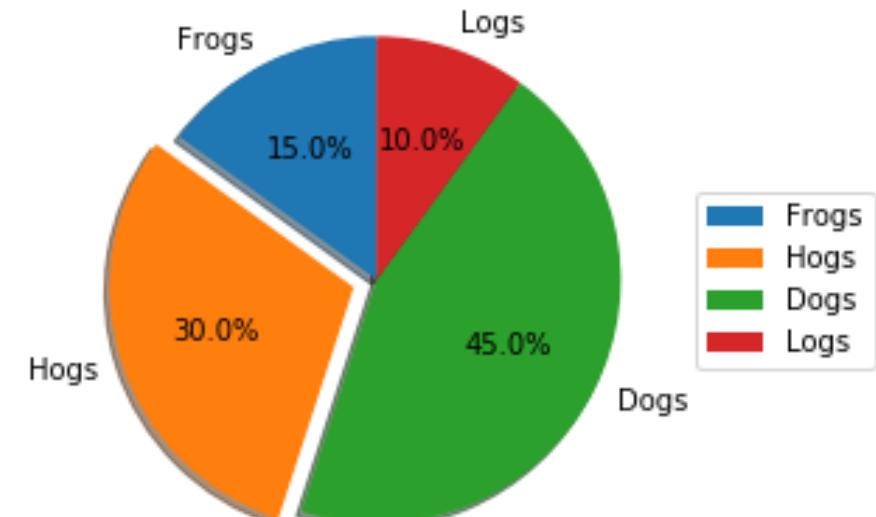
Pie Chart

pie()

- `plt.pie(x, [labels], [colors], [explode], [startangle], [autopct], ...)`

- Plot a pie chart
- `x`: the wedge sizes
- `labels`: a sequence of labels
- `colors`: a sequence of colors
- `explode`: the fraction of the radius with which to offset each wedge
- `startangle`: start pie chart with this degree
- `autopct`: label wedges with numeric value

```
labels = ['Frogs', 'Hogs', 'Dogs', 'Logs']
sizes = [15, 30, 45, 10]
ex = (0, 0.1, 0, 0)
plt.pie(sizes, explode=ex, labels=labels,
        autopct='%.1f%%', shadow=True, startangle=90)
plt.legend(loc='center right',
           bbox_to_anchor=(1.4, 0.5))
```

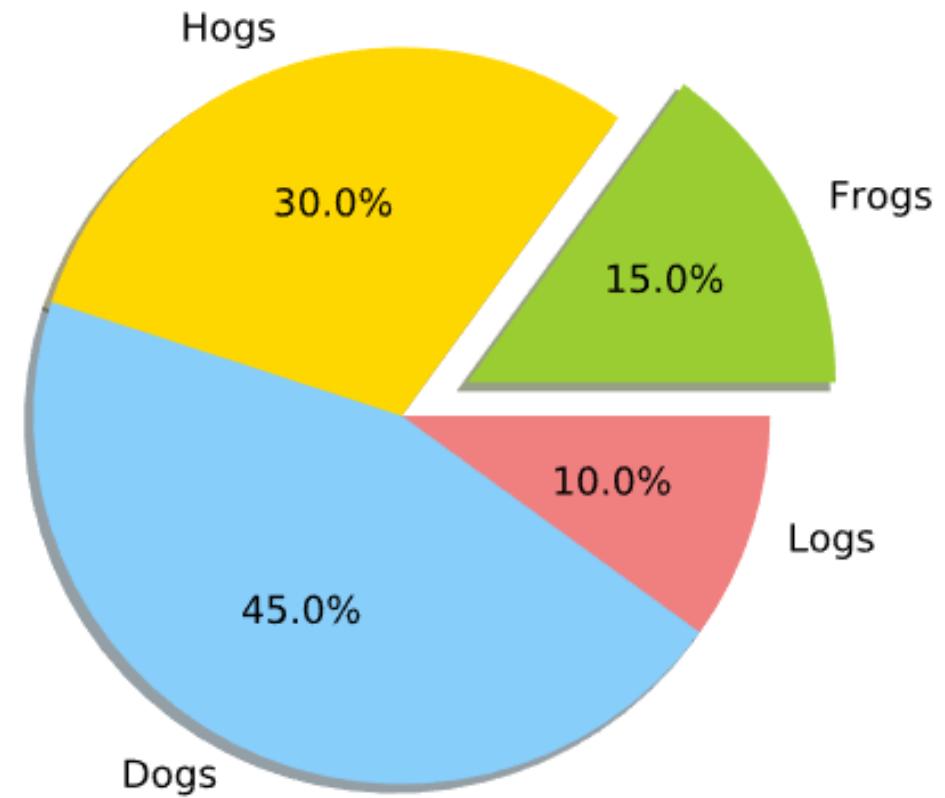


Using Specific Colors

```
labels = ['Frogs', 'Hogs', 'Dogs', 'Logs']
sizes = [15, 30, 45, 10]
ex = (0.2, 0, 0, 0)
c = ['yellowgreen', 'gold',
      'lightskyblue', 'lightcoral']

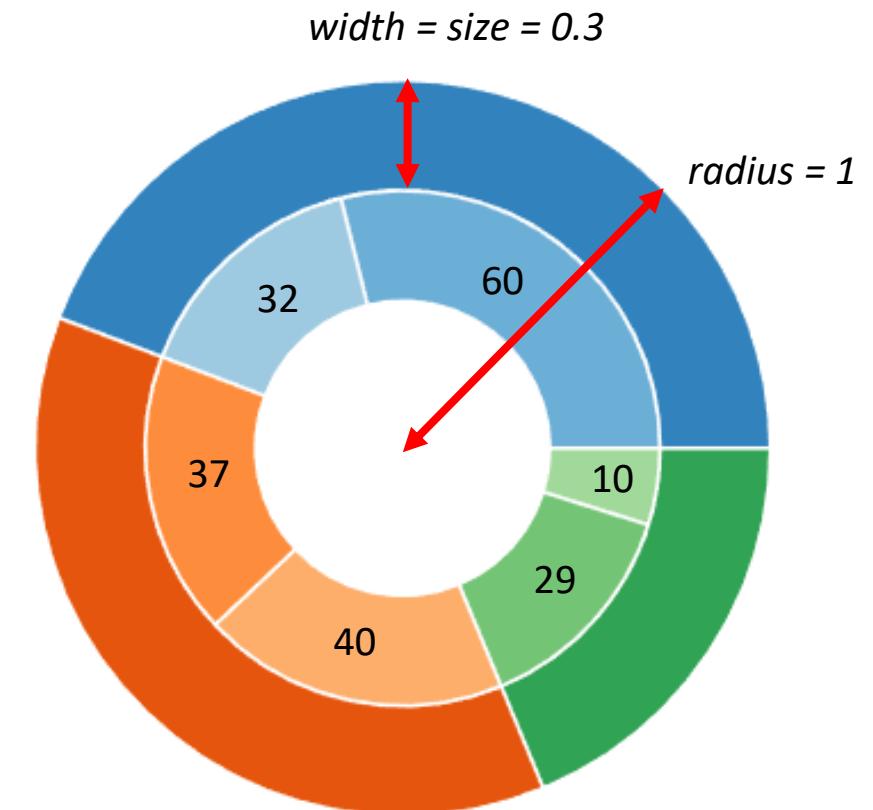
plt.pie(sizes, explode=ex, labels=labels,
        colors=c, autopct='%.1f%%',
        shadow=True, startangle=0)

plt.axis('equal')
```



Nested Pie Chart

```
size = 0.3  
vals = np.array([[60.,32.],[37.,40.],[29.,10.]])  
  
cmap = plt.get_cmap('tab20c')  
outer_colors = cmap(np.arange(3)*4)  
inner_colors = cmap(np.array([1,2,5,6,9,10]))  
  
plt.pie(vals.sum(axis=1), radius=1,  
         colors=outer_colors,  
         wedgeprops={'width':size, 'edgecolor':'w'})  
plt.pie(vals.flatten(), radius=1-size,  
         colors=inner_colors,  
         wedgeprops={'width':size, 'edgecolor':'w'})  
plt.axis('equal')
```



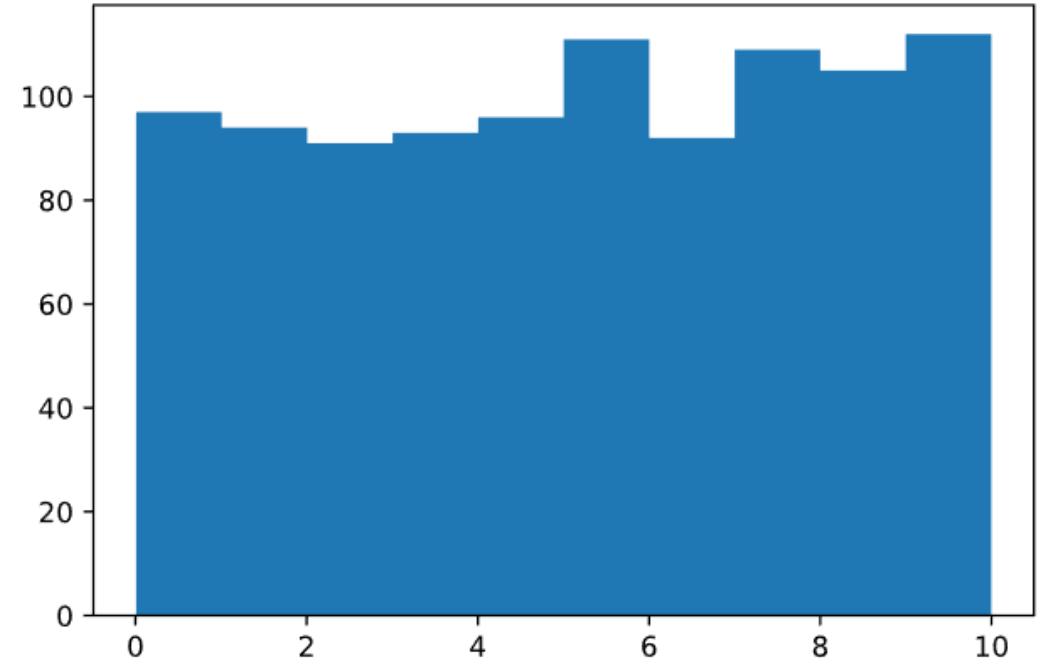
Histogram

hist()

- `plt.hist(x, [bins], [range], [cumulative], [bottom], ...)`

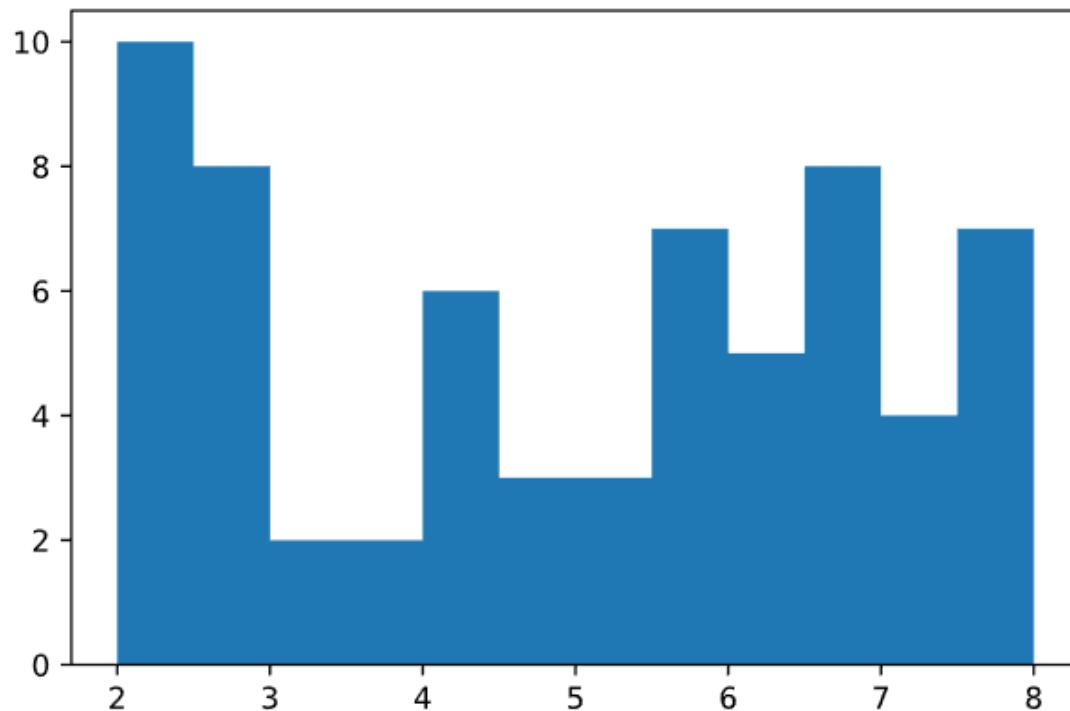
- Plot a histogram
- `x`: input values
- `bins`: number of bins (default: 10)
- `range`: the lower and upper range of the bins
- `cumulative`: if `True`, shows the cumulative sum
- `bottom`: the baseline of each bin

```
x = np.random.uniform(0., 10., 1000)  
plt.hist(x)
```



Range and Bins

```
x = np.random.uniform(0., 10., 100)  
plt.hist(x, range=[2., 8.], bins=12)
```



Split [2.0, 8.0] into 12 bins:

bin 0 has the count of $2 \leq x < 2.5$
bin 1 has the count of $2.5 \leq x < 3.0$

bin i has the count of $2 + 0.5i \leq x < 2.5 + 0.5i$

bin 11 has the count of $7.5 \leq x < 8.0$

Box Plot

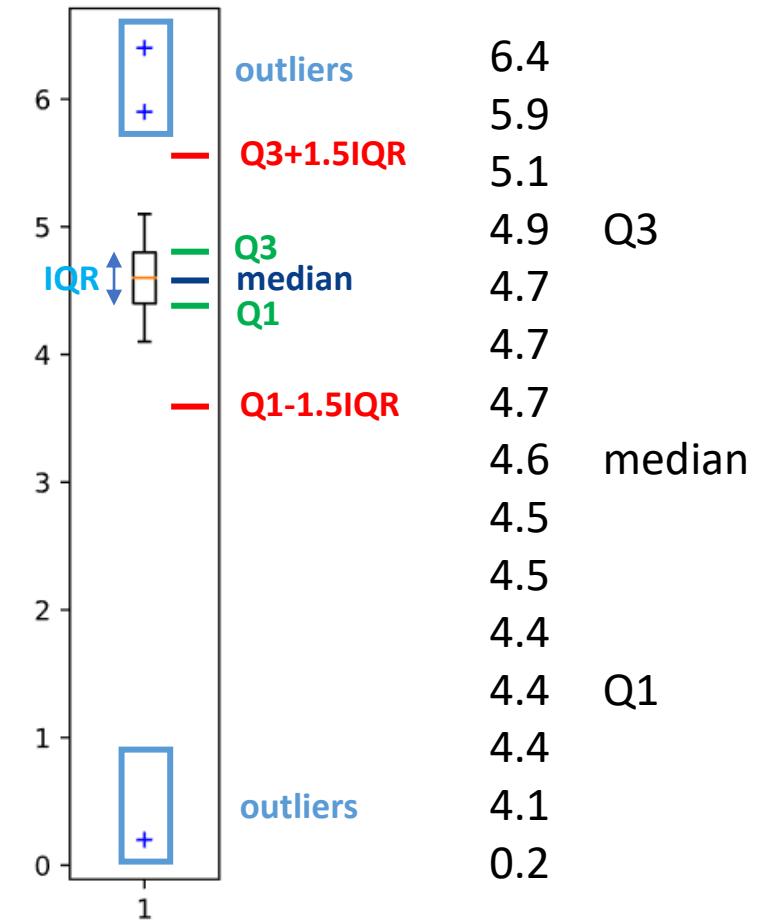
boxplot()

- `plt.boxplot(x, [notch], [sym], [vert], [whis], ...)`

- Make a box and whisker plot
- `x`: input data
- `notch`: If `True`, produce a notched box plot (default: `False`)
- `sym`: the symbol for outliers
- `vert`: If `True`, make the boxes vertical (default: `True`)
- `whis`: the reach of the whiskers (default: 1.5)

(cf.) IQR = InterQuartile Range

```
x = [0.2, 4.1, 4.4, 4.4, 4.4, 4.5, 4.5, 4.6,
      4.7, 4.7, 4.7, 4.9, 5.1, 5.9, 6.4]
plt.boxplot(x, sym='b+')
```



$$\text{IQR} = Q_3 - Q_1 = 4.9 - 4.4 = 0.5$$

$$Q_1 - 1.5\text{IQR} = 4.4 - 1.5 \cdot 0.5 = 3.65$$

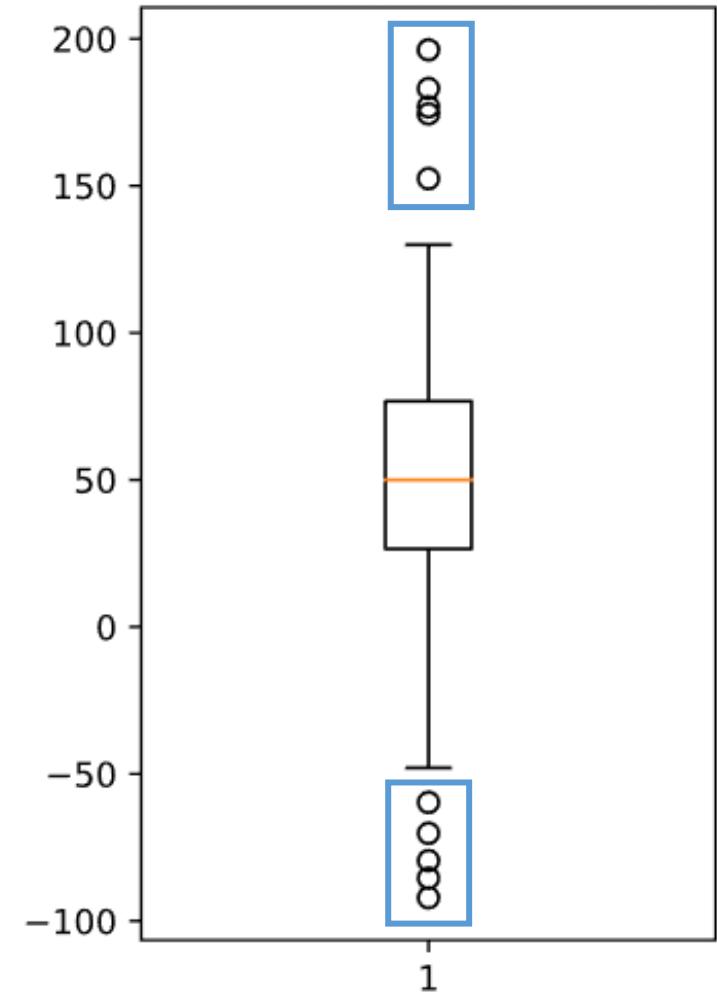
$$Q_3 + 1.5\text{IQR} = 4.9 + 1.5 \cdot 0.5 = 5.65$$

Box Plot Example

```
spread = np.random.rand(50)*100
center = np.ones(25)*50
high = np.random.rand(10)*100+100
low = np.random.rand(10)*(-100)

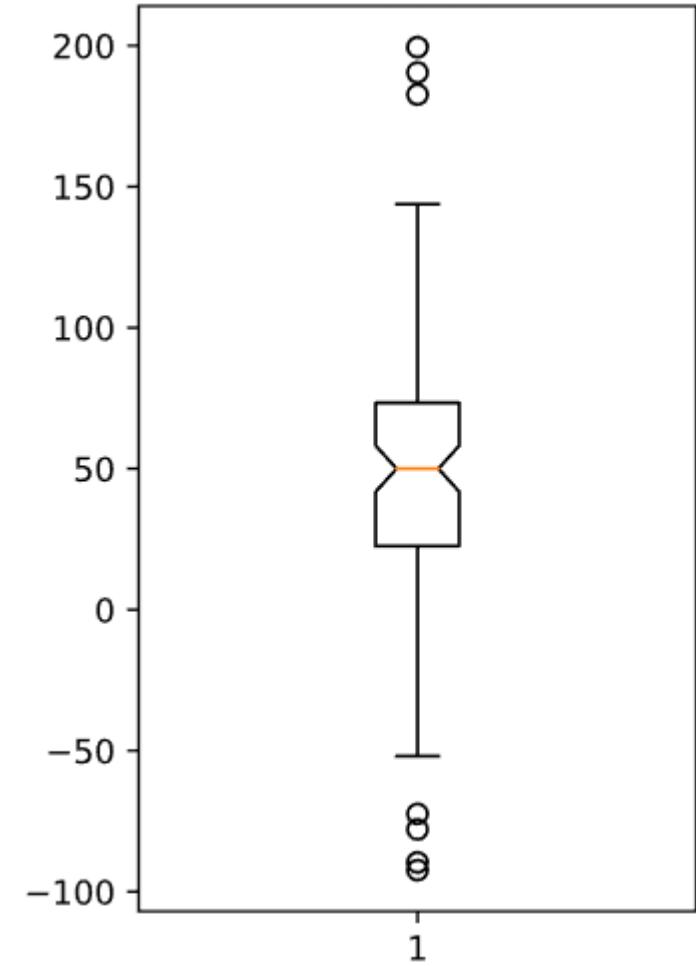
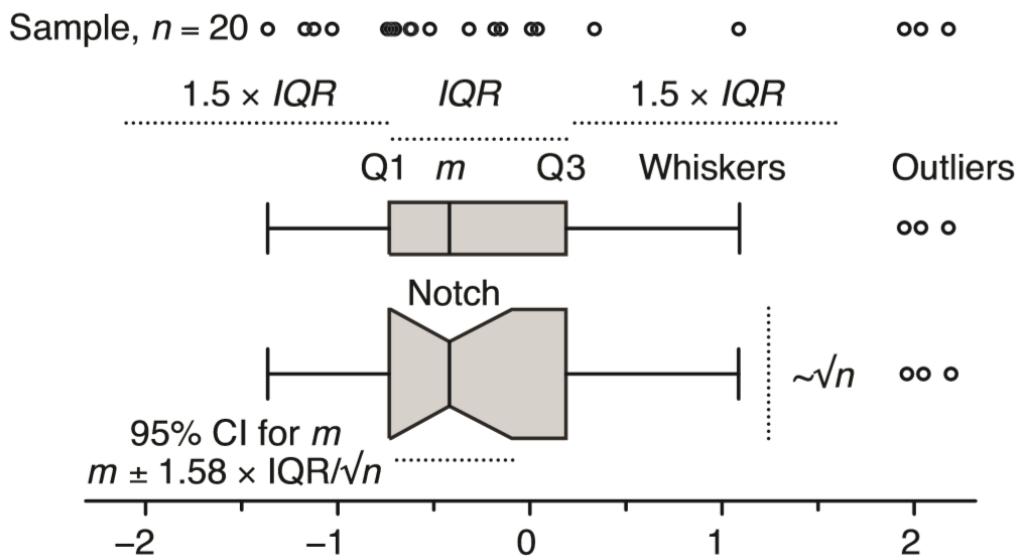
data = np.concatenate((spread, center, high, low), 0)
plt.figure(figsize=(3, 5)) # fig size in inches
plt.boxplot(data)
```

- spread: 50 numbers in $[0, 100]$
- center: 25 numbers with the value 50.0
- high: 10 numbers in $[100, 200]$
- low: 10 numbers in $(-100, 0]$



Notched Box Plot

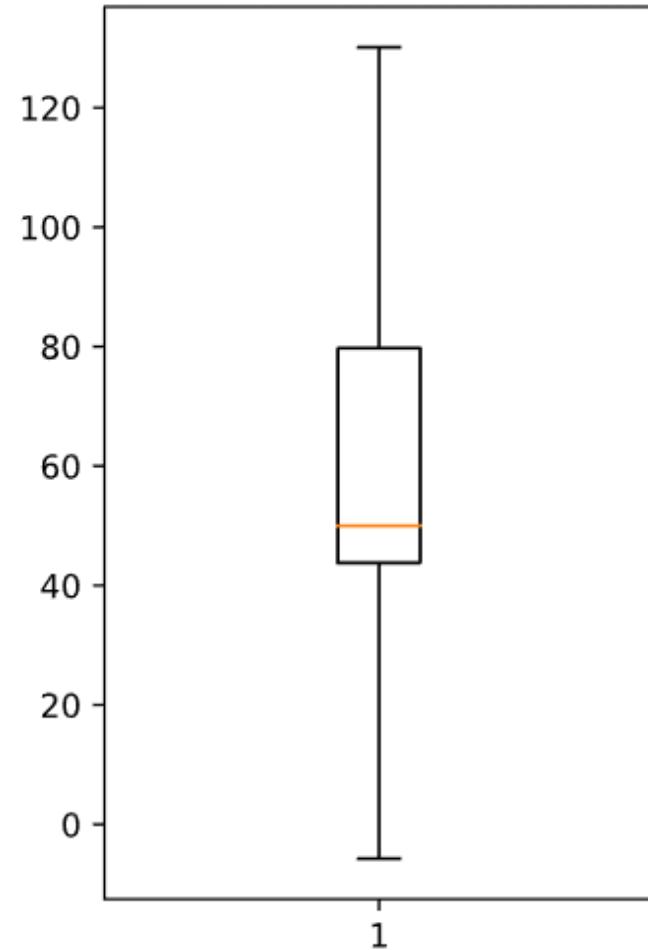
```
spread = np.random.rand(50)*100
center = np.ones(25)*50
high = np.random.rand(10)*100+100
low = np.random.rand(10)*(-100)
data = np.concatenate((spread, center, high, low), 0)
plt.figure(figsize=(3, 5)) # fig size in inches
plt.boxplot(data, notch=True)
```



Removing Outliers

```
spread = np.random.rand(50)*100
center = np.ones(25)*50
high = np.random.rand(10)*100+100
low = np.random.rand(10)*(-100)
data = np.concatenate((spread, center, high, low), 0)

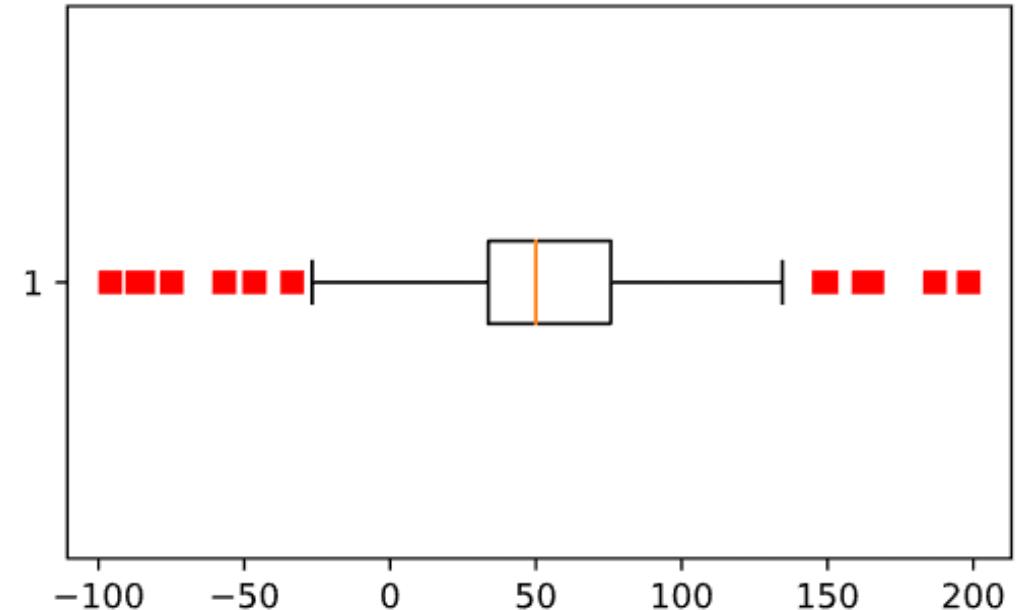
plt.figure(figsize=(3, 5)) # fig size in inches
plt.boxplot(data, sym=' ')
```



Horizontal Box Plot

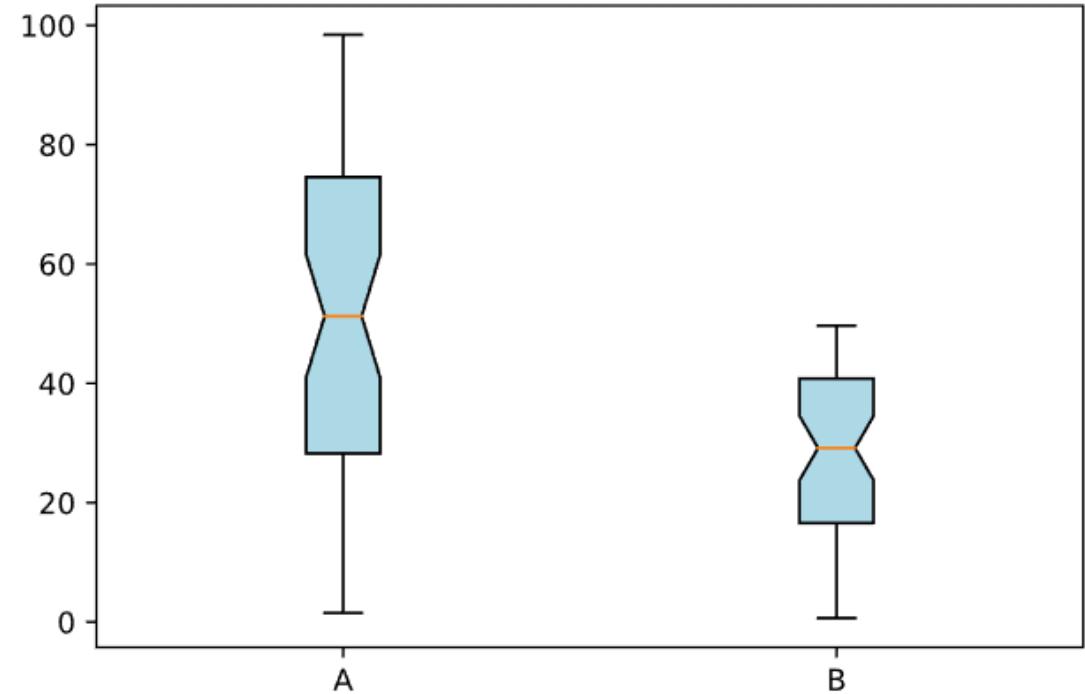
```
spread = np.random.rand(50)*100
center = np.ones(25)*50
high = np.random.rand(10)*100+100
low = np.random.rand(10)*(-100)
data = np.concatenate((spread, center,
    high, low), 0)

plt.figure(figsize=(5, 3))
plt.boxplot(data, vert=False, sym='rs')
```



Multiple Box Plots

```
p1 = np.random.rand(50)*100  
p2 = np.random.rand(50)*50  
data = [p1, p2]  
  
plt.boxplot(data, notch=True,  
            patch_artist=True,  
            boxprops={'facecolor':'lightblue'},  
            labels=['A', 'B'])
```



Subplots

figure()

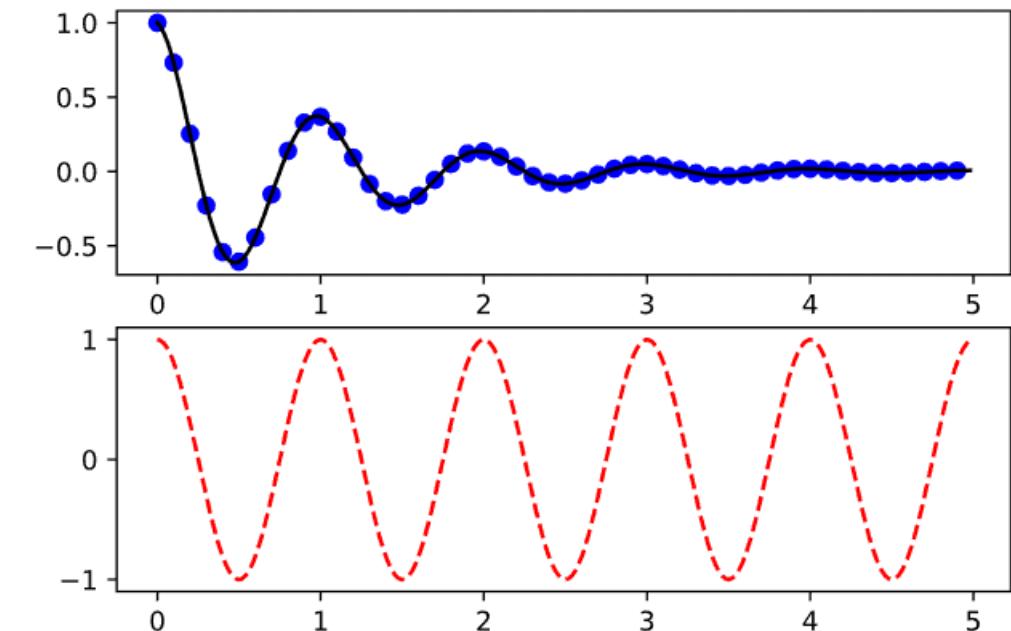
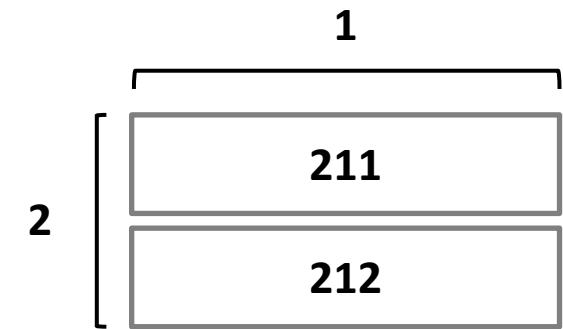
- `plt.figure([num], [figsize], [facecolor], [edgecolor], [frameon], ...)`
 - Create a new figure
 - `num`: If not provided, a new figure will be created with the figure number incremented. If provided, a figure with this id is created
 - `figsize`: (width, height) in inches
 - `facecolor`: the background color
 - `edgecolor`: the border color
 - `frameon`: If True, draw the figure frame (default:True)

subplot()

- **plt.subplot([pos], ...)**

- Add a subplot to the current figure
- *pos*: a three digit integer denoting the number of rows, the number of columns, and the index of the subplot
- Returns the axes of the subplot

