

Python 초급

2016 빅데이터 교육 콘텐츠

Contents

I 파이썬 소개 및 문법

1. 변수에 대한 이해
2. 데이터구조
3. 조건문과 반복문
4. 데이터 입출력

II Pandas

1. Pandas 소개와 패키지설치
2. 자료구조 : Series 와 Dataframe
3. 자료다루기
4. 기초분석
5. 핵심기능 – Group by

Contents

III

NumPy

1. NumPy 소개
2. ndarray 다루기
3. Wine Quality 데이터를 활용한 NumPy 실습

IV

Scikit-learn

1. Scikit-learn & Machine learning 개념소개
2. NumPy를 활용한 Neural network의 이해 및 실습
3. Scikit-learn과 NumPy를 활용한 Wine Quality 데이터 Neural network 실습

V

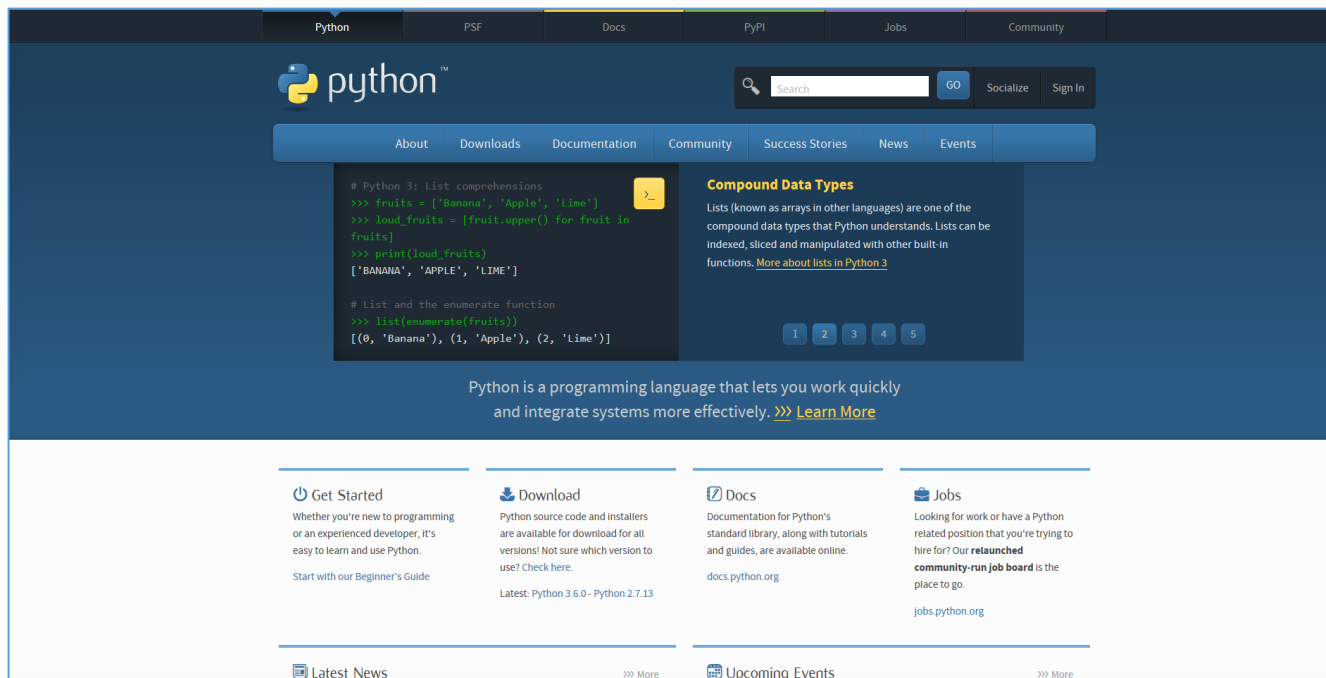
Case Study

1. Case Study 1(초급): Adult data set
2. Case Study 2(중급): German credit data set

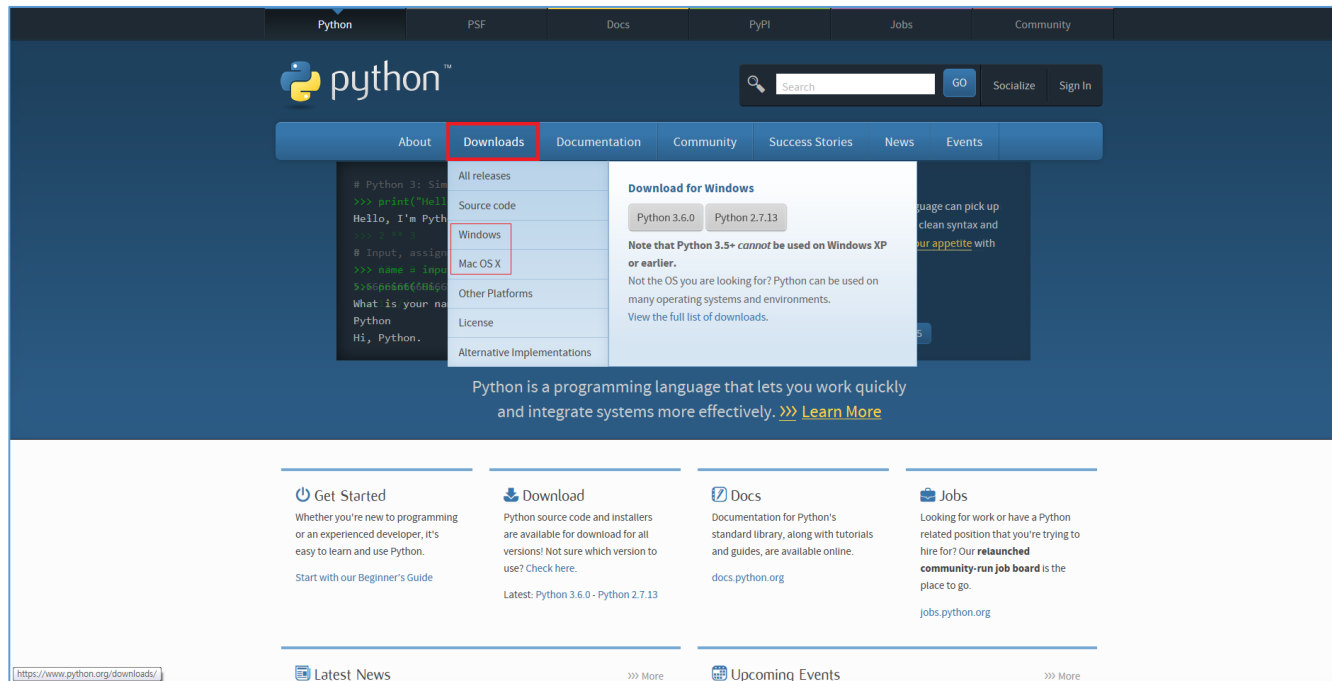
SECTION

파이썬 다운로드

- 파이썬 다운로드는 <https://www.python.org> 으로 들어가면 설치 파일을 다운로드 할 수 있다.



- 상단의 메뉴 중 Downloads를 들어간 다음 원하는 파이썬 버전을 선택하여 다운받으면 된다.



자신의 컴퓨터 OS에 따라 클릭하여 다운로드 페이지로 들어가면 된다.
본 강의에서는 Windows 기준으로 설명하였다.

SECTION

파이썬 설치

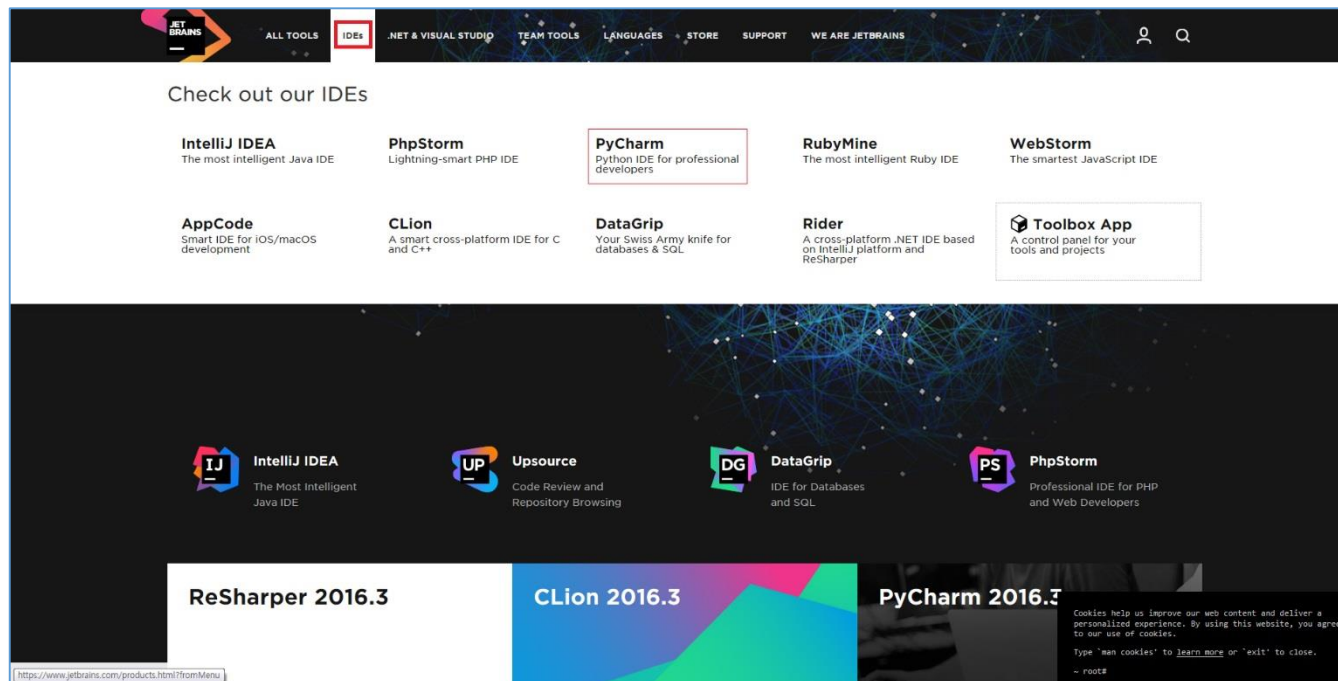
- 다운로드한 파이썬을 실행하여 Install Now를 클릭하여 설치를 완료한다.



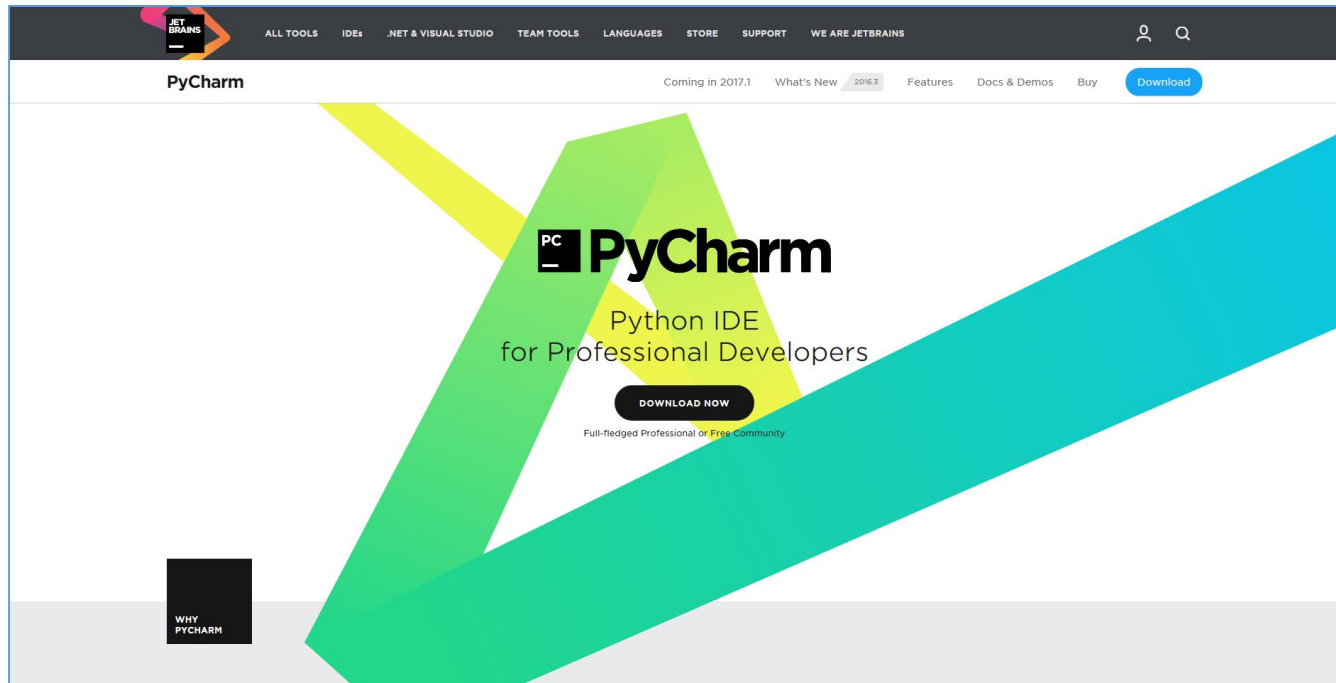
SECTION

Pycharm 다운로드

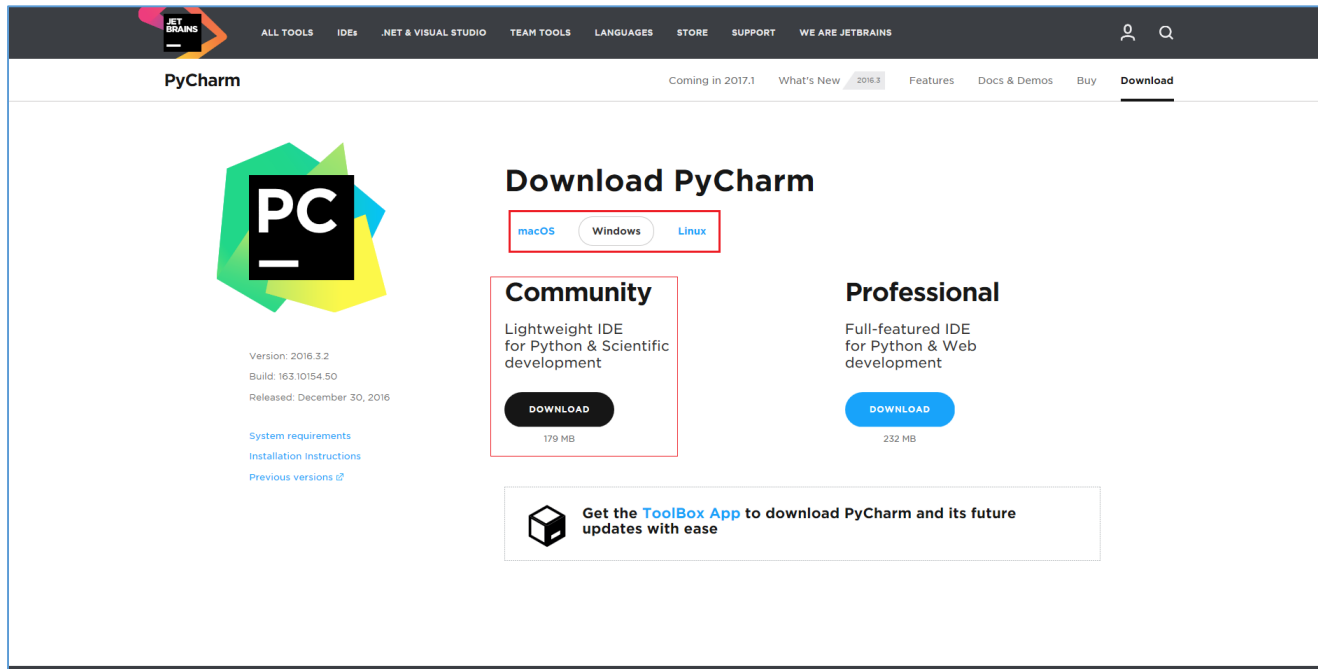
- Pycharm은 <https://www.jetbrains.com>으로 들어가서 다운 받을 수 있다.



- DOWNLOAD NOW를 클릭하여 사용자 컴퓨터 사양에 맞는 Pycharm을 다운로드 한다.



- Community 버전의 Download를 클릭하여 다운로드를 완료한다.



The screenshot shows the PyCharm download page. The top navigation bar includes links for ALL TOOLS, IDEs, .NET & VISUAL STUDIO, TEAM TOOLS, LANGUAGES, STORE, SUPPORT, and WE ARE JETBRAINS. The main content area features the PyCharm logo and version information (2016.3.2, Build: 163.10154.50, Released: December 30, 2016). Below the logo, there are links for System requirements, Installation instructions, and Previous versions. The 'Download PyCharm' section offers three options: macOS, Windows, and Linux. The 'Community' edition is highlighted with a red box, showing it is a 'Lightweight IDE for Python & Scientific development' with a 'DOWNLOAD' button and a size of 179 MB. The 'Professional' edition is also shown, described as a 'Full-featured IDE for Python & Web development' with a 'DOWNLOAD' button and a size of 232 MB. At the bottom, there is a promotional banner for the 'ToolBox App'.