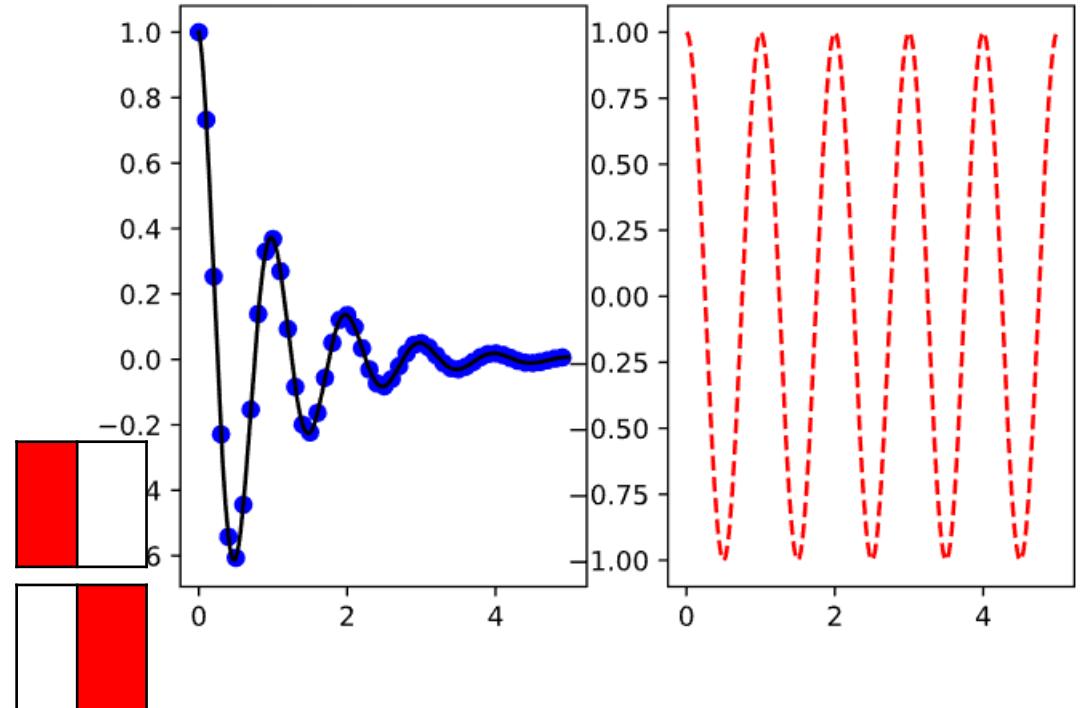
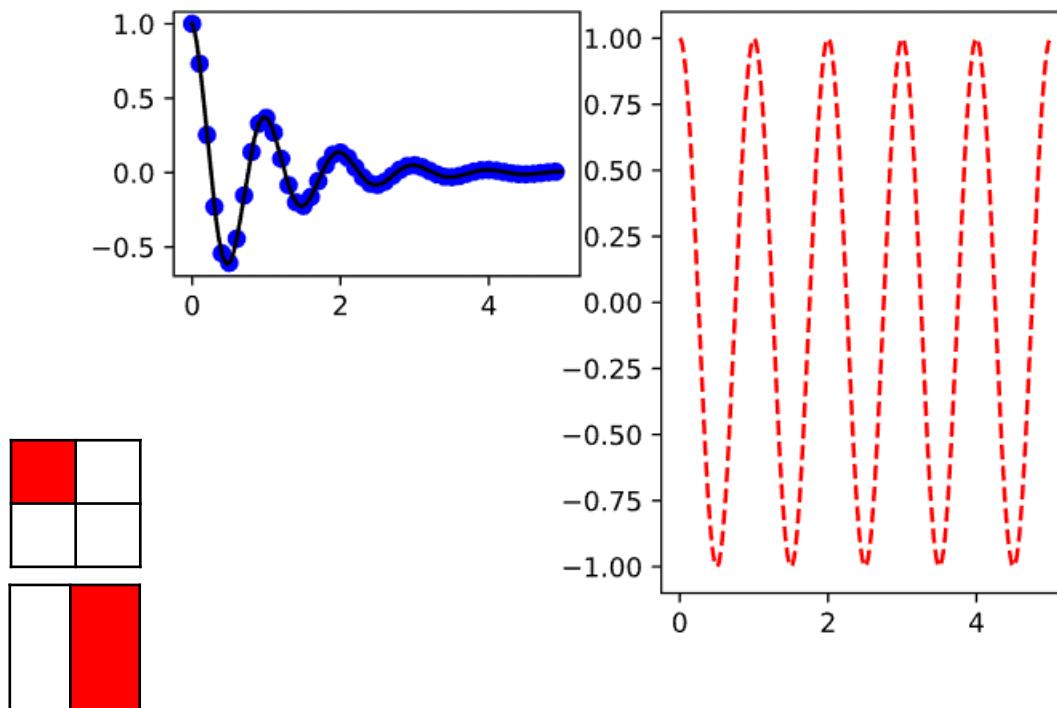
```
def f(t):  
    return np.exp(-t)*np.cos(2*np.pi*t)  
t1 = np.arange(0.0, 5.0, 0.1)  
t2 = np.arange(0.0, 5.0, 0.02)  
# plt.figure(1)  
plt.subplot(211)  
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')  
plt.subplot(212)  
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
```



Subplot Examples

```
plt.subplot(221)  
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')  
plt.subplot(122)  
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
```

```
plt.subplot(121)  
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')  
plt.subplot(122)  
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
```

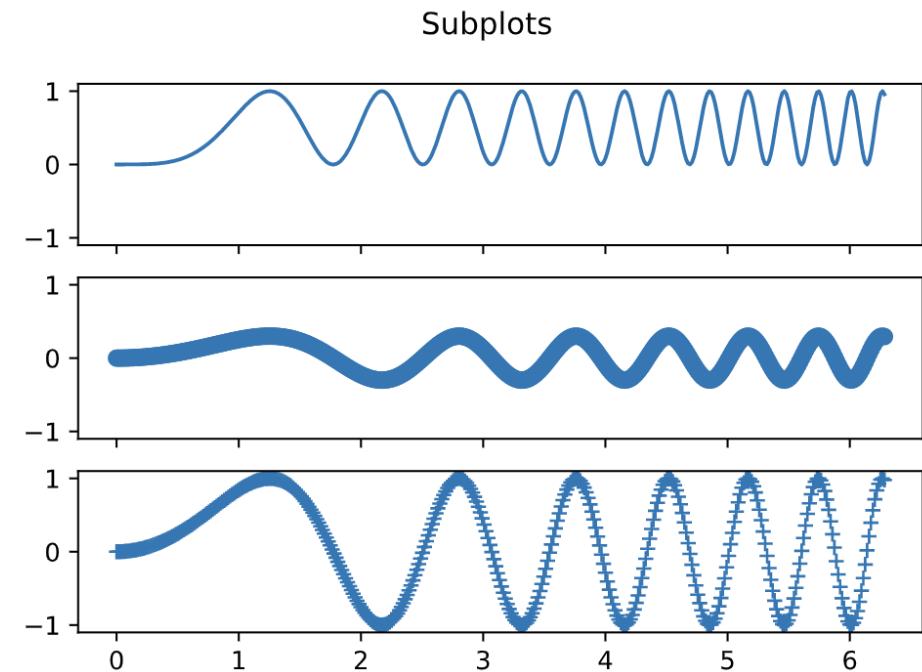


subplots()

- `plt.subplots([nrows], [ncols], [sharex], [sharey], ...)`
 - Create a figure and a set of subplots
 - `nrows, ncols`: number of rows/columns of the subplot grid
 - `sharex, sharey`: if True, x- or y- axis will be shared among all subplots (default: False)
 - Return figure and ax (or array of axes)

```
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)

fig, axs = plt.subplots(3, 1,
                      sharex=True, sharey=True)
fig.suptitle('Subplots')
axs[0].plot(x, y**2)
axs[1].plot(x, 0.3*y, 'o')
axs[2].plot(x, y, '+')
```

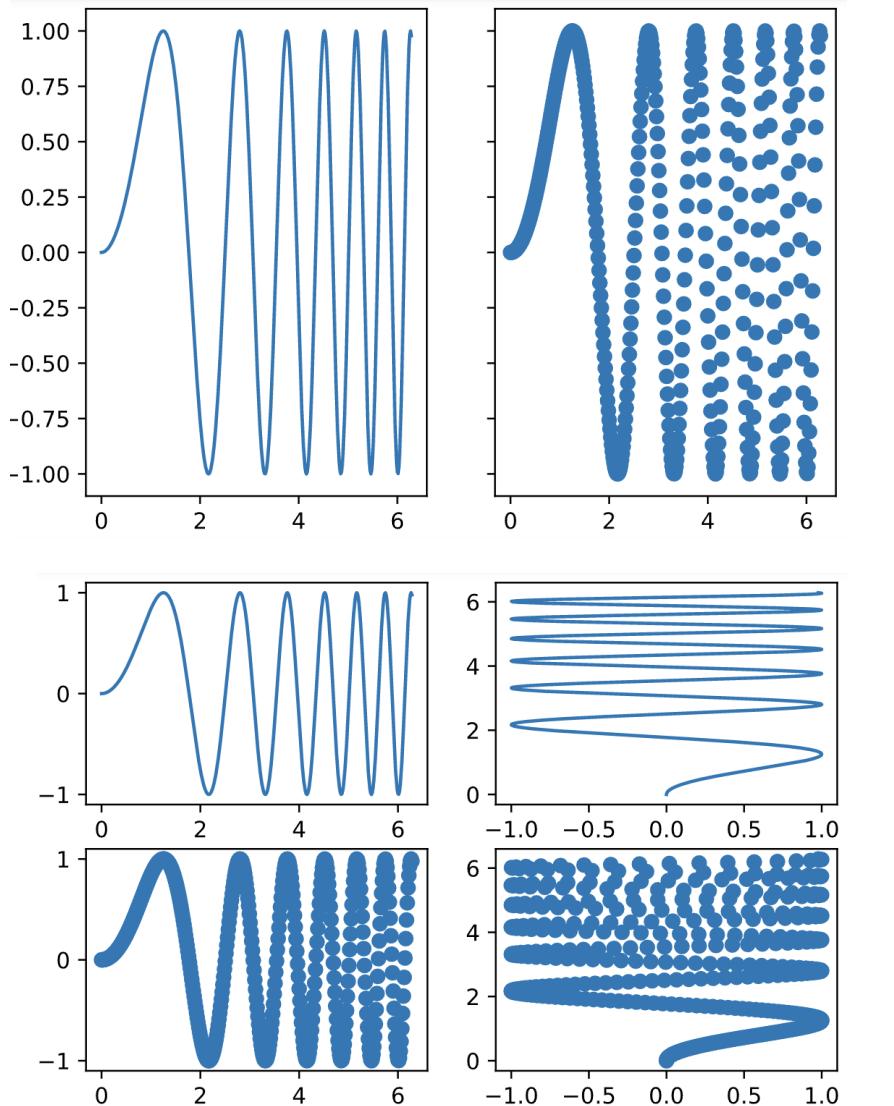


subplots(): More Examples

```
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)

# Create two subplots
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
ax1.plot(x, y)
ax2.scatter(x, y)

# Create four subplots
_, axs = plt.subplots(2, 2)    # axs: 2D ndarray
axs[0, 0].plot(x, y)
axs[0, 1].plot(y, x)
axs[1, 0].scatter(x, y)
axs[1, 1].scatter(y, x)
```



Seaborn

Seaborn

- Python visualization library based on matplotlib
 - More attractive and informative statistical graphics
 - Wrapper of matplotlib which improved the default styles
 - Even if plotting using only matplotlib.pyplot, importing seaborn will make nicer plots
- Dependency
 - Data structures: NumPy, Pandas
 - Statistical routines: SciPy
- Open source
 - <https://seaborn.pydata.org>

```
>>> import seaborn as sns
```

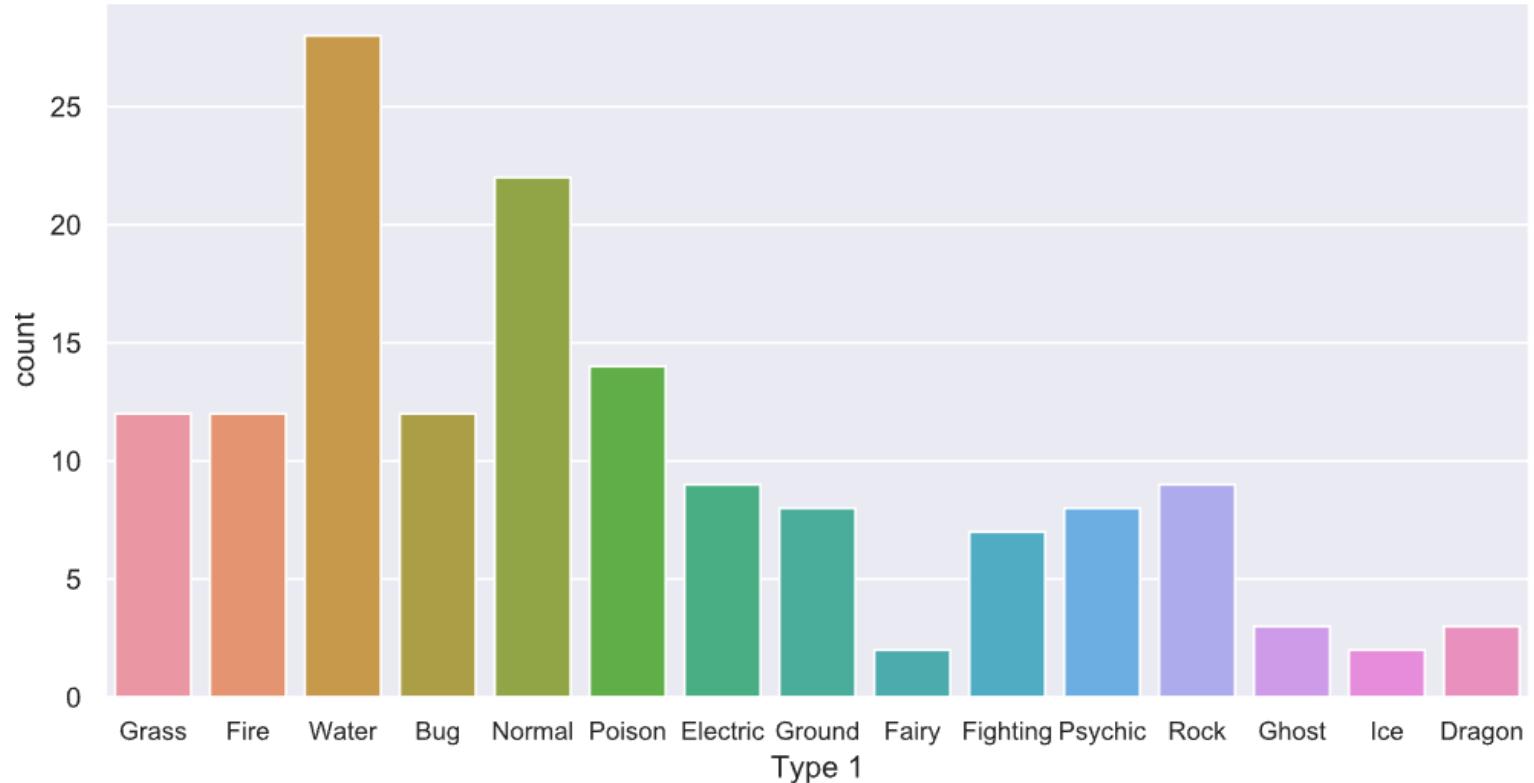
Pokemon Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('pokemon.csv', index_col=0)
df.head()
```

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Stage	Legendary
1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	2	False
3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	3	False
4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	2	False

Count Plot

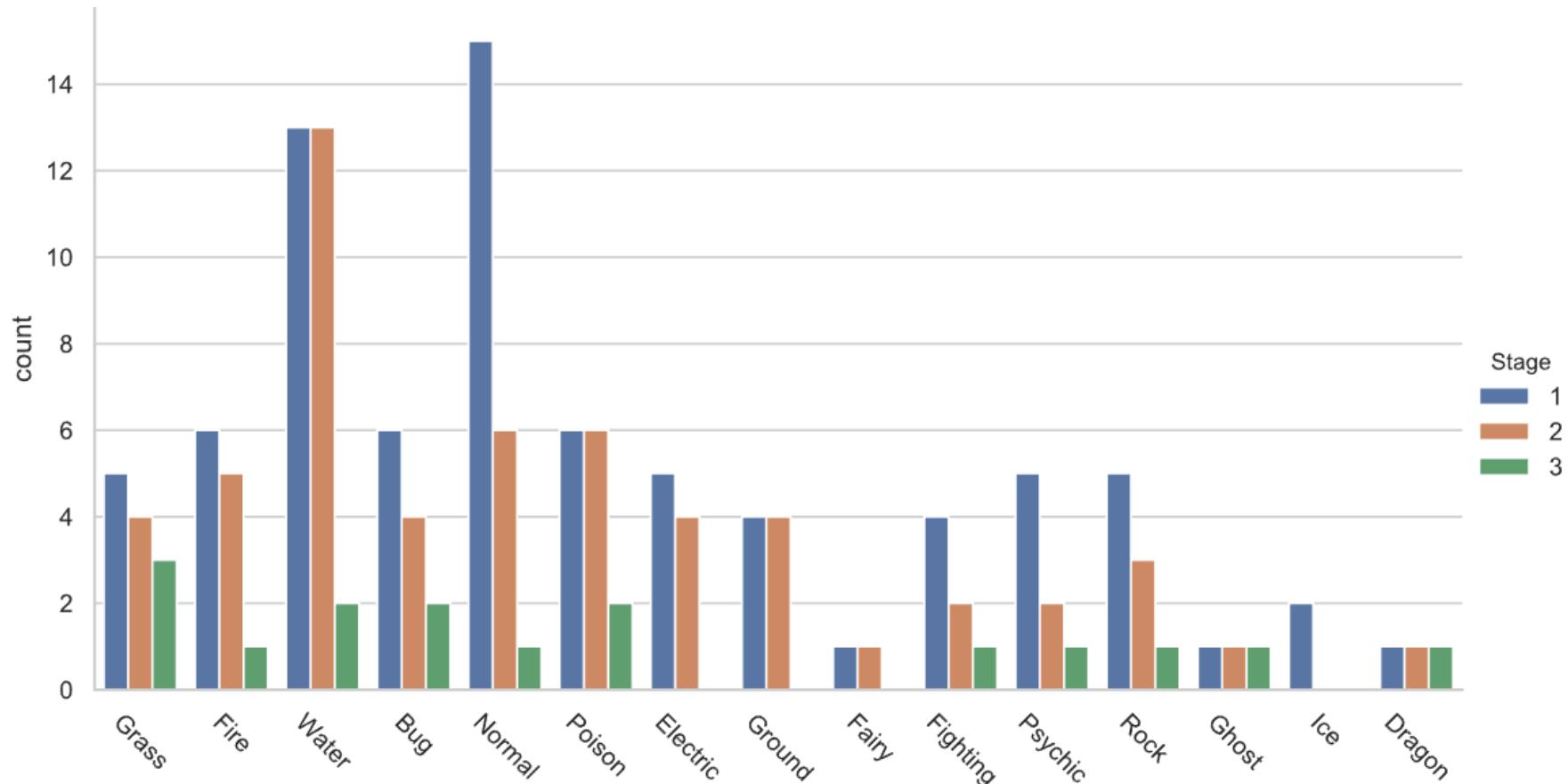
```
sns.set(style='darkgrid')
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10)
sns.countplot(x='Type 1', data=df)
```



Categorical Plot

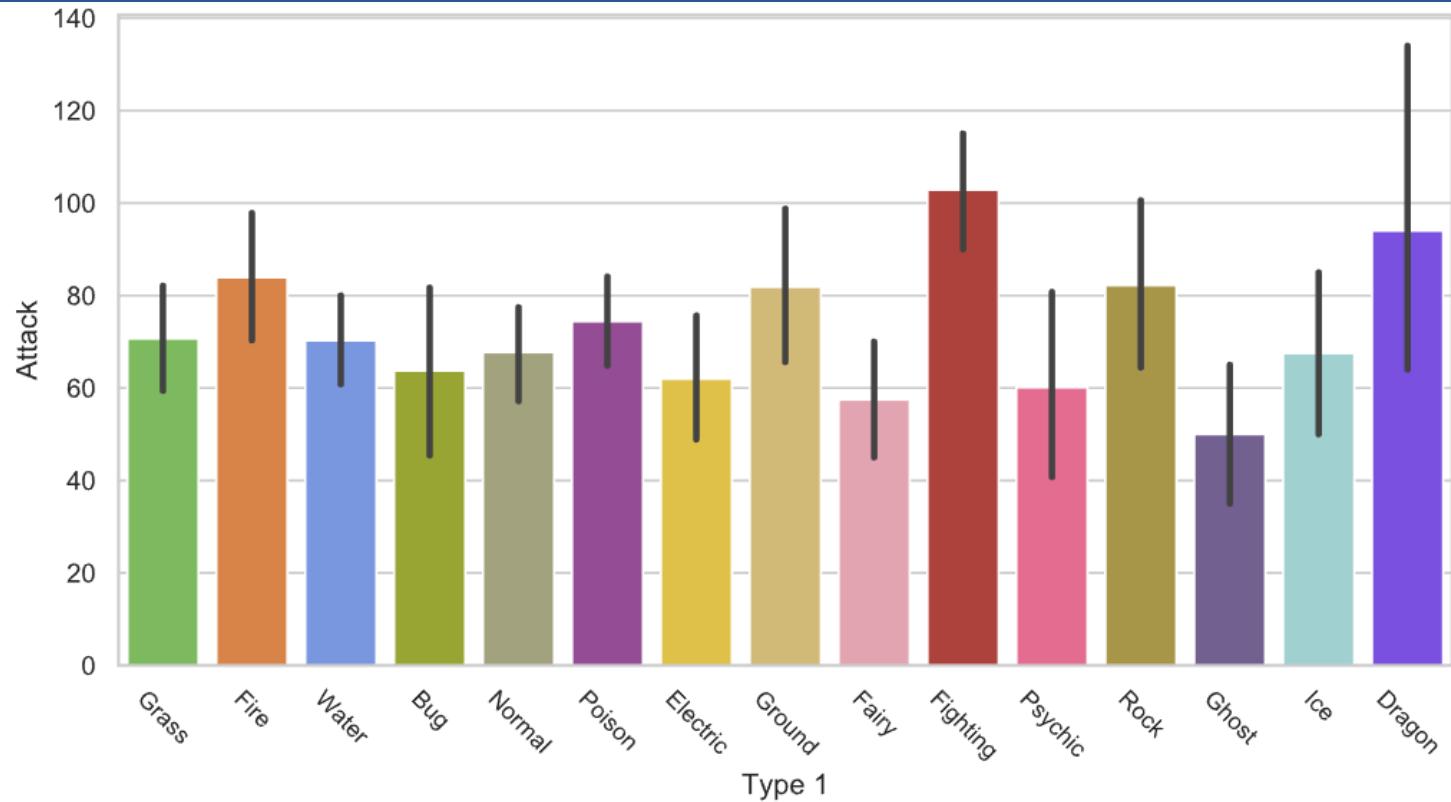
Group data

```
sns.catplot(x='Type 1', kind='count', hue='Stage', data=df, aspect=1.8)  
plt.xticks(rotation=-45)
```



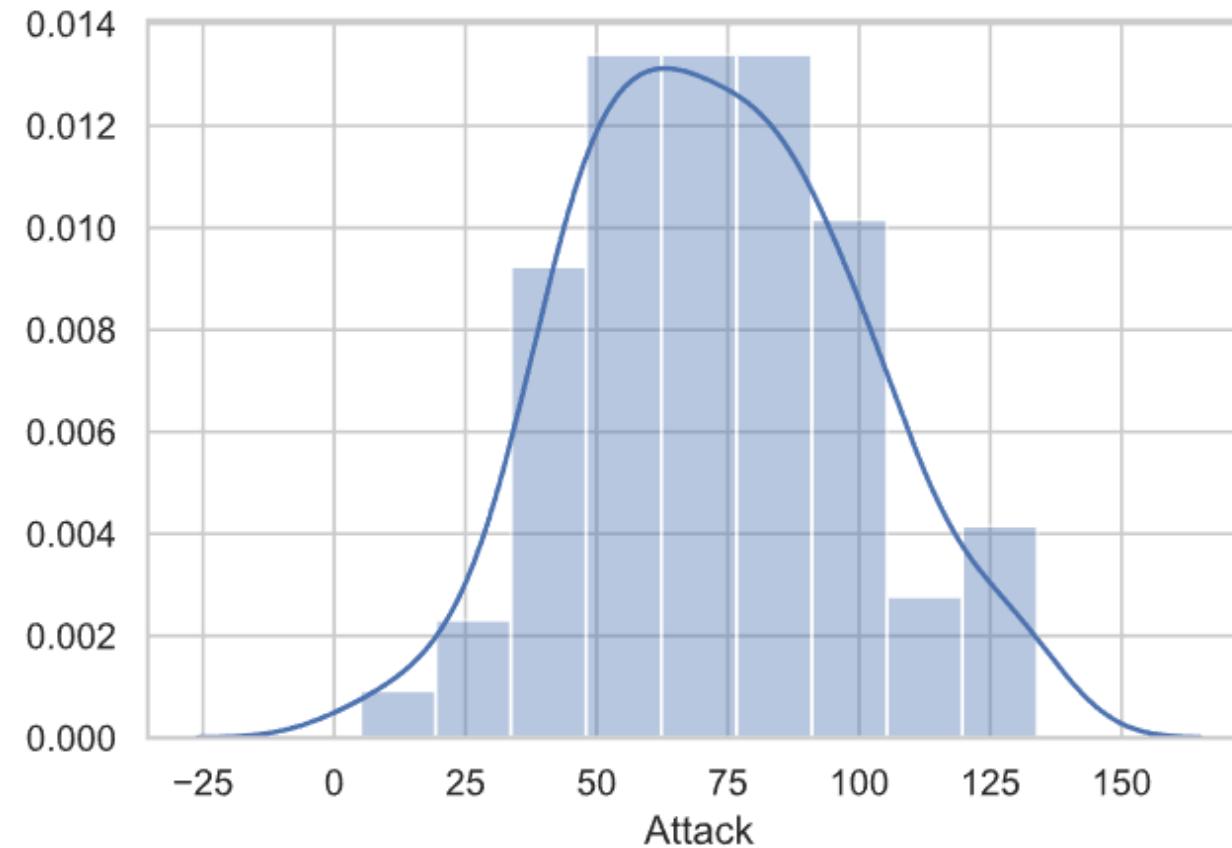
Bar Plot

```
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10, rotation=-45)
sns.barplot(x='Type 1', y='Attack', data=df, palette=poke_colors)
```



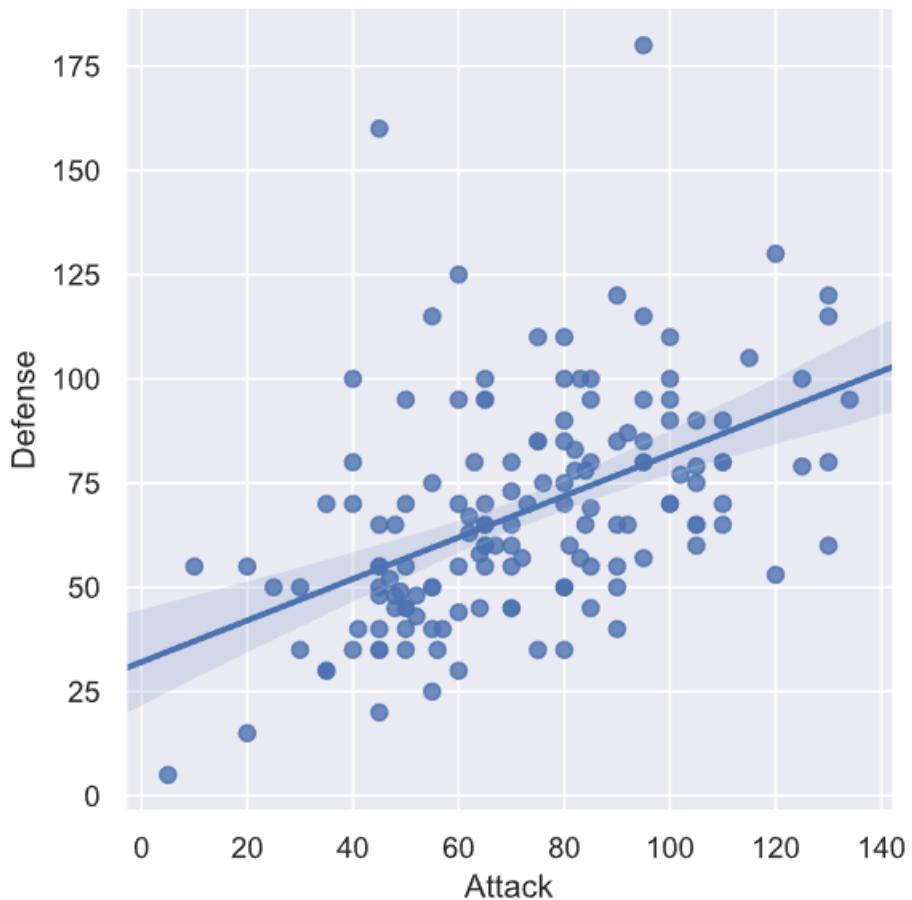
Histogram

```
sns.distplot(df.Attack)
```

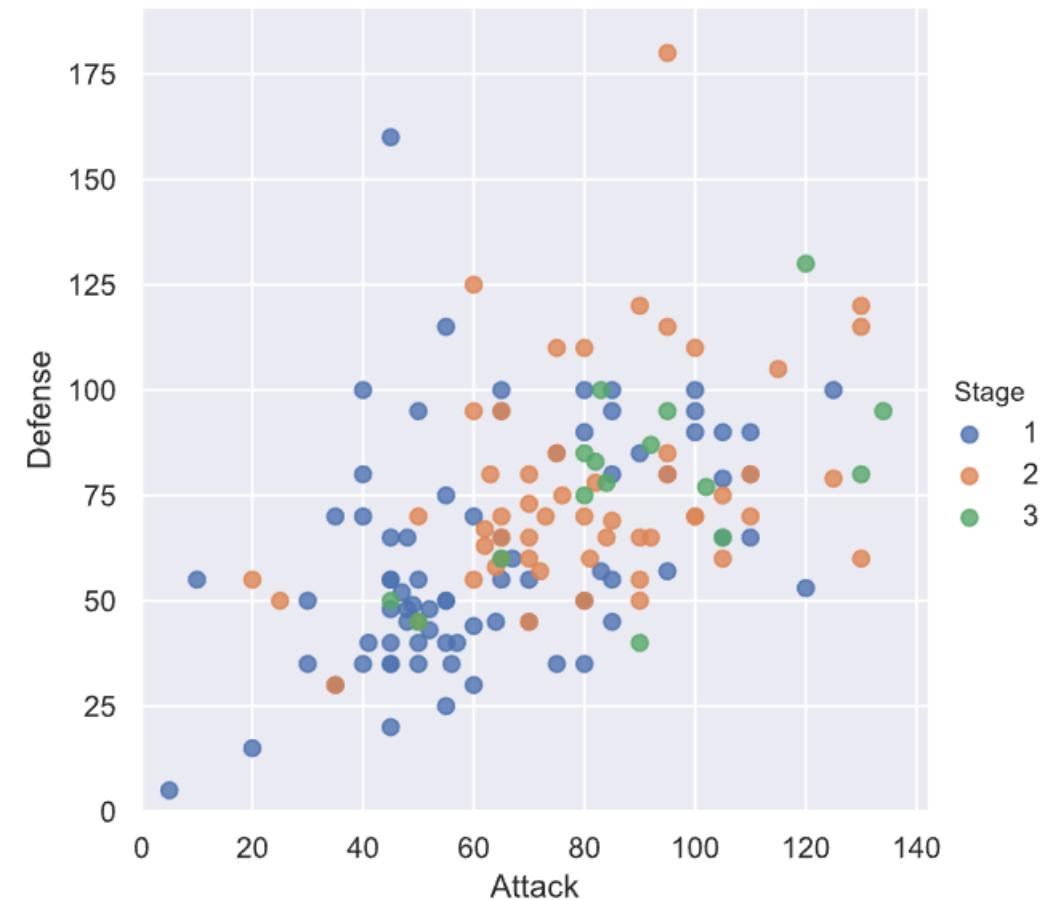


Scatter Plot

```
sns.regplot(x='Attack', y='Defense', data=df)
```

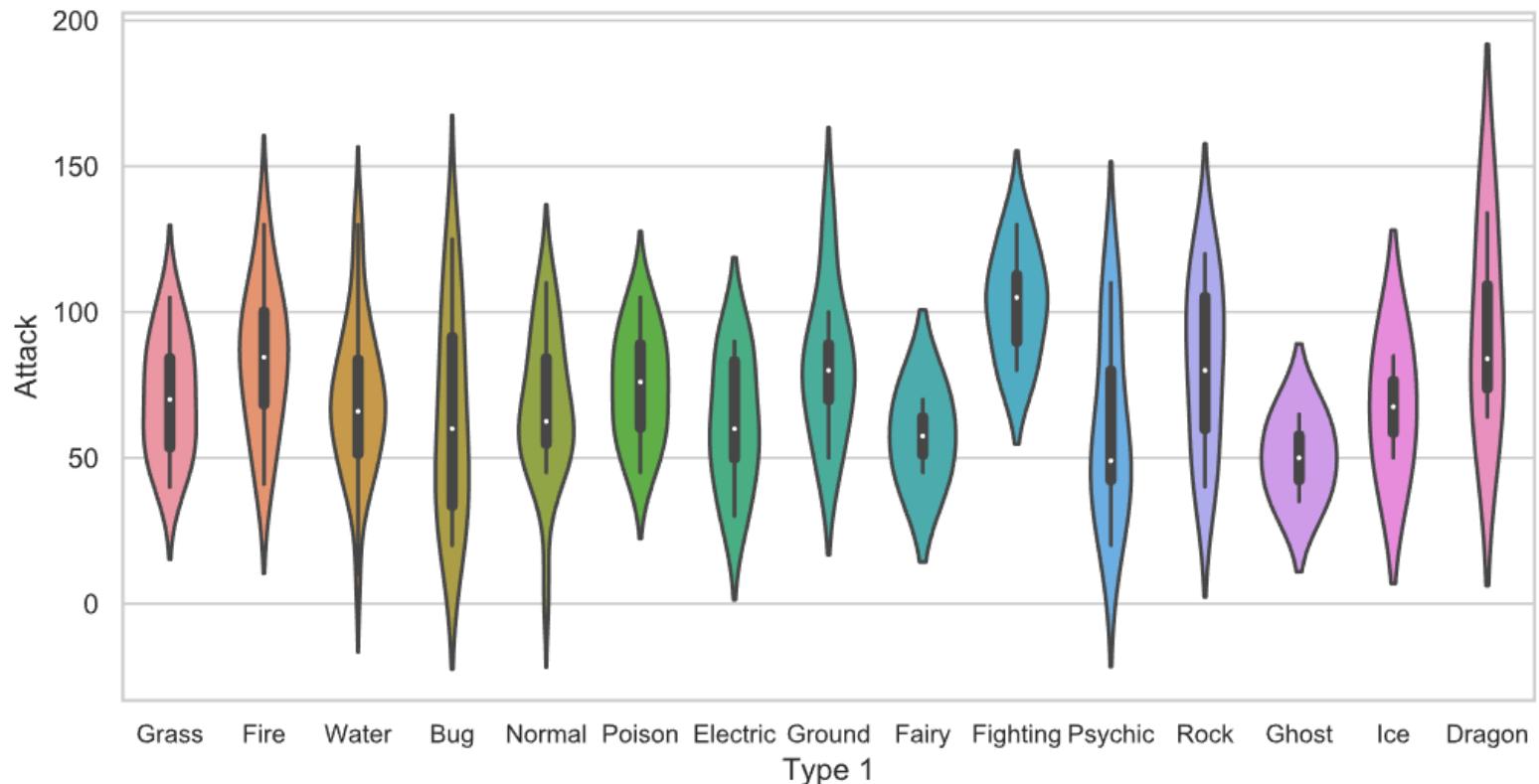


```
sns.lmplot(x='Attack', y='Defense',  
            fit_reg=False, hue='Stage', data=df)
```



Violin Plot

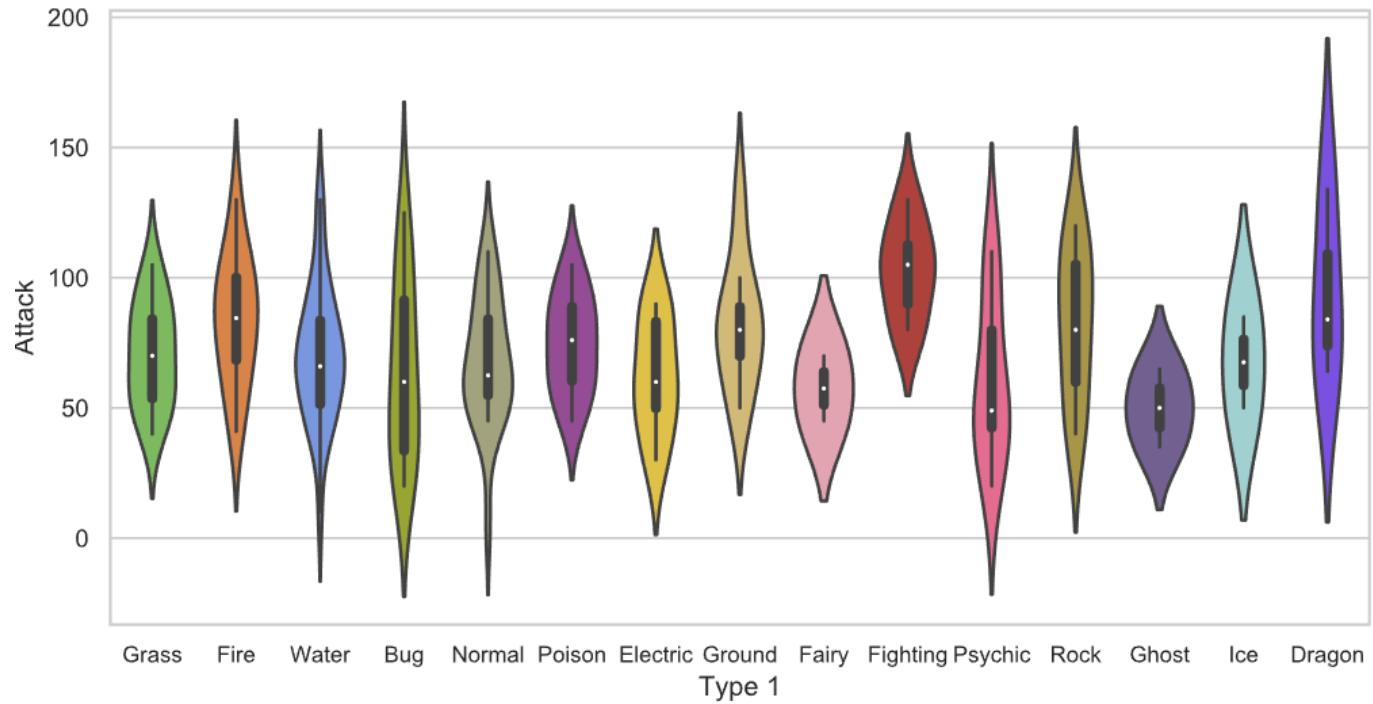
```
sns.set_style('whitegrid')
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10)
sns.violinplot(x='Type 1', y='Attack', data=df)
```



Violin Plot (with Customized Colors)

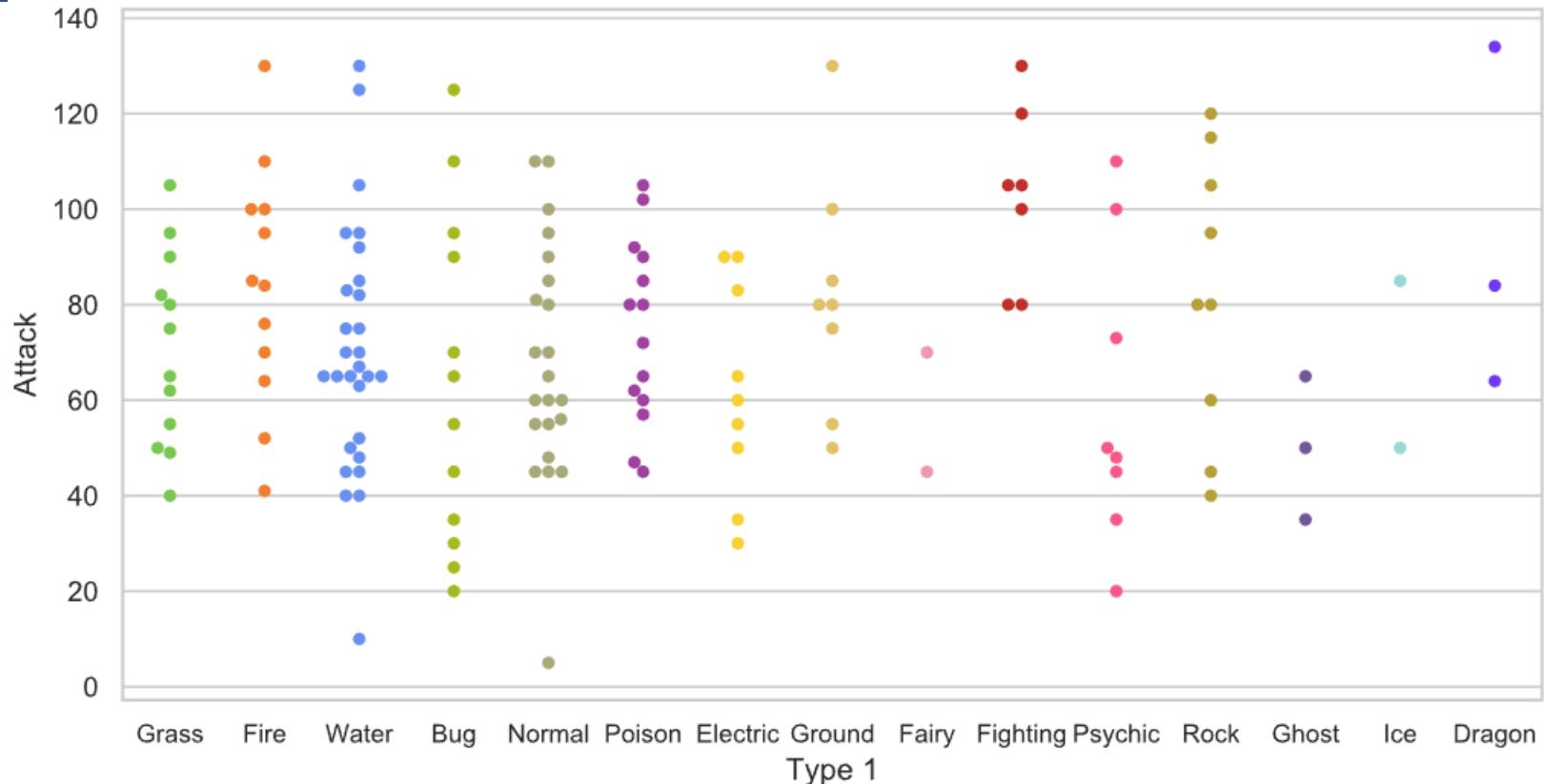
```
poke_colors = ['#78C850', # Grass  
               '#F08030', # Fire  
               '#6890F0', # Water  
               '#A8B820', # Bug  
               '#A8A878', # Normal  
               '#A040A0', # Poison  
               '#F8D030', # Electric  
               '#E0C068', # Ground  
               '#EE99AC', # Fairy  
               '#C03028', # Fighting  
               '#F85888', # Psychic  
               '#B8A038', # Rock  
               '#705898', # Ghost  
               '#98D8D8', # Ice  
               '#7038F8', # Dragon  
]
```

```
plt.figure(figsize=(10, 5))  
plt.xticks(fontsize=10)  
sns.violinplot(x='Type 1', y='Attack', data=df,  
                palette=poke_colors)
```



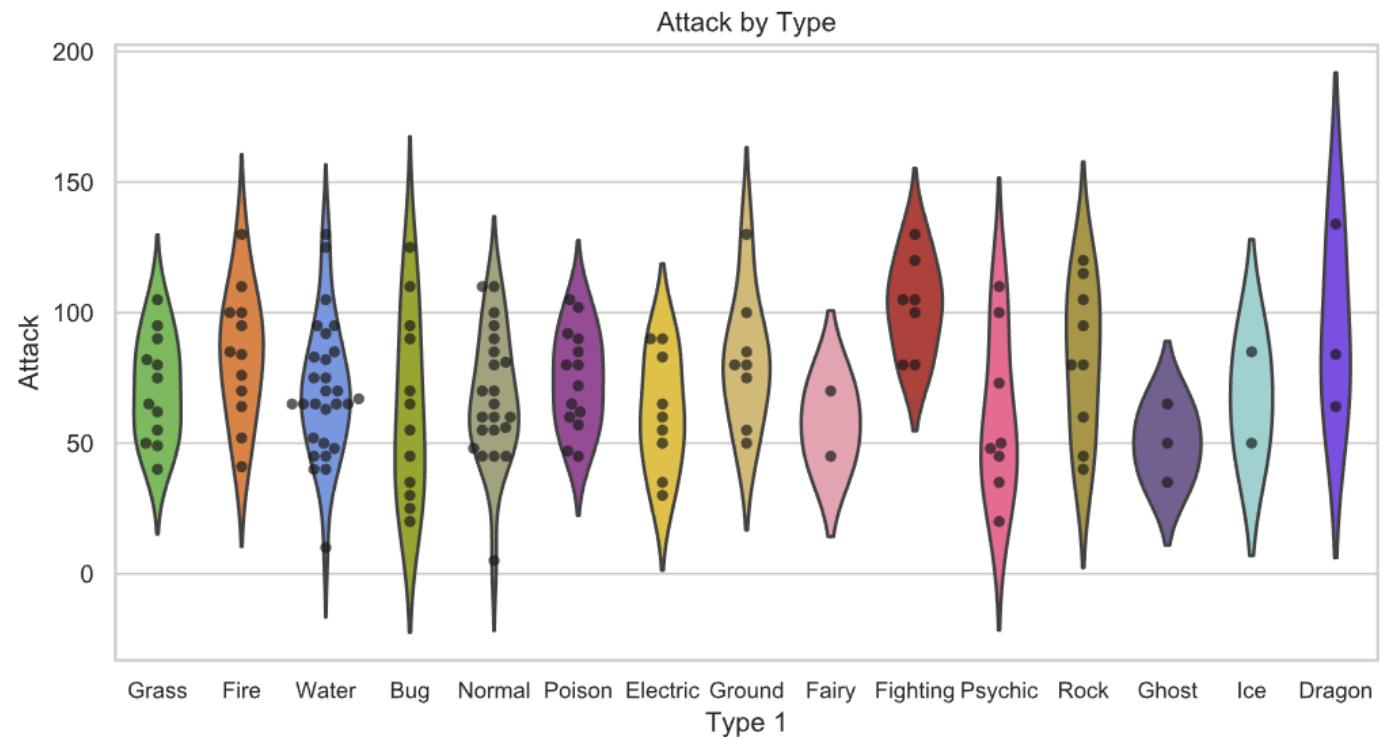
Swarm Plot

```
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10)
sns.swarmplot(x='Type 1', y='Attack', data=df, palette=poke_colors)
```



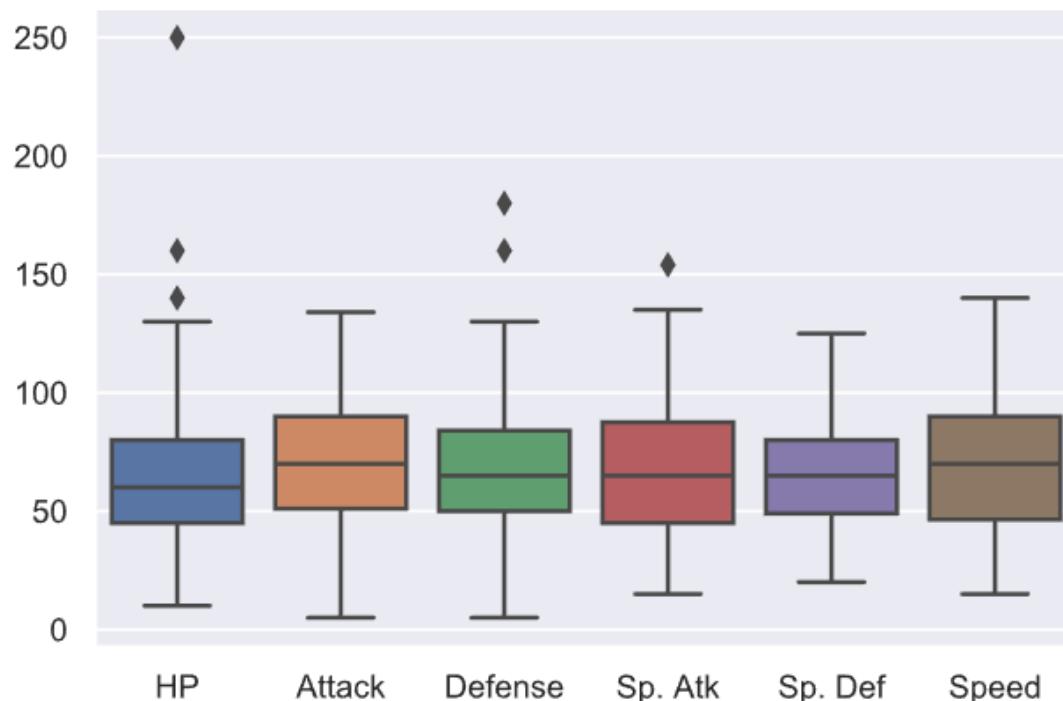
Violin + Swarm Plot

```
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10)
sns.violinplot(x='Type 1', y='Attack', data=df, palette=poke_colors, inner=None)
sns.swarmplot(x='Type 1', y='Attack', data=df, color='k', alpha=0.7)
plt.title('Attack by Type')
```

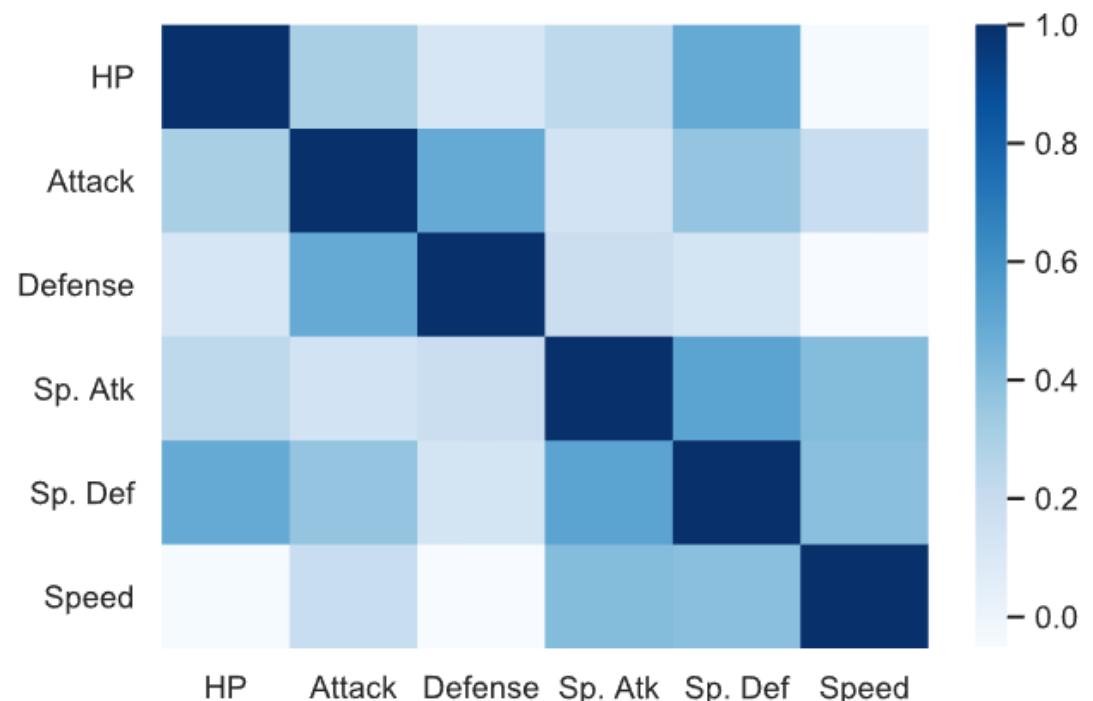


Box Plot and Heatmap

```
stats_df = df.drop(['Total', 'Stage',  
                    'Legendary'], axis=1)  
sns.boxplot(data=stats_df)
```

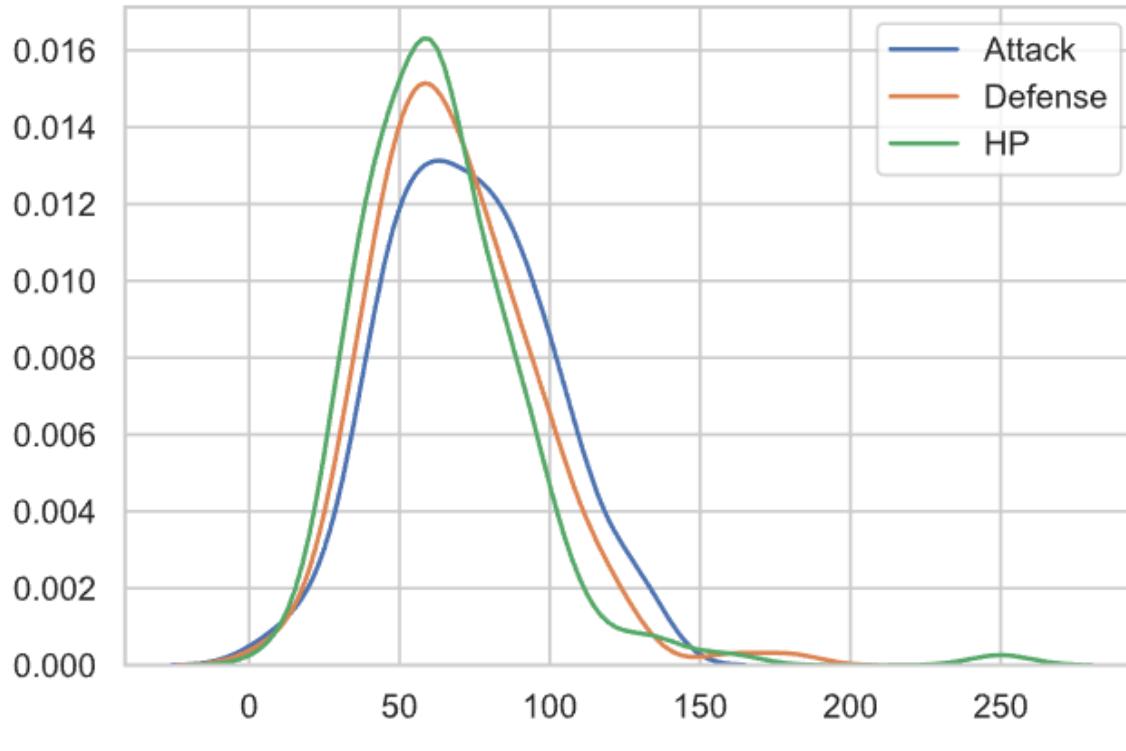


```
corr = stats_df.corr()  
sns.heatmap(corr, cmap='Blues')
```

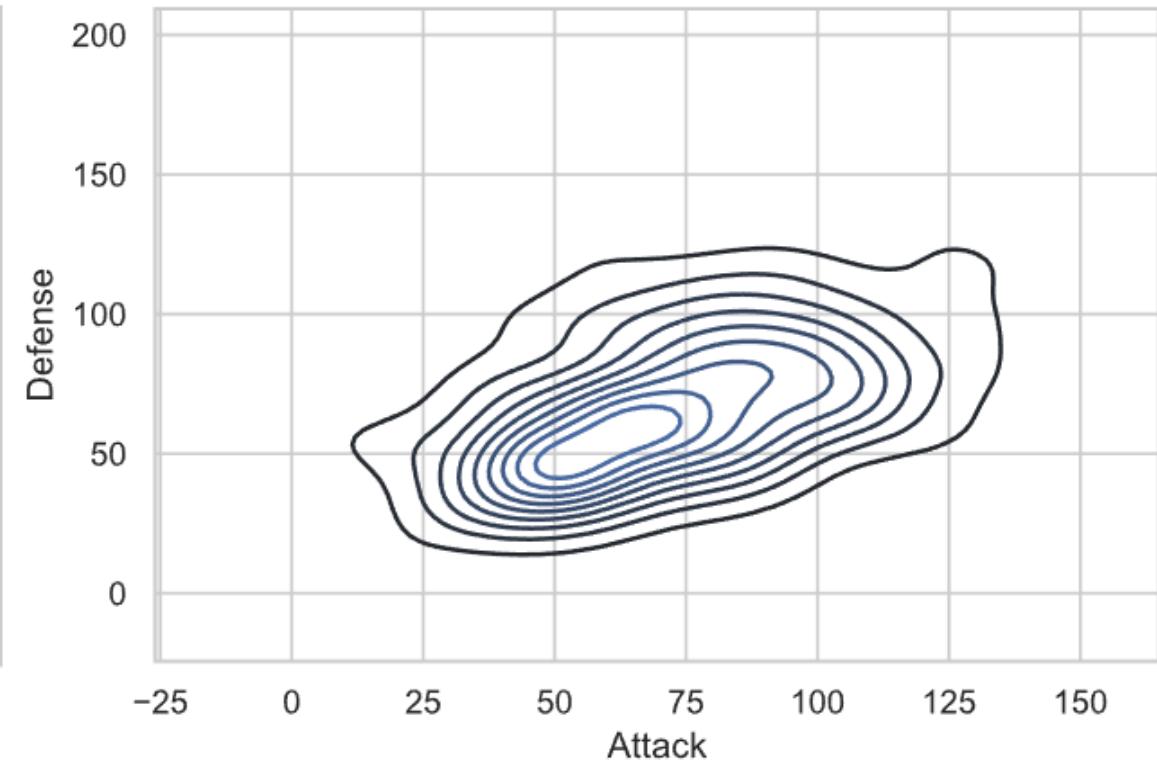


KDE Plot

```
sns.kdeplot(df.Attack)  
sns.kdeplot(df.Defense)  
sns.kdeplot(df.HP)
```

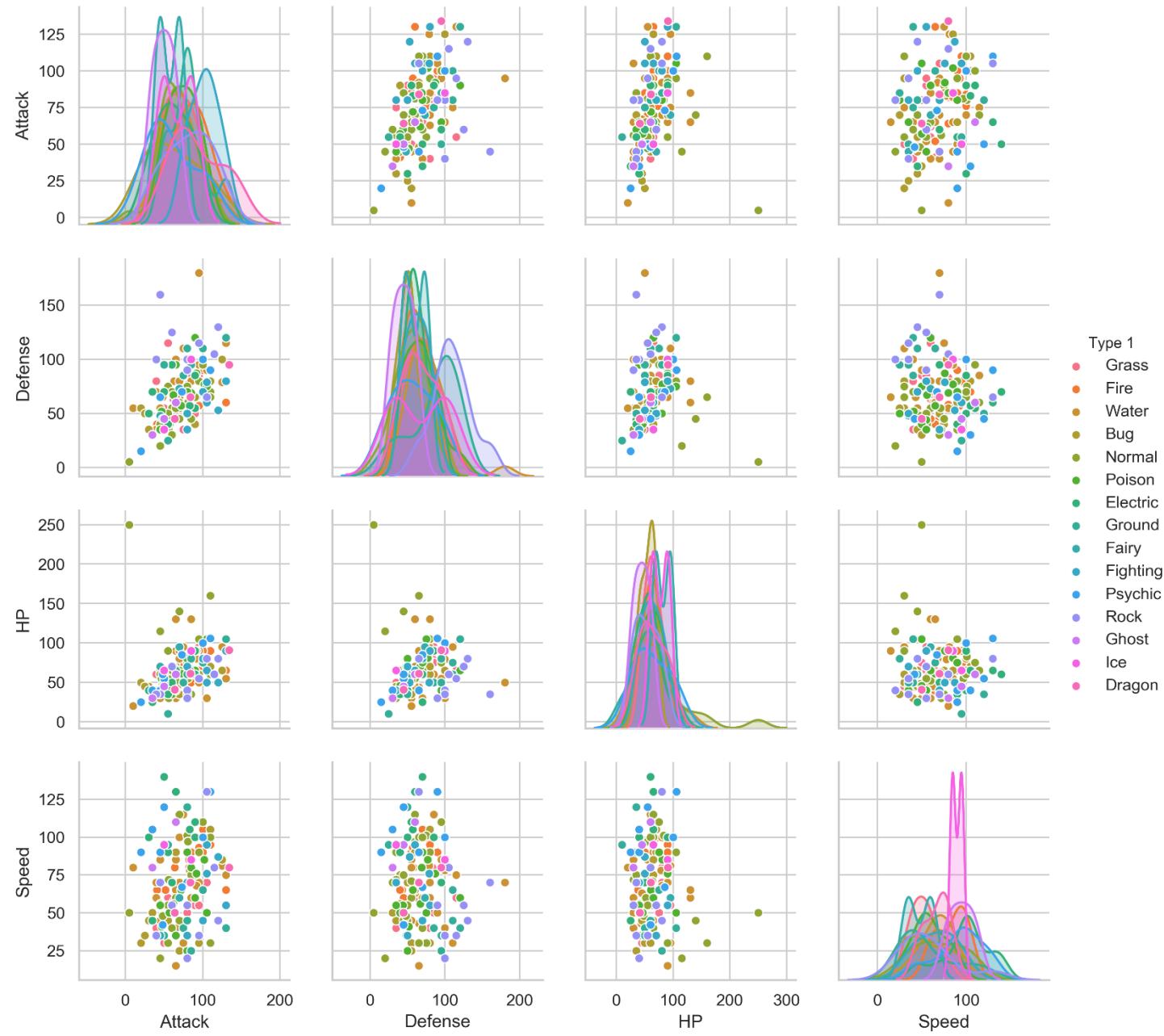


```
sns.kdeplot(df.Attack, df.Defense)
```



Pairplot

```
s.pairplot(df[['Type 1',  
               'Attack',  
               'Defense',  
               'HP',  
               'Speed']],  
           hue='Type 1')
```



Plotting in Pandas

Pandas plot()

- `df.plot(*args, **kwargs)` or `series.plot (*args, **kwargs)`

- Make plots of DataFrame or Series

<code>plot(kind=str)</code>	<code>subfunction</code>	<code>Graph produced</code>
'line'	<code>plot.line()</code>	Line plot (default)
'bar'	<code>plot.bar()</code>	Vertical bar plot
'barh'	<code>plot.barh()</code>	Horizontal bar plot
'hist'	<code>plot.hist()</code>	Histogram
'box'	<code>plot.box()</code>	Box plot
'kde' or 'density'	<code>plot.kde()</code>	Kernel Density Estimation plot
'area'	<code>plot.area()</code>	Area plot
'pie'	<code>plot.pie()</code>	Pie plot
'scatter'	<code>plot.scatter()</code>	Scatter plot (DataFrame only)

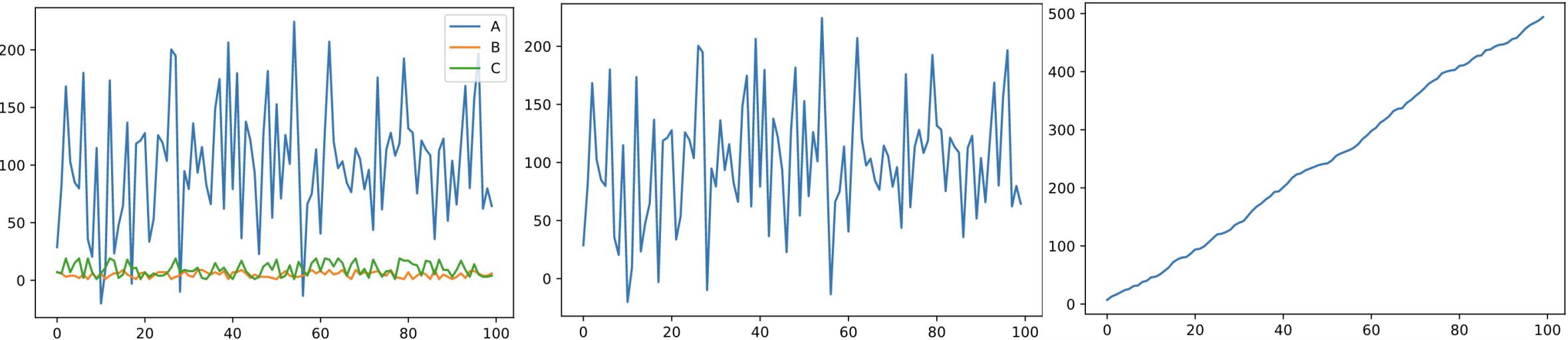
Line Plot

```
df = pd.DataFrame({'A': np.random.normal(100, 50, 100),  
                   'B': np.random.randint(1, 10, size=100),  
                   'C': np.random.randint(1, 20, size=100)})
```

```
df.plot()
```

```
df.A.plot()
```

```
df.B.cumsum().plot()
```

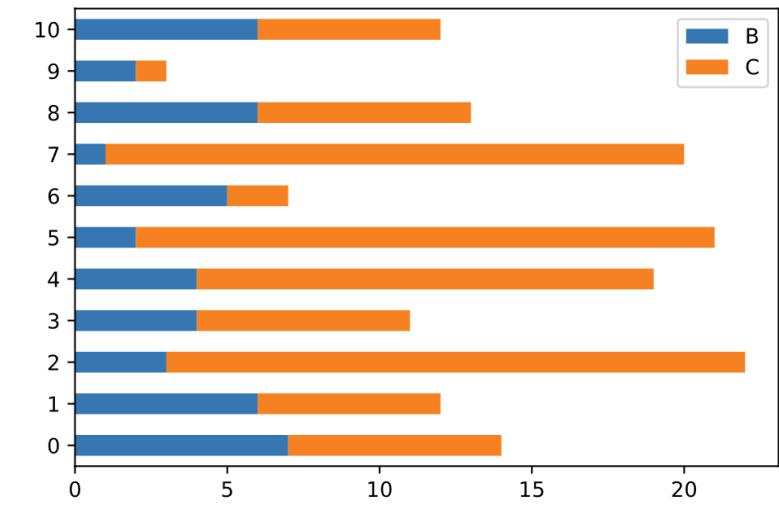
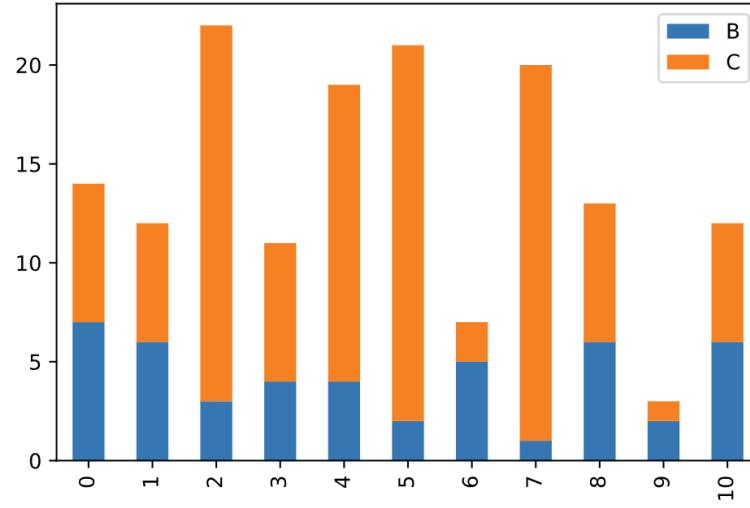
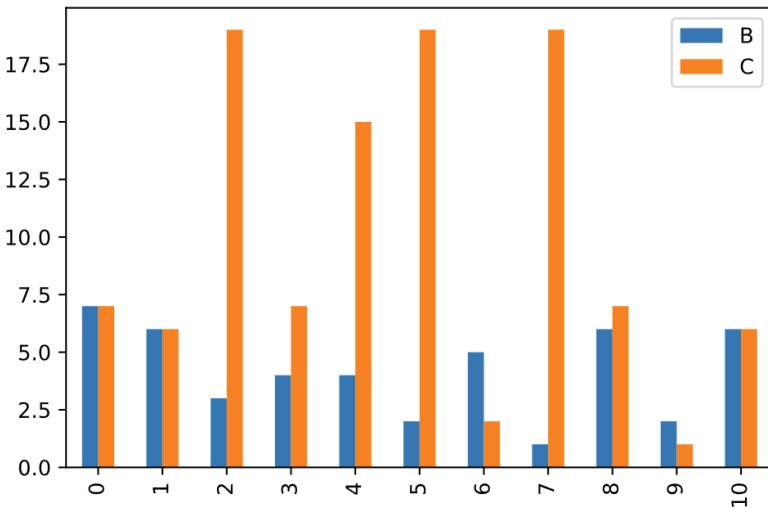


Bar Plot

```
df.loc[:10, ['B', 'C']].plot.bar()
```

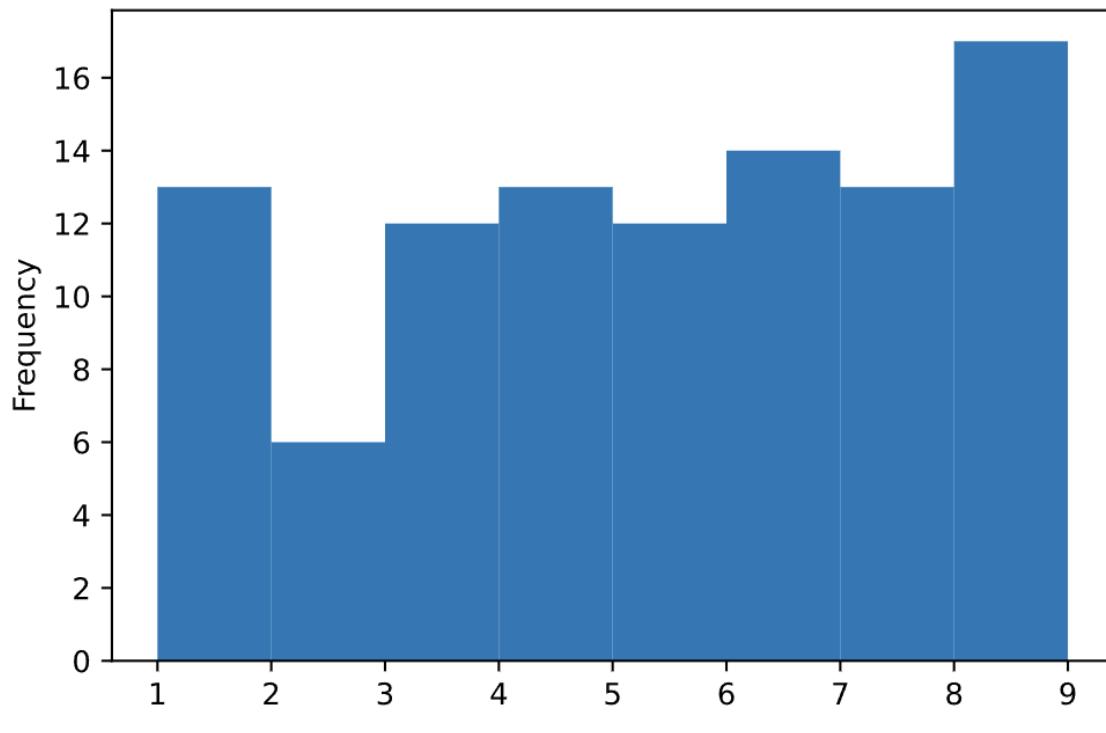
```
df.loc[:10, ['B', 'C']].plot.bar(stacked=True)
```

```
df.loc[:10, ['B', 'C']].plot.barh(stacked=True)
```

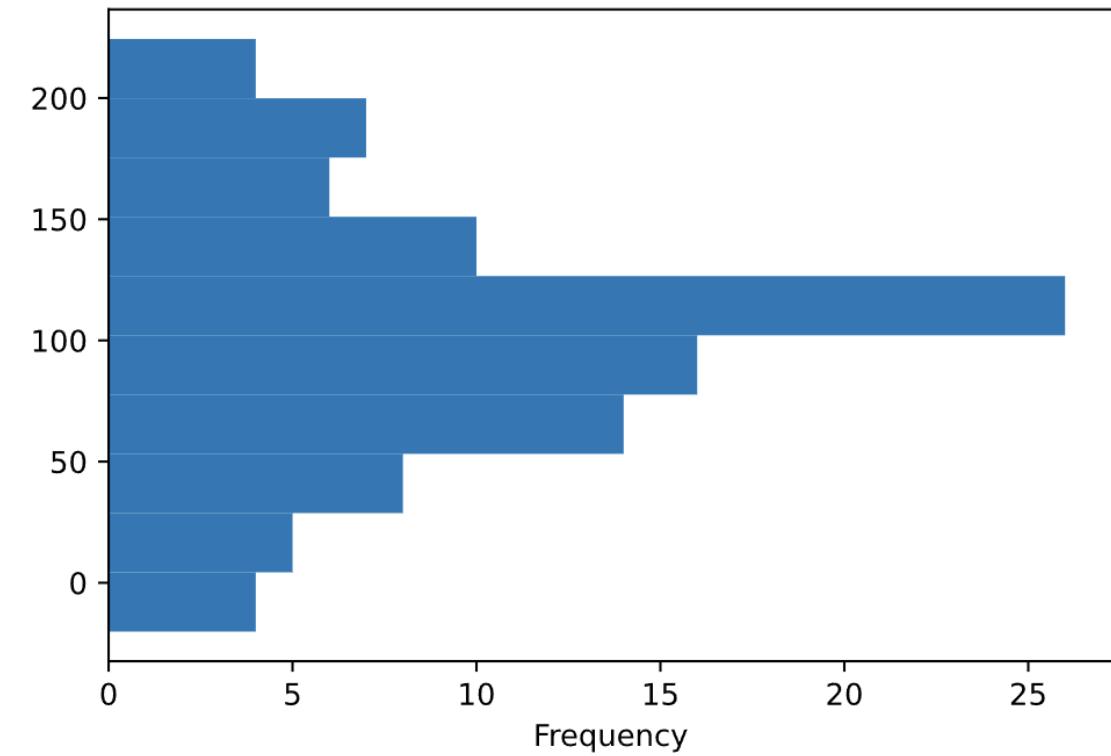


Histogram