PyCharm

Coming in 2017.1 What's New 2016.3 Features Docs & Demos Buy **Download**

Download PyCharm

macOS Windows Linux

Community
Lightweight IDE for Python & Scientific development
DOWNLOAD
179 MB

Professional
Full-featured IDE for Python & Web development
DOWNLOAD
232 MB

Version: 2016.3.2
Build: 163.10154.50
Released: December 30, 2016

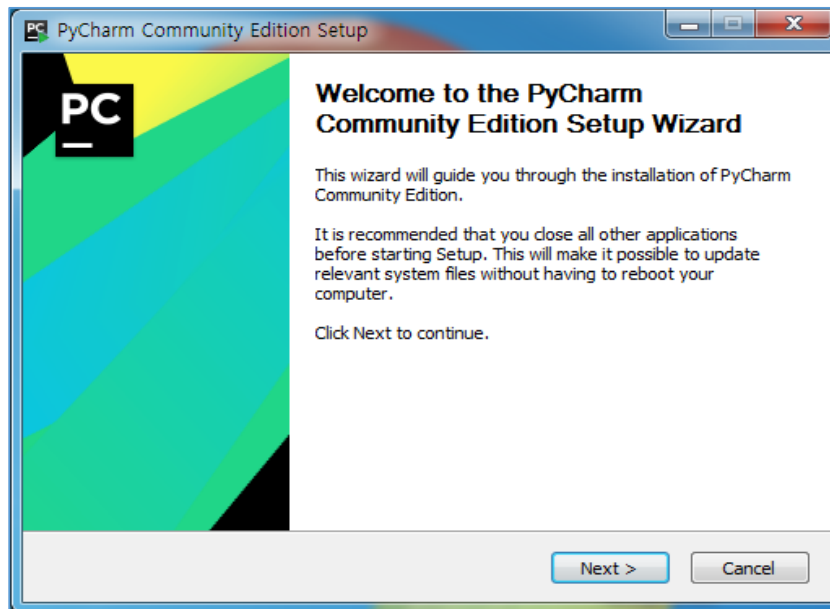
[System requirements](#)
[Installation instructions](#)
[Previous versions](#)

Get the **ToolBox App** to download PyCharm and its future updates with ease

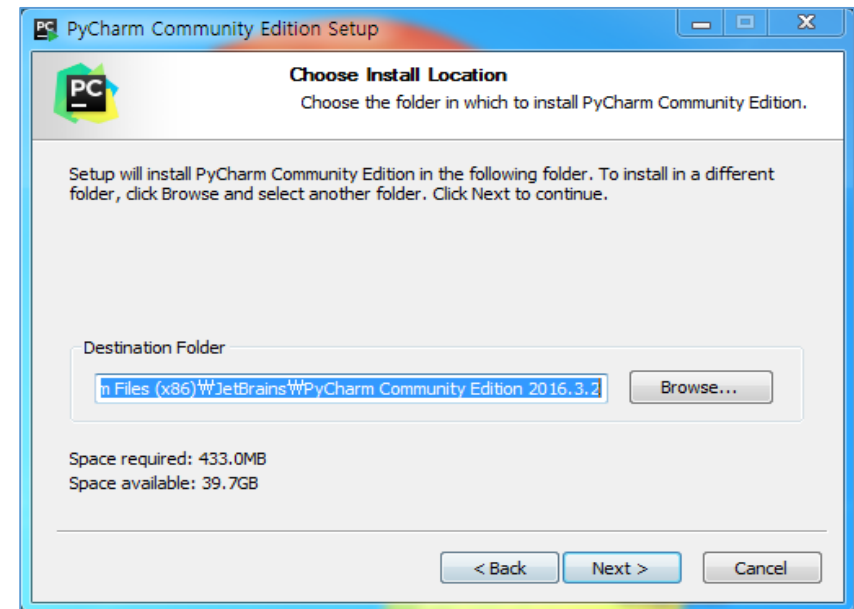
SECTION

Pycharm 설치

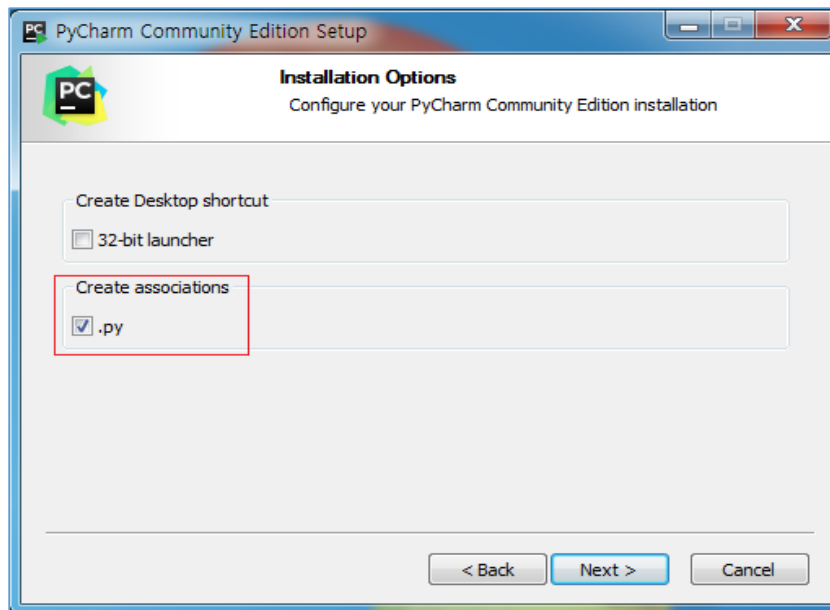
- 다운로드한 파일을 실행시켰을 때의 화면



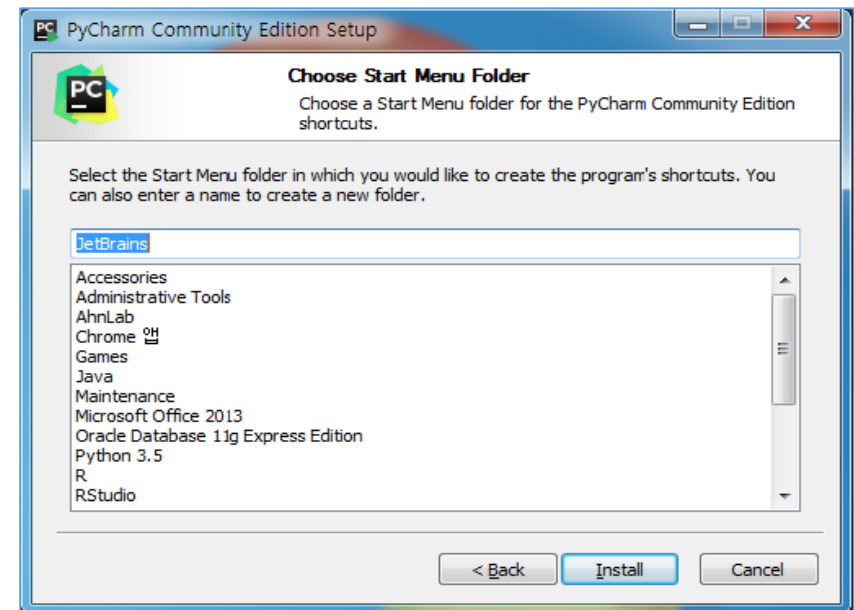
- 설치하고자 하는 파일의 경로 설정



- 기존의 파이썬 파일과의 연결



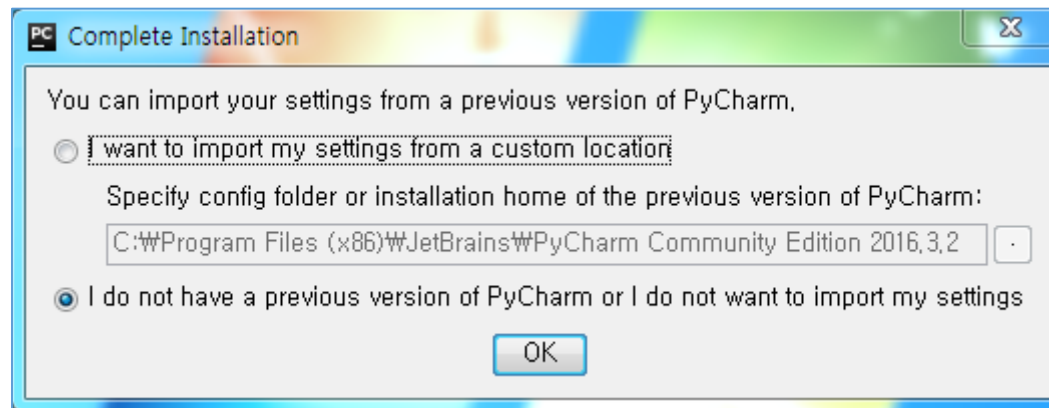
- 시작 프로그램 폴더 항목 지정



SECTION

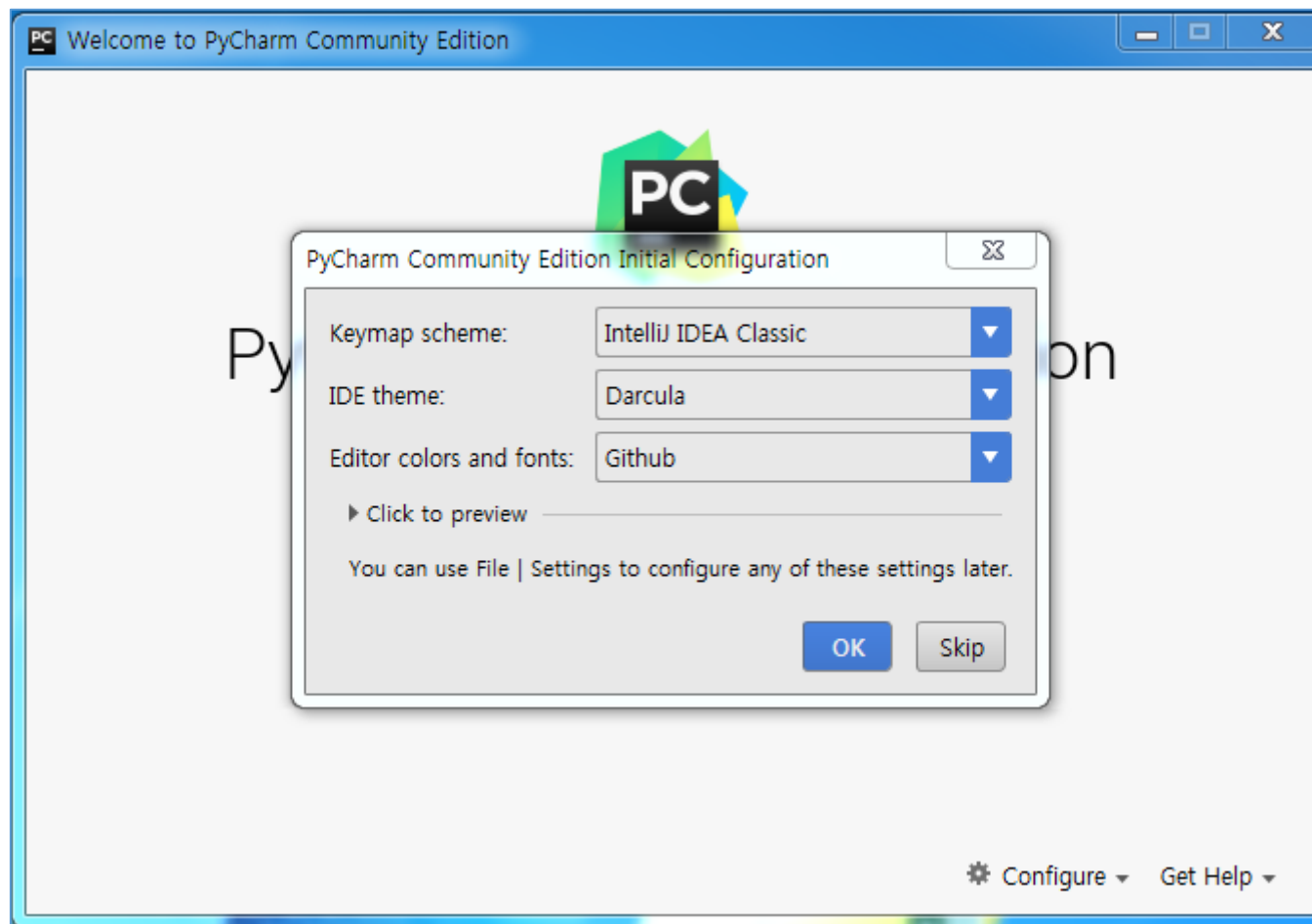
Pycharm 실행 및 설정

- Pycharm 처음 실행 시 화면

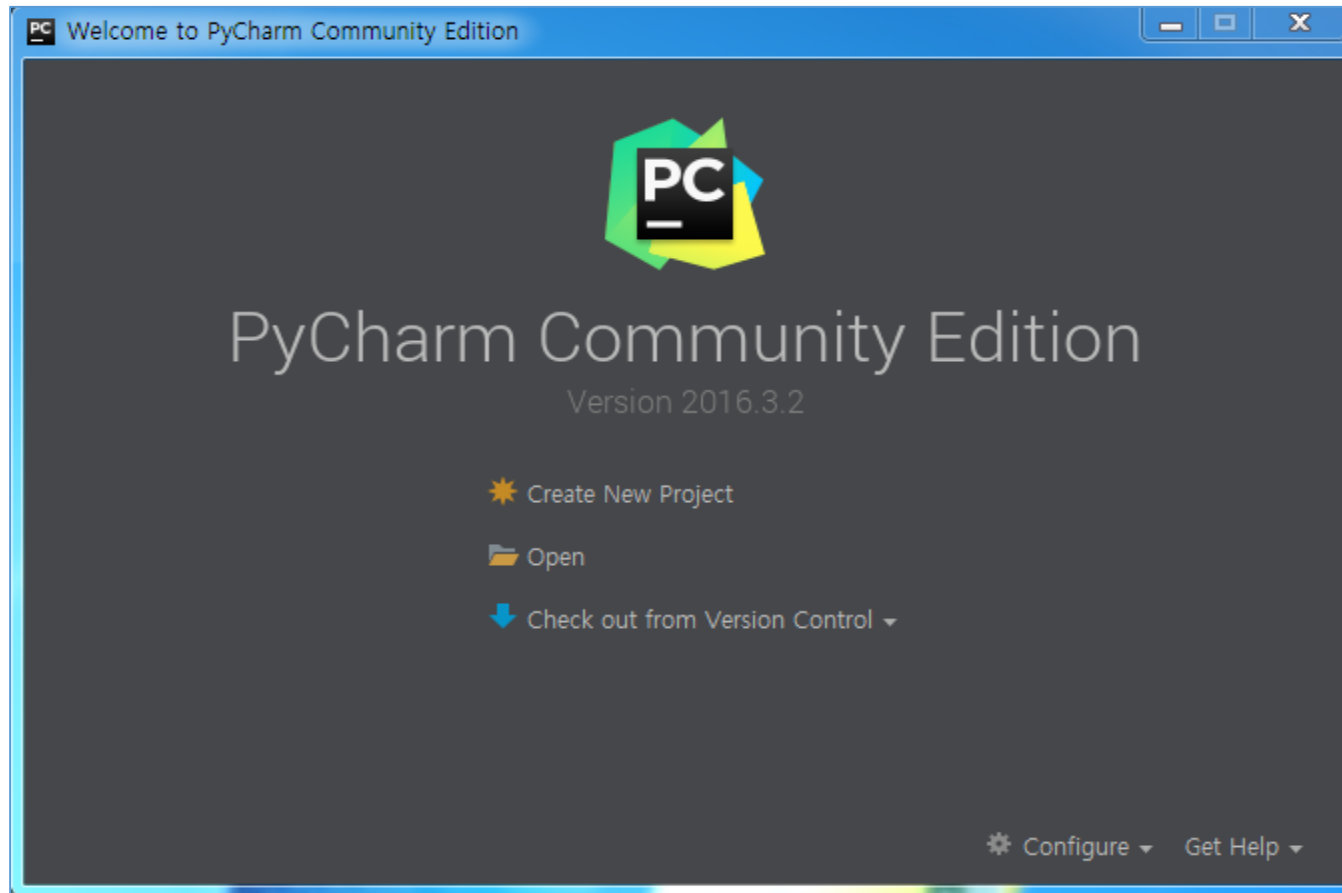


처음 실행 시 불러올 설정이 없으므로 아래 부분을 선택하고 OK를 누르면 된다.