```
df.B.plot(kind='hist', bins=8)
```

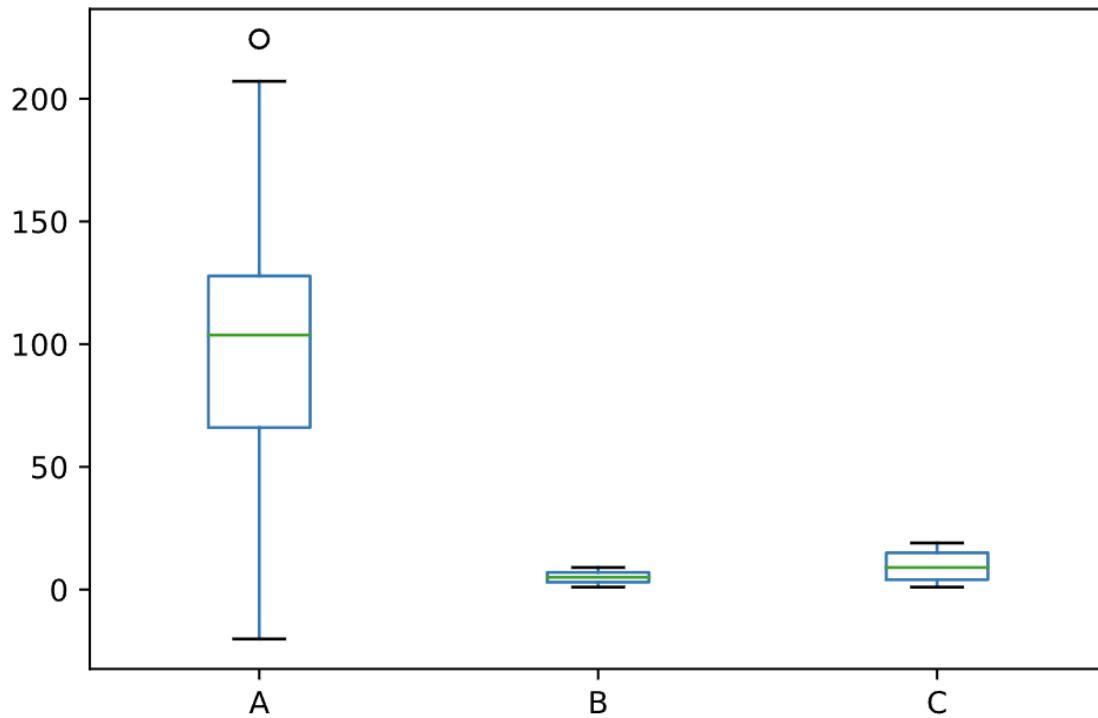


```
df.A.plot.hist(orientation='horizontal')
```

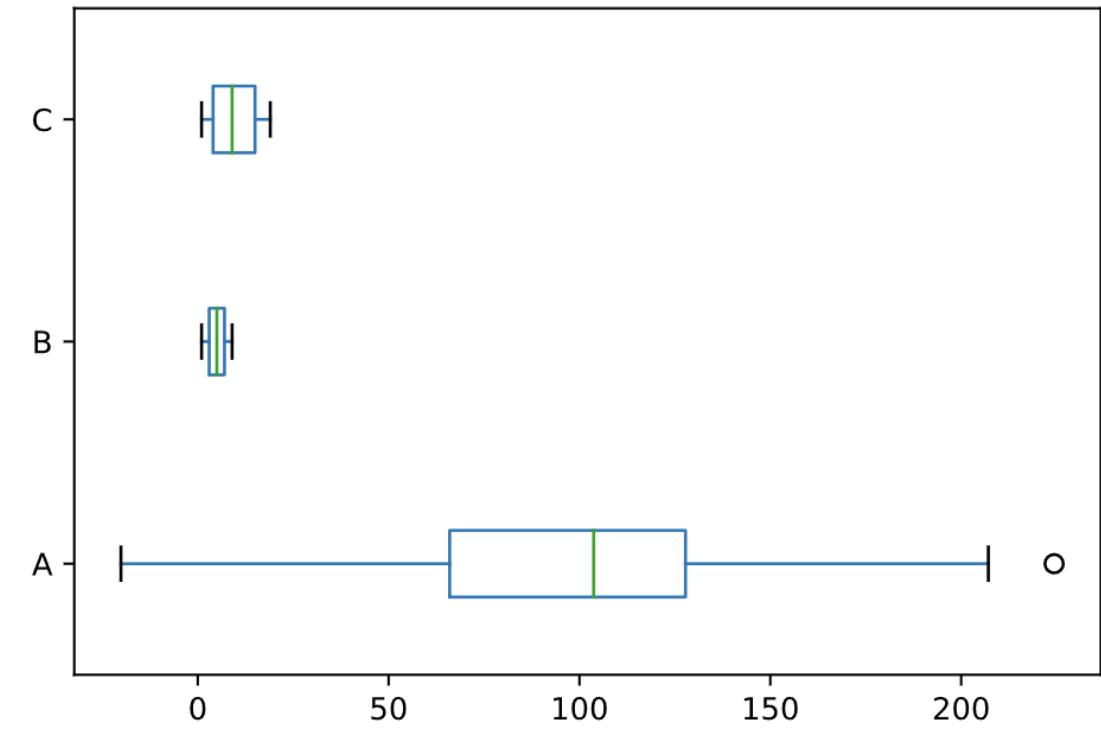


Box Plot

```
df.plot.box()
```

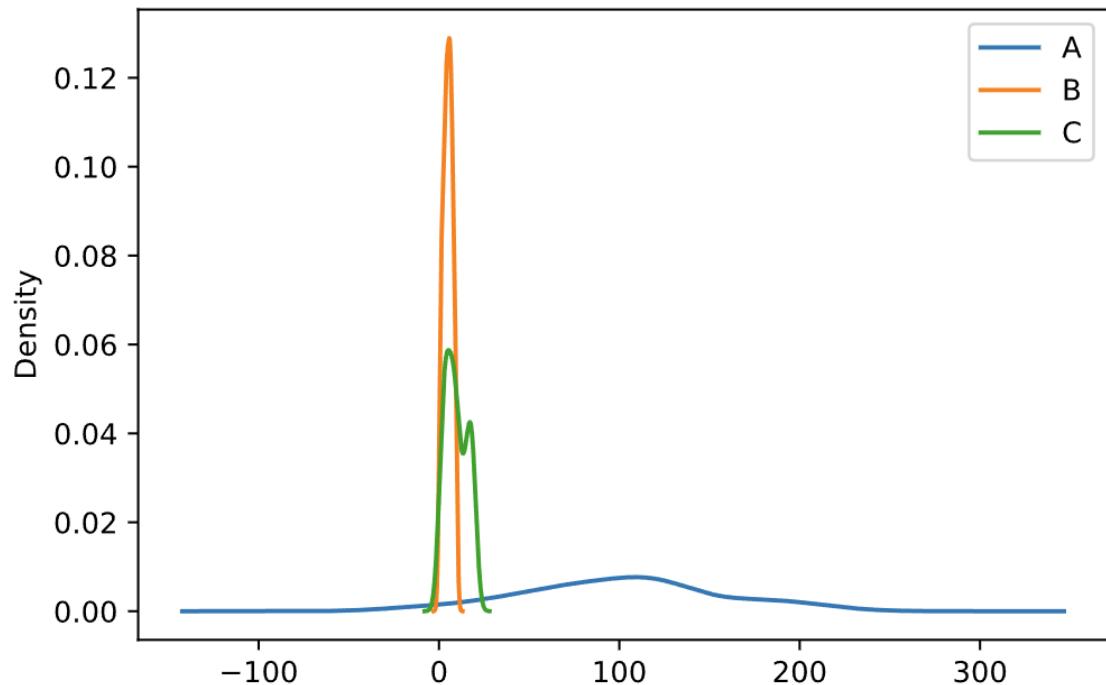


```
df.plot.box(vert=False)
```

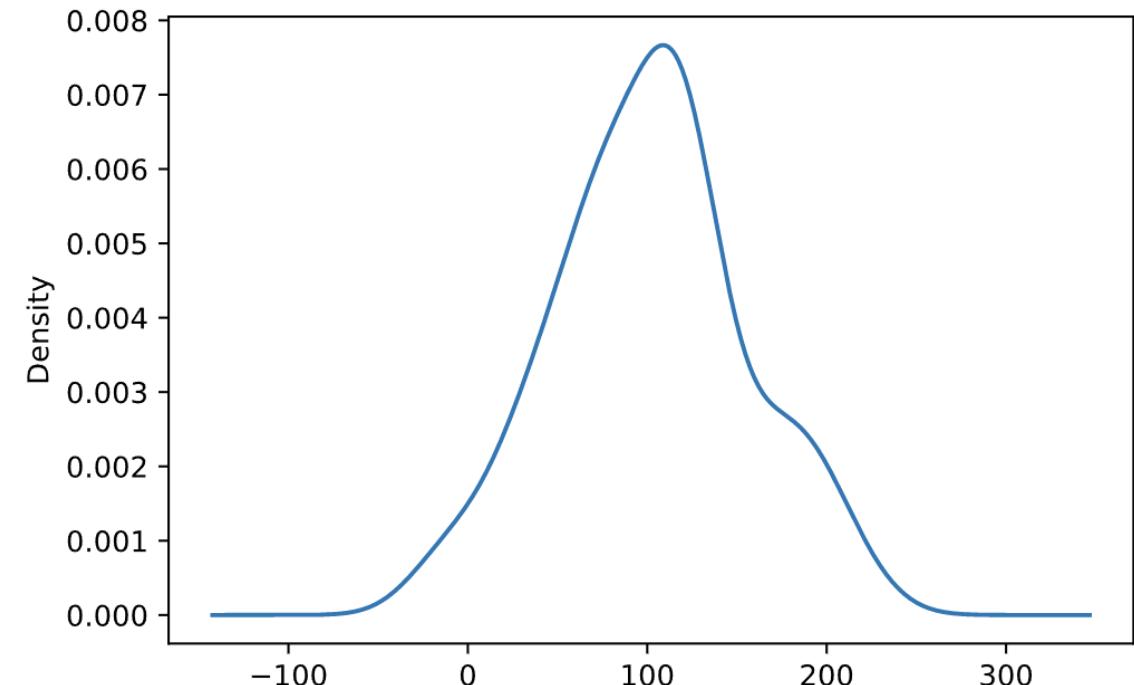


KDE Plot

```
df.plot.kde()
```

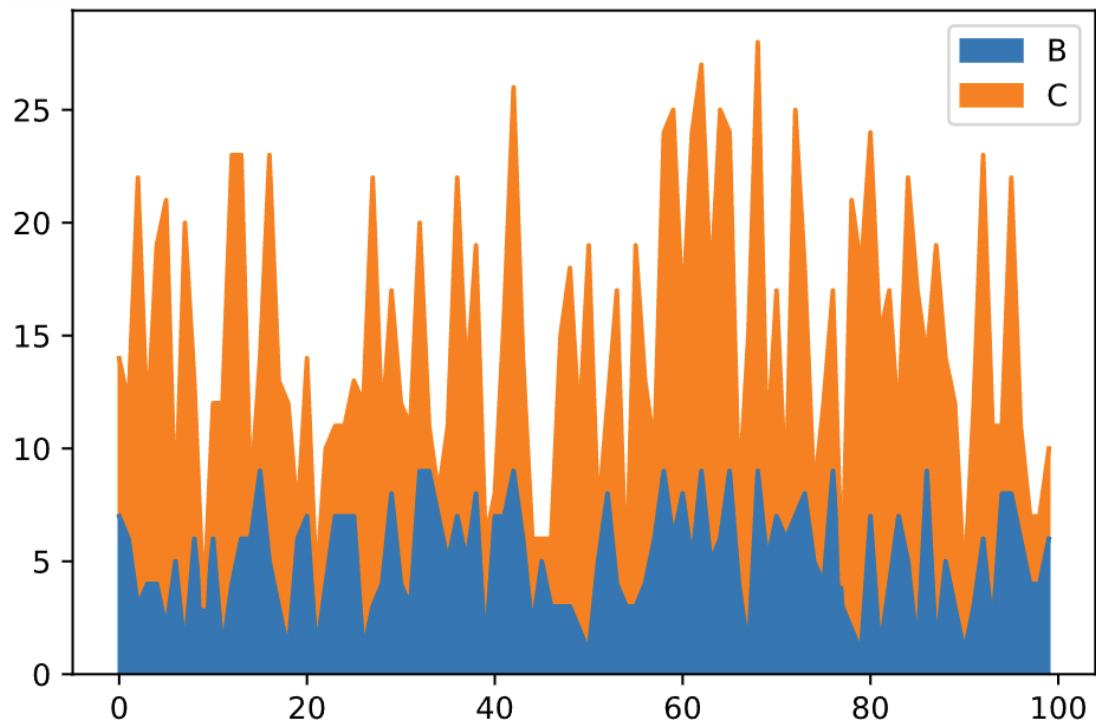


```
df.A.plot.kde()
```

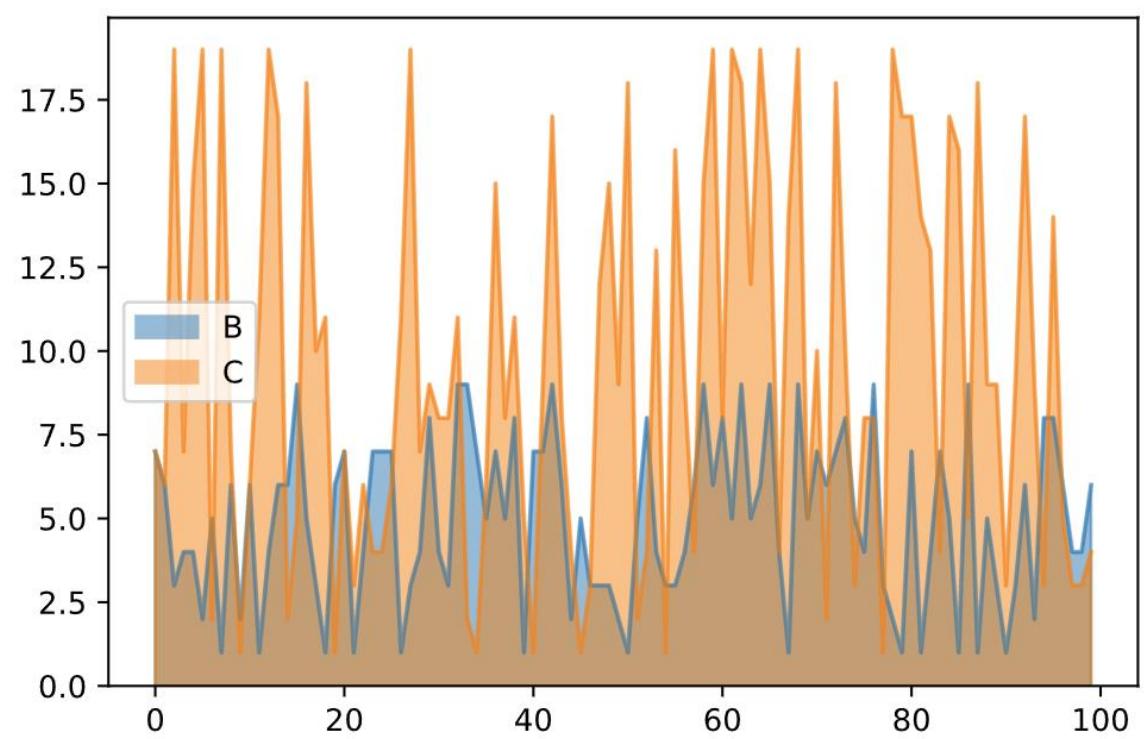


Area Plot

```
df[['B','C']].plot.area()
```

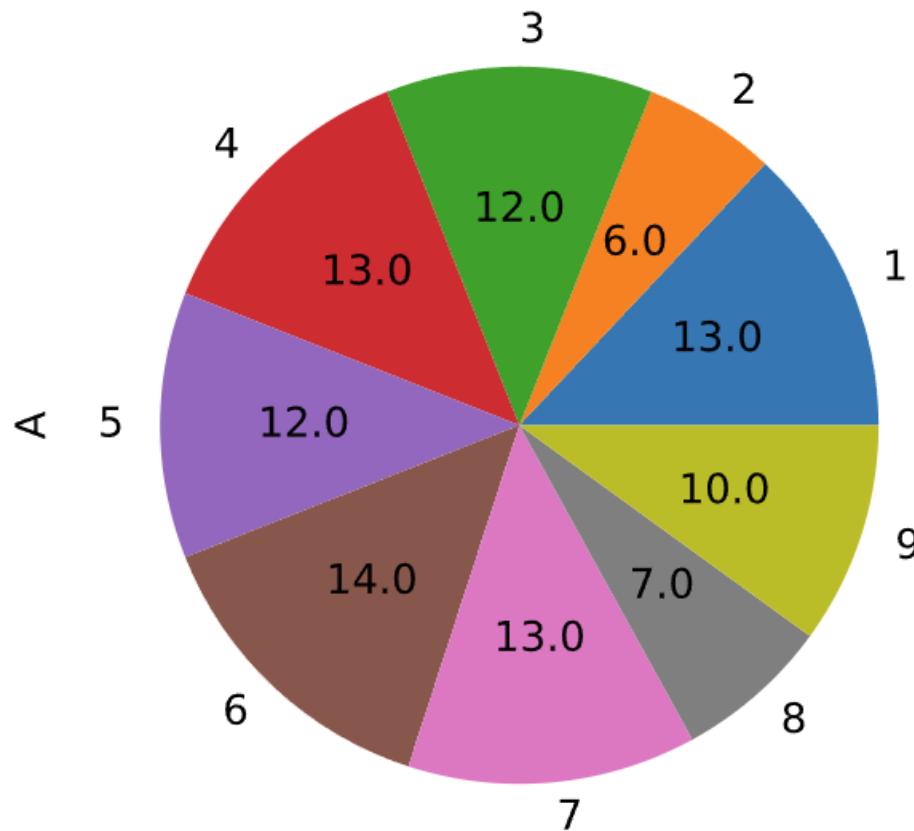


```
df[['B','C']].plot.area(stacked=False)
```



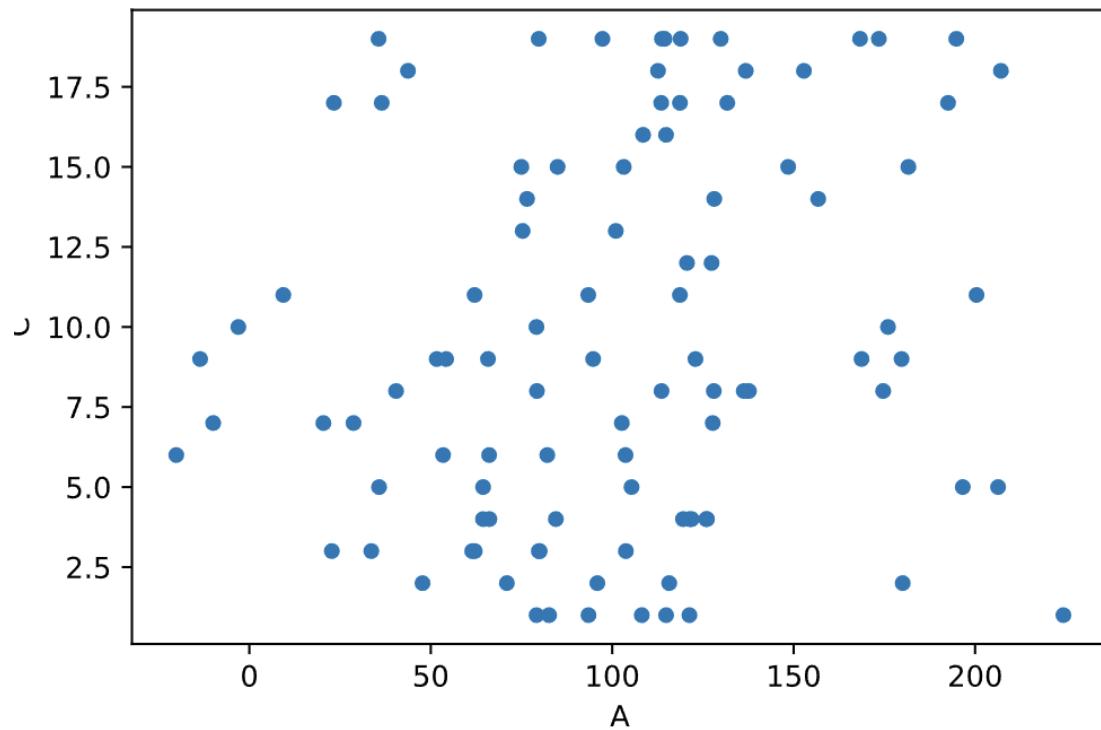
Pie Chart

```
df.groupby('B').count().plot.pie(y='A', legend=False, autopct=".1f")
```

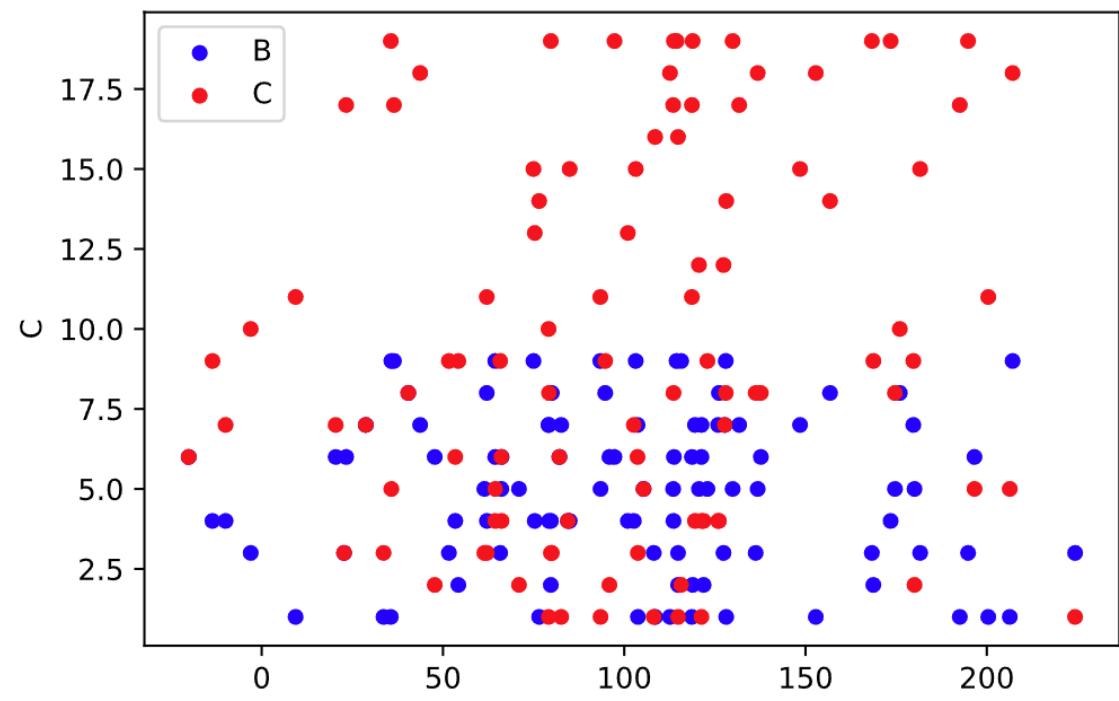


Scatter Plot (I)

```
df.plot(kind='scatter', x='A', y='C')
```

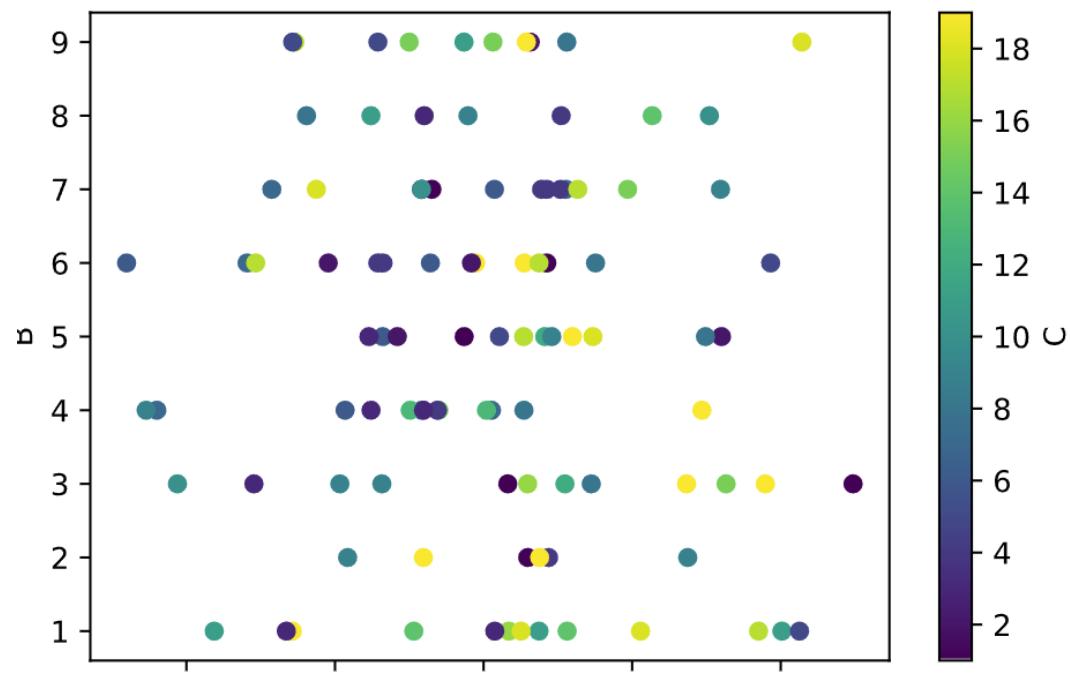


```
ax = df.plot.scatter(x='A', y='B',  
                     color='blue', label='B')  
df.plot.scatter(x='A', y='C',  
                     color='red', label='C', ax=ax)
```

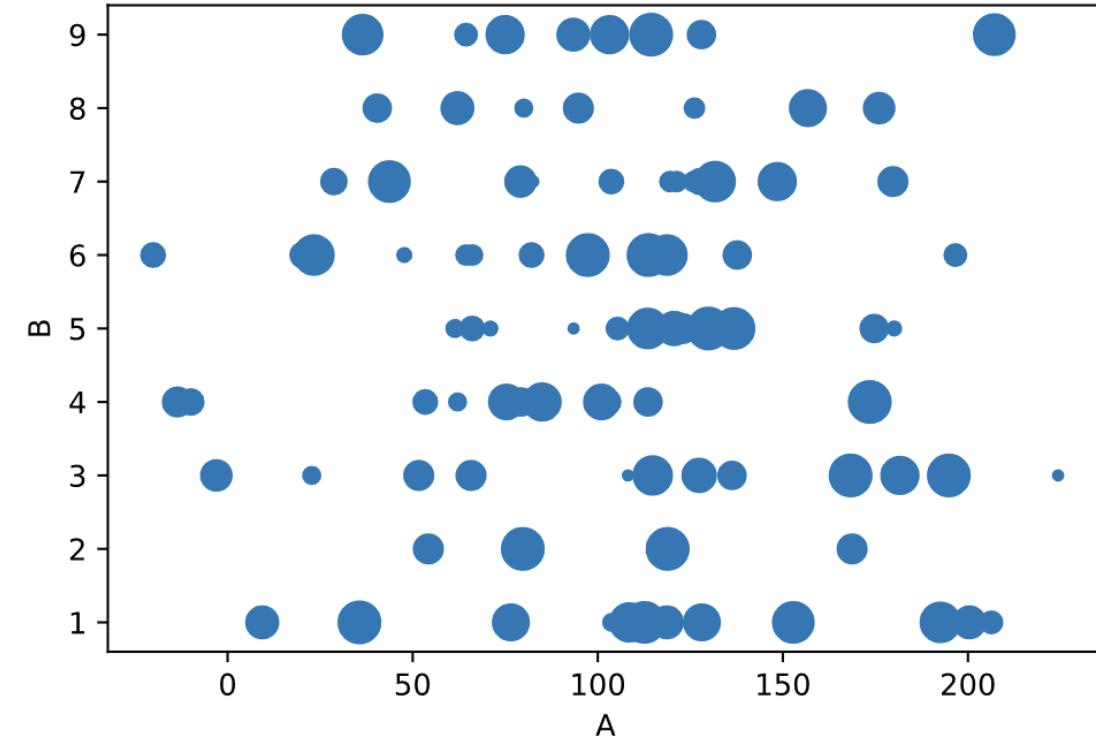


Scatter Plot (2)

```
df.plot.scatter(x='A', y='B', c='C',  
                cmap='viridis', s=30)
```



```
df.plot.scatter(x='A', y='B', s=df['C']*10)
```



Jin-Soo Kim
[\(jinsoo.kim@snu.ac.kr\)](mailto:jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022



Python for Data Analytics

Pandas III

Outline

- Why Pandas?
- Pandas Series
- Pandas DataFrame
 - Creating DataFrame
 - Manipulating Columns
 - Manipulating Rows
 - Arithmetic operations
 - Group Aggregation
 - Hierarchical Indexing
 - Combining and Merging
 - Time Series Data

Time Series Data

Time Series Data

- How to analyze time series data?

- Sample time series data

2011-01-01 00:00:00	-0.131254
2011-01-01 01:00:00	0.068876
2011-01-01 02:00:00	-0.207636
2011-01-01 03:00:00	1.388030
Timestamp Index → 2011-01-01 04:00:00	0.937158

- Time series data is the data with the timestamp index

- How to parse time series information from various sources and formats?
 - How to generate sequences of fixed-frequency dates and time spans
 - How to manipulate and convert date times with timezone information?
 - How to group data by time?
 - ...

Python datetime Module

- `datetime.datetime` class: a combination of date and time
 - year, month, day, hour, minute, second, microsecond, tzinfo
- `datetime.now()`: return the current local datetime

```
import numpy as np
import pandas as pd
```

```
import datetime as dt
now = dt.datetime.now()
print(now)
```

2021-01-09 22:56:48.365683

```
newyear = dt.datetime(2021, 1, 1)
print(newyear)
```

2021-01-01 00:00:00

```
print(now - newyear)
```

8 days, 22:56:48.365683

NumPy datetime64 Type

- NumPy supports datetime functionality with the data type called '**datetime64**'
 - No timezone support

```
import numpy as np
now = np.datetime64('now')
print(now)
```

```
2021-01-09T13:59:25
```

```
np.arange('2021-01', '2021-07', dtype='datetime64[M]')
```

```
array(['2021-01', '2021-02', '2021-03', '2021-04', '2021-05', '2021-06'],
      dtype='datetime64[M]')
```

```
newyear = np.datetime64('2021-1-1')
```

```
-----  
ValueError
```

```
Traceback (most recent call last)
```

```
<ipython-input-11-bc5dab85414d> in <module>
----> 1 newyear = np.datetime64('2021-1-1')
```

```
ValueError: Error parsing datetime string "2021-1-1" at position 5
```

Converting to Datetime

- Pandas supports extensive capabilities and features for working with time series data based on NumPy datetime64.
- `pd.to_datetime(arg, ...)`
 - Convert argument to datetime.
 - Return type can be a DatetimeIndex, Series, or Timestamp
 - `arg`: integer, float, string, datetime, list, tuple, 1-D array, Series

```
t = np.array(['1/11/2021', '2021/1/12', '20210113', '2021-1-14', '2021 1 15',
              '2021, 1, 16', 'Jan. 17 2021', '18 Jan 2021'])
pd.to_datetime(t)
```

```
DatetimeIndex(['2021-01-11', '2021-01-12', '2021-01-13', '2021-01-14',
                 '2021-01-15', '2021-01-16', '2021-01-17', '2021-01-18'],
                dtype='datetime64[ns]', freq=None)
```

Generating DatetimeIndex

- `pd.date_range(start=None, end=None, periods=None, freq=None, ...)`
 - Return a fixed frequency DatetimeIndex
 - `start`: left bound for generating dates
 - `end`: right bound for generating dates
 - `periods`: the number of datetime to generate
 - `freq`: the time interval between consecutive datetime values (default: 'D')

Freq string	Description	Freq string	Description
'D'	One absolute day	'M'	Calendar month end
'H'	One hour	'MS'	Calendar month begin
'T' or 'min'	One minute	'BM'	Business month end
'S'	One second	'BMS'	Business month begin
'B'	Business day (weekday)	'WOM-2THU'	Second Thursday of the month
'W'	One week	'1h30min'	One and half hour

date_range() Examples (I)

- Default: everyday

```
pd.date_range('2021-1-11', '2021-1-17')
```

```
DatetimeIndex(['2021-01-11', '2021-01-12', '2021-01-13', '2021-01-14',
                 '2021-01-15', '2021-01-16', '2021-01-17'],
                dtype='datetime64[ns]', freq='D')
```

- 7 days since 2021-1-11

```
pd.date_range('2021 1 11', periods=7)
```

```
DatetimeIndex(['2021-01-11', '2021-01-12', '2021-01-13', '2021-01-14',
                 '2021-01-15', '2021-01-16', '2021-01-17'],
                dtype='datetime64[ns]', freq='D')
```

date_range() Examples (2)

- Just weekdays

```
pd.date_range('2021 1 1', '2021/1/20', freq='B')
```

```
DatetimeIndex(['2021-01-01', '2021-01-04', '2021-01-05', '2021-01-06',
                 '2021-01-07', '2021-01-08', '2021-01-11', '2021-01-12',
                 '2021-01-13', '2021-01-14', '2021-01-15', '2021-01-18',
                 '2021-01-19', '2021-01-20'],
                dtype='datetime64[ns]', freq='B')
```

- Every Sunday

```
pd.date_range('2021-1-4', '2021-2-26', freq='W-SUN')
```

```
DatetimeIndex(['2021-01-10', '2021-01-17', '2021-01-24', '2021-01-31',
                 '2021-02-07', '2021-02-14', '2021-02-21'],
                dtype='datetime64[ns]', freq='W-SUN')
```

date_range() Examples (3)

- First business day every two months

```
pd.date_range('2021-1-1', '2021-12-31', freq='2BMS')
```

```
DatetimeIndex(['2021-01-01', '2021-03-01', '2021-05-03', '2021-07-01',
                 '2021-09-01', '2021-11-01'],
                dtype='datetime64[ns]', freq='2BMS')
```

- Every one and half hour

```
pd.date_range('2021-1-11 8:30', periods=7, freq='1h30min')
```

```
DatetimeIndex(['2021-01-11 08:30:00', '2021-01-11 10:00:00',
                 '2021-01-11 11:30:00', '2021-01-11 13:00:00',
                 '2021-01-11 14:30:00', '2021-01-11 16:00:00',
                 '2021-01-11 17:30:00'],
                dtype='datetime64[ns]', freq='90T')
```

Finding the Day of the Week

- `pd.DatetimeIndex.day_name(*args, ...)`

- Return the day names of the DatetimeIndex

```
idx = pd.date_range(start='2021-01-01', freq='D', periods=3)  
idx
```

```
DatetimeIndex(['2021-01-01', '2021-01-02', '2021-01-03'], dtype='datetime64[n  
s]', freq='D')
```

```
idx.day_name()
```

```
Index(['Friday', 'Saturday', 'Sunday'], dtype='object')
```

```
war = pd.date_range(start='1950-6-25', freq='D', periods=3)  
war.day_name()
```

```
Index(['Sunday', 'Monday', 'Tuesday'], dtype='object')
```

Creating Time Series Data

- Create a range of DatetimeIndex object

```
ts = pd.date_range('1/1/2021', periods=4, freq='2H')  
ts
```

```
DatetimeIndex(['2021-01-01 00:00:00', '2021-01-01 02:00:00',  
                 '2021-01-01 04:00:00', '2021-01-01 06:00:00'],  
                dtype='datetime64[ns]', freq='2H')
```

- Use the DatetimeIndex object as Pandas Series or DataFrame index

```
df = pd.DataFrame({'A':np.random.randn(len(ts)),  
                   'B':np.random.randn(len(ts))},  
                  index=ts)  
df
```

	A	B
2021-01-01 00:00:00	0.801227	1.536040
2021-01-01 02:00:00	0.004163	2.357660
2021-01-01 04:00:00	0.043119	-2.062572
2021-01-01 06:00:00	0.302858	0.446922

Example: Pandas Time Series Data (I)

- Create a dataframe: input dataset = <timestamp, access count>

```
import pandas as pd

data = {'date': ['2021-01-01 08:47:05.069722',
                 '2021-01-01 18:47:05.119994',
                 '2021-01-02 08:47:05.178768',
                 '2021-01-02 13:47:05.230071',
                 '2021-01-02 18:47:05.230071',
                 '2021-01-02 23:47:05.280592',
                 '2021-01-03 08:47:05.332662',
                 '2021-01-03 18:47:05.385109',
                 '2021-01-04 08:47:05.436523',
                 '2021-01-04 18:47:05.486877'],
        'counts': [34, 25, 26, 15, 15, 14, 26, 25, 62, 41]}
df = pd.DataFrame(data)
df
```

	date	counts
0	2021-01-01 08:47:05.069722	34
1	2021-01-01 18:47:05.119994	25
2	2021-01-02 08:47:05.178768	26
3	2021-01-02 13:47:05.230071	15
4	2021-01-02 18:47:05.230071	15
5	2021-01-02 23:47:05.280592	14
6	2021-01-03 08:47:05.332662	26
7	2021-01-03 18:47:05.385109	25
8	2021-01-04 08:47:05.436523	62
9	2021-01-04 18:47:05.486877	41

Example: Pandas Time Series Data (2)

- Convert df['date'] from string to datetime

```
df.date = pd.to_datetime(df.date)
```

date	counts
2021-01-01 08:47:05.069722	34
2021-01-01 18:47:05.119994	25
2021-01-02 08:47:05.178768	26
2021-01-02 13:47:05.230071	15
2021-01-02 18:47:05.230071	15
2021-01-02 23:47:05.280592	14
2021-01-03 08:47:05.332662	26
2021-01-03 18:47:05.385109	25
2021-01-04 08:47:05.436523	62
2021-01-04 18:47:05.486877	41

- Set df['date'] as the index

```
df = df.set_index('date')
```

Accessing Time Series Data (I)

- View data in 2021

```
df['2021']
```

	counts
date	
2021-01-01 08:47:05.069722	34
2021-01-01 18:47:05.119994	25
2021-01-02 08:47:05.178768	26
2021-01-02 13:47:05.230071	15
2021-01-02 18:47:05.230071	15
2021-01-02 23:47:05.280592	14
2021-01-03 08:47:05.332662	26
2021-01-03 18:47:05.385109	25
2021-01-04 08:47:05.436523	62
2021-01-04 18:47:05.486877	41

- View data in January 2021

```
df['2021-01']
```

	counts
date	
2021-01-01 08:47:05.069722	34
2021-01-01 18:47:05.119994	25
2021-01-02 08:47:05.178768	26
2021-01-02 13:47:05.230071	15
2021-01-02 18:47:05.230071	15
2021-01-02 23:47:05.280592	14
2021-01-03 08:47:05.332662	26
2021-01-03 18:47:05.385109	25
2021-01-04 08:47:05.436523	62
2021-01-04 18:47:05.486877	41

Accessing Time Series Data (2)

- Observations after 12:00, Jan. 3, 2021

```
df['2021/1/3 12:00':]
```

	counts
date	
2021-01-03 18:47:05.385109	25
2021-01-04 08:47:05.436523	62
2021-01-04 18:47:05.486877	41

These are the "views"!

- Observations between Jan. 1 - 2

```
df['1/1/2021':'1/2/2021']
```

	counts
date	
2021-01-01 08:47:05.069722	34
2021-01-01 18:47:05.119994	25
2021-01-02 08:47:05.178768	26
2021-01-02 13:47:05.230071	15
2021-01-02 18:47:05.230071	15
2021-01-02 23:47:05.280592	14

Accessing Time Series Data (3)

■ Extracting years

```
df['Year'] = df.index.year
```

	counts	Year
date		
2021-01-01 08:47:05.069722	34	2021
2021-01-01 18:47:05.119994	25	2021
2021-01-02 08:47:05.178768	26	2021
2021-01-02 13:47:05.230071	15	2021
2021-01-02 18:47:05.230071	15	2021
2021-01-02 23:47:05.280592	14	2021
2021-01-03 08:47:05.332662	26	2021
2021-01-03 18:47:05.385109	25	2021
2021-01-04 08:47:05.436523	62	2021
2021-01-04 18:47:05.486877	41	2021

.year
.month
.day
.hour
.minute
.second
.microsecond
...

■ Extracting months

```
df['Month'] = df.index.month
```

	counts	Month
date		
2021-01-01 08:47:05.069722	34	1
2021-01-01 18:47:05.119994	25	1
2021-01-02 08:47:05.178768	26	1
2021-01-02 13:47:05.230071	15	1
2021-01-02 18:47:05.230071	15	1
2021-01-02 23:47:05.280592	14	1
2021-01-03 08:47:05.332662	26	1
2021-01-03 18:47:05.385109	25	1
2021-01-04 08:47:05.436523	62	1
2021-01-04 18:47:05.486877	41	1

Accessing Time Series Data (4)

- Truncate observations after Jan. 3, 2021

```
df.truncate(after='1/3/2021')
```

	counts
date	
2021-01-01 08:47:05.069722	34
2021-01-01 18:47:05.119994	25
2021-01-02 08:47:05.178768	26
2021-01-02 13:47:05.230071	15
2021-01-02 18:47:05.230071	15
2021-01-02 23:47:05.280592	14

- Total counts per day

```
df.resample('D').sum()
```

	counts
date	
2021-01-01	59
2021-01-02	70
2021-01-03	51
2021-01-04	103

Resampling

- The process of converting a time series from one frequency to another
 - Downsampling (similar to groupby operation): aggregating higher frequency data to lower frequency
 - Upsampling: converting lower frequency to higher frequency
- `df.resample(rule, axis=0, closed=None, label=None, ...)`
 - `rule`: the offset string or object representing target conversion
 - `axis`: axis to use for up- or down-sampling
 - `closed`: which side of bin interval is closed (default: 'right' for 'M', 'A', 'Q', 'BM', 'BA', 'BQ', and 'W', and 'left' for others)
 - `label`: which bin edge label to label bucket with (same default value as `closed`)

Downsampling

<i>closed='left'</i>	counts
2021-01-01 00:00:00	0
2021-01-01 00:01:00	1
2021-01-01 00:02:00	2
2021-01-01 00:03:00	3
2021-01-01 00:04:00	4
2021-01-01 00:05:00	5
2021-01-01 00:06:00	6
2021-01-01 00:07:00	7
2021-01-01 00:08:00	8
2021-01-01 00:09:00	9
2021-01-01 00:10:00	10
2021-01-01 00:11:00	11

```
df.resample('5min').sum()
```

```
df.resample('5min', closed='right').sum()
```

closed='right'

```
df.resample('5min', closed='right', label='right').sum()
```

	counts
2021-01-01 00:00:00	0
2021-01-01 00:05:00	15
2021-01-01 00:10:00	40
2021-01-01 00:15:00	11

`closed='left'` 9:00 9:01 9:02 9:03 9:04 9:05

`closed='right'` 9:00 9:01 9:02 9:03 9:04 9:05

label='left'

label='right'

OHLC Resampling

■ Open-High-Low-Close resampling

- Used in finance

```
df.resample('5min').ohlc()
```

	counts			
	open	high	low	close
2021-01-01 00:00:00	0	4	0	4
2021-01-01 00:05:00	5	9	5	9
2021-01-01 00:10:00	10	11	10	11

	counts
2021-01-01 00:00:00	0
2021-01-01 00:01:00	1
2021-01-01 00:02:00	2
2021-01-01 00:03:00	3
2021-01-01 00:04:00	4
2021-01-01 00:05:00	5
2021-01-01 00:06:00	6
2021-01-01 00:07:00	7
2021-01-01 00:08:00	8
2021-01-01 00:09:00	9
2021-01-01 00:10:00	10
2021-01-01 00:11:00	11

Upsampling

```
df.resample('20s').ffill()
```

	counts
2021-01-01 00:00:00	0
2021-01-01 00:00:20	0
2021-01-01 00:00:40	0
2021-01-01 00:01:00	1
2021-01-01 00:01:20	1
2021-01-01 00:01:40	1
2021-01-01 00:02:00	2
2021-01-01 00:02:20	2
2021-01-01 00:02:40	2
2021-01-01 00:03:00	3

```
df.resample('20s').asfreq()
```

	counts
2021-01-01 00:00:00	0.0
2021-01-01 00:00:20	NaN
2021-01-01 00:00:40	NaN
2021-01-01 00:01:00	1.0
2021-01-01 00:01:20	NaN
2021-01-01 00:01:40	NaN
2021-01-01 00:02:00	2.0
2021-01-01 00:02:20	NaN
2021-01-01 00:02:40	NaN
2021-01-01 00:03:00	3.0

```
df.resample('20s').bfill(limit=1)
```

	counts
2021-01-01 00:00:00	0.0
2021-01-01 00:00:20	NaN
2021-01-01 00:00:40	1.0
2021-01-01 00:01:00	1.0
2021-01-01 00:01:20	NaN
2021-01-01 00:01:40	2.0
2021-01-01 00:02:00	2.0
2021-01-01 00:02:20	NaN
2021-01-01 00:02:40	3.0
2021-01-01 00:03:00	3.0

Downsampling with Periods

```
ts = pd.date_range('2021-01-01', periods=365, freq='D')
df = pd.DataFrame({'Date':np.random.randn(365)}, index=ts)
df
```

Date	
2021-01-01	0.242348
2021-01-02	-2.324153
2021-01-03	-0.829714
2021-01-04	1.253311
2021-01-05	-0.663266
...	...
2021-12-27	0.611949
2021-12-28	1.002227
2021-12-29	1.509228
2021-12-30	1.071371
2021-12-31	-1.015844
365 rows × 1 columns	

```
df.resample('M').mean()
```

Date	
2021-01-31	-0.171708
2021-02-28	-0.171608
2021-03-31	-0.077201
2021-04-30	0.072524
2021-05-31	0.342635
2021-06-30	0.158424
2021-07-31	0.013654
2021-08-31	0.037618
2021-09-30	-0.080903
2021-10-31	-0.009613
2021-11-30	-0.020476
2021-12-31	0.100228

```
df.resample('M', kind='period').mean()
```

Date	
2021-01	-0.171708
2021-02	-0.171608
2021-03	-0.077201
2021-04	0.072524
2021-05	0.342635
2021-06	0.158424
2021-07	0.013654
2021-08	0.037618
2021-09	-0.080903
2021-10	-0.009613
2021-11	-0.020476
2021-12	0.100228

```
df.resample('Q').mean()
```

Date	
2021-03-31	-0.139125
2021-06-30	0.192859
2021-09-30	-0.009105
2021-12-31	0.023856

```
df.resample('Q', kind='period').mean()
```

Date	
2021Q1	-0.139125
2021Q2	0.192859
2021Q3	-0.009105
2021Q4	0.023856

Aggregation Functions

function	Description
<code>.asfreq()</code>	Return the values at the new freq, essentially a reindex
<code>.fillna()</code>	Fill missing values introduced by upsampling ('ffill', 'bfill', 'nearest')
<code>.ffill() / .bfill()</code>	Forward (or backward) fill the values
<code>.first() / .last()</code>	Compute first (or last) of group values
<code>.min() / .max()</code>	Compute min (or max) of group values
<code>.mean() / .median()</code>	Compute mean (or median) of groups, excluding missing values
<code>.sum()</code>	Compute sum of group values
<code>.std() / .var()</code>	Compute standard deviation (or variance) of groups, excluding missing values
<code>.ohlc()</code>	Compute open, high, low and close values of a group, excluding missing values
<code>.count()</code>	Compute count of group, excluding missing values
<code>.nunique()</code>	Return number of unique elements in the group
<code>.quantile()</code>	Return value at the given quantile
<code>.interpolate()</code>	Interpolate values according to different methods

Moving Window Functions: `rolling()`

- `df.rolling(window, min_periods=None, ...)`
 - Provide rolling window calculations
 - `window`: size of the moving window
 - `min_periods`: minimum number of observations in window required to have a value

```
ts = pd.date_range('1/1/2022', periods=8, freq='D')
df = pd.DataFrame({'Counts': np.arange(len(ts))}, index=ts)
df
```

Counts
2022-01-01 0
2022-01-02 1
2022-01-03 2
2022-01-04 3
2022-01-05 4
2022-01-06 5
2022-01-07 6
2022-01-08 7

```
df.rolling(3).mean()
```

Counts
2022-01-01 NaN
2022-01-02 NaN
2022-01-03 1.0
2022-01-04 2.0
2022-01-05 3.0
2022-01-06 4.0
2022-01-07 5.0
2022-01-08 6.0

```
df.rolling(3, min_periods=2).sum()
```

Counts
2022-01-01 NaN
2022-01-02 1.0
2022-01-03 3.0
2022-01-04 6.0
2022-01-05 9.0
2022-01-06 12.0
2022-01-07 15.0
2022-01-08 18.0

Example: Stock Data (I)

LG Electronics Inc. (066570.KS) Stock

finance.yahoo.com/quote/066570.KS/history?p=066570.KS

Search for news, symbols or companies

Sign in | Mail

LG Electronics Inc. (066570.KS)

KSE - KSE Delayed Price. Currency in KRW

130,500.00 +500.00 (+0.38%)

At close: 03:30PM KST

Historical Data

Summary Chart Conversations Statistics Historical Data Profile Financials Analysis Options Holders Sustainability

한화생명 e재테크저축보험(무)

상세보기

Time Period: Jan 01, 2021 - Jan 31, 2022 Show: Historical Prices Frequency: Daily Apply

Download

Currency in KRW

Date	Open	High	Low	Close*	Adj Close**	Volume
Jan 11, 2022	129,500.00	131,500.00	127,500.00	130,500.00	130,500.00	972,850
Jan 10, 2022	135,500.00	136,000.00	129,000.00	130,000.00	130,000.00	1,784,965
Jan 07, 2022	136,500.00	138,000.00	135,000.00	137,500.00	137,500.00	808,243
Jan 06, 2022	136,000.00	139,000.00	134,500.00	135,000.00	135,000.00	1,038,777
Jan 05, 2022	142,000.00	142,000.00	137,500.00	138,500.00	138,500.00	965,541
Jan 04, 2022	143,000.00	145,000.00	140,000.00	142,000.00	142,000.00	1,419,997
Dec 30, 2021	140,000.00	140,500.00	138,000.00	138,000.00	138,000.00	703,821
Dec 29, 2021	138,500.00	141,500.00	138,000.00	140,500.00	140,500.00	1,104,635

Nano- & Micropositioning

SmarAct

People Also Watch

Symbol	Last Price	Change	% Change
005380...	210,500.00	+1,000.00	+0.48%
Hyundai Motor Company			
034220...	24,350.00	+550.00	+2.31%
LG Display Co., Ltd.			
000660...	128,000.00	+3,500.00	+2.81%
SK hynix Inc.			
005490...	300,000.00	-4,000.00	-1.32%
POSCO			
006400...	627,000.00	+3,000.00	+0.48%
Samsung SDI Co., Ltd.			

Example: Stock Data (2)

```
lge = pd.read_csv('066570.KS.csv')  
lge.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2021-01-04	136500	144000	136500	142000	142000	1868234
1	2021-01-05	139000	140000	137000	140000	140000	1602818
2	2021-01-06	148500	150500	137000	137500	137500	5283488
3	2021-01-07	139500	155000	137000	150000	150000	9782722
4	2021-01-08	145000	151000	140500	147500	147500	9472733

Example: Stock Data (3)

```
lge.tail()
```

	Date	Open	High	Low	Close	Adj Close	Volume
249	2022-01-05	142000	142000	137500	138500	138500	965541
250	2022-01-06	136000	139000	134500	135000	135000	1038777
251	2022-01-07	136500	138000	135000	137500	137500	808243
252	2022-01-10	135500	136000	129000	130000	130000	1784965
253	2022-01-11	129500	131500	127500	130500	130500	972850

Example: Stock Data (4)

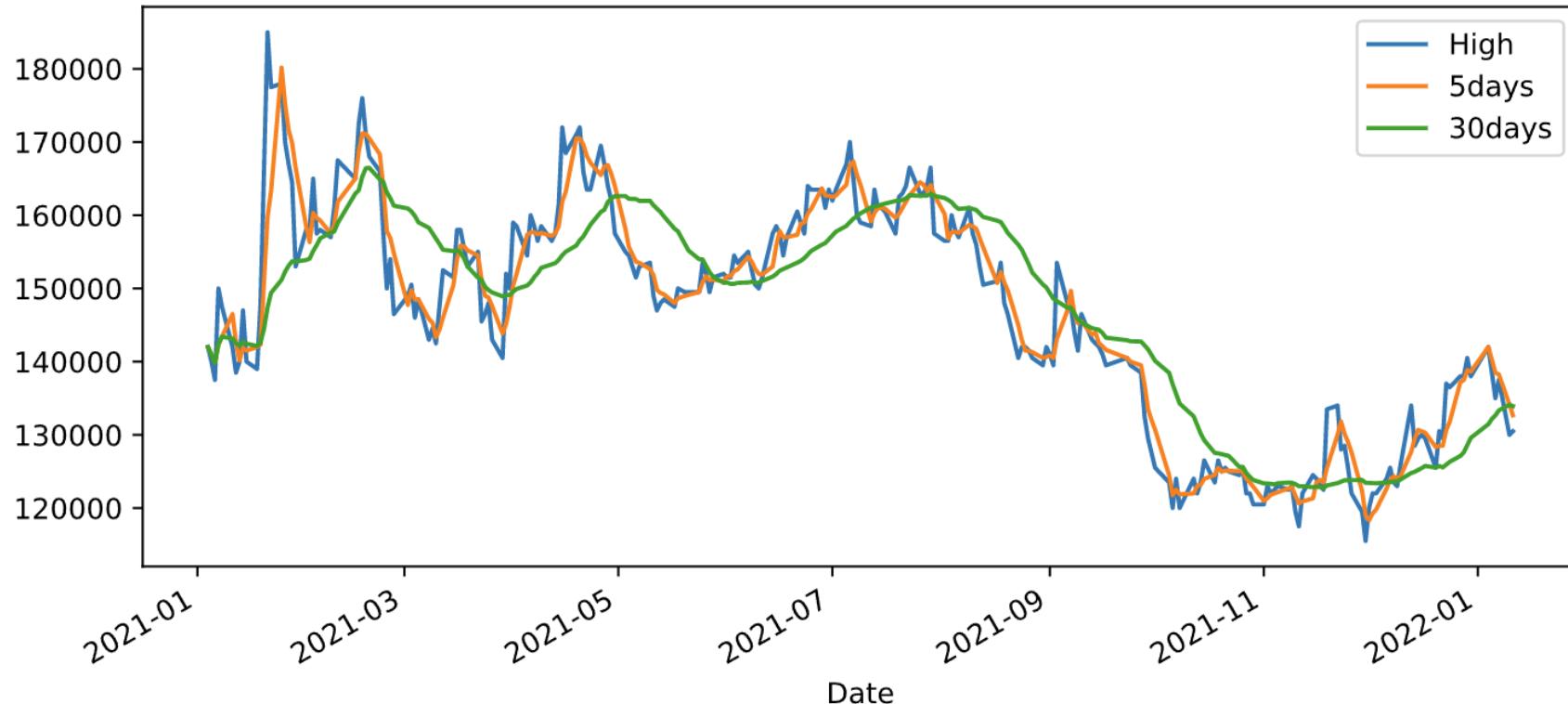
```
lge.Date = pd.to_datetime(lge.Date)
lge = lge.set_index('Date')
lge.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	254.000000	254.000000	254.000000	254.000000	254.000000	2.540000e+02
mean	147175.196850	149761.811024	144442.913386	146814.960630	146814.960630	1.620855e+06
std	15444.463011	15863.404024	14941.773980	15345.082403	15345.082403	1.693077e+06
min	116500.000000	119500.000000	115000.000000	115500.000000	115500.000000	3.600960e+05
25%	136500.000000	138625.000000	134125.000000	136625.000000	136625.000000	7.648182e+05
50%	149750.000000	152250.000000	147250.000000	149500.000000	149500.000000	1.108556e+06
75%	159000.000000	162000.000000	156500.000000	158000.000000	158000.000000	1.821710e+06
max	183000.000000	193000.000000	177500.000000	185000.000000	185000.000000	1.630495e+07

Example: Stock Data (5)

```
lge.Close.plot(figsize=(9,4))
lge.Close.rolling('5d').mean().plot()
lge.Close.rolling('30d').mean().plot()
plt.legend(['High', '5days', '30days'])
```

<matplotlib.legend.Legend at 0x21f3927f3a0>



Moving Window Functions: `ewm()`

- `df.ewm(alpha=None, min_periods=0, adjust=True, ...)`
 - Provide exponential weighted (EW) functions
 - `alpha`: specify smoothing factor α directly, $0 < \alpha \leq 1$
 - When `adjust=True` (default), the weighted moving average is calculated with $w_i = (1 - \alpha)^i$:

$$y_t = \frac{\sum_{i=0}^t w_i x_{t-i}}{\sum_{i=0}^t w_i} = \frac{x_t + (1 - \alpha)x_{t-1} + (1 - \alpha)^2x_{t-2} + \dots + (1 - \alpha)^t x_0}{1 + (1 - \alpha) + (1 - \alpha)^2 + \dots + (1 - \alpha)^t}$$

- When `adjust=False`, the weighted moving average is calculated recursively:

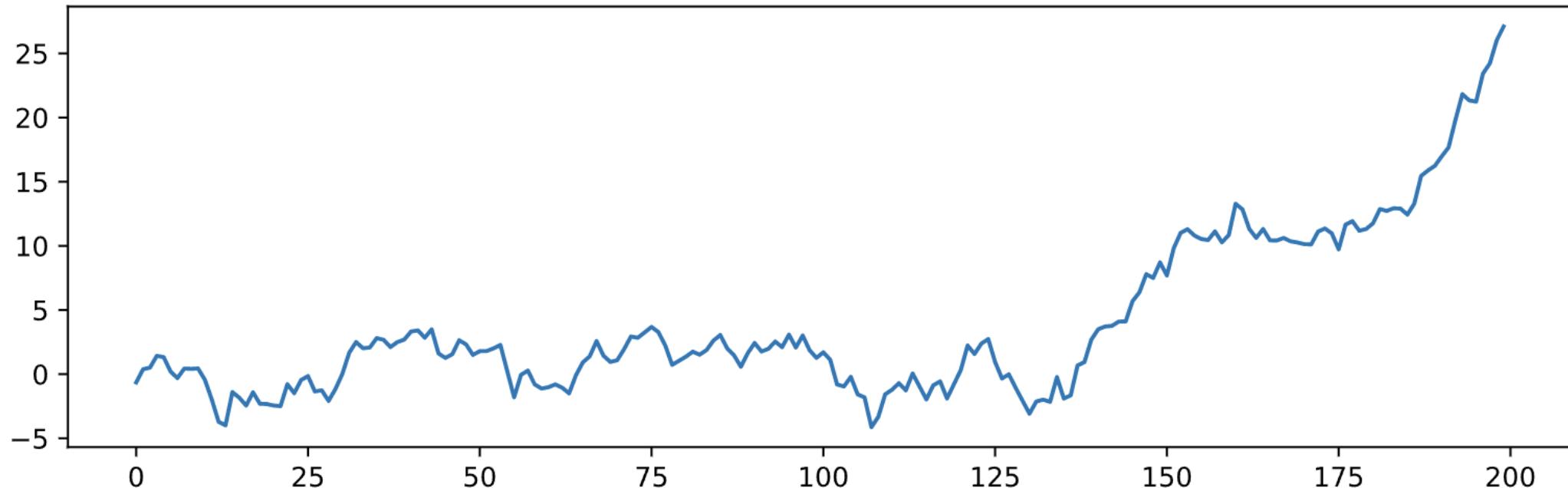
$$y_0 = x_0$$

$$y_t = (1 - \alpha)y_{t-1} + \alpha x_t$$

emw(): Example (I)

```
df = pd.DataFrame({'step': np.random.randn(200)})
df['dist'] = df['step'].cumsum()
df['dist'].plot(figsize=(10,3))
```

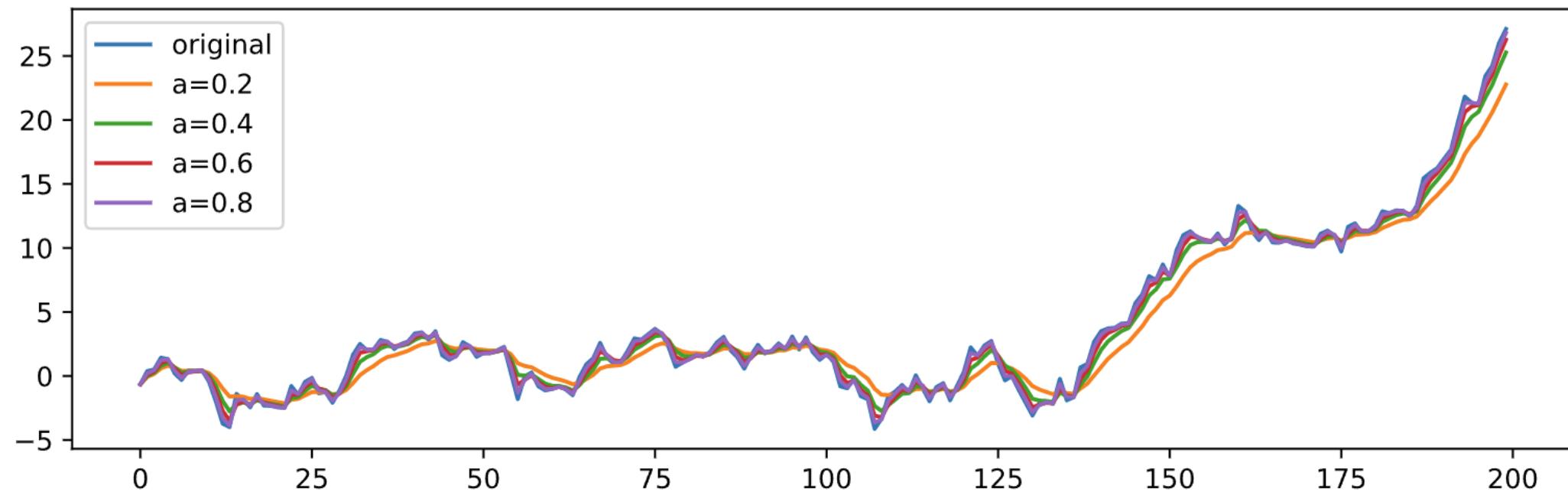
<AxesSubplot:>



emw(): Example (2)

```
df['dist'].plot(figsize=(10,3))
df['dist'].ewm(alpha=0.2).mean().plot()
df['dist'].ewm(alpha=0.4).mean().plot()
df['dist'].ewm(alpha=0.6).mean().plot()
df['dist'].ewm(alpha=0.8).mean().plot()
plt.legend(['original', 'a=0.2', 'a=0.4', 'a=0.6', 'a=0.8'])
```

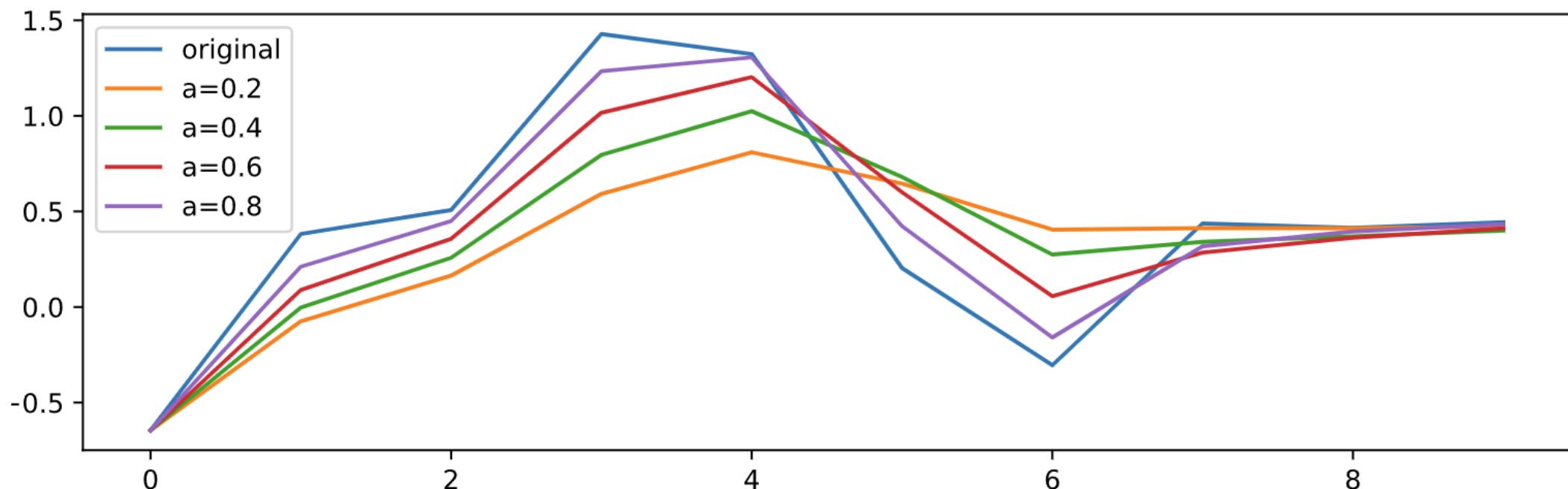
<matplotlib.legend.Legend at 0x21f391691c0>



emw(): Example (3)

```
df['dist'][:10].plot(figsize=(10,3))
df['dist'].ewm(alpha=0.2).mean()[:10].plot()
df['dist'].ewm(alpha=0.4).mean()[:10].plot()
df['dist'].ewm(alpha=0.6).mean()[:10].plot()
df['dist'].ewm(alpha=0.8).mean()[:10].plot()
plt.legend(['original', 'a=0.2', 'a=0.4', 'a=0.6', 'a=0.8'])
```

<matplotlib.legend.Legend at 0x21f39220100>



Jin-Soo Kim
[\(jinsoo.kim@snu.ac.kr\)](mailto:jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022



Python for Data Analytics

Data Preprocessing I

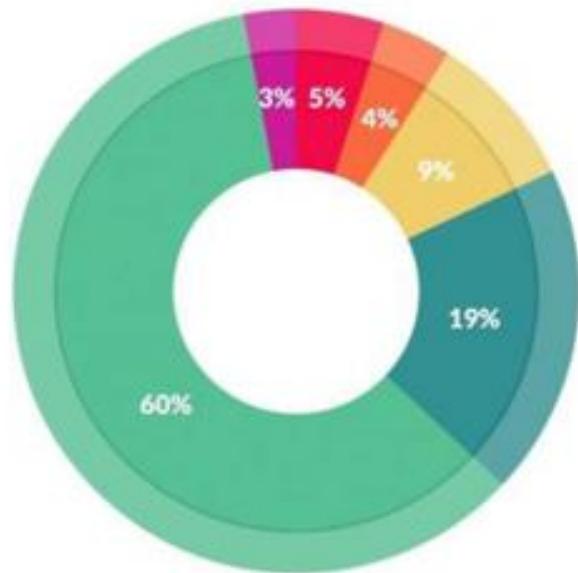
Data Collection for Data Analytics

- You will typically get data in one of four ways:
 1. Directly download a data file (or files) manually
 2. Query data from a database
 3. Query an API (usually web-based)
 4. Scrap data from a webpage

How to perform data preprocessing in Python?

Data Preprocessing

- The process of cleaning up the messy raw data for analysis
- Repetitive and tedious work
- Direct impact on analysis result and model performance
- Data collection and preprocessing occupy up to 80% of the analysis time



What data scientists spend the most time doing

- Building training set: 3%
- Cleaning and organizing data: 60%
- Collecting data set: 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%

Outline

- Handling Missing Values
- Handling Outliers
- SK-Learn
- Imputation of Missing Values
- Data Encoding

Handling Missing Values

Handling Missing Values

- Contact the data source to correct the missing values
 - Especially for human or mechanical errors
- Drop the column or row that contains missing values
 - Easy, but the dataset size should be large
- Replace the missing value with another value
 - For numerical data: mean, median, mode (most frequent) or zero
 - For categorical data: mode
 - Use interpolation
 - Requires domain knowledge
- Leave the missing value as it is

Dataset for Handling Missing Values

- User behavior and payment data in a shared file system
- 200,000 entries with 22 columns
- Column information

- rowid
- iduser: User ID
- mdutype: payment type
- group: payment info (mdu: paid user, sdu: free user)
- view/edit/share/search/cowork counts
- add/del/move/rename counts
- other user behaviors

```
df = pd.read_csv('testset.csv', index_col=0)
```

Dataset Head & Tail

```
df.head()
```

	iduser	mdutype	group	viewCount	editCount	shareCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
0	10100018739106	NaN	sdu	12.0	0.0	0.0	...	0.0	0.0	3504812.0	0.0	0.0	3504812.0
1	10100037810674	NaN	sdu	23.0	0.0	0.0	...	0.0	0.0	17123098.0	0.0	0.0	17123098.0
2	10100036273719	NaN	sdu	4.0	0.0	0.0	...	0.0	0.0	2234363.0	0.0	0.0	2234363.0
3	10100027752244	NaN	sdu	6.0	0.0	1.0	...	2.0	0.0	602361.0	210114.0	0.0	812475.0
4	10100000624840	NaN	sdu	Nan	Nan	Nan	...	Nan	Nan	Nan	Nan	Nan	Nan

5 rows × 22 columns

```
df.tail()
```

	iduser	mdutype	group	viewCount	editCount	shareCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
199995	10100014533282	NaN	sdu	37.0	0.0	2.0	...	7.0	0.0	13064406.0	1922364.0	0.0	14986770.0
199996	10100037382422	a2p	mdu	6.0	0.0	0.0	...	0.0	0.0	15936676.0	0.0	0.0	15936676.0
199997	10100024157271	NaN	sdu	32.0	0.0	0.0	...	0.0	0.0	7305871.0	0.0	0.0	7305871.0
199998	10100022150627	NaN	sdu	18.0	0.0	0.0	...	0.0	0.0	53352144.0	0.0	0.0	53352144.0
199999	10100021804275	NaN	sdu	3.0	0.0	0.0	...	0.0	0.0	95232.0	0.0	0.0	95232.0

5 rows × 22 columns

Set Index