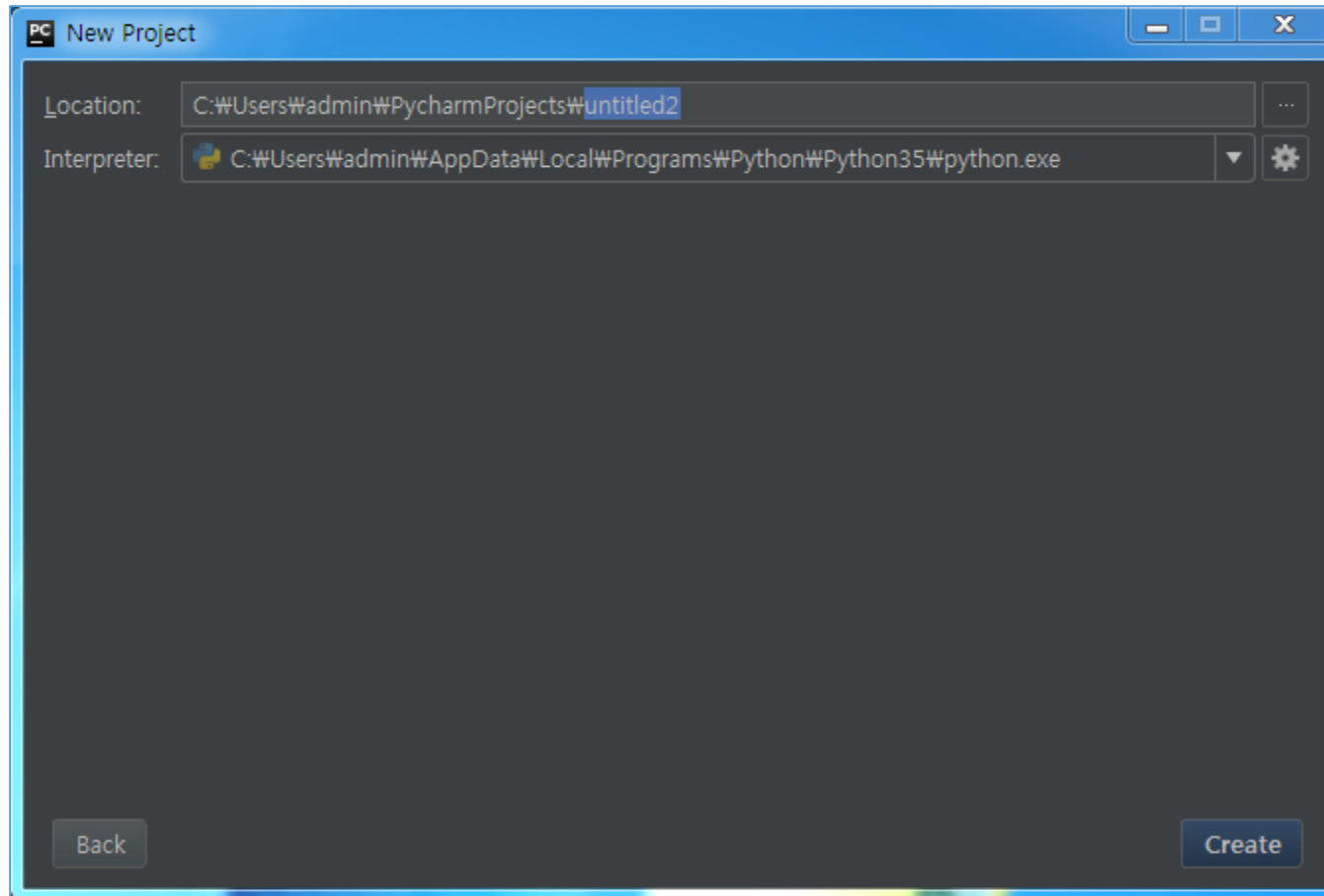
- Pycharm 테마 설정



- 테마 설정이 끝난 후의 Pycharm 실행 화면

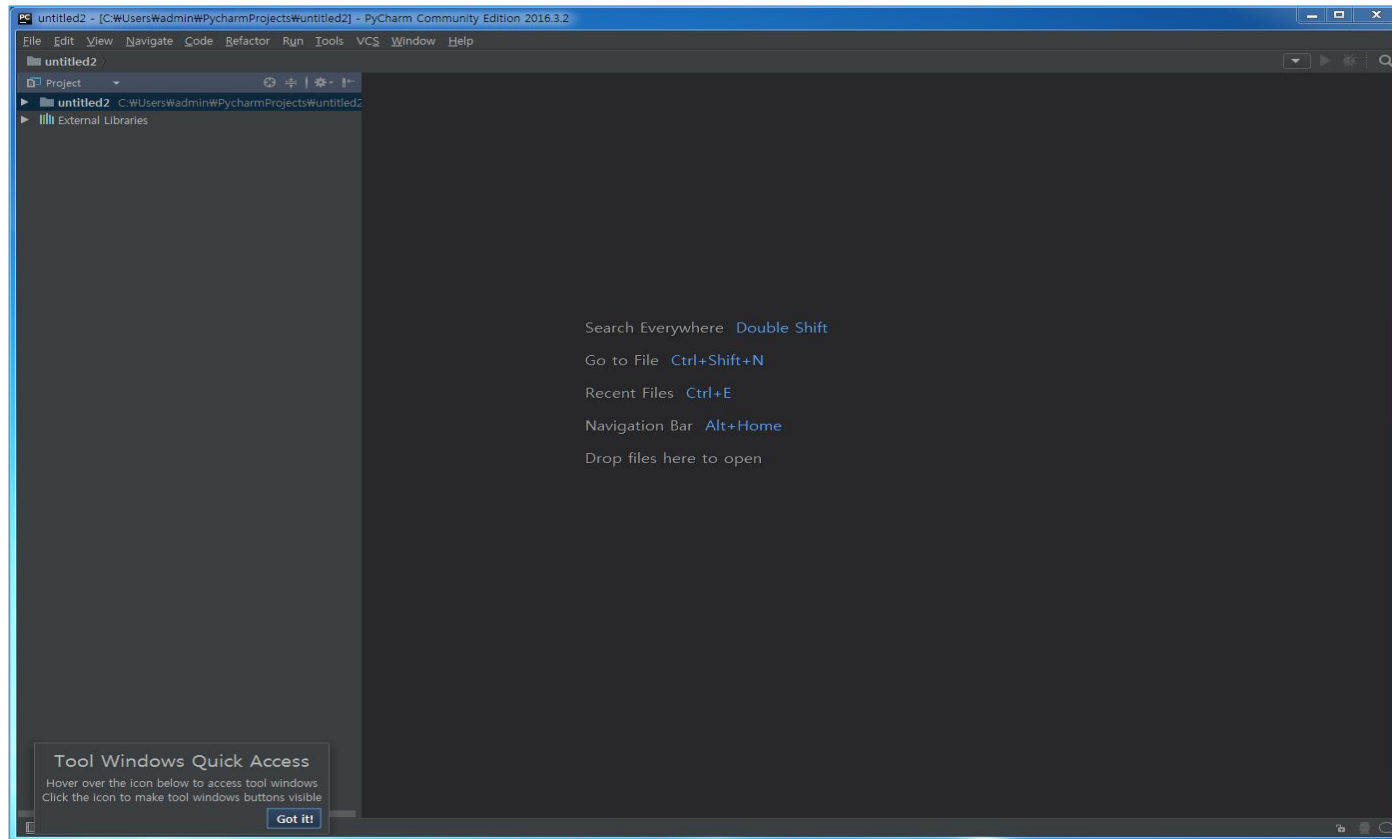


- 새 프로젝트 생성 및 경로 설정



프로젝트의 위치와 이름을 설정한 다음 오른쪽 하단의 CREATE를 클릭한다.

- 생성 및 설정이 완료된 후의 Pycharm 화면



본격적인 Python 프로그래밍이 가능한 에디터 창이 뜨는 것을 확인할 수 있다.

SECTION 1

변수에 대한 이해

파이썬에서 사용되는 '변수'는 모든 프로그래밍 언어에서 사용되는 기본 개념 중 하나로, 크게 숫자형과 문자형으로 나눌 수 있다. 이 장에서는 숫자형 변수의 계산과 문자형 변수의 형변환에 대해 다루고자 한다.

SECTION 1.1

변수

- 변수는 파이썬에서 가장 많이 사용되는 객체이다.
- 파이썬에서 변수 할당은 "=" 기호를 통해 이루어진다.

실행 코드

```
## 변수에 값 할당하기  
temp = 2  
print(temp)
```

출력 결과

2

SECTION 1.2

숫자

▪ 숫자의 사칙연산

실행 코드

```
## 사칙연산  
a=7  
b=3  
print(a+b)  
print(a-b)  
print(a*b)  
print(a/b)
```

출력 결과

```
10  
4  
21  
2.3333333333333335
```

- 몫 '//'

실행 코드

```
##### 몫 '//'  
a=7  
b=3  
print(a//b)
```

출력 결과

2

2.3333333 에서 몫 2만 출력된 것을 확인할 수 있다.

- 나머지 '%'

실행 코드

```
##### 나머지 '%'  
a=7  
b=3  
print(a%b)
```

출력 결과

1

몫이 2이고, 나머지가 1인 연산에서 나머지 1만 출력된 것을 확인할 수 있다.

- 지수 '**'

실행 코드

```
##### 지/수 '**'  
a=7  
b=3  
print(a**b)
```

출력 결과

343

7의 3제곱(7*7*7) 결과 343이 출력된 것을 확인할 수 있다.

SECTION 1.2

문자열

- 문자열

실행 코드

```
##### 문자열 할당  
temp = 'python is easy'  
temp
```

출력 결과

'python is easy'

print 함수를 이용하지 않고 변수만 출력하게 되면 문자열의 경우 ' ' 표시를 해줘 문장열임을 알 수 있다.
print 함수를 이용하면 ' ' 없이 할당되어있는 python is easy 라는 단어만 출력해 준다.
Ex) temp -> 'python is easy', print(temp) -> python is easy

- 문자열의 길이 len()

실행 코드

```
#### 문자열의 길이  
temp = 'python is easy'  
len(temp)
```

출력 결과

14

- 문자열의 결합

실행 코드

```
#### 공백없이  
'python'+'is easy'  
#### 공백있이  
'python '+'is easy'
```

출력 결과

```
'pythonis easy'  
'python is easy'
```

숫자와 마찬가지로 문자열도 변수에 지정한 후 함수를 사용 할 수 있다 예를 들어
a='python '
b='is easy'
a+b 의 결과는 'python is easy' 가 된다.

- 특정 문자 찾기 `index[]`

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|
| p | y | t | h | o | n | | i | s | | e | a | s | y |

실행 코드

```
##### index[ ]
temp='python is easy'
temp[3]
```

출력 결과

'h'

- 문자열 찾기 `slice[:]`

실행 코드

```
##### slice[: ]
temp='python is easy'
temp[0:5]
```

출력 결과

'pytho'

`slice[:]` 안에 값을 지정하지 않는 경우 끝까지 인식하여 값을 추출해 낸다.

Ex) `temp[:]` -> 'python is easy'
`temp[3:]` -> 'hon is easy'

- 문자열의 나누기 `split()`

실행 코드

```
##### split()  
temp='python is easy'  
temp.split()  
temp2='python,java,R,SAS'  
temp2.split(',')
```

출력 결과

```
['python', 'is', 'easy']  
['python', 'java', 'R', 'SAS']
```

- 문자열의 합치기 `join()`

실행 코드

```
##### join()  
temp2=['python', 'java', 'R', 'SAS']  
,'.join(temp2)
```

출력 결과

```
'python,java,R,SAS'
```

SECTION 2

데이터 구조 - 리스트, 튜플, 딕셔너리

데이터 구조는 1장에서 배운 숫자, 문자열과 같은 값들을 모아서 저장하는 방식을 의미하는 것으로 리스트, 튜플, 딕셔너리 각각에 따라 요구되는 형태와 사용방안이 있기 때문에 데이터 구조를 잘 이해하면 좀 더 파이썬을 간편하고 효율적으로 사용할 수 있다.

SECTION 2.1

리스트

- 리스트 생성하기 list()

실행 코드

```
#### 리스트 생성
number=['one','two','three','four']
print(number)
```

출력 결과

```
['one', 'two', 'three', 'four']
```


- 리스트 구조로 변환하기 list()

실행 코드

```
##### 리스트 구조로 변환
number_tuple=('one','two','three','four')
print(number_tuple)
list(number_tuple)
```

출력 결과

```
('one','two','three','four')
['one', 'two', 'three', 'four']
```

Tuple : 튜플 데이터 구조는 리스트 후에 다시 배우게 되는데, 리스트와 가장 큰 다른 점은 수정이 불가능하다는 것이다. 그리고 튜플은 리스트와 달리 대괄호 [] 를 사용하지 않고 소괄호 () 를 사용한다.

- 특정 값 찾기 index[], slice [:]

실행 코드

```
##### index[ ], slice[:]
number=['one','two','three','four']
number[1]
number[2:4]
```

출력 결과

```
'two'
['three', 'four']
```

파이썬에서 index 는 1이 처음 시작이 아니라 0으로 시작한다는 것이 특징이다.

문자열 slice 에서와 마찬가지로 step을 이용하여 [::2]일 경우 처음부터 마지막까지 2칸씩 항목을 추출하는 방법 등을 사용할 수 있다.

- 특정 값 변화 index[]

실행 코드

```
##### 리스트값 변환
number=['one','two','three','four']
number[0]='first'
print(number)
```

출력 결과

```
['first', 'two', 'three', 'four']
```

- 리스트 값 추가와 삭제 append(), del, remove()

실행 코드

```
##### append( )
number=['one','two','three','four']
number.append('five')
print(number)
```

출력 결과

```
['one', 'two', 'three', 'four', 'five']
```

- 리스트 값 추가와 삭제 `append()`, `del`, `remove()`

실행 코드

```
##### del  
number=['one','two','three','four']  
del number[1]  
print(number)
```

출력 결과

```
['one', 'three', 'four']
```

실행 코드

```
##### remove()  
number.remove('three')  
print(number)
```

출력 결과

```
['one', 'four']
```

- 리스트 병합 `extend()`

실행 코드

```
##### extend()  
number1=['one','two','three','four']  
number2=['six','seven','eight','nine']  
number1.extend(number2)  
print(number1)
```

출력 결과

```
['one', 'two', 'three', 'four', 'six', 'seven', 'eight', 'nine']
```

Tuple : 튜플 데이터 구조는 리스트 후에 다시 배우게 되는데, 리스트와 가장 큰 다른 점은 수정이 불가능하다는 것이다. 그리고 튜플은 리스트와 달리 대괄호 `[]` 를 사용하지 않고 소괄호 `()` 를 사용한다.

- 정렬하기 `sort()`, `sorted()`

실행 코드

```
##### sorted()  
number=['one','two','three','four']  
newnumber=sorted(number)  
print(newnumber)  
print(number)
```

출력 결과

```
['four', 'one', 'three', 'two']  
['one', 'two', 'three', 'four']
```

실행 코드

```
##### sort()  
number.sort()  
print(number)
```

출력 결과

```
['four', 'one', 'three', 'two']
```

숫자를 문자로 표현했기 때문에 숫자 순서가 아닌 문자열의 정렬 순서인 알파벳 순서로 정렬을 하게 된다.

- 정렬하기 `sort()`

실행 코드

```
##### sort(reverse=True)  
number=[1,2,3,4]  
number.sort(reverse=True)  
print(number)
```

출력 결과

```
[4, 3, 2, 1]
```

- 항목 개수 `len()`

실행 코드

```
##### len()  
number=['one', 'two', 'three', 'four']  
len(number)
```

출력 결과

```
4
```

SECTION 2.2

튜플

- 튜플은 수정이나 삭제, 추가 등이 불가능하다
- 공간 확보성, 비 변화성, 함수 인자로의 사용 등 이점이 있다.

- 튜플 생성하기 `tuple()`

실행 코드

```
##### tuple()  
number=(1,2,3,4)  
print(number)  
type(number)
```

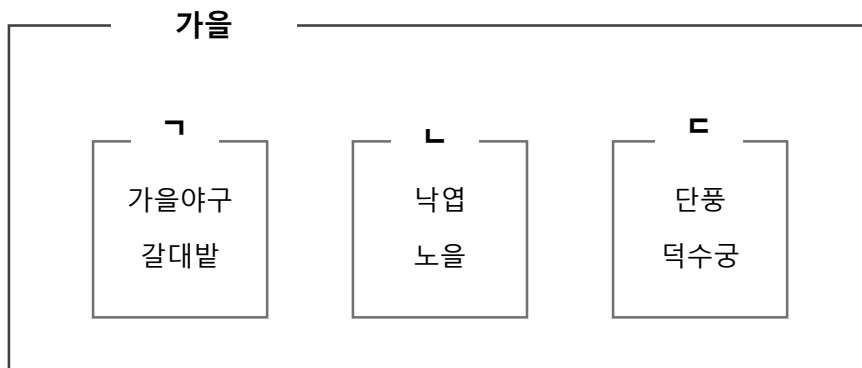
출력 결과

```
(1, 2, 3, 4)  
<class 'tuple'>
```

`type()` 함수를 이용하면 해당하는 객체의 데이터 구조가 무엇인지 알 수 있다.

SECTION 2.3

딕셔너리



- 딕셔너리 – 객체로서 구조의 종류중 하나 ex) 가을
- Key – 딕셔너리 안에서 사용 되는 인덱스 ex) 'ㄱ' 'ㄴ' 'ㄷ'
- 항목 – key 에 할당되어 있는 value ex) '가을야구' '갈대밭'

- 딕셔너리 구조로 변환하기 dict()

실행 코드

```
##### dict()  
fruit_price = [['apple',3500],['peer',2500],['cherry',5000]]  
fruit_price=dict(fruit_price)  
print(fruit_price)
```

출력 결과

```
{'cherry': 5000, 'peer': 2500, 'apple': 3500}
```

리스트형태를 딕셔너리로 변환하기 위해서는 리스트 안에 리스트 l이 존재해야 한다. 리스트 l은 리스트 안에서 하나의 항목을 다시 리스트로 묶어주는 형태이다.

- 키 변환하기 keys()

실행 코드

```
##### keys()  
fruit_price = {'apple' : 3500, 'peer' : 2500, 'cherry' : 5000}  
fruit_price.keys()
```

출력 결과

```
dict_keys(['peer', 'cherry', 'apple'])
```


- 항목 반환하기 index[]

실행 코드

```
##### index( )  
fruit_price = {'apple' : 3500, 'peer' : 2500, 'cherry' : 5000}  
fruit_price['peer']
```

출력 결과

2500

실행 코드

```
##### index( )  
school_class = {1: ['mina', 'jane', 'mark'], 2: ['peter', 'pay', 'parker'], 3: ['pay', 'joy', 'conan']}  
school_class.get(1)
```

출력 결과

['mina', 'jane', 'mark']

딕셔너리는 key가 존재하기 때문에 숫자로 index처리를 하지 못하지만 추후에 배울 pandas의 dataframe을 활용하면 숫자로도 index를 실행 할 수 있다.