```
df.set_index('iduser', inplace=True)
```

	iduser	mdutype	group	viewCount	editCount	shareCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
0	10100018739106	NaN	sdu	12.0	0.0	0.0	...	0.0	0.0	3504812.0	0.0	0.0	3504812.0
1	10100037810674	NaN	sdu	23.0	0.0	0.0	...	0.0	0.0	17123098.0	0.0	0.0	17123098.0
2	10100036273719	NaN	sdu	4.0	0.0	0.0	...	0.0	0.0	2234363.0	0.0	0.0	2234363.0
3	10100027752244	NaN	sdu	6.0	0.0	1.0	...	2.0	0.0	602361.0	210114.0	0.0	812475.0
4	10100000624840	NaN	sdu	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 22 columns



iduser	mdutype	group	viewCount	editCount	shareCount	searchCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	tra
10100018739106	NaN	sdu	12.0	0.0	0.0	0.0	...	0.0	0.0	3504812.0	0.0	0.0	3504812.0
10100037810674	NaN	sdu	23.0	0.0	0.0	1.0	...	0.0	0.0	17123098.0	0.0	0.0	17123098.0
10100036273719	NaN	sdu	4.0	0.0	0.0	0.0	...	0.0	0.0	2234363.0	0.0	0.0	2234363.0
10100027752244	NaN	sdu	6.0	0.0	1.0	0.0	...	2.0	0.0	602361.0	210114.0	0.0	812475.0
10100000624840	NaN	sdu	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 21 columns

Identifying Missing Values

- Check DataFrame information

```
df.info()  
  
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 200000 entries, 10100018739106 to 10100021804275  
Data columns (total 21 columns):  
 mdutype      9328 non-null object  
 group        200000 non-null object  
 viewCount    165369 non-null float64  
 editCount    165369 non-null float64  
 shareCount   165369 non-null float64  
 searchCount  165369 non-null float64  
 coworkCount  165369 non-null float64  
 add          63166 non-null float64  
 del          63166 non-null float64  
 move         63166 non-null float64  
 rename       63166 non-null float64  
 adddir       63166 non-null float64  
 movedir     63166 non-null float64  
 visdays     184306 non-null float64  
 openCount   149090 non-null float64  
 saveCount   149090 non-null float64  
 exportCount 149090 non-null float64  
 viewTraffic 149090 non-null float64  
 editTraffic 149090 non-null float64  
 exportTraffic 149090 non-null float64  
 traffic     149090 non-null float64  
 dtypes: float64(19), object(2)  
memory usage: 33.6+ MB
```

- Check per-column missing values

	df.isnull().sum()	df.isnull().sum()/len(df)*100
mdutype	190672	95.3360
group	0	0.0000
viewCount	34631	17.3155
editCount	34631	17.3155
shareCount	34631	17.3155
searchCount	34631	17.3155
coworkCount	34631	17.3155
add	136834	68.4170
del	136834	68.4170
move	136834	68.4170
rename	136834	68.4170
addir	136834	68.4170
movedir	136834	68.4170
visdays	15694	7.8470
openCount	50910	25.4550
saveCount	50910	25.4550
exportCount	50910	25.4550
viewTraffic	50910	25.4550
editTraffic	50910	25.4550
exportTraffic	50910	25.4550
traffic	50910	25.4550
		dtype: float64
		dtype: int64

Dropping Missing Values (I)

- Remove rows having missing values

```
df_droprows = df.dropna()  
df_droprows.isnull().sum()
```

```
mdutype      0  
group        0  
viewCount    0  
editCount    0  
shareCount   0  
searchCount  0  
coworkCount  0  
add          0  
del          0  
move         0  
rename       0  
addir        0  
movedir      0  
visdays      0  
openCount    0  
saveCount    0  
exportCount  0  
viewTraffic  0  
editTraffic  0  
exportTraffic 0  
traffic      0  
dtype: int64
```

```
df_droprows.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 2717 entries, 10100022538111 to 10100003355450  
Data columns (total 21 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          --          --          --  
 0   mdutype     2717 non-null   object    
 1   group       2717 non-null   object    
 2   viewCount   2717 non-null   float64  
 3   editCount   2717 non-null   float64  
 4   shareCount  2717 non-null   float64  
 5   searchCount 2717 non-null   float64  
 6   coworkCount 2717 non-null   float64  
 7   add          2717 non-null   float64  
 8   del          2717 non-null   float64  
 9   move         2717 non-null   float64  
 10  rename       2717 non-null   float64  
 11  adddir       2717 non-null   float64  
 12  movedir     2717 non-null   float64  
 13  visdays     2717 non-null   float64  
 14  openCount   2717 non-null   float64  
 15  saveCount   2717 non-null   float64  
 16  exportCount 2717 non-null   float64  
 17  viewTraffic 2717 non-null   float64  
 18  editTraffic 2717 non-null   float64  
 19  exportTraffic 2717 non-null   float64  
 20  traffic     2717 non-null   float64  
dtypes: float64(19), object(2)  
memory usage: 467.0+ KB
```

Dropping Missing Values (2)

- Remove the whole column containing missing values

```
df_dropcols = df.dropna(axis=1)
df_dropcols.isnull().sum()
```

```
group      0
dtype: int64
```

```
df_dropcols.head()
```

	group
	iduser
1	10100018739106 sdu
2	10100037810674 sdu
3	10100036273719 sdu
4	10100027752244 sdu
5	10100000624840 sdu

```
df_dropcols.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 10100021804275
Data columns (total 1 columns):
 #   Column   Non-Null Count   Dtype  
--- 
 0   group    200000 non-null   object 
dtypes: object(1)
memory usage: 3.1+ MB
```

Dropping Missing Values (3)

- Remove only missing rows related to the column 'viewCount'

```
df_dropView = df.dropna(subset=['viewCount'], axis=0)
df_dropView.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 165369 entries, 10100018739106 to 101000218
04275
Data columns (total 21 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   mdtype       7338 non-null   object 
 1   group        165369 non-null object 
 2   viewCount    165369 non-null float64
 3   editCount    165369 non-null float64
 4   shareCount   165369 non-null float64
 5   searchCount  165369 non-null float64
 6   coworkCount  165369 non-null float64
 7   add          61545 non-null   float64
 8   del          61545 non-null   float64
 9   move         61545 non-null   float64
 10  rename       61545 non-null   float64
 11  adddir       61545 non-null   float64
 12  movedir     61545 non-null   float64
 13  visdays     161772 non-null  float64
 14  openCount   149090 non-null  float64
 15  saveCount   149090 non-null  float64
 16  exportCount 149090 non-null  float64
 17  viewTraffic 149090 non-null  float64
 18  editTraffic 149090 non-null  float64
 19  exportTraffic 149090 non-null  float64
 20  traffic     149090 non-null  float64
dtypes: float64(19), object(2)
memory usage: 27.8+ MB
```

```
df_dropView.isnull().sum() / len(df_dropView) * 100
mdtype           95.562651
group            0.000000
viewCount        0.000000
editCount        0.000000
shareCount       0.000000
searchCount      0.000000
coworkCount     0.000000
add              62.783230
del              62.783230
move             62.783230
rename           62.783230
addir            62.783230
movedir          62.783230
visdays          2.175136
openCount         9.844046
saveCount         9.844046
exportCount       9.844046
viewTraffic       9.844046
editTraffic       9.844046
exportTraffic     9.844046
traffic           9.844046
dtype: float64
```

- subset: labels along the other axis to consider

Dropping Missing Values (4)

- Remove columns whose percentage of non-null values is below the threshold (e.g., 80%)
 - thresh*: require that many non-NA values

```
df_dropThre = df.dropna(thresh=0.8*len(df), axis=1)
df_dropThre.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 101000218
04275
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   group        200000 non-null   object  
 1   viewCount    165369 non-null   float64 
 2   editCount    165369 non-null   float64 
 3   shareCount   165369 non-null   float64 
 4   searchCount  165369 non-null   float64 
 5   coworkCount  165369 non-null   float64 
 6   visdays      184306 non-null   float64 
dtypes: float64(6), object(1)
memory usage: 12.2+ MB
```

```
df_dropThre.isnull().sum()/len(df_dropThre)*100
```

group	0.0000
viewCount	17.3155
editCount	17.3155
shareCount	17.3155
searchCount	17.3155
coworkCount	17.3155
visdays	7.8470

dtype: float64

Replacing Missing Values (I)

■ `df.select_dtypes([include], [exclude])`

- Return a subset of the DataFrame's columns based on the column dtypes
- *include, exclude*: A selection of dtypes or strings to be included/excluded
(`np.number`: all numeric types, `np.datetime64`: datetimes, `object`: strings etc.)

```
numeric = df.select_dtypes(include=np.number)
numeric_cols = numeric.columns
numeric_cols

Index(['viewCount', 'editCount', 'shareCount', 'searchCount', 'coworkCount',
       'add', 'del', 'move', 'rename', 'addir', 'movedir', 'visdays',
       'openCount', 'saveCount', 'exportCount', 'viewTraffic', 'editTraffic',
       'exportTraffic', 'traffic'],
      dtype='object')
```

Replacing Missing Values (2)

- Replace with a value using `fillna()`

- `fillna(value)`
- `fillna(method='ffill')`
== `fillna(method='pad')`
- `fillna(method='bfill')`
== `fillna(method='backfill')`
- `fillna(method='ffill', limit=3)`

```
df[numeric_cols] = df[numeric_cols].fillna(0)
df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 10100021804275
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   mdutype          9328 non-null    object 
 1   group            200000 non-null   object 
 2   viewCount        200000 non-null   float64
 3   editCount        200000 non-null   float64
 4   shareCount       200000 non-null   float64
 5   searchCount      200000 non-null   float64
 6   coworkCount      200000 non-null   float64
 7   add              200000 non-null   float64
 8   del              200000 non-null   float64
 9   move             200000 non-null   float64
 10  rename           200000 non-null   float64
 11  adddir           200000 non-null   float64
 12  movedir          200000 non-null   float64
 13  visdays          200000 non-null   float64
 14  openCount         200000 non-null   float64
 15  saveCount         200000 non-null   float64
 16  exportCount       200000 non-null   float64
 17  viewTraffic       200000 non-null   float64
 18  editTraffic        200000 non-null   float64
 19  exportTraffic      200000 non-null   float64
 20  traffic            200000 non-null   float64
dtypes: float64(19), object(2)
memory usage: 33.6+ MB
```

Replacing Missing Values (3)

■ Replace with the average value

```
df[numeric_cols] = df[numeric_cols].fillna(df.mean())
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 10100021804275
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   mdutype          9328 non-null    object  
 1   group            200000 non-null   object  
 2   viewCount        200000 non-null   float64 
 3   editCount        200000 non-null   float64 
 4   shareCount       200000 non-null   float64 
 5   searchCount      200000 non-null   float64 
 6   coworkCount      200000 non-null   float64 
 7   add              200000 non-null   float64 
 8   del              200000 non-null   float64 
 9   move             200000 non-null   float64 
 10  rename           200000 non-null   float64 
 11  adddir           200000 non-null   float64 
 12  movedir          200000 non-null   float64 
 13  visdays          200000 non-null   float64 
 14  openCount         200000 non-null   float64 
 15  saveCount         200000 non-null   float64 
 16  exportCount       200000 non-null   float64 
 17  viewTraffic       200000 non-null   float64 
 18  editTraffic       200000 non-null   float64 
 19  exportTraffic     200000 non-null   float64 
 20  traffic           200000 non-null   float64 
dtypes: float64(19), object(2)
memory usage: 33.6+ MB
```

```
df.isnull().sum()/len(df)*100
```

mdutype	95.336
group	0.000
viewCount	0.000
editCount	0.000
shareCount	0.000
searchCount	0.000
coworkCount	0.000
add	0.000
del	0.000
move	0.000
rename	0.000
addir	0.000
movedir	0.000
visdays	0.000
openCount	0.000
saveCount	0.000
exportCount	0.000
viewTraffic	0.000
editTraffic	0.000
exportTraffic	0.000
traffic	0.000

```
dtype: float64
```

Replacing Missing Values (4)

■ Use linear interpolation

```
df[numeric_cols] = df[numeric_cols].interpolate(method='linear', limit_direction='forward')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 10100021804275
Data columns (total 21 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   mdtype       9328 non-null   object 
 1   group        200000 non-null  object 
 2   viewCount    200000 non-null  float64
 3   editCount    200000 non-null  float64
 4   shareCount   200000 non-null  float64
 5   searchCount  200000 non-null  float64
 6   coworkCount  200000 non-null  float64
 7   add          199999 non-null  float64
 8   del          199999 non-null  float64
 9   move         199999 non-null  float64
 10  rename       199999 non-null  float64
 11  adddir       199999 non-null  float64
 12  movedir     199999 non-null  float64
 13  visdays     200000 non-null  float64
 14  openCount    200000 non-null  float64
 15  saveCount   200000 non-null  float64
 16  exportCount  200000 non-null  float64
 17  viewTraffic 200000 non-null  float64
 18  editTraffic  200000 non-null  float64
 19  exportTraffic 200000 non-null  float64
 20  traffic      200000 non-null  float64
dtypes: float64(19), object(2)
memory usage: 33.6+ MB
```

before

viewCount editCount		
iduser		
10100039309679	40.0	10.0
10100022148600	44.0	0.0
10100011509371	NaN	NaN
10100001192660	12.0	1.0
10100028428084	138.0	0.0

after

viewCount editCount		
iduser		
10100039309679	40.0	10.0
10100022148600	44.0	0.0
10100011509371	28.0	0.5
10100001192660	12.0	1.0
10100028428084	138.0	0.0

Replacing Missing Values (5)

- Replace the categorical data with the most frequent value

```
df.mdutype.fillna(df.mdutype.mode()[0], inplace=True)  
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 200000 entries, 10100018739106 to 10100021804275  
Data columns (total 21 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          -----          ----  
 0   mdutype     200000 non-null  object    
 1   group       200000 non-null  object    
 2   viewCount   200000 non-null  float64  
 3   editCount   200000 non-null  float64  
 4   shareCount  200000 non-null  float64  
 5   searchCount 200000 non-null  float64  
 6   coworkCount 200000 non-null  float64  
 7   add          200000 non-null  float64  
 8   del          200000 non-null  float64  
 9   move         200000 non-null  float64  
 10  rename       200000 non-null  float64  
 11  adddir       200000 non-null  float64  
 12  movedir     200000 non-null  float64  
 13  visdays     200000 non-null  float64  
 14  openCount    200000 non-null  float64  
 15  saveCount   200000 non-null  float64  
 16  exportCount  200000 non-null  float64  
 17  viewTraffic 200000 non-null  float64  
 18  editTraffic  200000 non-null  float64  
 19  exportTraffic 200000 non-null  float64  
 20  traffic      200000 non-null  float64  
dtypes: float64(19), object(2)  
memory usage: 33.6+ MB
```

before

```
df.mdutype.value_counts()
```

```
a2p    7094  
mul   1650  
p2a    584  
Name: mdutype, dtype: int64
```

after

```
df.mdutype.value_counts()
```

```
a2p    197766  
mul   1650  
p2a    584  
Name: mdutype, dtype: int64
```

```
df.isnull().sum()/len(df)*100
```

mdutype	0.0
group	0.0
viewCount	0.0
editCount	0.0
shareCount	0.0
searchCount	0.0
coworkCount	0.0
add	0.0
del	0.0
move	0.0
rename	0.0
addir	0.0
movedir	0.0
visdays	0.0
openCount	0.0
saveCount	0.0
exportCount	0.0
viewTraffic	0.0
editTraffic	0.0
exportTraffic	0.0
traffic	0.0

Duplicate Rows (I)

■ Finding duplicate rows

```
df.duplicated()
```

```
iduser
10100018739106    False
10100037810674    False
10100036273719    False
10100027752244    False
10100000624840    False
...
10100014533282    False
10100037382422    False
10100024157271    False
10100022150627    False
10100021804275    True
Length: 200000, dtype: bool
```

```
df.duplicated(['viewCount', 'editCount'])
```

```
iduser
10100018739106    False
10100037810674    False
10100036273719    False
10100027752244    False
10100000624840    False
...
10100014533282    True
10100037382422    True
10100024157271    True
10100022150627    True
10100021804275    True
Length: 200000, dtype: bool
```

```
df['viewCount'].duplicated().sum()
```

```
199512
```

```
len(df['viewCount'].unique())
```

```
488
```

```
df.duplicated().sum()
```

```
43139
```

```
df.duplicated(['viewCount', 'editCount']).sum()
```

```
194734
```

Duplicate Rows (2)

■ Extracting duplicate rows

```
df.loc[df.duplicated(),:]
```

	iduser	mdtype	group	viewCount	editCount	shareCount	searchCount	coworkCount	add	del	move	...	addir	movedir	visdays	openCount	:
	10100009612042	NaN	sdu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
	10100017397956	NaN	sdu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
	10100030949780	NaN	sdu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
	10100025107423	NaN	sdu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
	10100011509371	NaN	sdu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	4.0	NaN	

	10100030252788	NaN	sdu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	2.0	NaN	
	10100039519206	NaN	sdu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
	10100032389548	NaN	sdu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	9.0	NaN	
	10100022852685	NaN	sdu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	8.0	NaN	
	10100021804275	NaN	sdu	3.0	0.0	0.0	0.0	0.0	0.0	NaN	NaN	...	NaN	NaN	3.0	3.0	

43139 rows × 21 columns



Duplicate Rows (3)

■ Dropping duplicate rows

`df.drop_duplicates(['add', 'del'])`
for considering certain columns only

```
df.drop_duplicates()
```

iduser	mdtype	group	viewCount	editCount	shareCount	searchCount	coworkCount	add	del	move	... adddir	movedir	visdays	openCount	
10100018739106	NaN	sdu	12.0	0.0	0.0	0.0	0.0	NaN	NaN	NaN	...	NaN	NaN	6.0	12.0
10100037810674	NaN	sdu	23.0	0.0	0.0	1.0	0.0	13.0	0.0	0.0	...	0.0	0.0	8.0	23.0
10100036273719	NaN	sdu	4.0	0.0	0.0	0.0	0.0	NaN	NaN	NaN	...	NaN	NaN	4.0	4.0
10100027752244	NaN	sdu	6.0	0.0	1.0	0.0	0.0	NaN	NaN	NaN	...	NaN	NaN	5.0	6.0
10100000624840	NaN	sdu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	24.0	NaN
...
10100037511235	NaN	sdu	21.0	0.0	0.0	0.0	0.0	NaN	NaN	NaN	...	NaN	NaN	8.0	21.0
10100014533282	NaN	sdu	37.0	0.0	2.0	0.0	0.0	25.0	0.0	0.0	...	0.0	0.0	14.0	37.0
10100037382422	a2p	mdu	6.0	0.0	0.0	6.0	0.0	NaN	NaN	NaN	...	NaN	NaN	18.0	6.0
10100024157271	NaN	sdu	32.0	0.0	0.0	0.0	0.0	28.0	0.0	0.0	...	0.0	0.0	18.0	32.0
10100022150627	NaN	sdu	18.0	0.0	0.0	0.0	0.0	20.0	0.0	0.0	...	0.0	0.0	9.0	18.0

156861 rows × 21 columns

Handling Outliers

Visualizing Outliers (I)

■ Preparing dataset

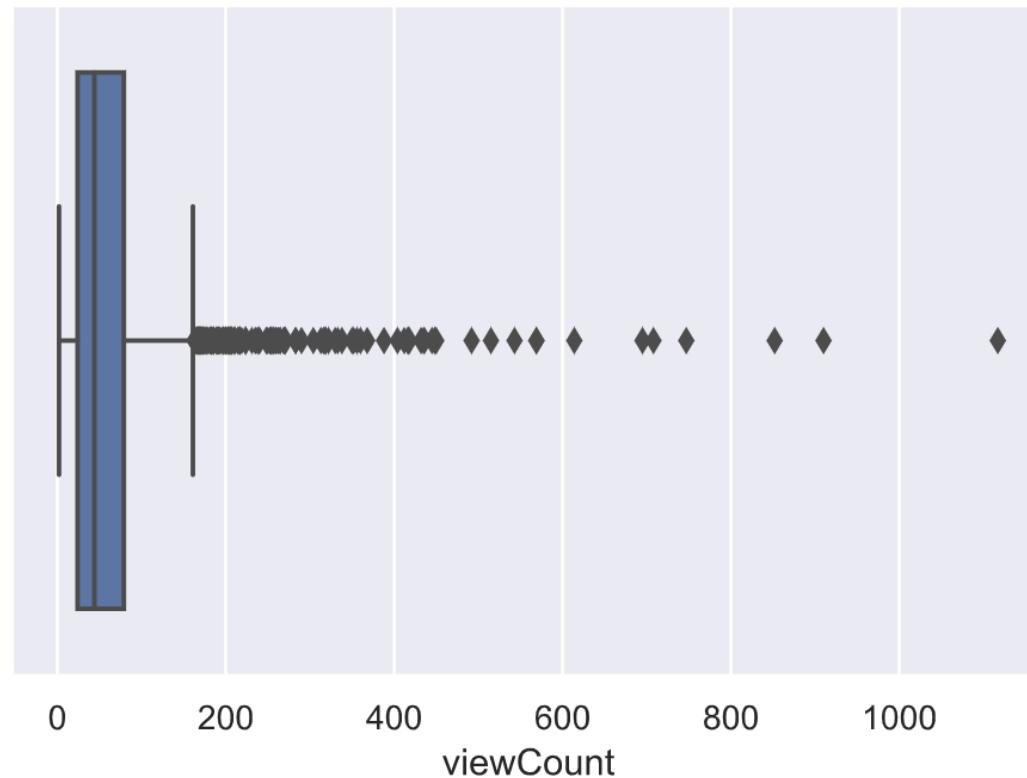
```
df = pd.read_csv('testset.csv', index_col=0)
df.set_index('iduser', inplace=True)
df = df.dropna(how='any')
df = df.select_dtypes(include=np.number)
df.head()
```

	viewCount	editCount	shareCount	searchCount	coworkCount	...	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
iduser											
10100022538111	35.0	68.0	1.0	0.0	0.0	...	0.0	934912.0	92672.0	0.0	1027584.0
10100039309679	40.0	10.0	2.0	3.0	0.0	...	1.0	2719076.0	88398.0	0.0	2807474.0
10100037687198	44.0	1.0	0.0	0.0	0.0	...	0.0	28866560.0	6246400.0	0.0	35112960.0
10100017371337	95.0	19.0	0.0	12.0	0.0	...	0.0	25970473.0	8772492.0	0.0	34742965.0
10100013627062	75.0	15.0	0.0	3.0	0.0	...	0.0	1289983.0	271360.0	0.0	1561343.0

Visualizing Outliers (2)

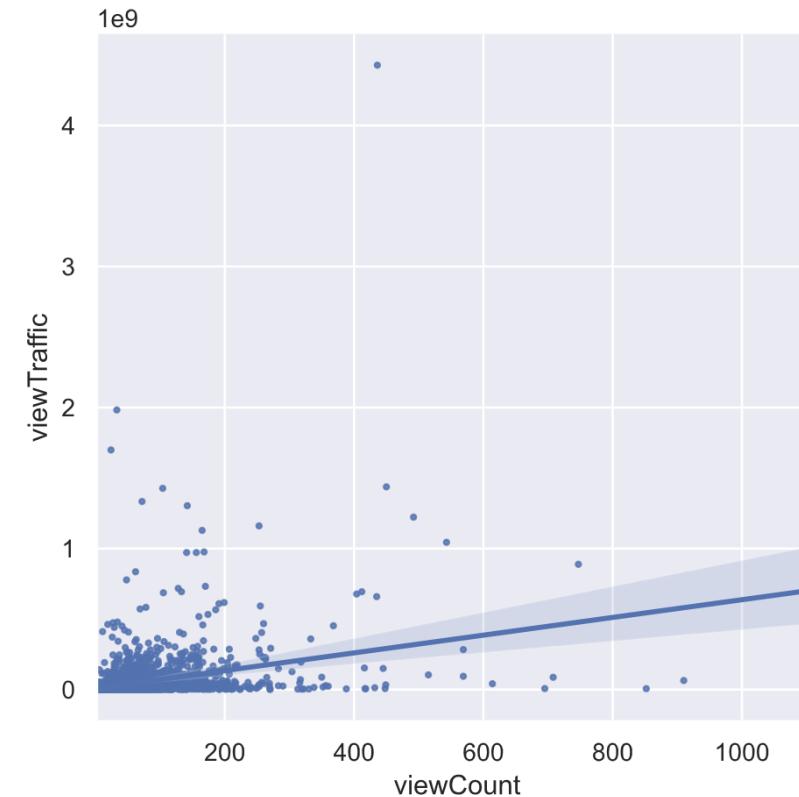
```
sns.set(style='darkgrid')
sns.boxplot(x=df.viewCount)
```

```
<AxesSubplot:xlabel='viewCount'>
```



```
sns.lmplot(x='viewCount', y='viewTraffic',
            data=df, scatter_kws={"s": 5})
```

```
<seaborn.axisgrid.FacetGrid at 0x226858376a0>
```

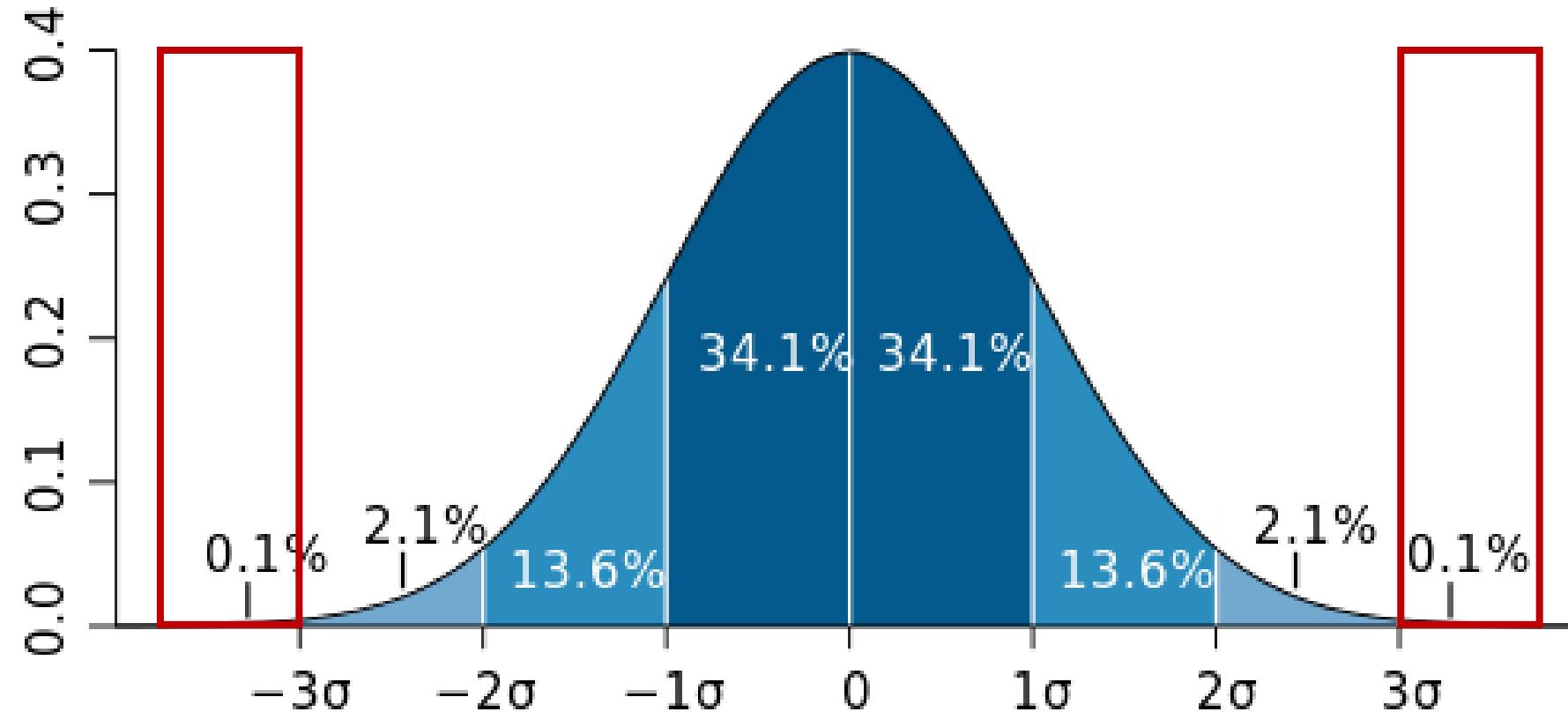


Handling Outliers

- Clipping using Normal Distribution
- Clipping using IQR Score

Clipping using Normal Distribution (I)

- Remove outliers outside of the $\mu \pm n\sigma$ (e.g., $n = 3$)



Clipping using Normal Distribution (2)

■ Clipping data

```
for c in df.columns:  
    df = df[np.abs(df[c] - df[c].mean()) <= 3 * df[c].std()]  
df.head()
```

$$= |x - \mu| \leq 3\sigma$$

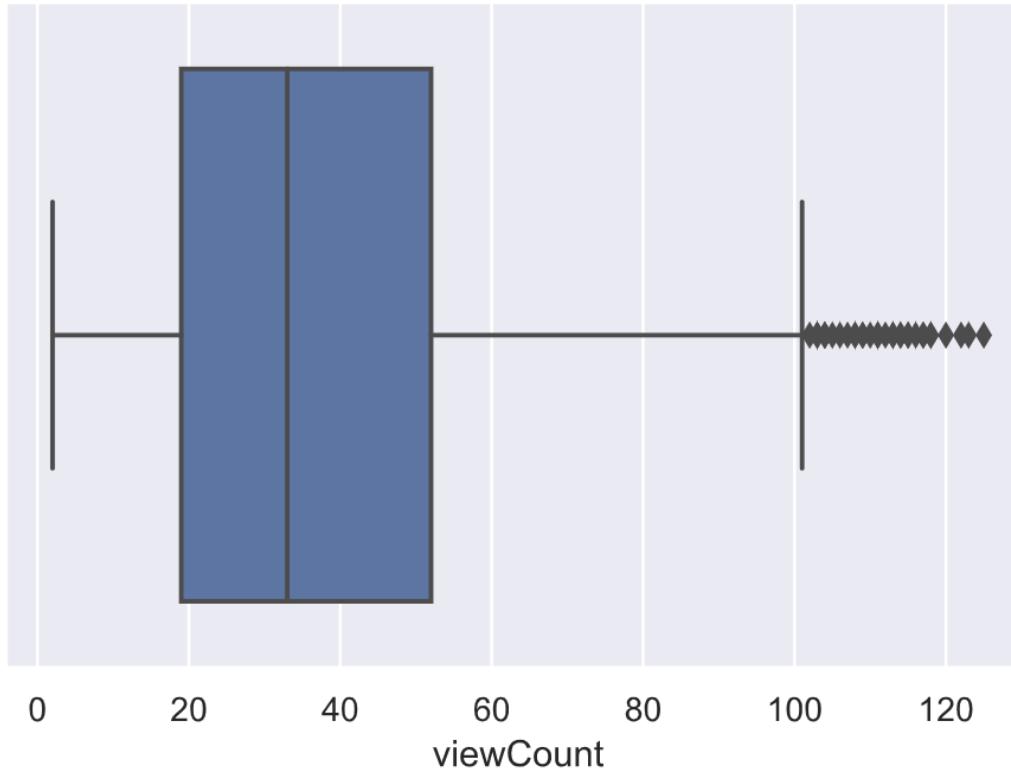
	viewCount	editCount	shareCount	searchCount	coworkCount	...	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
iduser											
10100039309679	40.0	10.0	2.0	3.0	0.0	...	1.0	2719076.0	88398.0	0.0	2807474.0
10100037687198	44.0	1.0	0.0	0.0	0.0	...	0.0	28866560.0	6246400.0	0.0	35112960.0
10100017371337	95.0	19.0	0.0	12.0	0.0	...	0.0	25970473.0	8772492.0	0.0	34742965.0
10100013627062	75.0	15.0	0.0	3.0	0.0	...	0.0	1289983.0	271360.0	0.0	1561343.0
10100012989173	49.0	0.0	2.0	13.0	0.0	...	0.0	2071600.0	51129.0	0.0	2122729.0

Clipping using Normal Distribution (3)

■ After clipping

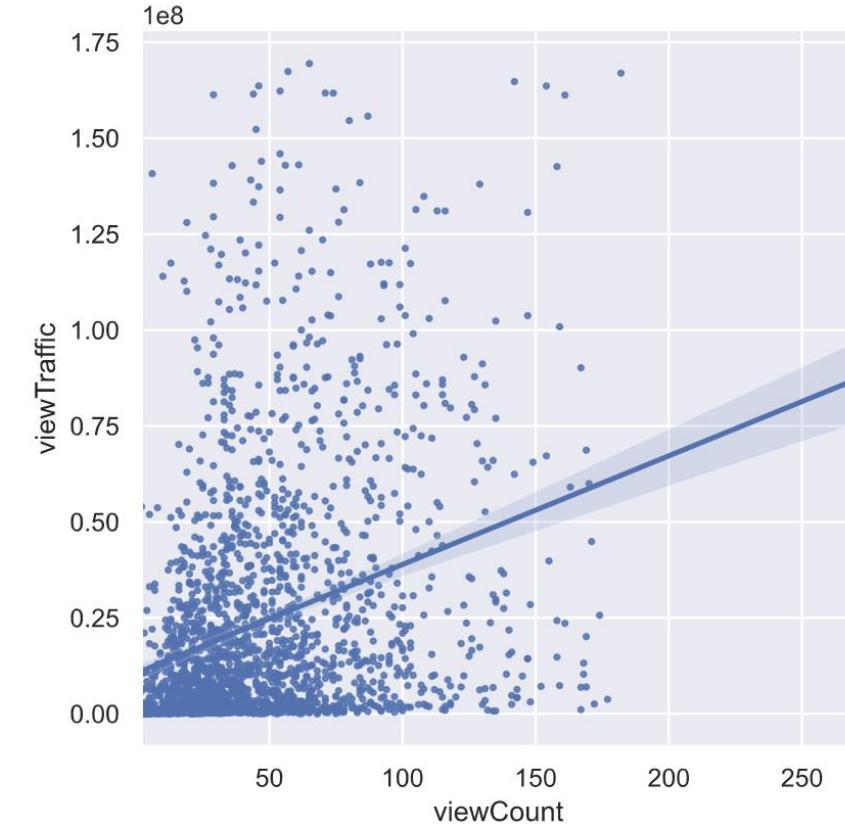
```
sns.boxplot(x=df.viewCount)
```

```
<AxesSubplot:xlabel='viewCount'>
```



```
sns.lmplot(x='viewCount', y='viewTraffic',  
           data=df, scatter_kws={"s": 5})
```

```
<seaborn.axisgrid.FacetGrid at 0x22685944f10>
```



Z-Score (Standard Score)

- The number of standard deviations by which the value of a raw score is above or below the mean value
 - Re-scale data to normal distribution
 - In most cases, the data with $|z| > 3$ are considered as outliers -- clipping those data is same as clipping data such as $|x - \mu| > 3\sigma$

$$z = \frac{x - \mu}{\sigma}$$

```
from scipy import stats

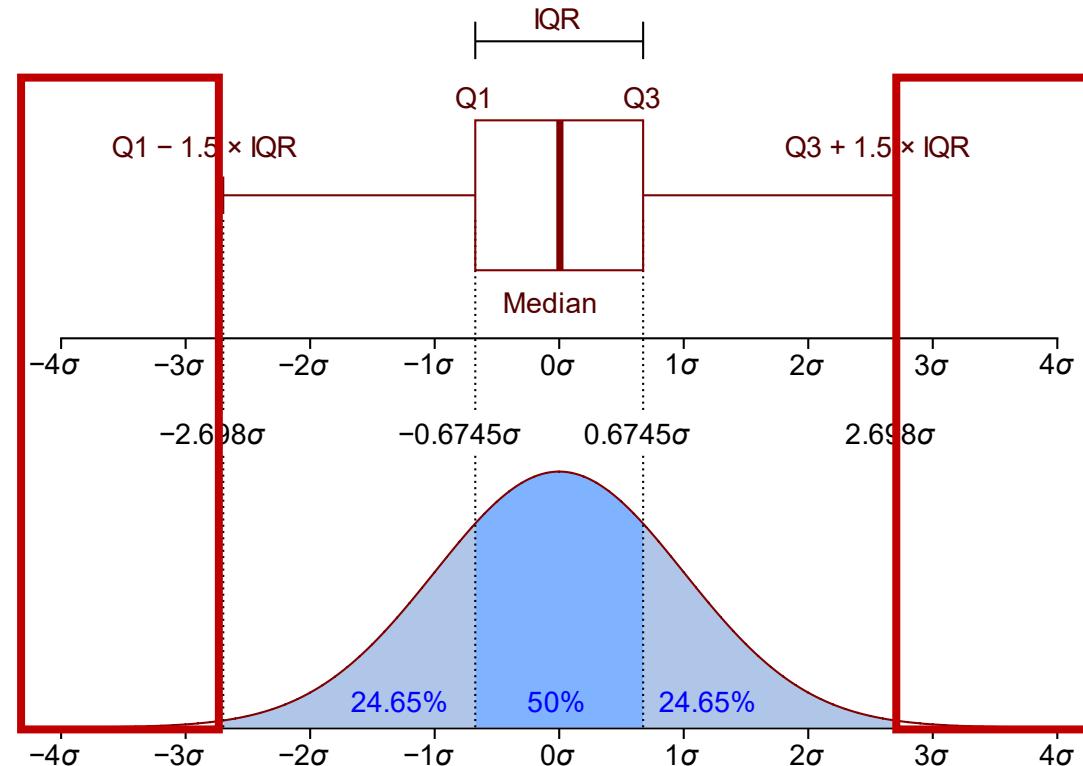
for c in df.columns:
    df = df[np.fabs(stats.zscore(df[c])) <= 3]
df.head()
```

	viewCount	editCount	shareCount	searchCount	coworkCount	...	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
iduser											
10100039309679	40.0	10.0	2.0	3.0	0.0	...	1.0	2719076.0	88398.0	0.0	2807474.0
10100037687198	44.0	1.0	0.0	0.0	0.0	...	0.0	28866560.0	6246400.0	0.0	35112960.0

Clipping using IQR Score (I)

- IQR (InterQuartile Range)

- $Q_1 = 25\%$, $Q_3 = 75\%$, $IQR = Q_3 - Q_1$
- Remove data points outside of $[Q_1 - 1.5IQR, Q_3 + 1.5IQR]$



Clipping using IQR Score (2)

- **`np.percentile(a, q, [axis], ...)`**

- Compute the q -th percentile of the data along the specified axis
- a : input array
- q : percentile or sequence of percentiles to compute in $[0, 100]$
- $axis$: axis or axes along which the percentiles are computed

- **`df.quantile(q, [axis], ...)`**

- Return values at the given quantile over requested axis

```
a = np.random.randint(0, 20, 10)
a = np.sort(a)
```

```
array([ 4,  5,  5,  9, 13, 13, 14, 14, 17, 18])
```

```
np.percentile(a, 50)
```

```
13.0
```

```
np.percentile(a, [25, 75])
```

```
array([ 6., 14.])
```

```
a[np.where(a > 14)]
```

```
array([17, 18])
```

Clipping using IQR Score (3)

■ Clipping data

```
for c in df.columns:  
    Q1 = df[c].quantile(0.25)  
    Q3 = df[c].quantile(0.75)  
    IQR = Q3 - Q1  
    df = df[(df[c] >= (Q1 - 1.5 * IQR)) & (df[c] <= (Q3 + 1.5 * IQR))]  
df.head()
```

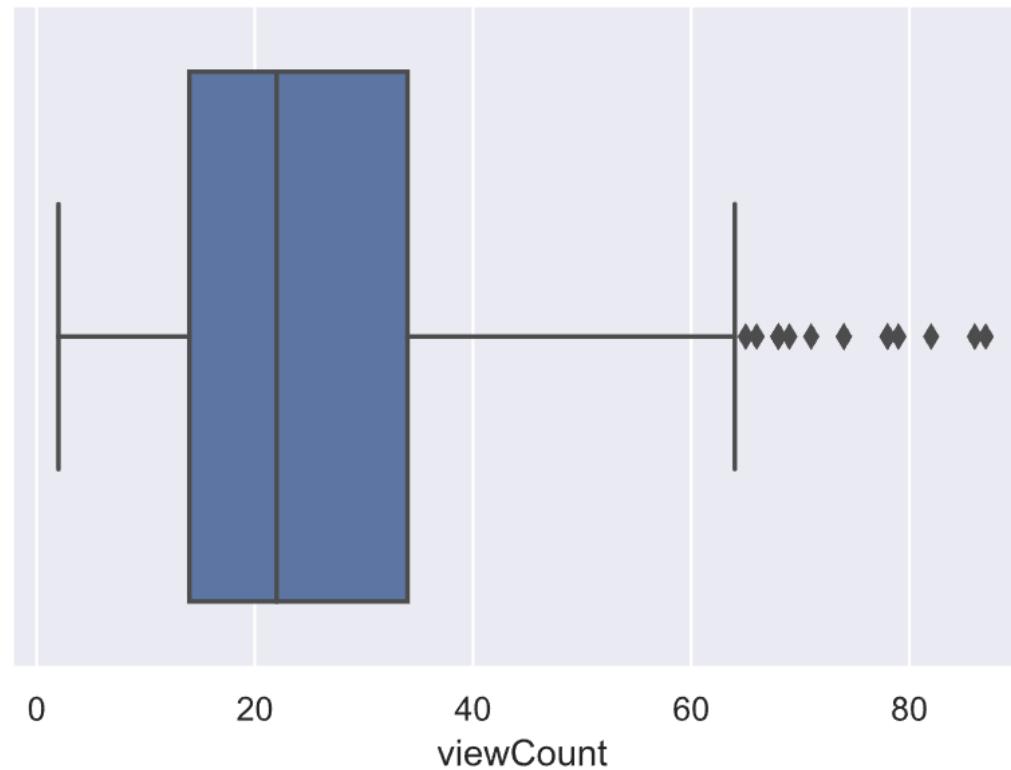
	viewCount	editCount	shareCount	searchCount	coworkCount	...	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
iduser											
10100034231482	29.0	0.0	0.0	0.0	0.0	...	0.0	6401717.0	0.0	0.0	6401717.0
10100011294549	74.0	6.0	0.0	0.0	0.0	...	0.0	8749387.0	435164.0	0.0	9184551.0
10100020127806	26.0	0.0	0.0	0.0	0.0	...	0.0	10857632.0	0.0	0.0	10857632.0
10100030084140	28.0	6.0	0.0	1.0	0.0	...	0.0	37385701.0	252178.0	0.0	37637879.0
10100027809274	14.0	0.0	0.0	0.0	0.0	...	0.0	4257850.0	304324.0	0.0	4562174.0

Clipping using IQR Score (4)

■ After clipping

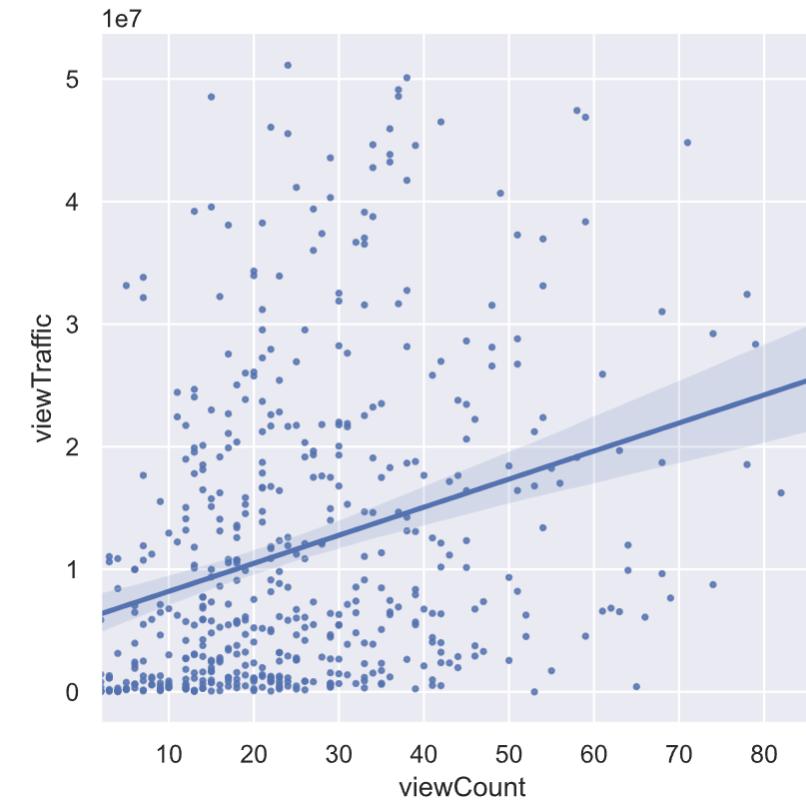
```
sns.boxplot(x=df.viewCount)
```

```
<AxesSubplot:xlabel='viewCount'>
```



```
sns.lmplot(x='viewCount', y='viewTraffic',  
           data=df, scatter_kws={"s": 5})
```

```
<seaborn.axisgrid.FacetGrid at 0x1d58ab66040>
```



SK-Learn

SK-Learn

- **SciKit (SciPy Toolkit)-learn, Sklearn, or SK-Learn**
- Open source machine learning library for Python
- Built on top of SciPy
 - Designed to interoperate with Python numerical and scientific library
- Dependency
 - NumPy, SciPy, Matplotlib
- Open source (<https://scikit-learn.org>)
 - Initially developed by David Cournapeau as a "Google Summer of Code" project in 2007
 - Still under active development (v1.0.2 as of December 2021)

SK-Learn Modules

- **Classification:** identify to which category an object belongs to
 - Regression: predict continuous-valued attribute (linear, logistic, etc.)
 - SVM, Decision tree, Neural nets, Nearest neighbors, ...
- **Clustering:** grouping of similar objects
 - K-means, Hierarchical clustering, etc.
- **Model selection:** validate and choosing parameters and model
 - Cross validation, metrics, etc.
- **Preprocessing:** feature extraction & normalization
- **Dimensionality reduction:** reducing number of variables
 - PCA, Feature selection, etc.
- **Datasets**

Structure of SK-Learn

- **.fit()**

- Build a model using the (training) data X
- Requires the y values for classification and prediction

.fit(X , $y=None$)

- **.predict()**

- Predict the y values for the test data X based on the model
- Perform classification, regression, clustering, etc.

.transform(X)

- **.transform()**

- Transform the input data X based on the model
- Perform preprocessing, dimensionality reduction, feature extraction, feature selection, etc.



Example: StandardScaler()

- StandardScaler transform data using the following equation:

$$\tilde{x}_i = \frac{x_i - \text{mean}(x)}{\text{std}(x)}$$

- `scaler.fit()`
 - Build a model using the given data
 - Compute mean and variance
- `scaler.transform()`
 - Transform X to $X_{\text{transformed}}$ using the model

```
import numpy as np
import sklearn.preprocessing as prep

X = np.arange(5, dtype='float').reshape(5, 1)
```

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

```
scaler = prep.StandardScaler()
scaler.fit(X)
scaler.mean_, scaler.var_
(array([2.]), array([2.]))
```

```
X_ = scaler.transform(X)
X_

array([-1.41421356,
       -0.70710678,
       [ 0.        ],
       [ 0.70710678],
       [ 1.41421356]])
```

Example: LinearRegression()

- Linear regression

- `regr.fit()`

- Build a model for linear regression
- $y = -0.42 + 3.58x_1 - 0.69x_2 - 1.22x_3$

- `regr.predict()`

- Predict the y value for the test data X using the model

```
import numpy as np
from sklearn import linear_model

X = np.random.random((5,3))
y = np.random.random((5,1))
X, y

array([[0.34523274, 0.03465153, 0.49879222], array([[0.02940408],
[0.34154268, 0.558655 , 0.05047529], [0.41239473],
[0.55781272, 0.93527388, 0.59078667], [0.1845568 ],
[0.7568254 , 0.57266255, 0.90788885], [0.78603924],
[0.42153745, 0.09800326, 0.69636864]]])
```

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
regr.coef_, regr.intercept_

(array([[ 3.58372982, -0.68867795, -1.21704883]]), array
([-0.41676285]))
```

```
test = np.random.random((2,3))
print('test\n', test)
regr.predict(test)

test
[[0.1394362  0.8556813  0.437153 ]
[0.56918605 0.05113164  0.42308297]]

array([[-1.03838658],
 [ 1.07292029]])
```

Imputation of Missing Values

Simple Imputer

- `sklearn.impute.SimpleImputer(missing_values=nan, strategy='mean', fill_value=None, ...)`
 - Imputation transformer for completing missing values
 - `missing_values`: the placeholder for the missing values
 - `strategy`: 'mean', 'median', 'most_frequent', 'constant' (use `fill_value`)

```
from sklearn.impute import SimpleImputer
simp = SimpleImputer(strategy='mean')
X = [[3, 6, 2], [1, np.nan, 4], [np.nan, 9, 5]]
simp.fit_transform(X)
```

```
array([[3. , 6. , 2. ],
       [1. , 7.5, 4. ],
       [2. , 9. , 5. ]])
```

Iterative Imputer (Experimental)

- **`sklearn.impute.IterativeImputer(...)`**

- Multivariate imputer that estimates each feature from all the others
- Model each feature with missing values as a function of other features in a round-robin fashion

```
:> from sklearn.experimental import enable_iterative_imputer
:| from sklearn.impute import IterativeImputer
:| iimp = IterativeImputer()
:| X = [[1, 2, 3], [2, 4, np.nan], [3, 6, 15], [4, np.nan, 21]]
:| iimp.fit_transform(X)
:|
:| array([[ 1.          ,  2.          ,  3.          ],
:|        [ 2.          ,  4.          ,  9.00000022],
:|        [ 3.          ,  6.          , 15.          ],
:|        [ 4.          ,  7.99999948, 21.          ]])
```

KNN Imputer

- **sklearn.impute.KNNImputer(*n_neighbors*=5, *weights*='uniform', ...)**

- Impute missing values using k-Nearest Neighbors
- *n_neighbors*: number of neighboring samples to use for imputation
- *weights*: weight function used in prediction ('uniform', 'distance', etc.)

```
from sklearn.impute import KNNImputer
kimp = KNNImputer(n_neighbors=2)
X = [[1, 2, 3], [2, 4, np.nan], [3, 6, 15], [4, np.nan, 21]]
kimp.fit_transform(X)
```

```
array([[ 1.,  2.,  3.],
       [ 2.,  4.,  9.],
       [ 3.,  6., 15.],
       [ 4.,  5., 21.]])
```

```
from sklearn.metrics.pairwise import nan_euclidean_distances
nan_euclidean_distances(X, X)
```

array([[0. , 2.73861279, 12.80624847, 22.34949664], [2.73861279, 0. , 2.73861279, 3.46410162], [12.80624847, 2.73861279, 0. , 7.44983221], [22.34949664, 3.46410162, 7.44983221, 0.]])
--

Data Encoding

Approaches

■ Label Encoding

- Substitute categorical data with a corresponding number
- Simplest
- Can be "misinterpreted" by the algorithms

■ Ordinal Encoding

- Encode ordinal data as numbers

■ One Hot Encoding

- Most common, correct way to deal with non-ordinal categorical data
- Represent a label as a vector of 0's and 1's

Label	Encoding
'dog'	0
'cat'	1
'rabbit'	2

Label	Encoding
'cold'	0
'warm'	1
'hot'	2

Label	Encoding
'dog'	(1, 0, 0)
'cat'	(0, 1, 0)
'rabbit'	(0, 0, 1)

SK-Learn LabelEncoder()

- `sklearn.preprocessing.LabelEncoder()`
 - Encode target labels (ID array) with value between 0 and `n_classes`-1
- Methods
 - `fit(X, [y])`: fit label encoder
 - `transform(X)`: transform labels to normalized encoding
 - `inverse_transform(y)`: transform labels back to original encoding

```
from sklearn.preprocessing import LabelEncoder  
X = ['A', 'B', 'A', 'A', 'B', 'C', 'C', 'A', 'C', 'B']  
le = LabelEncoder()
```

```
le.fit_transform(X)
```

```
array([0, 1, 0, 0, 1, 2, 2, 0, 2, 1], dtype=int64)
```

SK-Learn OrdinalEncoder()

- `sklearn.preprocessing.OrdinalEncoder(categories='auto', ...)`
 - Encode categorical features (2D array) with value between 0 and `n_classes-1`
 - `categories`: categories (unique values) per feature

```
from sklearn.preprocessing import OrdinalEncoder
X = [['hot'], ['warm'], ['cold'], ['hot'], ['warm']]
oe = OrdinalEncoder()
oe.fit_transform(X)

array([[1.],
       [2.],
       [0.],
       [1.],
       [2.]])
```

```
oe = OrdinalEncoder(categories=[[ 'cold', 'warm', 'hot']])
oe.fit_transform(X)

array([[2.],
       [1.],
       [0.],
       [2.],
       [1.]])
```



```
oe.inverse_transform([[0],[1],[2]])

array([['cold'],
       ['warm'],
       ['hot']], dtype=object)
```

SK-Learn OneHotEncoder()

- **`sklearn.preprocessing.OneHotEncoder([sparse], ...)`**
 - Encode categorial features using a one-hot numeric array
 - `sparse`: if True, return sparse matrix else return an array (default:True)
- **Attributes**
 - `categories_`: the categories of each feature determined during fitting
- **Methods**
 - `fit(X, [y])`: fit OneHotEncoder to X
 - `transform(X)`: transform X using one-hot encoding
 - `inverse_transform(X)`: convert the data back to the original representation

OneHotEncoder(): ID Data

The result of OneHotEncoder()
is a SciPy's sparse matrix

```
from sklearn.preprocessing import OneHotEncoder  
  
ohe = OneHotEncoder()  
X = np.array(['a', 'b', 'a', 'c']).reshape(-1, 1)  
ohe.fit(X)  
X_encode = ohe.transform(X)  
type(X_encode)
```

→ scipy.sparse.csr.csr_matrix

toarray() converts sparse
matrix to dense matrix

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

```
ohe.inverse_transform([[0., 1., 0.]])  
  
array([['b']], dtype='<U1')
```

OneHotEncoder(): 2D Data

```
X = np.array([[0, 0, 4],  
             [1, 1, 0],  
             [0, 2, 1],  
             [1, 0, 2]])  
  
ohe.fit(X)  
X_encode = ohe.transform(X)  
X_encode.toarray()
```

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

```
ohe.inverse_transform([[1., 0., 0., 0., 1.], [0., 0., 0., 1., 0.]])
```

```
array([[0, 2, 4]], dtype=int32)
```

Using OneHotEncoder() in Pandas

```
from sklearn.preprocessing import OneHotEncoder  
  
ohe = OneHotEncoder()  
df_encode = pd.DataFrame(ohe.fit_transform(df[['Animal']]).toarray())  
df_ohe = df.join(df_encode)  
df_ohe
```

	Animal	0	1	2
0	cat	1.0	0.0	0.0
1	dog	0.0	1.0	0.0
2	rabbit	0.0	0.0	1.0
3	cat	1.0	0.0	0.0
4	dog	0.0	1.0	0.0

	df	Animal
0		cat
1		dog
2		rabbit
3		cat
4		dog

Pandas One-Hot Encoding

- ***pd.get_dummies(data, [columns], ...)***
 - Convert categorical variable into dummy/indicator variables
 - *data*: data of which to get dummy indicators
 - *columns*: column names in the DataFrame to be encoded

```
df_encode = pd.get_dummies(df, columns=['Animal'])
df_ohe = df.join(df_encode)
df_ohe
```

df	Animal	Animal	Animal_cat	Animal_dog	Animal_rabbit
		0	1	0	0
	cat	0	cat	1	0
	dog	1	dog	0	1
	rabbit	2	rabbit	0	1
	cat	3	cat	1	0
	dog	4	dog	0	1

get_dummies() vs. OneHotEncoder()

X_train

Animal

0 dog

1 cat

2 cat

3 rabbit

4 dog

5 rabbit

X_test

Animal

0 cat

1 dog

2 rabbit

3 horse

```
encoder = OneHotEncoder(handle_unknown='ignore')  
encoder.fit(X_train)
```

```
X_train_encode = encoder.transform(X_train)  
X_train.join(pd.DataFrame(X_train_encode.toarray()))
```

	Animal	0	1	2
0	dog	0.0	1.0	0.0
1	cat	1.0	0.0	0.0
2	cat	1.0	0.0	0.0
3	rabbit	0.0	0.0	1.0
4	dog	0.0	1.0	0.0
5	rabbit	0.0	0.0	1.0

```
X_test_encode = encoder.transform(X_test)  
X_test.join(pd.DataFrame(X_test_encode.toarray()))
```

	Animal	0	1	2
0	cat	1.0	0.0	0.0
1	dog	0.0	1.0	0.0
2	rabbit	0.0	0.0	1.0
3	horse	0.0	0.0	0.0

X_train.join(pd.get_dummies(X_train))

Animal Animal_cat Animal_dog Animal_rabbit

0	dog	0	1	0
1	cat	1	0	0
2	cat	1	0	0
3	rabbit	0	0	1
4	dog	0	1	0
5	rabbit	0	0	1

X_test.join(pd.get_dummies(X_test))

Animal Animal_cat Animal_dog Animal_horse Animal_rabbit

0	cat	1	0	0	0
1	dog	0	1	0	0
2	rabbit	0	0	0	1
3	horse	0	0	1	0

SK-Learn Binarizer()

- `sklearn.preprocessing.Binarizer([threshold], ...)`
 - Binarizer data (set feature values to 0 or 1) according to a threshold
 - `threshold`: feature values below or equal to this are replaced by 0, above it by 1 (default: 0.0)
- Methods
 - `fit(X, [y])`: do nothing
 - `transform(X)`: binarize each element of X

Binarizer() Example

```
from sklearn.preprocessing import Binarizer
X = np.array([[1., -1., 2.],
              [2., -3., 1.],
              [0., 1., -1.]])
bin = Binarizer()
bin.transform(X)
```

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

```
bin = Binarizer(threshold = 1)
bin.transform(X)
```

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

Jin-Soo Kim
[\(jinsoo.kim@snu.ac.kr\)](mailto:jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022



Python for Data Analytics

Data Preprocessing II

Data Scaling

Why Data Scaling?

- Features in a dataset can have very different scales
- Unscaled data can degrade the predictive performance of many machine learning algorithms
 - Many estimators assume that each feature takes values close to zero and all features vary on comparable scales
 - Metric-based and gradient-based estimators often assume approximately standardized data (normal distribution)
 - (cf.) Decision tree-based estimators are robust to arbitrary scaling of the data
- Unscaled data can slow down or even prevent the convergence of many gradient-based estimators

Data Scaling

- Standard scaling: $\rightarrow \tilde{x}_i \sim \text{Normal distribution } (\mu = 0, \sigma = 1)$
$$\tilde{x}_i = \frac{x_i - \text{mean}(x)}{\text{std}(x)}$$
- Min-Max Scaling: $\rightarrow \tilde{x}_i \text{ in } [0, 1]$
$$\tilde{x}_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$
- Max-Abs Scaling: $\rightarrow |\tilde{x}_i| \leq 1$
$$\tilde{x}_i = \frac{x_i}{\max(|x|)}$$
- Robust Scaling: \rightarrow Based on median and IQR
$$\tilde{x}_i = \frac{x_i - \text{median}(x)}{Q3(x) - Q1(x)}$$

Data Scaling supported by SK-Learn

Scaling	Function	Class
Standard	<code>scale(x)</code>	<code>StandardScaler</code>
Min-Max	<code>minmax_scale()</code>	<code>MinMaxScaler</code>
Max-Abs	<code>maxabs_scale()</code>	<code>MaxAbsScaler</code>
Robust	<code>robust_scale()</code>	<code>RobustScaler</code>

Standard Scaling

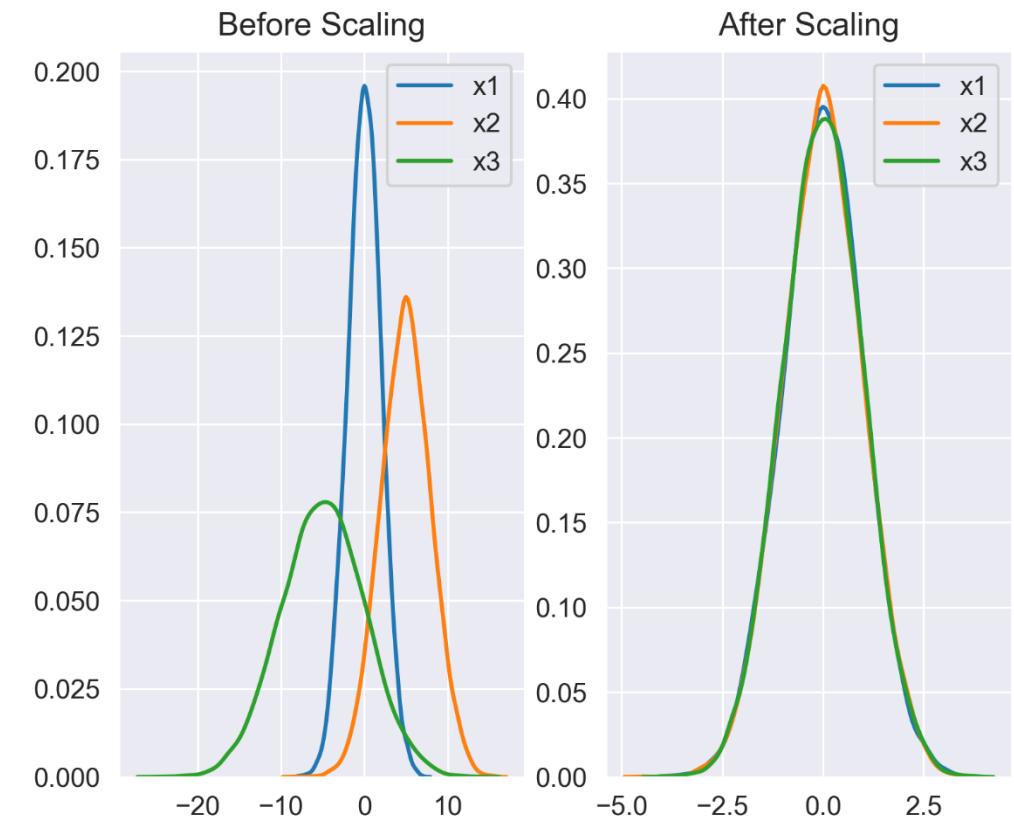
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing

df = pd.DataFrame({
    'x1': np.random.normal(0, 2, 10000),
    'x2': np.random.normal(5, 3, 10000),
    'x3': np.random.normal(-5, 5, 10000)
})
scaler = preprocessing.StandardScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])

sns.set_style('darkgrid')
_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)

ax2.set_title('After Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
```

$$\tilde{x}_i = \frac{x_i - \text{mean}(x)}{\text{std}(x)}$$

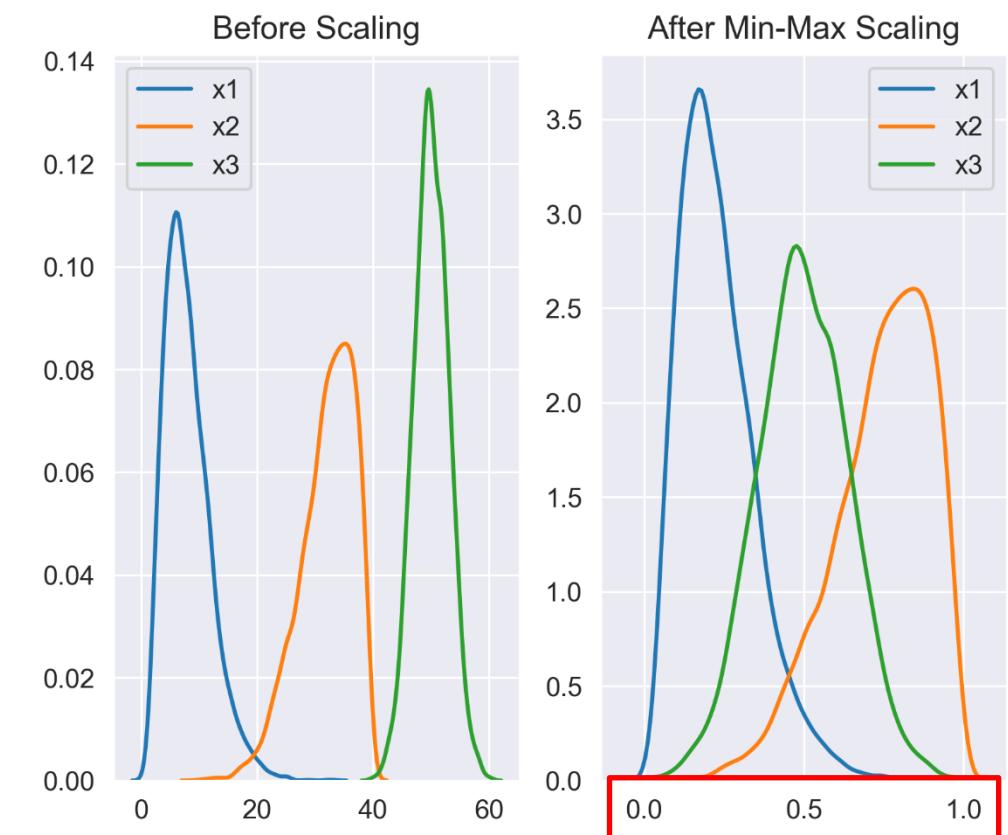


Min-Max Scaling

- All values are mapped in the range [0, 1]
- Very sensitive to the presence of outliers

```
df = pd.DataFrame({  
    'x1': np.random.chisquare(8, 10000),      # positive skew  
    'x2': np.random.beta(8, 2, 10000)*40,     # negative skew  
    'x3': np.random.normal(50, 3, 10000)       # no skew  
})  
  
scaler = preprocessing.MinMaxScaler()  
scaled_df = scaler.fit_transform(df)  
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])  
  
_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))  
ax1.set_title('Before Scaling')  
sns.kdeplot(df['x1'], ax=ax1)  
sns.kdeplot(df['x2'], ax=ax1)  
sns.kdeplot(df['x3'], ax=ax1)  
  
ax2.set_title('After Min-Max Scaling')  
sns.kdeplot(scaled_df['x1'], ax=ax2)  
sns.kdeplot(scaled_df['x2'], ax=ax2)  
sns.kdeplot(scaled_df['x3'], ax=ax2)
```

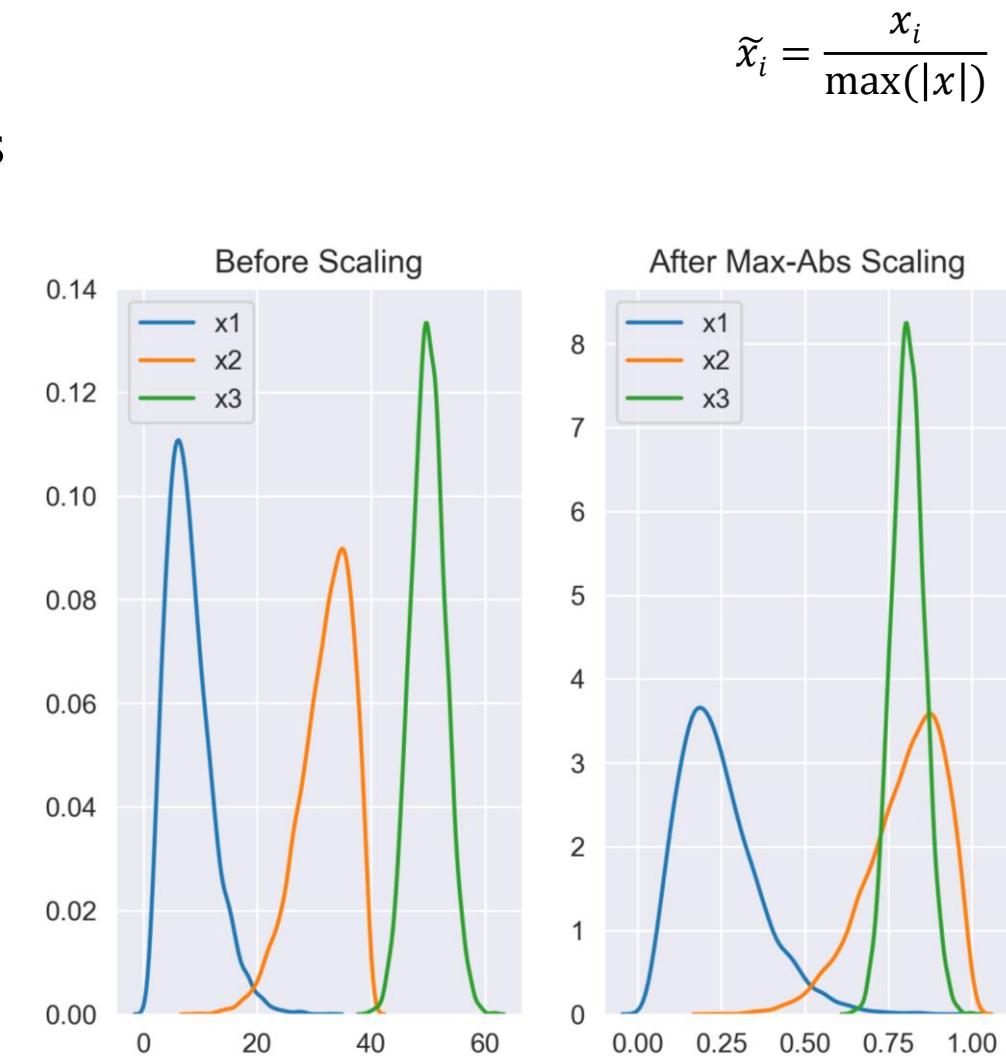
$$\tilde{x}_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$



Max-Abs Scaling

- Doesn't change the shape of the distribution
- Also suffers from the presence of large outliers

```
df = pd.DataFrame({  
    'x1': np.random.chisquare(8, 10000),      # positive skew  
    'x2': np.random.beta(8, 2, 10000)*40,     # negative skew  
    'x3': np.random.normal(50, 3, 10000)       # no skew  
})  
  
scaler = preprocessing.MaxAbsScaler()  
scaled_df = scaler.fit_transform(df)  
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])  
  
_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))  
ax1.set_title('Before Scaling')  
sns.kdeplot(df['x1'], ax=ax1)  
sns.kdeplot(df['x2'], ax=ax1)  
sns.kdeplot(df['x3'], ax=ax1)  
  
ax2.set_title('After Max-Abs Scaling')  
sns.kdeplot(scaled_df['x1'], ax=ax2)  
sns.kdeplot(scaled_df['x2'], ax=ax2)  
sns.kdeplot(scaled_df['x3'], ax=ax2)
```



Robust Scaling (I)

- Based on percentiles
- Not influenced by a few number of very large marginal outliers

```
df = pd.DataFrame({  
    # distribution with Lower outliers  
    'x1': np.hstack((np.random.normal(20,1,1000),  
                      np.random.normal(1,1,25))),  
    # distribution with upper outliers  
    'x2': np.hstack((np.random.normal(30,1,1000),  
                      np.random.normal(50,1,25)))  
})  
  
robust_scaler = preprocessing.RobustScaler()  
robust_df = robust_scaler.fit_transform(df)  
robust_df = pd.DataFrame(robust_df, columns=['x1', 'x2'])  
  
minmax_scaler = preprocessing.MinMaxScaler()  
minmax_df = minmax_scaler.fit_transform(df)  
minmax_df = pd.DataFrame(minmax_df, columns=['x1', 'x2'])
```

$$\tilde{x}_i = \frac{x_i - \text{median}(x)}{Q3(x) - Q1(x)}$$