- 딕셔너리 병합 update ()

실행 코드

```
##### update()  
fruit_price = {'apple' : 3500, 'peer' : 2500, 'cherry' : 5000}  
others = {'banana':3200,'grape':2000}  
fruit_price.update(others)  
print(fruit_price)
```

출력 결과

```
{'peer': 2500, 'cherry': 5000, 'apple': 3500, 'banana': 3200, 'grape': 2000}
```

실행 코드

```
##### update() - 리스트 항목  
school_class = {1: ['mina', 'jane', 'mark'], 2 : ['peter', 'pay', 'parker']}  
other_class = {3:['john','eric','terry']}  
school_class.update(other_class)  
print(school_class)
```

출력 결과

```
{1: ['mina', 'jane', 'mark'], 2: ['peter', 'pay', 'parker'], 3: ['john', 'eric', 'terry']}
```

- 삭제하기 `del`

실행 코드

```
##### del
fruit_price = {'apple' : 3500, 'peer' : 2500, 'cherry' : 5000}
del fruit_price['apple']
print(fruit_price)
```

출력 결과

```
{'peer': 2500, 'cherry': 5000}
```

실행 코드

```
##### del - 리스트 항목
school_class = {1: ['mina', 'jane', 'mark'], 2: ['peter', 'pay', 'parker']}
del school_class[3]
print(school_class)
```

출력 결과

```
{1: ['mina', 'jane', 'mark'], 2: ['peter', 'pay', 'parker']}
```

SECTION 3

조건문과 반복문

조건문과 반복문은 모든 언어에서 가장 기본이 되고 많이 사용하는 명령어이다. 특히, 최근 데이터들이 점점 big data화 됨에 원시 데이터를 그대로 분석을 할 수 없어 데이터의 핸들링이 중요해 지고 있기 때문에 이런 상황에 조건문과 반복문 등의 사용능력은 점점 중요해 지는 실정이다.

SECTION 3.1

boolean

▪ boolean 데이터 타입

실행 코드

```
#### boolean  
python = True  
type(python)
```

출력 결과

```
<class 'bool'>
```

- 연산자 정리

| 비교 연산자 | 작용원리 | 예제 |
|--------------------|-----------------|---------------------------------------|
| <code>==</code> | 같다 | <code>(a==b)=False</code> |
| <code>!=</code> | 다르다 | <code>(a!=b)=True</code> |
| <code><</code> | 보다 작다 | <code>(a<b)=True</code> |
| <code><=</code> | 보다 작거나 같다 | <code>(a<=b)=True</code> |
| <code>></code> | 보다 크다 | <code>(a>b)=False</code> |
| <code>>=</code> | 보다 크거나 같다 | <code>(a>=b)=False</code> |
| <code>in</code> | 이들 중 같은 값이 존재한다 | <code>(a in (10, 20, 30))=True</code> |

| 논리 연산자 | 작용원리 | 예제 |
|--------------------|--------|--|
| <code>and</code> | 그리고 | <code>(a==b and a==c)= False</code> |
| <code>&</code> | 그리고 | <code>(a==b) & (a==c)=False</code> |
| <code>Or</code> | 또는 | <code>(a==b) or (a==c)=True</code> |
| <code> </code> | 또는 | <code>(a==b) (a==c)=True</code> |
| <code>not</code> | ~이 아니다 | <code>not(a==b)=True</code> |

| 식별 연산자 | 작용원리 | 예제 |
|---------------------|--------|--------------------------------|
| <code>is</code> | ~이다 | <code>(a is b)=False</code> |
| <code>is not</code> | ~이 아니다 | <code>(a is not b)=True</code> |

SECTION 3.2

조건문 if, elif, else

▪ If 문

실행 코드

```
##### if  
python = True  
if(python):  
    print("조건문이 참입니다.")
```

출력 결과

조건문이 참입니다.

if 뒤 괄호 () 안에 조건을 작성하며 그 내용은 주로 비교, 논리, 식별연산자들을 이용하여 사용한다.

- else 문

실행 코드

```
##### else
python = False
if(python) :
    print("조건문이 참입니다.")
else :
    print("조건문이 참이 아닙니다.")
```

출력 결과

조건문이 참이 아닙니다.

- elif 문

실행 코드

```
##### elif
python = False
pycharm = True
if(python) :
    print("첫번째 조건문이 참입니다.")
elif (pycharm) :
    print("첫번째 조건문은 참이 아니고 두번째 조건문은 참입니다.")
else :
    print("조건문이 참이 아닙니다.")
```

출력 결과

조건문이 참이 아닙니다.

SECTION 3.3

반복문 for

▪ for 문

실행 코드

```
##### for  
for i in range(1,10) :  
    print(i)
```

출력 결과

```
1  
2  
3  
4  
5  
6  
7  
8  
9
```


- 리스트 for 문

실행 코드

```
##### 리스트 for문
fruit = ['apple', 'peer', 'cherry']
type(fruit)for i in fruit:
    print(i)
```

출력 결과

```
<class 'list'>apple
peer
cherry
```

- 딕셔너리 for 문

실행 코드

```
##### 딕셔너리 for문 key()
fruit_price = {'apple' : 3500, 'peer' : 2500, 'cherry' : 5000}
for i in fruit_price.keys():
    print(i)
```

출력 결과

```
apple
peer
cherry
```

딕셔너리 구조에서 keys() 의 역할은 딕셔너리의 key들을 모두 반환해주는 역할을 한다.

실행 코드

```
##### 딕셔너리 for문 values()  
fruit_price = {'apple' : 3500, 'peer' : 2500, 'cherry' : 5000}  
for i in fruit_price.values():  
    print(i)
```

출력 결과

```
3500  
2500  
5000
```

딕셔너리 구조에서 values()의 역할은 딕셔너리의 항목들을 모두 반환해주는 역할을 한다.

실행 코드

```
fruit_price = {'apple' : 3500, 'peer' : 2500, 'cherry' : 5000}  
for item in fruit_price.items():  
    print(item)
```

출력 결과

```
peer 2500  
cherry 5000  
apple 3500
```

딕셔너리 구조에서 items()의 역할은 딕셔너리의 키와 항목을 모두 반환해 주는 역할을 하기 때문에 처음의 key와 value인 peer 2500을 반환해 주고 그 후 순서대로 결과를 표출해 준다.

- for 문 활용

실행 코드

```
#### for문의 활용
high_price=[]
low_price=[]
fruit_price ={'apple' : 3500, 'peer' : 2500, 'cherry' : 5000}
for item in fruit_price.items():
    if item[1] >3000 :
        high_price.append(item[0])
    else :
        low_price.append(item[0])
print(high_price)
print(low_price)
```

출력 결과

```
['apple', 'cherry']
['peer']
```

```
# high_price 와 low_price에 빈 리스트 할당
# fruit_price 딕셔너리를 순회 하면서 가격이 3000원 이상인 과일은 high_price 그 외 과일은 low_price에 할당
# append( ) 함수를 이용하여 순회하며 해당 리스트에 추가
```

SECTION 3

데이터 입출력

기술의 발달과 함께 점점 데이터는 빅데이터화 되어가고 있고, 많은 현장에서는 데이터베이스와 연동하는 등의 방법을 통해 데이터를 다루고 있다. 이 점에 대해선 추후 더욱 자세히 언급하고, 이번 장에서는 txt, csv 등 파일단위를 통한 파일 입출력을 다루겠다.

SECTION 3.1

데이터 출력

▪ txt 파일 출력

실행 코드

```
#### print( ) 데이터출력 - txt
fruit = open('D://fruit.txt','wt')
fruit_price = {'apple' : 3500, 'peer' : 2500, 'cherry' : 5000}
for item in fruit_price.items():
    print(item,file=fruit)
fruit.close()
```

실행 코드

```
##### write( ) 데이터출력 - txt
fruit = open('D://fruit2.txt','wt')
fruit.write('apple 3500Wn')
fruit.write('peer 2500Wn')
fruit.write('cherr 5000Wn')
fruit.close()
```

- Csv 파일 출력

출력 코드

```
##### write( ) 데이터출력 - csv
import csv
with open('D://fruit.csv','w',newline='') as fruit:
    writer = csv.writer(fruit,delimiter=',')
    writer.writerow(['apple'])
    writer.writerow(['peer'])
    writer.writerow(['cherry'])
    writer.writerow(['banana'])
    writer.writerow(['peach'])
```

SECTION 3.2

데이터 입력

▪ txt 파일 입력

실행 코드

```
##### open( ), read 데이터 입력 - txt  
fruit_path = open('d:\wwfruit.txt','rt')  
fruit_path  
  
fruit = fruit_path.readlines()  
print(fruit)
```

출력 결과

```
[ "('apple', 3500)\n", "('cherry', 5000)\n", "('peer', 2500)\n"]
```

- csv 파일 입력

실행 코드

```
import csv

f = open('d:WWfruit.csv','rt')
csvRead = csv.reader(f)
csvRead
<_csv.reader object at 0x02E58EB0>

fruit = []
for i in csvRead:
    fruit.append(csvRead)
    print(i)
```

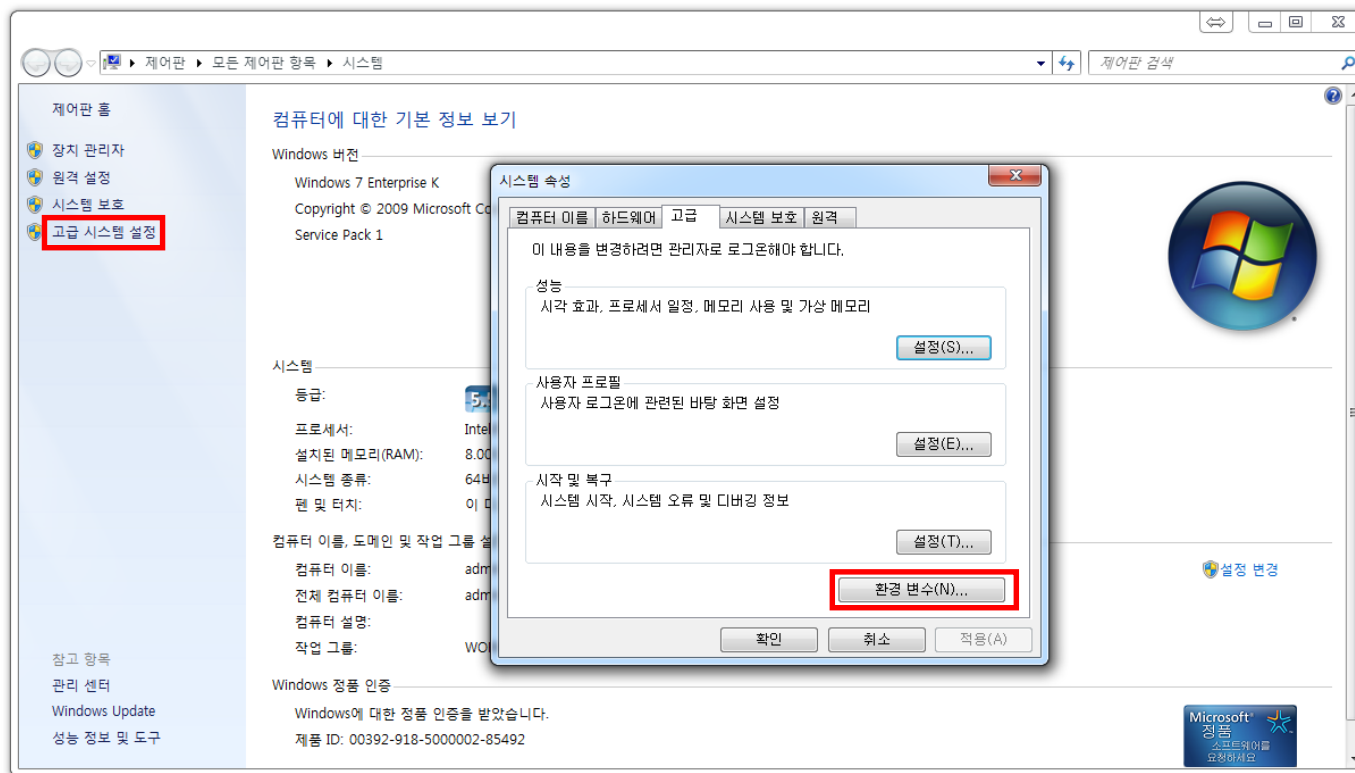
출력 결과

```
['apple']
['peer']
['cherry']
['banana']
['peach']
```

SECTION 1.1

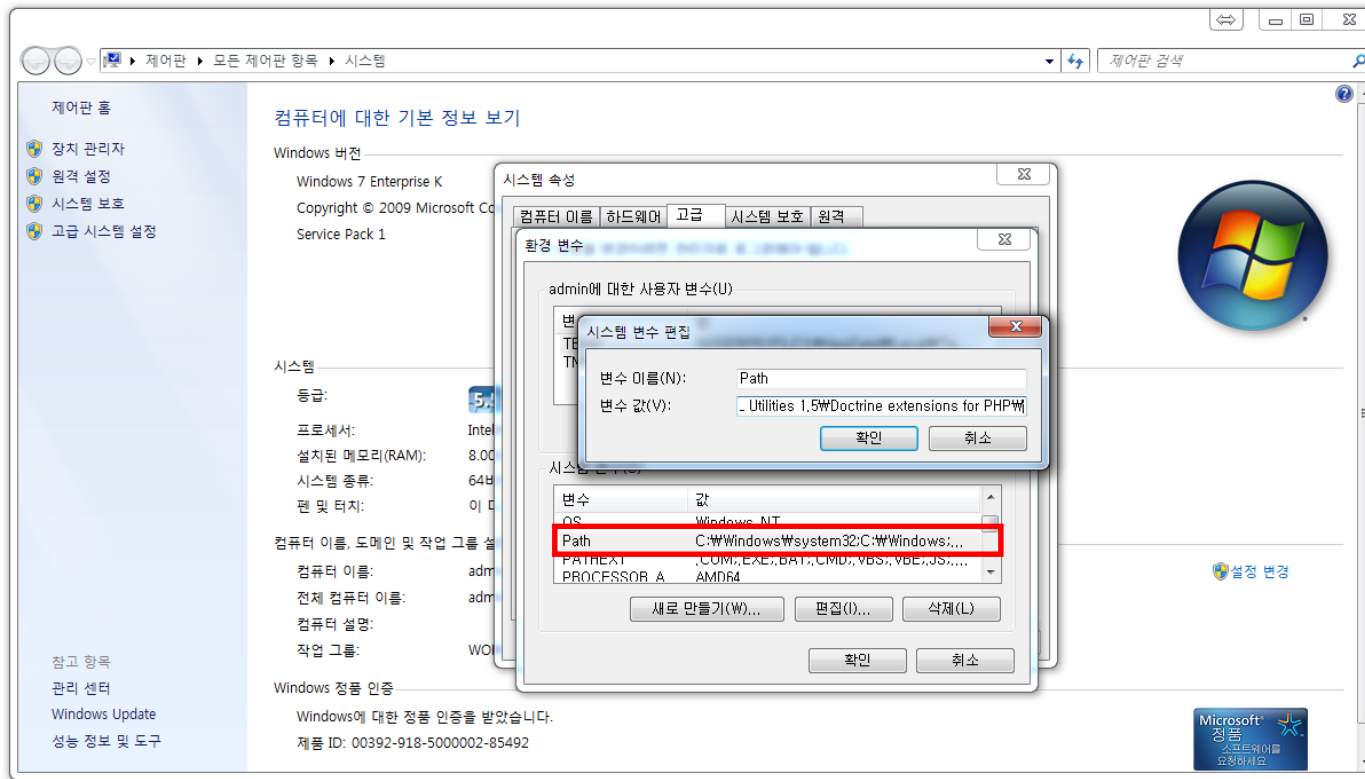
패키지 설치

■ 환경변수 설정



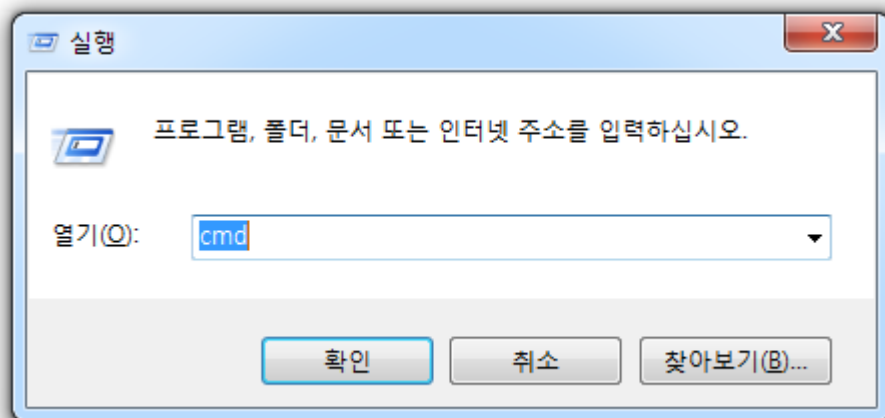
고급 시스템 설정 > 환경변수

- 환경변수 설정

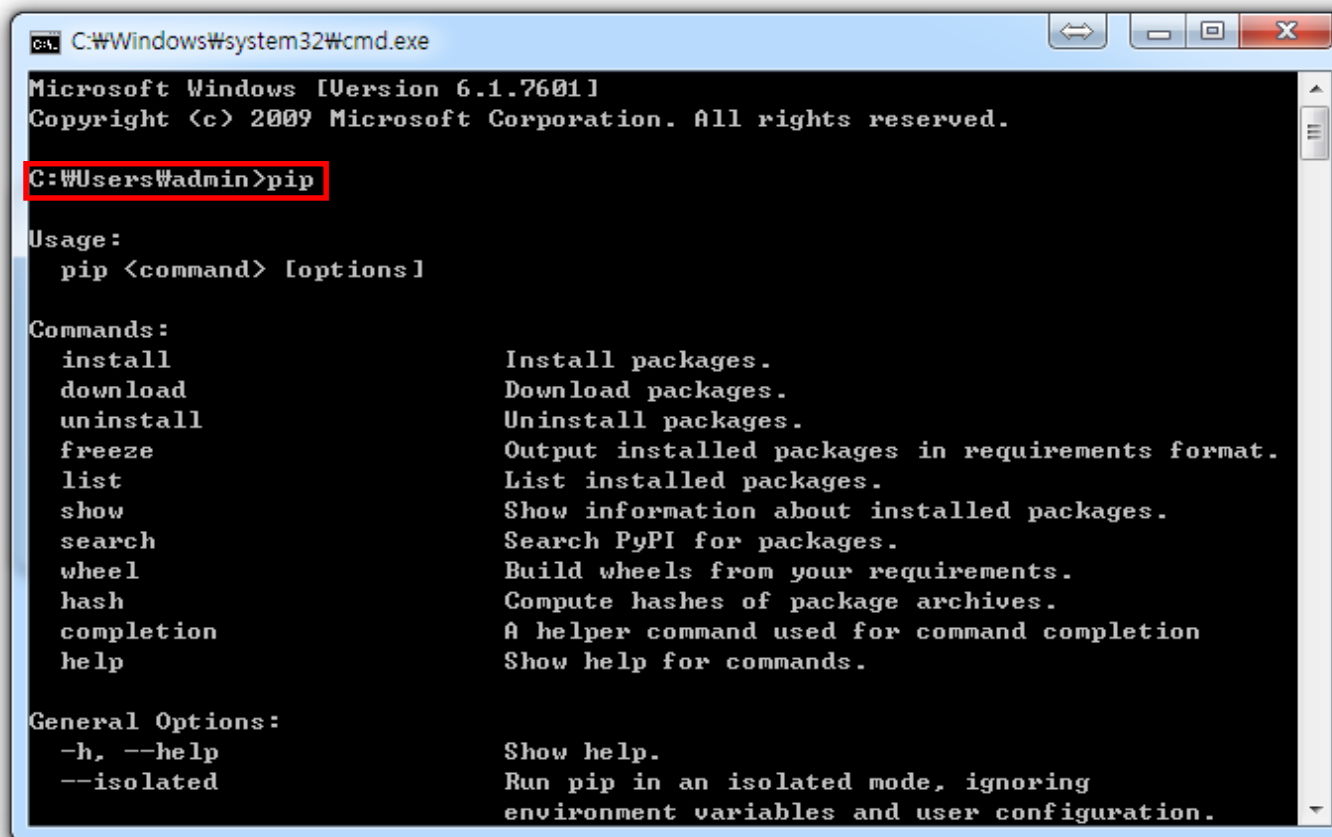


변수 값 뒤에 python의 script 경로 추가

- pip 을 활용한 패키지 설치



- pip 을 활용한 패키지 설치



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\admin>pip

Usage:
  pip <command> [options]

Commands:
  install           Install packages.
  download          Download packages.
  uninstall         Uninstall packages.
  freeze            Output installed packages in requirements format.
  list              List installed packages.
  show              Show information about installed packages.
  search            Search PyPI for packages.
  wheel             Build wheels from your requirements.
  hash              Compute hashes of package archives.
  completion        A helper command used for command completion
  help              Show help for commands.

General Options:
  -h, --help        Show help.
  --isolated         Run pip in an isolated mode, ignoring
                    environment variables and user configuration.
```

- pip 을 활용한 패키지 설치

```
C:\Windows\system32\cmd.exe

download. Implied with --no-index.

C:\Users\Wadmin>pip install pandas
Collecting pandas
  Downloading pandas-0.19.0-cp35-cp35m-win32.whl (6.4MB)
    100% |#####| 6.4MB 210kB/s
Collecting numpy>=1.7.0 (from pandas)
  Downloading numpy-1.11.2-cp35-none-win32.whl (6.6MB)
    100% |#####| 6.6MB 193kB/s
Collecting pytz>=2011k (from pandas)
  Downloading pytz-2016.7-py2.py3-none-any.whl (480kB)
    100% |#####| 481kB 2.2MB/s
Collecting python-dateutil>=2 (from pandas)
  Downloading python_dateutil-2.5.3-py2.py3-none-any.whl (201kB)
    100% |#####| 204kB 3.3MB/s
Collecting six>=1.5 (from python-dateutil>=2->pandas)
  Downloading six-1.10.0-py2.py3-none-any.whl
Installing collected packages: numpy, pytz, six, python-dateutil, pandas
Successfully installed numpy-1.11.2 pandas-0.19.0 python-dateutil-2.5.3 pytz-2016.7 six-1.10.0
You are using pip version 8.1.1, however version 8.1.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\Wadmin>
```

SECTION 2

자료 구조 : Seires 와 Datafarme

Pandas에서 제공하는 데이터 자료구조는 Series와 Dataframe 두가지가 존재하는데 Series는 시계열과 유사한 데이터로서 index와 value 가 존재하고 Dataframe은 딕셔너리데이터를 매트릭스 형태로 만들어 준 것과 같은 frame 을 가지고 있다. 이런 데이터 구조를 통해 시계열, 비시계열 데이터를 통합하여 다룰 수 있고 이를 좀더 자세히 살펴보자.

SECTION 2.1

Series

▪ Series 구조

실행 코드

```
##### import
from pandas import Series, DataFrame
import pandas as pd
##### series
fruit = Series([2500,3800,1200,6000],index=['apple','banana','peer','cherry'])
print(fruit)
```

출력 결과

```
apple    2500
banana   3800
peer     1200
cherry   6000
```

- 특정값 추출 values, index

실행 코드

```
##### values index
fruit = Series([2500,3800,1200,6000],index=['apple','banana','peer','cherry'])
print(fruit.values)
print(fruit.index)
```

출력 결과

```
[2500 3800 1200 6000]
Index(['apple', 'banana', 'peer', 'cherry'], dtype='object')
```

Series의 형태는 딕셔너리와 매우 유사하여 key 가 index로 바뀌었다고 착각할 수 있지만 사실상 데이터 구조 자체가 매우 다르기 때문에 딕셔너리에서 사용하는 방식이 조금씩 다르기 때문에 주의해야 한다.

- 특정값 추출 values, index

실행 코드

```
##### Seires( )
fruitData = {'apple': 2500,'banana':3800,'peer':1200,'cherry':6000}
fruit = Series(fruitData)
type(fruitData)
type(fruit)
```

출력 결과

```
<class 'dict'>
<class 'pandas.core.series.Series'>
```

- Value와 index 이름 지정하기 name

실행 코드

```
##### name
fruit = Series([2500, 3800, 1200, 6000], index=['apple', 'banana', 'peer', 'cherry'])
fruit.name = 'fruitPrice'
fruit.index.name = 'fruitName'
print(fruit)
```

출력 결과

```
fruitName
apple    2500
banana   3800
peer     1200
cherry    6000
Name: fruitPrice, dtype: int64
```

index에 해당하는 fruitName이 index위에 지정이 되고 value의 역할인 fruitPrice는 아래에서 표출 된다.

SECTION 2.2

Dataframe

▪ Dataframe 구조

실행 코드

```
##### Dataframe()  
fruitData = {'fruitName':['apple','banana','cherry','peer'],  
             'fruitPrice':[2500,3800,6000,1200],  
             'num':[10,5,3,8]}  
  
fruitFrame = DataFrame(fruitData)  
print(fruitFrame)
```

출력 결과

| | fruitName | fruitPrice | num |
|---|-----------|------------|-----|
| 0 | apple | 2500 | 10 |
| 1 | banana | 3800 | 5 |
| 2 | cherry | 6000 | 3 |
| 3 | peer | 1200 | 8 |

딕셔너리의 각 항목들이 데이터 프레임의 하나의 컬럼의 형태로 변환이 가능하다.

- 칼럼순서 지정하기 `columns`

실행 코드

```
##### columns
fruitData = {'fruitName':['apple','banana','cherry','peer'],
             'fruitPrice':[2500,3800,6000,1200],
             'num':[10,5,3,8]}
fruitFrame = DataFrame(fruitData, columns = ['fruitPrice','num','fruitName'])
print(fruitFrame)
```

출력 결과

| | fruitprice | num | fruitName |
|---|------------|-----|-----------|
| 0 | NaN | 10 | apple |
| 1 | NaN | 5 | banana |
| 2 | NaN | 3 | cherry |
| 3 | NaN | 8 | peer |

- 특정항목 추출하기

실행 코드

```
#### 사전형식  
fruitFrame['fruitName']
```

출력 결과

```
0    apple  
1    banana  
2    cherry  
3     peer  
Name: fruitName, dtype: object
```

실행 코드

```
#### 속성형식  
fruitFrame.fruitName
```

출력 결과

```
0    apple  
1    banana  
2    cherry  
3     peer  
Name: fruitName, dtype: object
```

- 컬럼 추가하기

실행 코드

```
#### 컬럼 추가하기 - 같은 값  
fruitFrame['year'] = 2016  
print(fruitFrame)
```

출력 결과

| | fruitName | fruitPrice | num | year |
|---|-----------|------------|-----|------|
| 0 | apple | 2500 | 10 | 2016 |
| 1 | banana | 3800 | 5 | 2016 |
| 2 | cherry | 6000 | 3 | 2016 |
| 3 | peer | 1200 | 8 | 2016 |

데이터 프레임이 기존의 파이썬의 데이터 구조보다 조금 더 간편한 것 중 하나가 바로 이런 데이터 추가방식이다. 기존의 데이터 구조 리스트나 딕셔너리의 경우 추가하는 function을 이용하여 추가하는 것이 번거롭지만 데이터 프레임의 경우 위와 같이 새로운 칼럼을 지정만 해주면 된다.

실행 코드

```
#### 컬럼 추가하기 - 다른 값  
variable=Series([4,2,1],index=[0,2,3])  
fruitFrame['stock'] = variable  
print(fruitFrame)
```

출력 결과

| | fruitName | fruitPrice | num | year | stock |
|---|-----------|------------|-----|------|-------|
| 0 | apple | 2500 | 10 | 2016 | 4.0 |
| 1 | banana | 3800 | 5 | 2016 | NaN |
| 2 | cherry | 6000 | 3 | 2016 | 2.0 |
| 3 | peer | 1200 | 8 | 2016 | 1.0 |

데이터프레임은 index가 존재하기 때문에 index를 지정할 수 있는 Series를 통해 새로운 컬럼을 정의한다.
새로 추가하는 index의 값이 존재하지 않을경우 결측치라는 의미로 NaN을 자동으로 지정한다.

SECTION 3

자료 다루기

앞 장에서 Pandas의 자료 구조 형태인 Series와 Dataframe에 대한 생성 방식과 기본적인 데이터 구조 다루는 방법에 대해서 언급했고 이번 장은 좀 더 자세히 Series와 Dataframe을 이용하여 분석용 데이터를 만들기 위한 자료 다루는 방법에 대해서 배우겠다.

SECTION 3.1

데이터 구조의 항목삭제

▪ Series 데이터 row 삭제하기 `drop()`

실행 코드

```
##### row 삭제하기 drop()  
fruit = Series([2500,3800,1200,6000],index=['apple','banana','peer','cherry'])  
new_fruit = fruit.drop('banana')  
print(fruit)  
print(new_fruit)
```

출력 결과

```
##### fruit  
apple    2500  
banana   3800  
peer     1200  
cherry   6000
```

```
##### new_fruit  
apple    2500  
peer     1200  
cherry   6000
```

banana의 row가 사라진 것을 확인 할 수 있다.

- Dataframe row 삭제하기 drop()

실행 코드

```
##### row 삭제하기 drop()  
fruitData = {'fruitName':['apple','banana','cherry','peer'],  
             'fruitPrice':[2500,3800,6000,1200],  
             'num':[10,5,3,8]}  
fruitName=fruitData['fruitName']  
fruitFrame = DataFrame(fruitData,index = fruitName,columns=['fruitPrice','num'])  
fruitFrame2=fruitFrame.drop(['apple','cherry'])  
  
print(fruitFrame)  
print(fruitFrame2)
```

출력 결과

```
##### fruitFrame  
      fruitPrice  num  
apple        2500   10  
banana        3800    5  
cherry        6000    3  
peer         1200    8  
  
##### fruitFrame2  
      fruitPrice  num  
banana        3800    5  
peer         1200    8
```

Dataframe 의 인자에 index를 추가하면 원하는 index값을 지정 할 수 있다.
index는 columns와 다르게 DataFrame을 만들고자 하는 데이터에서 가져올 수 없고 따로 객체가 존재해야 한다.

- DataFrame column 삭제하기 drop()

실행 코드

```
##### column 삭제하기 drop()  
fruitFrame3=fruitFrame.drop('num',axis=1)  
print(fruitFrame)  
print(fruitFrame3)
```

출력 결과

```
##### fruitFrame  
      fruitPrice  num  
apple         2500   10  
banana        3800    5  
cherry        6000    3  
peer         1200    8  
  
##### fruitFrame3  
      fruitPrice  
apple         2500  
banana        3800  
cherry        6000  
peer         1200
```

row삭제와 마찬가지로 여러 개의 컬럼을 삭제하려면 ['num','fruitPrice']처럼 리스트형태로 값을 지정하면 된다.

SECTION 3.2

항목 추출하기 slice

▪ Series 의 slice

실행 코드

```
#### Series Slice  
fruit = Series([2500,3800,1200,6000],index=['apple','banana','peer','cherry'])  
fruit['apple':'peer']
```

출력 결과

```
apple    2500  
banana   3800  
peer     1200  
dtype: int64
```

Series에서 slice는 각 항목이 아닌 시작점과 끝점을 표시하여 slice를 실시한다

▪ DataFrame 의 slice

실행 코드

```
##### dataframe Slice
fruitData = {'fruitName':['apple','banana','cherry','peer'],
             'fruitPrice':[2500,3800,6000,1200],
             'num':[10,5,3,8]}
fruitName=fruitData['fruitName']
fruitFrame = DataFrame(fruitData,index = fruitName,columns=['fruitPrice','num'])
print(fruitFrame)
fruitFrame['fruitPrice']
fruitFrame['apple':'banana']
```

출력 결과

```
##### print(fruitFrame)
   fruitPrice  num
apple      2500   10
banana     3800    5
cherry     6000    3
peer       1200    8
```

```
##### fruitFrame['fruitPrice']
apple      2500
banana     3800
cherry     6000
peer       1200
Name: fruitPrice, dtype: int64
```

```
##### fruitFrame['apple':'banana']
   fruitPrice  num
apple      2500   10
banana     3800    5
```

SECTION 3.3

데이터의 기본연산

▪ Series의 연산

실행 코드

```
##### Series 연산
fruit1 = Series([5,9,10,3],index = ['apple','banana','cherry','peer'])
fruit2 = Series([3,2,9,5,10],index = ['apple','orange','banana','cherry','mango'])
print(fruit1)
print(fruit2)
fruit1 + fruit2
```

출력 결과

```
##### print(fruit1)
apple      5
banana     9
cherry    10
peer       3
dtype: int64
```

```
##### print(fruit2)
apple      3
orange     2
banana     9
cherry     5
mango     10
dtype: int64
```

```
##### fruit1 + fruit2
apple      8.0
banana    18.0
cherry    15.0
mango      NaN
orange      NaN
peer       NaN
dtype: float64
```

연산하는 Series들의 인덱스 중 하나라도 NaN(결측치)가 존재하면 연산 결과도 무조건 NaN으로 나타난다.

▪ Dataframe 의 연산

실행 코드

```
##### Dataframe 연산
fruitData1 = {'Ohio' : [4,8,3,5], 'Texas' : [0,1,2,3]}
fruitFrame1 = DataFrame(fruitData1, columns=['Ohio', 'Texas'], index = ['apple', 'banana', 'cherry', 'peer'])
fruitData2 = {'Ohio' : [3,0,2,1,7], 'Colorado' : [5,4,3,6,0]}
fruitFrame2 = DataFrame(fruitData2, columns = ['Ohio', 'Colorado'], index = ['apple', 'orange', 'banana', 'cherry', 'mango'])
print(fruitFrame1)
print(fruitFrame2)
fruitFrame1 + fruitFrame2
```

출력 결과

print(fruitFrame1)

| | Ohio | Texas |
|--------|------|-------|
| apple | 4 | 0 |
| banana | 8 | 1 |
| cherry | 3 | 2 |
| peer | 5 | 3 |

print(fruitFrame2)

| | Ohio | Colorado |
|--------|------|----------|
| apple | 3 | 5 |
| orange | 0 | 4 |
| banana | 2 | 3 |
| cherry | 1 | 6 |
| mango | 7 | 0 |

fruitFrame1 + fruitFrame2

| | Colorado | Ohio | Texas |
|--------|----------|------|-------|
| apple | NaN | 7.0 | NaN |
| banana | NaN | 10.0 | NaN |
| cherry | NaN | 4.0 | NaN |
| mango | NaN | NaN | NaN |
| orange | NaN | NaN | NaN |
| peer | NaN | NaN | NaN |

Dataframe 은 column 이 존재하기 때문에 column 까지 같이 모두 함께 연산된다.
컬럼 또한 NaN이 하나라도 존재하는 row는 모두 NaN 처리가 된다.

SECTION 3.4

데이터의 정렬

▪ Series의 정렬

실행 코드

```
#### Series 정렬
```

```
fruit = Series([2500,3800,1200,6000],index=['apple','banana','peer','cherry'])  
fruit.sort_values(ascending=False)
```

출력 결과

```
cherry    6000  
banana    3800  
apple     2500  
peer      1200
```

- DataFrame의 정렬

실행 코드

```
##### DataFrame 정렬
fruitData = {'fruitName':['peer','banana','apple','cherry'],
             'fruitPrice':[2500,3800,6000,1200],
             'num':[10,5,3,8]}
fruitName=fruitData['fruitName']
fruitFrame = DataFrame(fruitData,index = fruitName,columns=['num','fruitPrice'])

print(fruitFrame)
fruitFrame.sort_index()
fruitFrame.sort_index(axis = 1)
```

출력 결과

```
##### print(fruitFrame)
```

| | num | fruitPrice |
|--------|-----|------------|
| peer | 10 | 2500 |
| banana | 5 | 3800 |
| apple | 3 | 6000 |
| cherry | 8 | 1200 |

```
##### fruitFrame.sort_index()
```

| | num | fruitPrice |
|--------|-----|------------|
| apple | 3 | 6000 |
| banana | 5 | 3800 |
| cherry | 8 | 1200 |
| peer | 10 | 2500 |

```
##### fruitFrame.sort_index(axis = 1)
```

| | fruitPrice | num |
|--------|------------|-----|
| peer | 2500 | 10 |
| banana | 3800 | 5 |
| apple | 6000 | 3 |
| cherry | 1200 | 8 |

- Dataframe 의 값 정렬

실행 코드