```
_, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(9,5))  
ax1.set_title('Before Scaling')  
sns.kdeplot(df['x1'], ax=ax1)  
sns.kdeplot(df['x2'], ax=ax1)  
  
ax2.set_title('After Robust Scaling')  
sns.kdeplot(robust_df['x1'], ax=ax2)  
sns.kdeplot(robust_df['x2'], ax=ax2)  
  
ax3.set_title('After Min-Max Scaling')  
sns.kdeplot(minmax_df['x1'], ax=ax3)  
sns.kdeplot(minmax_df['x2'], ax=ax3)
```

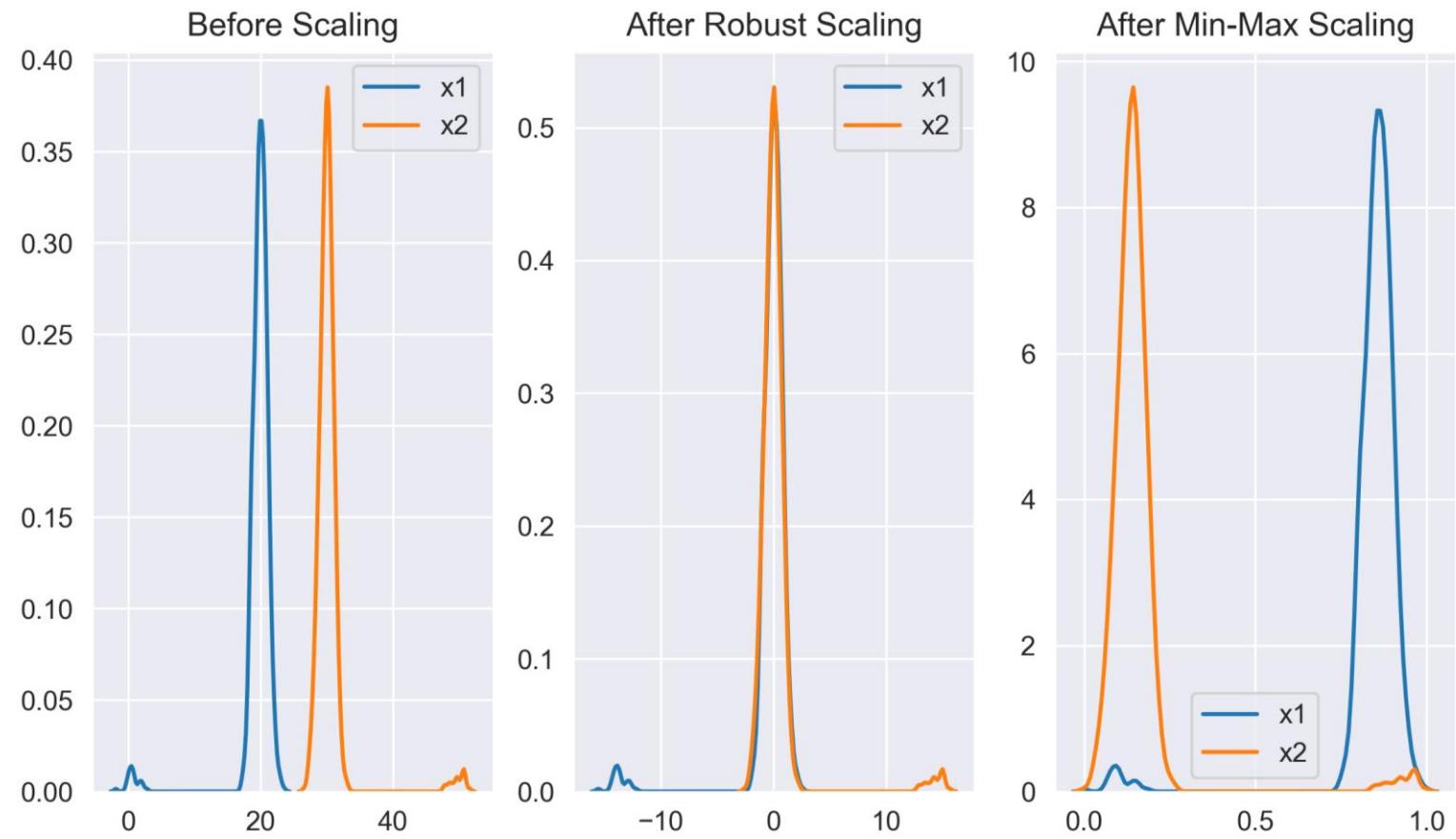
Robust Scaling (2)

■ Min-Max Scaling

- Significantly affected by outliers

■ Robust Scaling

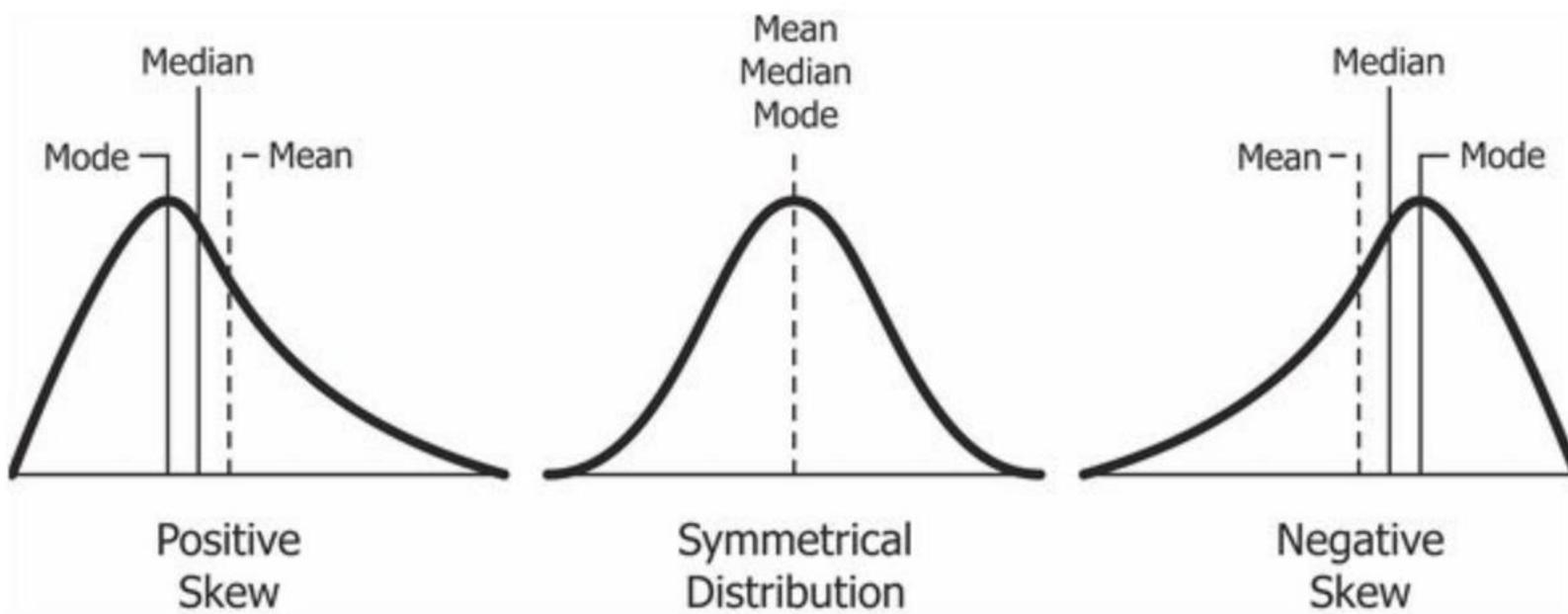
- Inliers are in $[-2, 2]$
- Outliers still exist at the end of each distribution



Data Standardization

Data Skewness

- A measure of asymmetry of a distribution
 - Symmetrical (skewness = 0, e.g., normal distribution): mean == median == mode
 - Positive skew (skewness > 0): tail at right, mode < median < mean
 - Negative skew (skewness < 0): tail at left, mean < median < mode



Measuring Data Skewness

- `df.skew([axis], [skipna], ...)`
 - Return unbiased skew over requested axis
 - `axis`: axis for the function to be applied on
 - `skipna`: if `True`, exclude null values when computing the result (default `True`)
- Meaning of skewness value
 - $-0.5 \leq \text{skewness} \leq 0.5$: fairly symmetrical
 - $-1 < \text{skewness} < -0.5$ or $0.5 < \text{skewness} < 1$: moderately skewed
 - $\text{skewness} < -1$ or $\text{skewness} > 1$: highly skewed

Handling Data Skewness

- Linear model performs better when the dataset follows normal distribution
- Dealing with positive skewness
 - Square root transformation (x to $x^{1/2}$)
 - Cube root transformation (x to $x^{1/3}$)
 - Log transformation (x to $\log_2 x, \log_e x, \ln x, \dots$)
- Dealing with negative skewness
 - Square transformation (x^2)
 - Cube transformation (x^3)
 - Reflect the values and apply the methods used to reduce the positive skewness

The Boston Housing Dataset

- Dataset for housing values in areas of Boston in 70's
- 506 rows, 14 columns (13 attributes + housing value)
- Available in the SK-Learn datasets

CRIM:	범죄율
ZN:	25,000ft ² 초과 거주지역 비율
INDUS:	비소매상업지역 면적 비율
CHAS:	찰스강 경계에 위치한 경우 1
NOX:	일산화질소 농도
AGE:	1940년 이전 건축된 주택 비율
RM:	주택당 방 수
RAD:	방사형 고속도로까지의 거리
LSTAT:	인구 중 하위 계층 비율
DIS:	직업 센터의 거리
B:	인구 중 흑인 비율
TAX:	재산세율
PTRATIO:	학생/교사 비율
MEDV:	주택 가격의 median (단위: \$1,000)

```
from sklearn import datasets
import pandas as pd
boston = datasets.load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df.head()
```

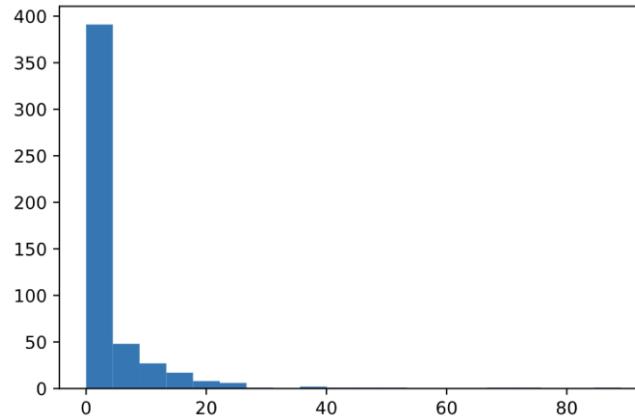
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Skewness in Boston Housing Dataset

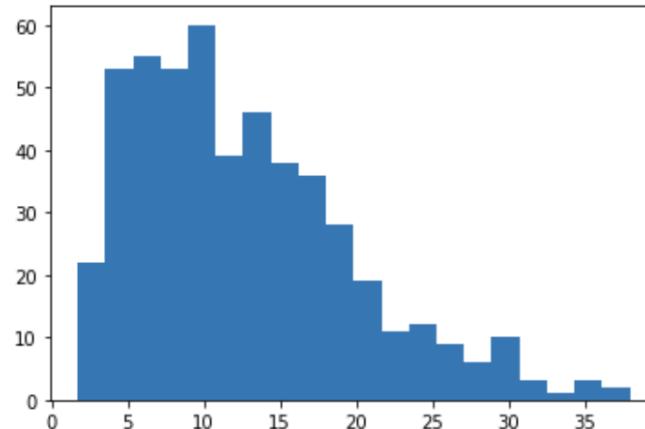
```
df.skew()
```

CRIM	5.223149
ZN	2.225666
INDUS	0.295022
CHAS	3.405904
NOX	0.729308
RM	0.403612
AGE	-0.598963
DIS	1.011781
RAD	1.004815
TAX	0.669956
PTRATIO	-0.802325
B	-2.890374
LSTAT	0.906460
dtype:	float64

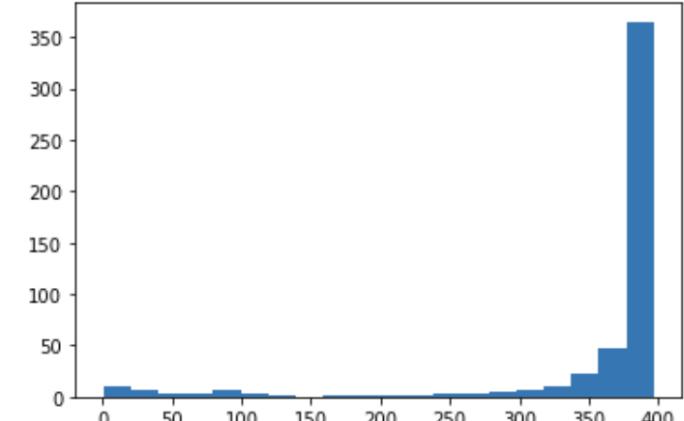
```
plt.hist(df.CRIM, bins=20)
```



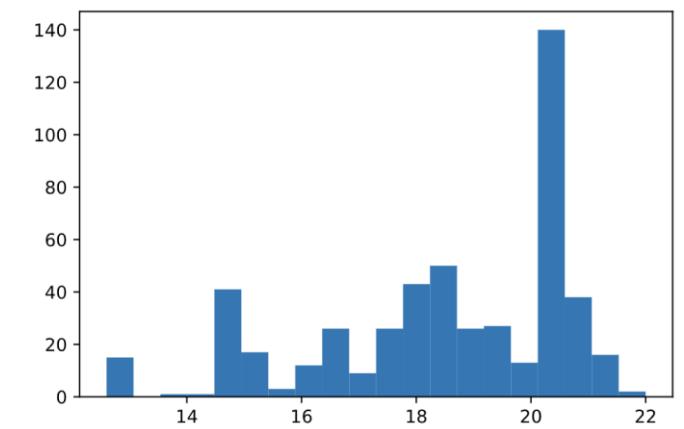
```
plt.hist(df.LSTAT, bins=20)
```



```
plt.hist(df.B, bins=20)
```



```
plt.hist(df.PTRATIO, bins=20)
```



Transforming Data (I)

- `sklearn.preprocessing.scale(X, ...)`
 - Standardize a dataset along any axis (standard scaler)
 - Center to the zero mean and component wise scale to unit variance
 - `X`: the data to center and scale

```
from sklearn import preprocessing

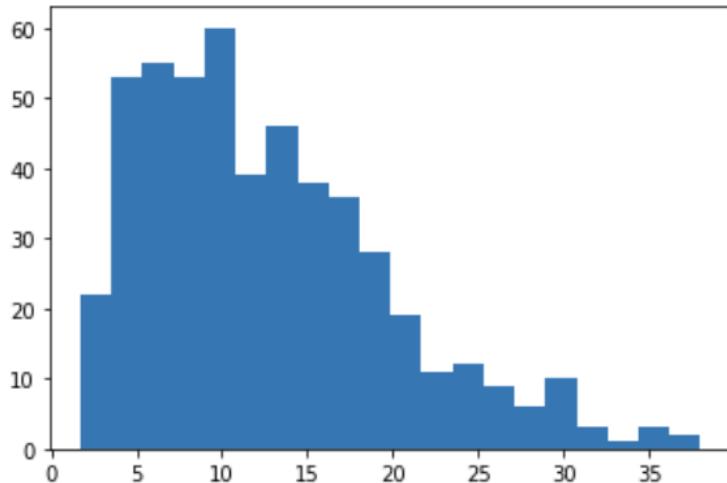
df['LSTAT_log'] = preprocessing.scale(np.log(df['LSTAT']+1))
df['LSTAT_sqrt'] = preprocessing.scale(np.sqrt(df['LSTAT']+1))
df[['LSTAT', 'LSTAT_log', 'LSTAT_sqrt']].skew()
```

```
LSTAT          0.906460
LSTAT_log     -0.187195
LSTAT_sqrt     0.359606
dtype: float64
```

Transforming Data (2)

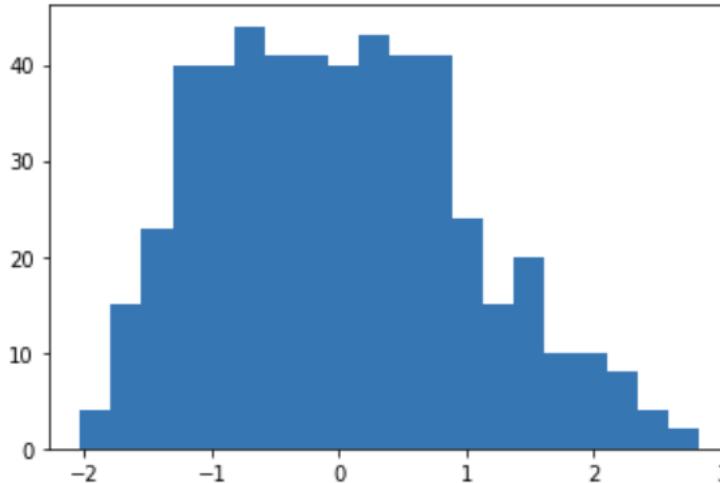
Original data

```
import matplotlib.pyplot as plt  
  
plt.hist(df['LSTAT'], bins=20)  
plt.show()
```



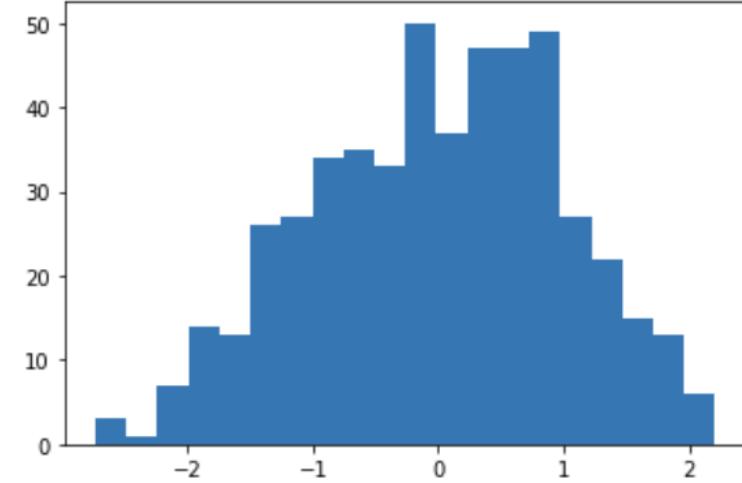
Square root transformation

```
import matplotlib.pyplot as plt  
  
plt.hist(df['LSTAT_sqrt'], bins=20)  
plt.show()
```



Log transformation

```
import matplotlib.pyplot as plt  
  
plt.hist(df['LSTAT_log'], bins=20)  
plt.show()
```



Sampling for Imbalanced Data

Imbalanced Data

- A problem with classification where the classes are not represented equally
 - The model will be mostly tuned for the majority class
- Example:
 - A dataset with Class A : Class B = 9 : 1
 - The percentage of correct answers in the test dataset will also be 9 : 1
 - Even if a model classifies everything to Class A, it will have a 90% of accuracy
- Solutions: Balance data using sampling
 - Oversampling: increase the amount of minority class
 - Undersampling: use only part of majority class

imbalanced-learn module

- A python package offering a number of re-sampling techniques
- Commonly used for datasets showing strong between-class imbalance
- Part of scikit-learn-contrib projects
- <https://github.com/scikit-learn-contrib/imbalanced-learn>
- Installation
 - pip install -U imbalanced-learn
 - conda install -c conda-forge imbalanced-learn

```
>>> import imblearn.under_sampling  
>>> import imblearn.over_sampling
```

Creating Imbalanced Data

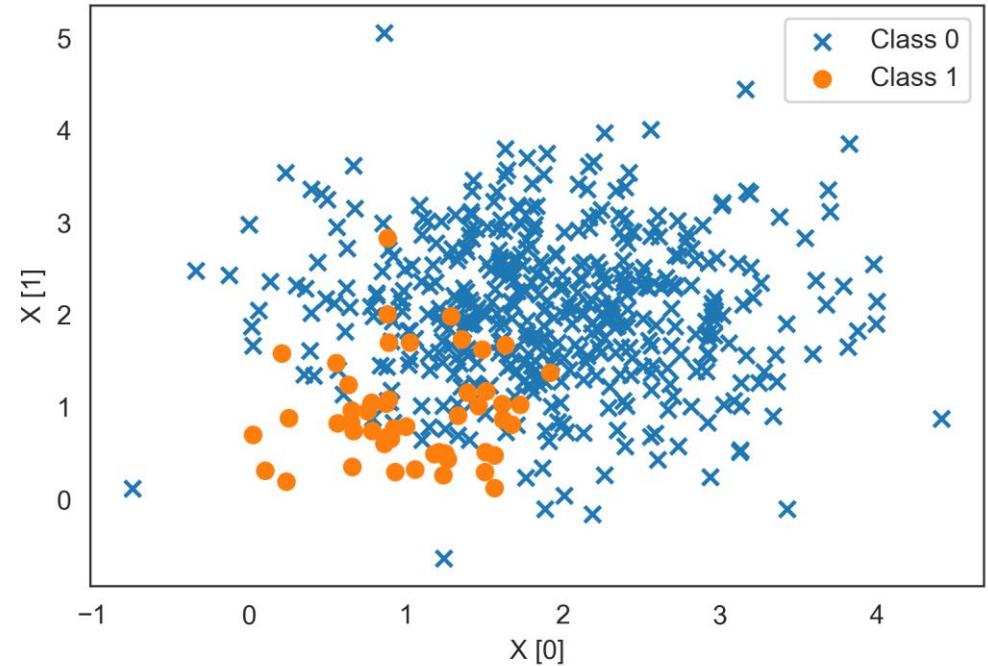
```
def plot(X, y):
    plt.scatter(X[y==0, 0], X[y==0, 1], marker='x', label='Class 0')
    plt.scatter(X[y==1, 0], X[y==1, 1], marker='o', label='Class 1')
    plt.xlabel('X [0]')
    plt.ylabel('X [1]')
    plt.legend()

n0 = 450
n1 = 50

a = np.random.randn(n0, 2)*0.8 + 2 # N(2, 0.8)
b = np.random.randn(n1, 2)*0.5 + 1 # N(1, 0.5)

X = np.vstack([a, b])
y = np.hstack([np.zeros(n0), np.ones(n1)])

plot(X, y)
```

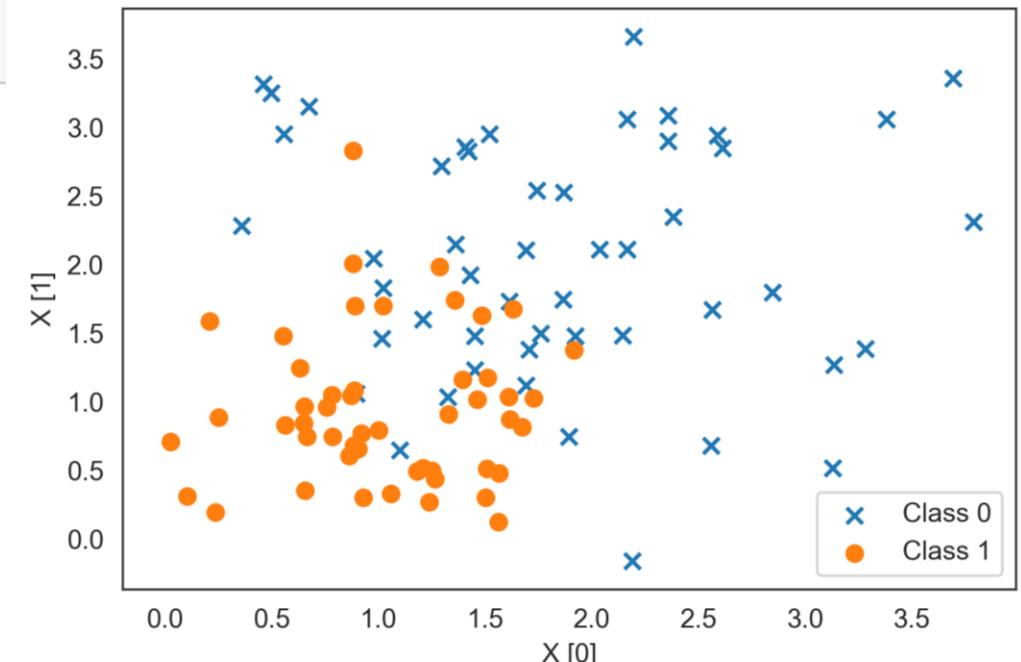


Undersampling: RandomUnderSampler()

- Under-sample the majority class by randomly picking samples

```
from imblearn.under_sampling import RandomUnderSampler  
  
X_samp, y_samp = RandomUnderSampler(random_state=0).fit_sample(X, y)  
print(X_samp.shape, y_samp.shape)  
plot(X_samp, y_samp)
```

(100, 2) (100,)



Undersampling: EditedNearestNeighbours()

- Keep a sample if all or majority of the NN's belong to the same class

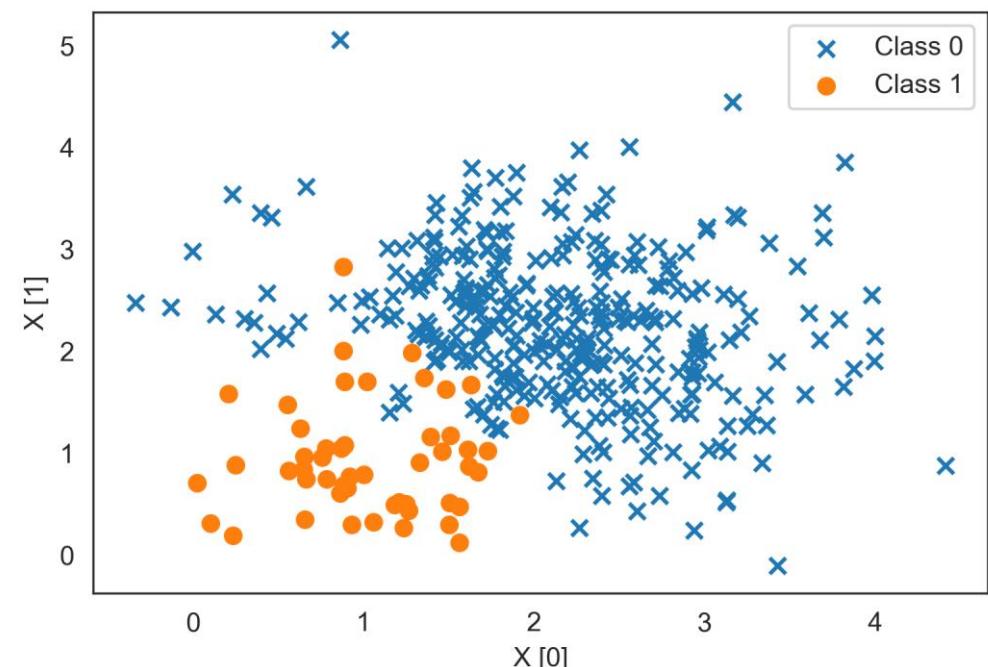
```
from imblearn.under_sampling import EditedNearestNeighbours

X_samp, y_samp = EditedNearestNeighbours(kind_sel='all', n_neighbors=10,
                                           random_state=0).fit_sample(X, y)

print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

(388, 2) (388,)

- n_neighbors*: size of the neighbourhood to consider to compute the nearest neighbors
- kind_sel*: 'all' (all have to agree to keep), 'mode' (majority vote to keep)



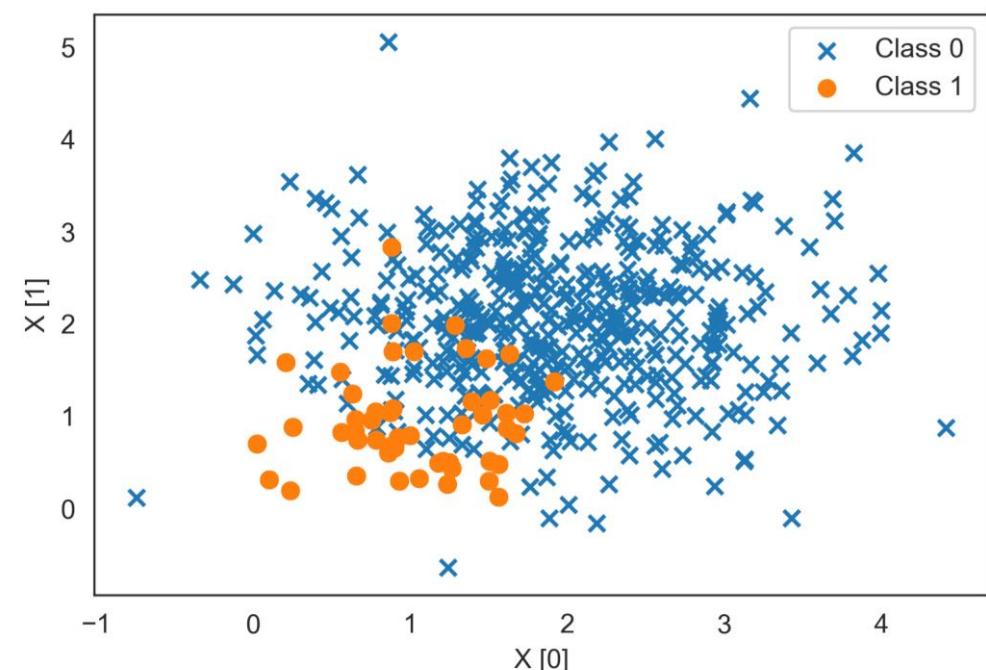
Oversampling: RandomOverSampler()

- Over-sample the minority class by picking samples at random

```
from imblearn.over_sampling import RandomOverSampler  
  
X_samp, y_samp = RandomOverSampler(random_state=0).fit_sample(X, y)  
print(X_samp.shape, y_samp.shape)  
plot(X_samp, y_samp)
```

(900, 2) (900,)

- Graph looks same, but the count has increased to 450



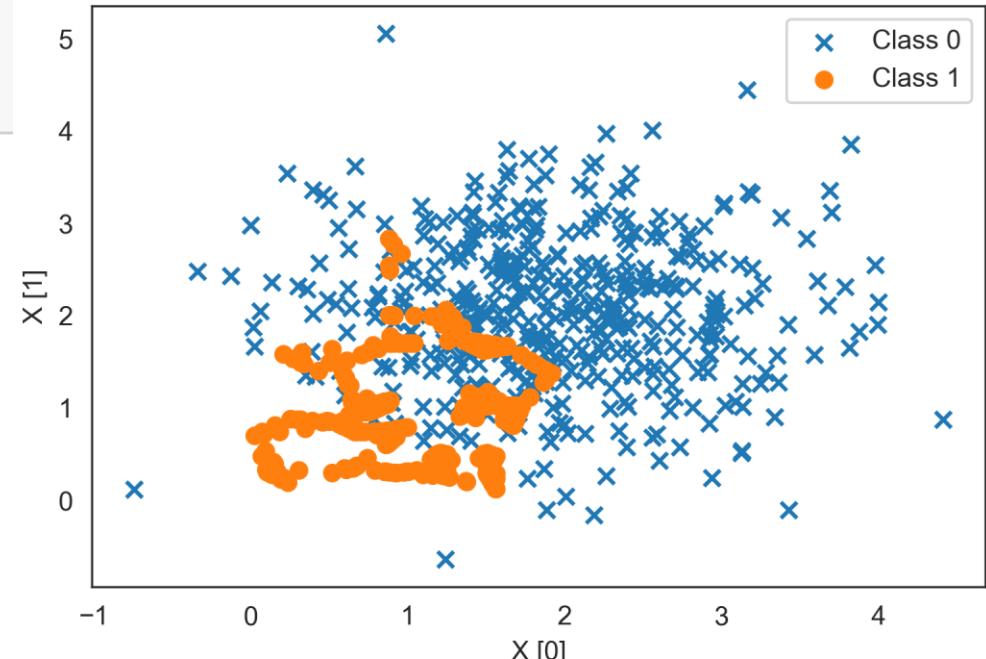
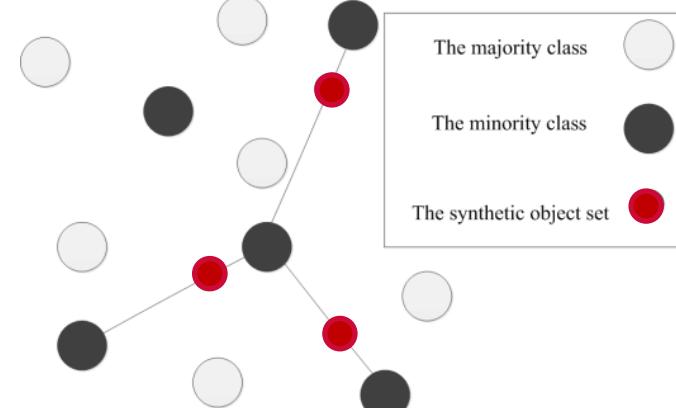
Oversampling: SMOTE()

- A sample is created at a randomly selected point between a minority sample and its neighbor which is randomly selected among k neighbors

```
from imblearn.over_sampling import SMOTE
```

```
X_samp, y_samp = SMOTE(k_neighbors=3).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

(900, 2) (900,)



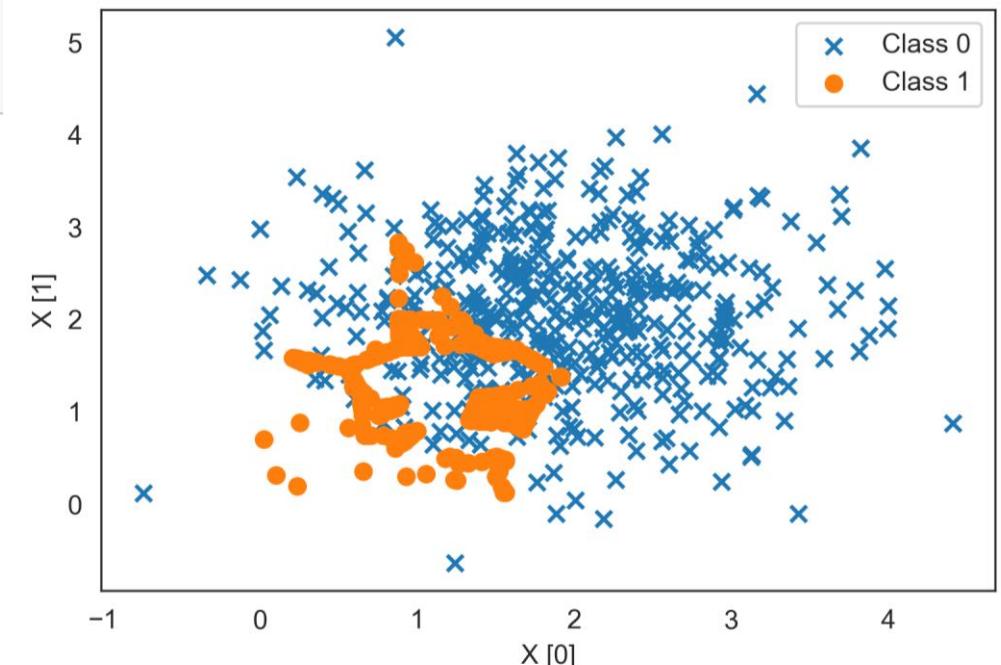
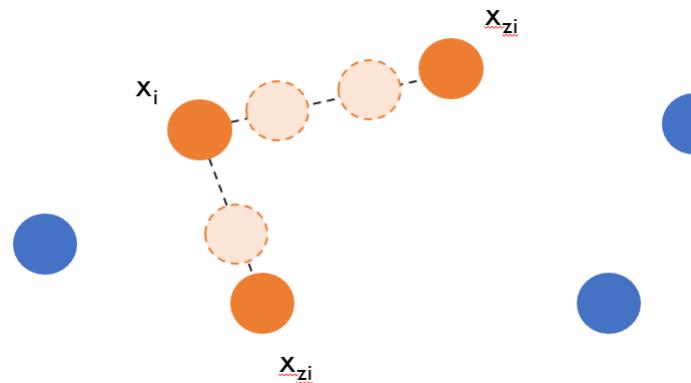
Oversampling:ADASYN()

- For a minority sample dominated by majority class samples, more synthetic minority class samples are generated

```
from imblearn.over_sampling import ADASYN
```

```
X_samp, y_samp = ADASYN(n_neighbors=3, random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

(908, 2) (908,)



Thank You!