```
##### Dataframe 값 정렬  
fruitFrame.sort_values(by=['fruitPrice'])
```

출력 결과

| num | fruitPrice |
|--------|-------------|
| cherry | 8 1200 |
| peer | 10 2500 |
| banana | 5 3800 |
| apple | 3 6000 |

기준을 여러칼럼으로 하고 싶을 때는 by 안의 리스트에 여러 개의 column 이름을 지정하면 된다.

SECTION 4

기초분석

Pandas는 데이터를 보다 좀 더 편하게 다룰 수 있게 하는 데이터 구조 측면에서의 장점을 가진 패키지로 Pandas에서 제공하는 통계분석은 기본적인 기술 통계 및 데이터 요약 이다. 고급 통계 기법은 scikit-learn 이나 다른 통계 패키지를 이용하여 수행 할 수 있다.

SECTION 4.1

기술통계량

▪ 기술통계량

| 함 수 | 작용원리 |
|----------|-------------|
| count | NA 를 제외한 개수 |
| min, max | 최소, 최대값 |
| sum | 합 |
| cumprod | 누적합 |
| mean | 평균 |
| median | 중앙값 |
| quantile | 분위수 |
| Var | 표본분산 |
| std | 표본 정규분산 |
| Describe | 요약통계량 |

- German credit data 입력

실행 코드

```
#### Pandas를 이용한 데이터 입력
german=pd.read_csv("http://freakonometrics.free.fr/german_credit.csv")
list(german.columns.values)
```

출력 결과

```
['Creditability', 'Account Balance', 'Duration of Credit (month)', 'Payment Status of Previous Credit', 'Purpose', 'Credit Amount', 'Value Savings/Stocks', 'Length of current employment', 'Instalment per cent', 'Sex & Marital Status', 'Guarantors', 'Duration in Current address', 'Most valuable available asset', 'Age (years)', 'Concurrent Credits', 'Type of apartment', 'No of Credits at this Bank', 'Occupation', 'No of dependents', 'Telephone', 'Foreign Worker']
```

```
# pd.read_csv function을 통해 개인 컴퓨터의 데이터도 가져올 수 있지만, 웹의 데이터도 가져올 수 있다.
# cloumns.values function을 통해 어떤 컬럼들이 있는지 알아볼 수 있다.
```


- 기초통계분석

실행 코드

```
##### min  
german_sample.min()
```

출력 결과

| | |
|----------------------------|-----|
| Creditability | 0 |
| Duration of Credit (month) | 4 |
| Purpose | 0 |
| Credit Amount | 250 |

실행 코드

```
##### max  
german_sample.max()
```

출력 결과

| | |
|----------------------------|-------|
| Creditability | 1 |
| Duration of Credit (month) | 72 |
| Purpose | 10 |
| Credit Amount | 18424 |

실행 코드

```
#### mean
german_sample.mean()
```

출력 결과

| | |
|----------------------------|----------|
| Creditability | 0.700 |
| Duration of Credit (month) | 20.903 |
| Purpose | 2.828 |
| Credit Amount | 3271.248 |

실행 코드

```
#### 요약통계량 describe
german_sample.describe()
```

출력 결과

| | Creditability | Duration of Credit (month) | Purpose | Credit Amount |
|-------|---------------|----------------------------|-------------|---------------|
| count | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 |
| mean | 0.700000 | 20.903000 | 2.828000 | 3271.24800 |
| std | 0.458487 | 12.058814 | 2.744439 | 2822.75176 |
| min | 0.000000 | 4.000000 | 0.000000 | 250.00000 |
| 25% | 0.000000 | 12.000000 | 1.000000 | 1365.50000 |
| 50% | 1.000000 | 18.000000 | 2.000000 | 2319.50000 |
| 75% | 1.000000 | 24.000000 | 3.000000 | 3972.25000 |
| max | 1.000000 | 72.000000 | 10.000000 | 18424.00000 |

SECTION 4.2

상관관계와 공분산

- 상관관계와 공분산

실행 코드

```
##### corr cov
german=pd.read_csv("http://freakonometrics.free.fr/german_credit.csv")
german_sample=german[['Duration of Credit (month)','Credit Amount','Age (years)']]
german_sample.corr()
german_sample.cov()
```

출력 결과

```
##### corr
```

| | Duration of Credit (month) | Credit Amount | Age (years) |
|----------------------------|----------------------------|---------------|-------------|
| Duration of Credit (month) | 1.000000 | 0.624988 | -0.037550 |
| Credit Amount | 0.624988 | 1.000000 | 0.032273 |
| Age (years) | -0.037550 | 0.032273 | 1.000000 |


```
##### cov
```

| | Duration of Credit (month) | Credit Amount | Age (years) |
|----------------------------|----------------------------|---------------|-------------|
| Duration of Credit (month) | 145.415006 | 2.127401e+04 | -5.140567 |
| Credit Amount | 21274.007063 | 7.967927e+03 | 1034.202787 |
| Age (years) | -5.140567 | 1.034203e+03 | 128.883119 |

SECTION 5

핵심기능 Group by

Group by는 데이터를 구분 할 수 있는 column 의 값들을 이용하여 데이터를 여러 기준에 의해 구분 한 뒤 계산 및 순회 등 함수의 계산을 할 수 있는 방법이다. 이런 Group by를 통해 계산 하고 반복문을 활용하는 방법에 대해 배우겠다.

SECTION 5.1

Group by 를 이용한 계산 및 요약통계

▪ 한 개 그룹 요약통계

실행 코드

```
#### 한개 group  
german_grouped=german_sample['Credit Amount'].groupby(german_sample['Type of apartment'])  
german_grouped.mean()
```

출력 결과

```
Type of apartment  
1    3122.553073  
2    3067.257703  
3    4881.205607
```

▪ 두 개 그룹 요약통계

실행 코드

```
##### 두개 group
german_grouped2=german_sample['Credit Amount'].groupby([german_sample['Purpose'],german_sample['Type of apartment']])
german_grouped2.mean()
```

출력 결과

| Purpose | Type of apartment | |
|---------|-------------------|-------------|
| 0 | 1 | 2597.225000 |
| | 2 | 2811.024242 |
| | 3 | 5138.689655 |
| 1 | 1 | 5037.086957 |
| | 2 | 4915.222222 |
| | 3 | 6609.923077 |
| 2 | 1 | 2727.354167 |
| | 2 | 3107.450820 |
| | 3 | 4100.181818 |

group을 두 가지 이상으로 지정하고 싶을 때는 리스트를 통해 그룹을 두 개로 지정하면 된다.

SECTION 5.2

Group 간 반복하기

한 개 그룹반복

실행 코드

```
#### group 한 개 반복
german=pd.read_csv("http://freakonometrics.free.fr/german_credit.csv")
german_sample=german[['Type of apartment','Sex & Marital Status','Credit Amount']]
for type , group in german_sample.groupby('Type of apartment'):
    print(type)
    print(group.head(n=3))
```

출력 결과

```
1
Type of apartment Sex & Marital Status Credit Amount
0      1      2      1049
1      1      3      2799
2      1      2       841
```

```
3
Type of apartment Sex & Marital Status Credit Amount
29      3      3      4796
44      3      3     123969
```

```
2
Type of apartment Sex & Marital Status Credit Amount
4      2      3      2171
6      2      3      3398
7      2      3      1361
```

```
3      3      2032
```

head(n=x) function은 데이터의 처음 n개의 값을 나타내 준다.

- 두 개 그룹반복

실행 코드

```
##### group 두개 반복
for (type,sex) , group in german_sample.groupby(['Type of apartment','Sex & Marital Status']):
    print((type,sex))
    print(group.head(n=3))
```

출력 결과

```
(1, 1)
  Type of apartment  Sex & Marital Status  Credit Amount
369                1                1          3021
777                1                1          3384
797                1                1          2319(1, 2)
  Type of apartment  Sex & Marital Status  Credit Amount
0                1                2          1049
2                1                2           841
9                1                2          3758(1, 3)
  Type of apartment  Sex & Marital Status  Credit Amount
1                1                3          2799
3                1                3          2122
5                1                3          2241(1, 4)
  Type of apartment  Sex & Marital Status  Credit Amount
11                1                4          6187
14                1                4          1936
17                1                4          3213
```

```
(2, 1)
  Type of apartment  Sex & Marital Status  Credit Amount
50                2                1           640
97                2                1          4455
112               2                1          2366
```

⋮

SECTION 1

Numpy 소개

NumPy(Numerical Python)는 파이썬에서 과학적 계산을 위한 핵심 라이브러리이다. NumPy는 다차원 배열 객체와 배열과 함께 작동하는 도구들을 제공한다. 하지만 NumPy 자체로는 고수준의 데이터 분석 기능을 제공하지 않기 때문에 NumPy 배열과 배열 기반 컴퓨팅의 이해를 통해 Pandas와 같은 도구를 좀 더 효율적으로 사용하는 것이 필요하다.

SECTION 2.1

ndarray 생성

- array함수를 사용하여 배열 생성하기

실행 코드

```
import numpy as np

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

출력 결과

```
[1 2 3 4]
```

NumPy의 array 함수를 사용하여 배열을 만들 수 있다. (1,2,3,4로 이루어진 랭크 1의 배열)

- zeros, ones, empty 함수를 사용하여 배열 생성하기

실행 코드

```
## zeros, ones, empty 함수를 사용한 배열 생성
np.zeros((3,3))
np.ones((2,2))
np.empty((4,4))
```

출력 결과

```
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
array([[ 1.,  1.],
       [ 1.,  1.]])
array([[ 0.00000000e+000,  0.00000000e+000,  2.19578773e-314,
         2.19582881e-314],
       [ 2.19584693e-314,  2.19582995e-314,  7.12067159e-091,
         0.00000000e+000],
       [ 2.13222891e-314,  6.36598737e-311,  0.00000000e+000,
         0.00000000e+000],
       [ 2.19582011e-314,  2.75859453e-313,  0.00000000e+000,
         1.11253706e-308]])
```

zeros 함수를 사용하여 모든 값이 0인 배열을 만들 수 있다.
ones 함수를 사용하여 모든 값이 1인 배열을 만들 수 있다.
empty 함수를 사용하여 값이 초기화되지 않은 배열을 만들 수 있다.

- `arange` 함수를 사용하여 배열 생성하기

실행 코드

```
## arange 함수를 사용한 배열 생성  
np.arange(10)
```

출력 결과

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

`arange` 함수를 사용하여 범위 값을 순차적으로 갖는 배열을 만들 수 있다.

SECTION 2.2

ndarray 배열의 모양, 차수, 데이터 타입 확인

- Shape, ndim, dtype 함수를 사용하여 모양, 차수, 데이터 타입 확인하기

실행 코드

```
## shape, ndim, dtype 함수를 사용한 모양 차원 데이터타입 확인
arr = np.array([[1,2,3],[4,5,6]]) # (2,3)모양의 2차원 정수형 값들로 이루어진 배열 생성
print(arr) # 배열 확인
arr.shape
arr.ndim
arr.dtype
```

출력 결과

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

(2, 3)

2

dtype('int64')
```

SECTION 2.3

ndarray 배열의 타입 변환

- `astype` 함수를 사용하여 정수형 데이터 배열을 실수형 데이터 배열로 변환하기

실행 코드

```
## astype 함수를 사용한 데이터 타입 변환  
arr_int = np.array([1,2,3,4]) # 정수형 데이터 배열 생성  
arr_int.dtype  
arr_float = arr_int.astype(np.float64) # 실수형 데이터 배열로 변환  
arr_float.dtype
```

출력 결과

```
dtype('int64')  
dtype('float64')
```

- 문자형 데이터 배열을 정수형 데이터 배열로 변환하기

실행 코드

```
## 문자형 데이터 배열을 정수형 데이터 배열로 변환
arr_str = np.array(['1','2','3']) # 문자형 데이터 배열 생성
arr_str.dtype
arr_int = arr_str.astype(np.int64) # 정수형 데이터 배열로 변환
arr_int.dtype
```

출력 결과

```
dtype('<U1')
dtype('int64')
```

[표 1, 2] Built-in Python types, ndarray data types

| 'b' | 'i' | 'u' | 'f' | 'c' | 'm' |
|----------|-----------------|------------------|----------------|------------------------|-----------|
| Boolean | (signed)integer | unsigned integer | floating-point | complex-floating point | timedelta |
| 'M' | 'O' | 'S', 'a' | 'U' | 'V' | |
| Datetime | (python)objects | (byte-)string | Unicode | raw data(void) | |

| Type | Description |
|-------------------|--|
| bool | Boolean (True or False) stored as a bit |
| int8 | Byte (-128 to 127) |
| int16 | Integer (-32768 to 32767) |
| int32 | Integer (-2^{31} to $2^{31} - 1$) |
| int64 | Integer (-2^{63} to $2^{63} - 1$) |
| uint8 | Unsigned integer (0 to 255) |
| uint16 | Unsigned integer (0 to 65535) |
| uint32 | Unsigned integer (0 to $2^{32} - 1$) |
| uint64 | Unsigned integer (0 to $2^{64} - 1$) |
| float16 | Half precision float: sign bit, 5b expo, 10b mantissa |
| float32 | Single precision float: sign bit, 8b expo, 23b mantissa |
| float64 | Double precision float: sign bit, 11b expo, 52b mantissa |
| complex64 | Complex number, represented by two 32-bit floats (real & imag) |
| complex128 | Complex number, represented by two 64-bit floats (real & imag) |

SECTION 2.4

ndarray 배열의 연산

▪ 기본 연산자와 함수를 통해 배열 연산하기

실행 코드

```
## 배열 연산
arr1 = np.array([[1,2],[3,4]])
arr2 = np.array([[5,6],[7,8]])

arr1 + arr2
np.add(arr1, arr2)

arr1 * arr2
np.multiply(arr1, arr2)
```

출력 결과

```
[[ 6  8]
 [10 12]]
[[ 6  8]
 [10 12]]

[[ 5 12]
 [21 32]]
[[ 5 12]
 [21 32]]
```

Numpy 배열의 연산은 연산자(+, -, *, /)나 함수(add_덧셈, subtract_뺄셈, multiply_곱셈, divide_나눗셈)로 가능하다.

- dot 함수를 사용하여 행렬의 곱 계산하기

실행 코드

```
## dot 함수를 사용한 행렬의 곱 계산  
arr1.dot(arr2) # 배열 객체의 인스턴스 메소드로 dot 함수 사용  
np.dot(arr1, arr2) # Numpy 모듈 함수로 dot 함수 사용
```

출력 결과

```
array([[19, 22],  
       [43, 50]])  
  
array([[19, 22],  
       [43, 50]])
```

앞에서 살펴본 것과 같이 Numpy 배열에 기본 연산자(*)를 통한 곱은 행렬의 곱이 아닌 배열 요소의 곱이다.
따라서 행렬의 곱을 수행하기 위해서는 dot 함수를 사용해야 한다.

SECTION 2.5

ndarray 배열 슬라이싱 하기

▪ ndarray 배열 슬라이싱 하기

실행 코드

```
## 배열 슬라이싱  
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])  
arr_1 = arr[:2,1:3] # 첫행(0)부터 2-1행까지, 1열부터 3-1열까지 슬라이싱  
print(arr_1)
```

출력 결과

```
[[2 3]  
 [5 6]]
```

슬라이싱한 배열은 원본 배열의 뷰이다.(슬라이싱한 배열을 수정하면 원본 배열도 바뀐다)
뷰가 아닌 새로운 배열을 생성하려면 arr[:2,1:3].copy() 와 같은 함수를 사용해야 한다.

▪ ndarray 정수 배열 인덱싱 하기

실행 코드

```
## 정수 배열 인덱싱  
arr = np.array([[1,2,3], [4,5,6], [7,8,9]])  
  
arr[0,2] # 0행 2열 값에 접근  
arr[[0,1,2],[2,0,1]] # 0,1,2행의 2,0,1열 값에 접근
```

출력 결과

```
3  
array([3, 4, 8])
```

슬라이싱한 배열은 원본 배열의 뷰였던 것과 달리, 정수 배열 인덱싱은 새로운 배열을 생성한다.
따라서 정수 인덱싱으로 생성한 배열을 수정하여도 원본 배열이 수정되지 않는다.

▪ Boolean 배열 인덱싱하기

실행 코드

```
## Boolean 배열 인덱싱  
arr = np.array([[1,2,3],[4,5,6]])  
idx = arr > 3 # arr에서 3보다 큰 값을 찾아서 같은 Shape의 True or False 값을 가진 배열로 생성  
print(idx)  
  
print(arr[idx]) # 만들어 놓은 idx를 통해 조건에 맞는 배열 구성
```

출력 결과

```
array([[False, False, False],  
       [ True,  True,  True]], dtype=bool)  
  
[4 5 6]
```

idx의 값들은 arr의 각 값이 3보다 큰 것인지(True), 같거나 작은 것인지(False) 나타낸다.

NumPy는 sum, mean과 같은 기본적인 수학 메서드와 통계 메서드를 제공하고 있다. 이를 활용하여 Wine Quality 데이터의 기초 통계 분석을 실습한다.

SECTION 3.1

Wine Quality 데이터 설명

- Wine Quality 데이터에는 포르투갈의 레드 와인과 화이트 와인 두 개의 데이터 셋이 있다. 각각의 데이터 셋 안에는 물리화학적 실험에 대한 12개의 변수들이 있다. 변수에 대한 설명은 다음과 같다.

[표 1] Wine Quality 데이터 변수 설명

| No | 변수명 | 변수 설명 |
|----|----------------------|--------------------|
| 1 | fixed acidity | 결합 산도 |
| 2 | volatile acidity | 휘발성산 |
| 3 | citric acid | 시트르산 |
| 4 | residual sugar | 발효 후 와인 속에 남아있는 당분 |
| 5 | chlorides | 염화물 |
| 6 | free sulfur dioxide | 유리 이산화황 |
| 7 | total sulfur dioxide | 총 이산화황 |
| 8 | density | 농도 |
| 9 | pH | 산도 |
| 10 | sulphates | 황산염 |
| 11 | alcohol | 알코올 |
| 12 | quality | 품질 |

SECTION 3.2

Wine Quality 데이터 불러오기

- 실습은 Wine Quality 데이터(red, white) 중 winequality-red.csv 파일만 사용한다.

실행 코드

```
import numpy as np
```

```
## winequality-red.csv 파일 불러오기
```

```
redwine = np.loadtxt(fname='c:/Users/seoseon/Desktop/winequality-red.csv', delimiter=';', skiprows=1)  
print(redwine)
```

출력 결과

```
[[ 7.4  0.7  0.  ..., 0.56  9.4  5. ]  
 [ 7.8  0.88 0.  ..., 0.68  9.8  5. ]  
 [ 7.8  0.76 0.04 ..., 0.65  9.8  5. ]  
 ...,  
 [ 6.3  0.51 0.13 ..., 0.75 11.  6. ]  
 [ 5.9  0.645 0.12 ..., 0.71 10.2  5. ]  
 [ 6.  0.31 0.47 ..., 0.66 11.  6. ]]
```

numpy라이브러리 안에 loadtxt함수를 실행하여 텍스트 파일을 불러올 수 있다.

fname에는 winequality-red.csv파일이 위치한 디렉토리를 입력한다.

csv파일의 구분자가 ; 이기 때문에 delimiter=';'을 해주고, 변수인 첫 행은 불러 오지 않기 때문에 skiprows=1를 해준다.

SECTION 3.3

Wine Quality 데이터를 활용한 기초 통계 분석

- NumPy에는 기본적인 배열 통계 메서드가 있다. `sum`, `mean`과 같은 기본적인 통계 메서드를 통해 3.2에서 불러온 `redwine` 데이터의 기초적인 통계 분석을 해본다. 실습할 통계 메서드는 다음과 같다.

[표 2] 배열 통계 메서드

| No | 메서드 | 메서드 설명 |
|----|-------------------|-----------------------------------|
| 1 | <code>sum</code> | 배열 전체 혹은 특정 축에 대한 모든 원소의 합을 계산 |
| 2 | <code>mean</code> | 산술 평균을 계산 |
| 3 | <code>std</code> | 표준 편차를 계산 |
| 4 | <code>var</code> | 분산을 계산, <code>std</code> 의 제곱과 같다 |
| 5 | <code>min</code> | 최소값 |
| 6 | <code>max</code> | 최대값 |

실행 코드

```
## redwine 데이터셋의 합(sum)
print(redwine.sum())
## redwine 데이터셋의 평균(mean)
print(redwine.mean())
print(redwine.mean(axis=0))
print(redwine[:,0].mean())
```

출력 결과

```
152084.78194
7.92603616531
[ 8.31963727  0.52782051  0.27097561  2.5388055  0.08746654
 15.87492183 46.46779237  0.99674668  3.3111132  0.65814884
 10.42298311  5.63602251]
8.3196372733
```

```
# redwine.sum() : redwine 데이터 셋에 있는 모든 데이터의 합
# redwine.mean() : redwine 데이터 셋에 있는 모든 데이터의 평균
# redwine.mean(axis=0) : redwine 데이터 셋의 열별(axis=0은 열을 뜻한다) 평균, 이를 통해 각 변수별 평균을 알 수 있다. Fixed acidity의 평균은 8.31963727
 이며, volatile acidity의 평균은 0.52782051이다.
# redwine[:,0].mean() : redwine 데이터 셋의 첫번째 열(0번째 열)의 모든 값들의 평균을 뜻한다. 즉, fixed acidity의 평균만 알 수 있다.
```

실행 코드

```
## redwine 데이터셋의 최대값(max)
print(redwine.max(axis=0))
## redwine 데이터셋의 최소값(min)
print(redwine.min(axis=0))
```

출력 결과

```
[ 15.9      1.58      1.      15.5      0.611    72.      289.
   1.00369   4.01      2.      14.9      8.      ]
[ 4.6      0.12      0.      0.9      0.012    1.      6.      0.99007
  2.74     0.33     8.4     3.      ]
```

```
# redwine.max(axis=0) : redwine 데이터 셋의 열별(변수별) 최대값
# redwine.min(axis=0) : redwine 데이터 셋의 열별(변수별) 최소값
```

SECTION 1

scikit-learn & Machine learning 개념 소개

머신러닝은 샘플 데이터를 통해 컴퓨터를 지속적으로 학습시켜 적절한 답을 찾아내는 기술이다. Scikit-learn은 머신러닝을 위한 파이썬 패키지이며 Sample dataset, Data preprocessing 기능, Supervised learning, Unsupervised learning, 모델 평가 기능 등을 담고 있다. Scikit-learn의 특징은 다양한 머신러닝 알고리즘을 하나의 패키지 안에서 모두 제공해준다는 점이다.

[표 1] Scikit-learn 패키지에서 제공하는 머신러닝 알고리즘(<http://scikit-learn.org> 참고)

| Supervised learning | Unsupervised learning |
|--|---|
| Generalized Linear Models | Gaussian mixture models |
| Linear and Quadratic Discriminant Analysis | Manifold learning |
| Kernel ridge regression | Clustering |
| Support Vector Machines | Biclustering |
| Stochastic Gradient Descent | Decomposing signals in components (matrix factorization problems) |
| Nearest Neighbors | Covariance estimation |
| Gaussian Processes | Novelty and Outlier Detection |
| Cross decomposition | Density Estimation |
| Naive Bayes | Neural network models (unsupervised) |
| Decision Trees | |
| Ensemble methods | |
| Multiclass and multilabel algorithms | |
| Feature selection | |
| Semi-Supervised | |
| Isotonic regression | |
| Probability calibration | |
| Neural network models (supervised) | |

SECTION 2

Scikit-learn 샘플 데이터 사용법과 전처리

Scikit-learn의 서브패키지 `sklearn.datasets`는 실습을 위한 샘플용 dataset을 제공하고 있다. 샘플용 Dataset은 기본적으로 Scikit-learn 패키지 안에 내장되어있는 형태(load명령으로 import), 인터넷에서 다운로드하여 사용하는 형태(fetch명령으로 import), 그리고 새로운 dataset을 생성시켜 사용하는 형태(make명령으로 생성)로 접근할 수 있다. Scikit-learn 패키지에는 데이터 전처리를 위한 preprocessing, feature_extraction 서브 패키지가 있다. 패키지를 활용하여 스케일링(Scaling), 인코딩(Encoding), 결측값 처리(Imputation)가 가능하다.

SECTION 2.1

샘플용 Data 소개

[표2] sklearn 샘플 Dataset

| load 계열 | fetch 계열 | make 계열 |
|---|--|---|
| <code>load_boston()</code> : 보스턴 집값 데이터 | <code>fetch_covtype()</code> : 토지 조사 데이터 | <code>make_regression()</code> : regression용 데이터 생성 |
| <code>load_diabetes()</code> : 당뇨병 관련 데이터 | <code>fetch_20newsgroups()</code> : 뉴스 텍스트 데이터 | <code>make_classification()</code> : classification용 데이터 생성 |
| <code>load_iris()</code> : iris 데이터 | <code>fetch_rcv1()</code> : 로이터 뉴스 말뭉치 | <code>make_blobs()</code> : clustering용 데이터 생성 |
| | <code>fetch_california_housing</code> : 주택 데이터 | |

sklearn 샘플 Dataset 객체는 다음과 같은 속성으로 구성되어 있다.

data: 독립 변수의 ndarray 배열 형태

target: 종속 변수의 ndarray 배열 형태

feature_names: 독립 변수 이름의 리스트 형태

target_names: 종속 변수 이름의 리스트 형태

DESCR: 데이터에 대한 설명

SECTION 2.2

Scikit_learn 샘플 Dataset 예시

▪ 샘플 Dataset 불러오기

실행 코드

```
from sklearn.datasets import fetch_california_housing
import pandas as pd
```

```
ch = fetch_california_housing()
```

```
x = pd.DataFrame(ch.data, columns=ch.feature_names)
```

```
y = pd.DataFrame(ch.target, columns=["AveInc"])
```

```
data = pd.concat([x, y], axis = 1)
```

```
data.tail() ## 데이터 확인
```

```
data.describe() ## 통계 요약 정보 확인
```

출력 결과

```
>>> data.tail() ## 데이터 확인
```

| | MedInc | HouseAge | AveRooms | AveBedrms | Population | AveOccup | Latitude \ |
|-------|--------|----------|----------|-----------|------------|----------|------------|
| 20635 | 1.5603 | 25.0 | 5.045455 | 1.133333 | 845.0 | 2.560606 | 39.48 |
| 20636 | 2.5568 | 18.0 | 6.114035 | 1.315789 | 356.0 | 3.122807 | 39.49 |
| 20637 | 1.7000 | 17.0 | 5.205543 | 1.120092 | 1007.0 | 2.325635 | 39.43 |
| 20638 | 1.8672 | 18.0 | 5.329513 | 1.171920 | 741.0 | 2.123209 | 39.43 |
| 20639 | 2.3886 | 16.0 | 5.254717 | 1.162264 | 1387.0 | 2.616981 | 39.37 |

SECTION 2.3.1 스케일링(Scaling)

- Scikit-learn에서 제공하는 `scale(x)`, `robust_scale(x)`, `minmax_scale(x)`, `maxabs_scale(x)` 함수 실습

실행 코드

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import scale, robust_scale, minmax_scale, maxabs_scale

x = (np.arange(5, dtype=np.float)).reshape(-1, 1)
# scaling
test = pd.DataFrame(np.hstack([x, scale(x), robust_scale(x), minmax_scale(x), maxabs_scale(x)]),
                    columns=["x", "scale(x)", "robust_scale(x)", "minmax_scale(x)", "maxabs_scale(x)"])
print(test)
```

출력 결과

| x | scale(x) | robust_scale(x) | minmax_scale(x) | maxabs_scale(x) |
|---|----------|-----------------|-----------------|-----------------|
| 0 | 0.0 | -1.414214 | -1.0 | 0.00 |
| 1 | 1.0 | -0.707107 | -0.5 | 0.25 |
| 2 | 2.0 | 0.000000 | 0.0 | 0.50 |
| 3 | 3.0 | 0.707107 | 0.5 | 0.75 |
| 4 | 4.0 | 1.414214 | 1.0 | 1.00 |

```
# scale(x) : Standard normal Gaussian 기본 스케일링
# robust_scale(x) : median과 interquartile range를 사용하여 스케일링
# minmax_scale(x) : 최대값과 최소값을 사용하여 스케일링
# maxabs_scale(x) : 최대절대값을 사용하여 스케일링
```

SECTION 2.3.2

인코딩(Encoding)

- Label Encoding 실습, Label Encoding은 실제 값에 상관없이 0~K-1까지의 정수로 변환

실행 코드

```
from sklearn.preprocessing import LabelEncoder  
le = LabelEncoder()
```

```
le.fit(["seoul", "busan", "busan", "daejeon"])  
print(le.classes_)
```

출력 결과

```
['busan' 'daejeon' 'seoul']
```

classes_ : 변환된 규칙을 확인

실행 코드

```
print(le.transform(["seoul", "busan", "busan", "daejeon"]))  
print(list(le.inverse_transform([2, 0, 1])))
```

출력 결과

```
[2 0 0 1]  
['seoul', 'busan', 'daejeon']
```

transform_ : 0~K-1 까지의 정수로 변환

inverse_transform : 역변환

SECTION 2.3.3

결측값 처리(Imputation)

- Imputer를 통해 누락된 정보(결측값)를 채운다.

실행 코드

```
import numpy as np
from sklearn.preprocessing import Imputer

# imputer - mean
imp_mean = Imputer(missing_values='NaN', strategy='mean', axis=0)
print(imp_mean.fit_transform([[1, 5], [2, np.nan], [3, 3]]))
```

출력 결과

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

```
# missing_values : 결측값
# strategy='mean' : mean(평균)을 사용하여 결측값을 채움
```

실행 코드

```
# imputer - median
imp_median = Imputer(missing_values='NaN', strategy='median', axis=0)
print(imp_median.fit_transform([[1, 5], [2, np.nan], [3, 3]]))
```

출력 결과

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

strategy='median' : median(중앙값)을 사용하여 결측값을 채움

실행 코드

```
# imputer - most_frequent
imp_freq = Imputer(missing_values='NaN', strategy='most_frequent', axis=0)
print(imp_freq.fit_transform([[1, 5], [2, np.nan], [3, 3]]))
```

출력 결과

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

strategy='most_frequent' : most_frequent(최빈값)을 사용하여 결측값을 채움

SECTION 3

Scikit-learn을 활용한 Wine Quality Data set Neural network 실습

Scikit-learn에는 Neural Network를 적용할 수 있는 MLPClassifier 함수가 있다. WineQuality 데이터 셋에 MLPClassifier 함수를 적용하여, Wine의 특징을 통해 Wine의 타입(White Wine 또는 Red Wine)을 맞추는 Neural Network를 만들어 본다.

SECTION 3.1

Winequality 데이터 설명

- 3.3 Wine Quality 데이터를 활용한 NumPy 실습에서 사용한 데이터와 동일하다. 단, quality변수를 제거하고 type변수를 추가하였다. type변수에서 0은 White-Wine을 뜻하고 1은 Red-wine을 의미한다.

[그림 1] Winequality_train.csv 데이터 형태

| | A | B | C | D | E | F | G | H | I | J | K | L |
|----|---------------|------------------|-------------|----------------|-----------|---------------------|----------------------|---------|------|-----------|---------|------|
| 1 | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | type |
| 2 | 10 | 0.35 | 0.45 | 2.5 | 0.092 | 20 | 88 | 0.99918 | 3.15 | 0.43 | 9.4 | 1 |
| 3 | 8.3 | 0.6 | 0.25 | 2.2 | 0.118 | 9 | 38 | 0.99616 | 3.15 | 0.53 | 9.8 | 1 |
| 4 | 7.1 | 0.37 | 0.67 | 10.5 | 0.045 | 49 | 155 | 0.9975 | 3.16 | 0.44 | 8.7 | 0 |
| 5 | 8.6 | 0.47 | 0.27 | 2.3 | 0.055 | 14 | 28 | 0.99516 | 3.18 | 0.8 | 11.2 | 1 |
| 6 | 5.9 | 0.3 | 0.23 | 4.2 | 0.038 | 42 | 119 | 0.9924 | 3.15 | 0.5 | 11 | 0 |
| 7 | 6.6 | 0.23 | 0.32 | 0.9 | 0.041 | 25 | 79 | 0.9926 | 3.39 | 0.54 | 10.2 | 0 |
| 8 | 10.6 | 0.42 | 0.48 | 2.7 | 0.065 | 5 | 18 | 0.9972 | 3.21 | 0.87 | 11.3 | 1 |
| 9 | 6 | 0.36 | 0.39 | 3.2 | 0.027 | 20 | 125 | 0.991 | 3.38 | 0.39 | 11.3 | 0 |
| 10 | 7.7 | 0.62 | 0.04 | 3.8 | 0.084 | 25 | 45 | 0.9978 | 3.34 | 0.53 | 9.5 | 1 |
| 11 | 7.8 | 0.42 | 0.26 | 9.2 | 0.058 | 34 | 199 | 0.9972 | 3.14 | 0.55 | 9.3 | 0 |
| 12 | 9.7 | 0.55 | 0.17 | 2.9 | 0.087 | 20 | 53 | 1.0004 | 3.14 | 0.61 | 9.4 | 1 |
| 13 | 6.3 | 0.33 | 0.27 | 1.2 | 0.046 | 34 | 175 | 0.9934 | 3.37 | 0.54 | 9.4 | 0 |

SECTION 3.2

Winequality 데이터 불러오기

- Winequality_Train.csv, Winequality_Test.csv 파일을 pandas 패키지를 활용하여 불러온 후, 트레이닝셋과 테스트셋을 나눈다.

실행 코드

```
import pandas as pd

train = pd.read_csv('디렉토리/Winequality_Train.csv')
test = pd.read_csv('디렉토리/Winequality_Test.csv')

train_x = train.iloc[:, :-1] ## 트레이닝셋 input data
train_y = train.iloc[:, -1] ## 트레이닝셋 target data

test_x = test.iloc[:, :-1] ## 테스트셋 input data
test_y = test.iloc[:, -1]
```

출력 결과

```
>>> print(train_x)
```

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | ₩ |
|---|---------------|------------------|-------------|----------------|-----------|---|
| 0 | 10.0 | 0.350 | 0.45 | 2.50 | 0.092 | |
| 1 | 8.3 | 0.600 | 0.25 | 2.20 | 0.118 | |
| 2 | 7.1 | 0.370 | 0.67 | 10.50 | 0.045 | |
| 3 | 8.6 | 0.470 | 0.27 | 2.30 | 0.055 | |
| 4 | 5.9 | 0.300 | 0.23 | 4.20 | 0.038 | |
| 5 | 6.6 | 0.230 | 0.32 | 0.90 | 0.041 | |
| 6 | 10.6 | 0.420 | 0.48 | 2.70 | 0.065 | |

```
# pd.read_csv : 판다스 패키지의 read_csv 함수를 사용하여 csv 파일을 불러온다.
# train_x = train.iloc[:, :-1] : 마지막 컬럼(type, 타겟변수)을 제외한 데이터를 train_x에 할당한다.
# train_y = train.iloc[:, -1] : 마지막 컬럼(type, 타겟변수)만 train_y에 할당한다.
# 테스트 데이터셋에도 동일하게 적용한다.
```

SECTION 3.3

MLPClassifier 함수를 사용한 Neural Network 모델링

- sklearn패키지의 MLPClassifier함수를 사용하여 Neural Network 모델을 만든다.

실행 코드

```
from sklearn.neural_network import MLPClassifier
```

```
mlp = MLPClassifier(hidden_layer_sizes=(50,30))
mlp.fit(train_x, train_y)
print("Training score: %s" % mlp.score(train_x, train_y))
```

출력 결과

```
>>> mlp = MLPClassifier(hidden_layer_sizes=(50,30))>>> mlp.fit(train_x, train_y)
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(50, 30), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)>>> mlp.fit(train_x, train_y)
print("Training score: %s" % mlp.score(train_x, train_y))
```

Training score: 0.932857142857

mlp = MLPClassifier(hidden_layer_sizes=(50,30)) : 은닉층이 2개이며, 노드 수가 각각 50, 30개인 뉴럴네트워크 모델을 생성한다.
 # 사용한 코드는 hidden_layer_sizes 파라미터만을 지정하였으며, 그 이외의 파라미터들은 default값이 적용되었다. MLPClassifier 함수의 파라미터들은 http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html 에 자세히 설명되어 있다.
 # mlp.fit(train_x, train_y) : 생성한 모델에 트레이닝 데이터셋을 적용한다.
 # mlp.score(train_x, train_y) : 모델에 트레이닝 셋을 적용하였을 때 train_y(타겟변수)와 비교하여 평균 정확도를 반환한다.

SECTION 3.4

Neural Network 모델 평가

- 다음은 일반적인 분류 모형의 평가 방법이다. Confusion Matrix를 출력해보고 이를 통해 Accuracy를 계산해본다.

[표1] 분류 모형의 평가

| | | 예측값 | |
|-----|----------|----------|----------|
| | | Positive | Negative |
| 실제값 | Positive | a | b |
| | Negative | c | d |

| 평가 방법 | 계산 |
|---------------------------------|---------------------------------------|
| 정분류율(Accuracy) | $(a+d) / (a+b+c+d)$ |
| 오분류율(Error rate) | $(b+c) / (a+b+c+d)$ |
| 민감도(Sensitivity) 또는 재현율(recall) | $a / (a+b)$ |
| 특이도(Specificity) | $d / (c+d)$ |
| 정밀도(Precision) | $a / (a+c)$ |
| F 값 | $2 * \{ (정밀도 * 재현율) / (정밀도 + 재현율) \}$ |

SECTION 3.2

Winequality 데이터 불러오기

- Winequality_Train.csv, Winequality_Test.csv 파일을 pandas 패키지를 활용하여 불러온 후, 트레이닝셋과 테스트셋을 나눈다.

실행 코드

```
pred = mlp.predict(test_x)

confusion_matrix = pd.crosstab(test_y, pred, rownames=['True'],
                               colnames=['Predicted'], margins=True) ## crosstab
print(confusion_matrix)
accuracy = (confusion_matrix[0][0]+confusion_matrix[1][1]) / 600
print(accuracy)
```

출력 결과

```
>>> pred = mlp.predict(test_x)
confusion_matrix = pd.crosstab(test_y, pred, rownames=['True'],
                               colnames=['Predicted'], margins=True) ## crosstab
print(confusion_matrix)

Predicted   0   1  All
True
0         296  11  307
1          21 272  293
All        317 283 600>>> accuracy = (confusion_matrix[0][0]+confusion_matrix[1][1]) / 600
print(accuracy)

0.9466666666666667
```

mlp.predict(test_x) : 생성된 뉴럴네트워크 모델에 타겟변수를 제외한 테스트 데이터(test_x)를 적용한다.
 # pd.crosstab(test_y, pred, rownames=['True'], colnames=['Predicted'], margins=True) : pandas의 crosstab함수를 사용하여 Confusion Matrix를 생성한다.
 # 정분류율(Accuracy) = $(a+d) / (a+b+c+d)$ 를 활용하여 accuracy를 계산한다.

Scikit-learn.ensemble 에는 Randomforest 기법을 적용할 수 있는 RandomForestClassifier 함수가 있다. breast-cancer-wisconsin 데이터 셋에 RandomForestClassifier 함수를 적용하여, 여러 변수들의 특징을 클래스(양성, 음성)를 맞추는 Randomforest 모델을 만들어 본다.

SECTION 4.1

breast-cancer-Wisconsin 데이터 설명

- Breast-cancer-wisconsin 데이터는 University of Wisconsin Hospitals에서 수집된 환자 데이터이다. 데이터셋은 환자들의 ID number와 가지고 있는 특징에 대한 변수 9개, 음성과 양성을 나누는 Class 변수(Target 변수)로 구성되어 있다. 변수 목록과 범위는 다음과 같다. 자세한 설명은 UCI Machine Learning Repository 에서 제공하고 있다.

<http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>

[표1] breast-cancer-Wisconsin 데이터 형태

| Attribute | Domain |
|-----------------------------|---------------------------------|
| Sample code number | id number |
| Clump Thickness | 1 - 10 |
| Uniformity of Cell Size | 1 - 10 |
| Uniformity of Cell Shape | 1 - 10 |
| Marginal Adhesion | 1 - 10 |
| Single Epithelial Cell Size | 1 - 10 |
| Bare Nuclei | 1 - 10 |
| Bland Chromatin | 1 - 10 |
| Normal Nucleoli | 1 - 10 |
| Mitoses | 1 - 10 |
| Class | (2 for benign, 4 for malignant) |

SECTION 4.2

breast-cancer-Wisconsin 데이터 불러오기

- Breast-cancer-Wisconsin.data 파일을 pandas 패키지를 활용하여 불러온다.

실행 코드

```
import pandas as pd
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier

df = pd.read_csv('breast-cancer-wisconsin.data', header=None,
                 names=['ID', 'Clump Thickness', 'Uniformity of Cell Size',
                        'Uniformity of Cell Shape', 'Marginal Adhesion',
                        'Single Epithelial Cell Size', 'Bare Nuclei',
                        'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class'])
pd.set_option('display.max_rows', 10)
print(df)
```

출력 결과

| ID | Clump Thickness | Uniformity of Cell Size | W |
|----|-----------------|-------------------------|---|
| 0 | 1000025 | 5 | 1 |
| 1 | 1002945 | 5 | 4 |
| 2 | 1015425 | 3 | 1 |
| 3 | 1016277 | 6 | 8 |
| 4 | 1017023 | 4 | 1 |

..

[699 rows x 11 columns]

```
# pd.read_csv을 통해 breast-cancer-wisconsin.data 파일을 불러오며, 컬럼명을 직접 지정해준다.  
# pd.set_option('display.max_rows', 10)을 통해 DataFrame을 출력했을 때 보여지는 행을 10개로 제한한다.  
# print(df)로 대략적인 데이터 형태를 확인한다.
```

SECTION 4.3

데이터 전처리

- 원본 데이터에서 분석에 사용하지 않을 부분을 제거하거나 필요한 형태로 변환한다.

실행 코드

데이터 전처리

```
df = df.drop('ID', axis = 1)
df['Class'] = df['Class'].replace(2, 0)
df['Class'] = df['Class'].replace(4, 1)
df.replace('?', np.nan, inplace=True)
df.dropna(inplace=True)
```

출력 결과

| | Clump Thickness | Uniformity of Cell Size | Uniformity of Cell Shape | W |
|----|-----------------|-------------------------|--------------------------|---|
| 0 | 5 | 1 | 1 | |
| 1 | 5 | 4 | 4 | |
| 2 | 3 | 1 | 1 | |
| 3 | 6 | 8 | 8 | |
| 4 | 4 | 1 | 1 | |
| .. | ... | ... | ... | |

[683 rows x 10 columns]

pd.drop('ID', axis = 1) : 판다스에 내장되어있는 drop함수를 사용하여 분석에 사용하지 않는 ID열을 제거한다.
 # df['Class'].replace(2,0) : 판다스에 내장되어있는 replace함수를 사용하여 Class열의 숫자 2를 숫자 0으로 변환한다.
 # df['Class'].replace(4,1) : 판다스에 내장되어있는 replace함수를 사용하여 Class열의 숫자 4를 숫자 1로 변환한다.
 # df.replace('?', np.nan, inplace = True) : 판다스에 내장되어있는 replace함수를 사용하여 데이터셋 내의 ? 값을 넘파이의 nan으로 변환한다. inplace 옵션은 변수의 재할당을 자동으로 수행한다. (default = False)
 # df.dropna(inplace = True) : 판다스에 내장되어있는 dropna함수를 사용하여 결측치를 제거한다.

SECTION 4.4

트레이닝셋과 테스트셋 분리

- `train-testsplit` 함수를 사용하여 트레이닝셋과 테스트셋을 분리한다.

실행 코드

```
## split training & test set
```

```
y = df['Class']
```

```
x = df.drop('Class', axis=1)
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

`train_test_split(x, y, test_size = 0.3, random_state = 0)` : Class(타겟변수)만이 할당되어있는 y와 Class(타겟변수)를 제외한 데이터들이 할당되어있는 x를 7:3의 비율로 분리하여 x_train, x_test, y_train, y_test 변수에 할당한다. random_state 옵션을 통해 seed를 지정할 수 있다. 0으로 지정한다면 항상 같은 샘플을 분리해낼 수 있다.

SECTION 4.5

모델링

- RandomForestClassifier 함수를 사용하여 Randomforest 모델을 만든다.

실행 코드

모델링

```
forest = RandomForestClassifier(n_estimators = 10000, random_state=0)
forest.fit(x_train, y_train)
```

출력 결과

>>> ## 모델링

```
forest = RandomForestClassifier(n_estimators = 10000, random_state=0)

forest.fit(x_train, y_train)
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=None, max_features='auto', max_leaf_nodes=None,
                        min_impurity_split=1e-07, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        n_estimators=10000, n_jobs=1, oob_score=False, random_state=0,
                        verbose=0, warm_start=False)
```

forest = RandomForestClassifier(n_estimators = 10000, random_state = 0) : forest에서 tree의 갯수가 10000개인 Randomforest 모델을 생성한다. random_state는 랜덤넘버 생성에서 seed로 사용된다. 이외의 파라미터들은 default값이 적용되었고, RandomForestClassifier 함수의 파라미터들은 <http://scikit-learn.org/s-table/modules/generated/sklearn.ensemble.RandomForestClassifier.html>에 자세히 설명되어 있다.

SECTION 4.6

Randomforest 모델 평가

- 모델 사용 결과를 Confusion Matrix로 만들어 출력해보고 Accuracy를 계산한다.

실행 코드

```
## 모델 평가
pred = forest.predict(x_test)

confusion_matrix = pd.crosstab(y_test, pred, rownames=['True'],
                               colnames=['Predicted'], margins=True) ## crosstab
print(confusion_matrix)

total = confusion_matrix[0][0] + confusion_matrix[0][1] + confusion_matrix[1][0] + confusion_matrix[1][1]
accuracy = (confusion_matrix[0][0]+confusion_matrix[1][1]) / total
print(accuracy)
```

출력 결과

```
Predicted   0   1  All
True
0          126   4  130
1           6  69   75
All         132  73  205
0.951219512195
```

forest.predict(x_test): 생성된 Randomforest 모델에 타겟변수를 제외한 테스트 데이터(x_test)를 적용한다.
 # pd.crosstab(y_test, pred, rownames=['True'], colnames=['Predicted'], margins=True): pandas의 crosstab함수를 사용하여 Confusion Matrix를 생성한다.
 # 생성된 Confusion Matrix에서 accuracy를 계산한다.

Scikit-learn.tree에는 Decision Tree 기법을 적용할 수 있는 DecisionTreeClassifier 함수가 있다. Adult 데이터 셋에 DecisionTreeClassifier 함수를 적용하여, Income-level(>50K, <=50K)을 분리하는 Decision Tree 모델을 만들어 본다.

SECTION 1.1 Adult 데이터 설명

- Adult 데이터는 인구 조사 데이터를 통하여 연평균 수입이 5만 달러 이상인 사람을 분류하는데 사용한다. 데이터셋은 총 15개의 변수를 가지고 있으며, 6개의 연속형 변수와 9개의 범주형 변수를 가진다. 목표 변수는 income-level이며, 변수 목록과 범위는 다음과 같다.

| Attribute | Domain |
|----------------|---|
| age | continuous |
| workclass | Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked |
| fnlwgt | continuous |
| education | Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool |
| education-num | continuous |
| marital-status | Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse |
| occupation | Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces |
| relationship | Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried |
| Race | White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black |
| Sex | Female, Male |
| capital-gain | continuous |
| capital-loss | continuous |
| hours-per-week | continuous |
| native-country | United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad&Tobago, Peru, Hong, Holand-Netherlands. |
| Income-level | >50K, <=50K |

SECTION 1.2

adult.data.txt, adult.test.txt 데이터 불러오기

- adult.data.txt, adult.test.txt 파일을 pandas 패키지를 활용하여 불러온다.

실행 코드

```
import numpy as np
import pandas as pd
from sklearn import linear_model, preprocessing, tree

## 열이름 정의
col = ["age", "workclass", "fnlwgt", "education", "education-num",
       "marital-status", "occupation", "relationship", "race", "sex",
       "capital-gain", "capital-loss", "hours-per-week", "native-country",
       "income-level"]

## 정의한 열 이름을 사용하여 데이터 읽기
df_train = pd.read_csv('adult.data.txt', names = col)
df_test = pd.read_csv('adult.test.txt', names = col, skiprows =1)

## 읽어온 데이터 길이와 형태 확인
print(len(df_train))
print(len(df_test))
print(df_train.head(5))
print(df_test.head(5))
```

출력 결과

```
## 읽어온 데이터 길이와 형태 확인
```

```
>>> print(len(df_train))
```

```
32561
```

```
>>> print(len(df_test))
```

```
16281
```

```
>>> print(df_train.head(5))
```

| | age | workclass | fnlwgt | education | education-num \ |
|----|-----|------------------|--------|-----------|-----------------|
| 0 | 39 | State-gov | 77516 | Bachelors | 13 |
| 1 | 50 | Self-emp-not-inc | 83311 | Bachelors | 13 |
| 2 | 38 | Private | 215646 | HS-grad | 9 |
| 3 | 53 | Private | 234721 | 11th | 7 |
| 4 | 28 | Private | 338409 | Bachelors | 13 |
| .. | .. | .. | .. | .. | .. |

```
# 열이름을 정의한 후 pd.read_csv을 통해 adult.data.txt, adult.test.txt 파일을 불러온다.
```

```
# print(len(df_train)) : df_train의 길이를 출력한다.
```

```
# print(df_train.head(5)) : df_train의 데이터 형태를 보기 위해 상위 5개의 행만을 출력한다.
```

SECTION 1.3

데이터 전처리

- adult.data.txt 데이터와 adult.test.txt 데이터의 income-level 값들의 차이 없애기(ex "<=50K"와 "<=50K."의 차이)
- 카테고리형 변수 데이터를 숫자형으로 변경하기

실행 코드

```
## 데이터 전처리
### 타겟변수 일치시키기
df_test["income-level"] = df_test["income-level"].str.replace('.', '')
print(df_test.head(5))

### 카테고리형 변수 데이터를 숫자형으로 변경하기(0,1,2,3)
categorical_col = ["workclass", "education", "marital-status", "occupation",
                  "relationship", "race", "sex", "native-country", "income-level"]

LabelEncoder = preprocessing.LabelEncoder()

for i in categorical_col:
    if df_train[i].dtypes == 'object':
        category = df_train[i].append(df_test[i])
        LabelEncoder.fit(category.values)
        df_train[i] = LabelEncoder.transform(df_train[i])
        df_test[i] = LabelEncoder.transform(df_test[i])

print(df_train.head(5))
```

출력 결과

```
print(df_test.head(5))
```

| | age | workclass | fnlwgt | education | education-num | marital-status \ |
|---|-----|-----------|--------|--------------|---------------|--------------------|
| 0 | 25 | Private | 226802 | 11th | 7 | Never-married |
| 1 | 38 | Private | 89814 | HS-grad | 9 | Married-civ-spouse |
| 2 | 28 | Local-gov | 336951 | Assoc-acdm | 12 | Married-civ-spouse |
| 3 | 44 | Private | 160323 | Some-college | 10 | Married-civ-spouse |
| 4 | 18 | ? | 103497 | Some-college | 10 | Never-married |

```
>>> print(df_train.head(5))
```

| | age | workclass | fnlwgt | education | education-num | marital-status \ |
|----|-----|-----------|--------|-----------|---------------|------------------|
| 0 | 39 | 7 | 77516 | 9 | 13 | 4 |
| 1 | 50 | 6 | 83311 | 9 | 13 | 2 |
| 2 | 38 | 4 | 215646 | 11 | 9 | 0 |
| 3 | 53 | 4 | 234721 | 1 | 7 | 2 |
| 4 | 28 | 4 | 338409 | 9 | 13 | 2 |
| 4 | | 40 | 5 | 0 | | |
| .. | | ... | | ... | | |

preprocessing.LabelEncoder(): 실제 값에 상관 없이 0~K-1까지의 정수로 변환하는 함수(4.2.3.2 Encoding 참조)

정의한 카테고리컬 변수 리스트(Categorical_col)에서 하나씩 요소를 꺼내 반복문을 실행.

반복문 구성

df_train에서 카테고리컬 변수에 해당하는 열을 찾고, 그 값들의 dtype이 'object'라면 하위 코드를 실행.(조건문)

df_train[i].append(df_test[i]): df_train의 해당 열 값들과 df_test의 해당 열 값들을 합쳐 'category'변수에 할당.

LabelEncoder.fit(category.values): 카테고리 변수에서 사용된 값목록을 추출하여 인코딩.

df_train[i] = LabelEncoder.transform(df_train[i]): df_train의 해당 열의 인코딩을 수행하고 재할당

print(df_train.head(5)): 상위 5열만을 출력하여 df_train의 전처리된 형태 확인.

SECTION 1.4

트레이닝셋과 테스트셋 분리

- 타겟변수를 분리하여 트레이닝셋과 테스트셋 구성하기

실행 코드

```
## 트레이닝셋과 테스트셋의 income-level(타겟변수) 분리하기
x_test = np.array(df_test.drop(['income-level'], 1))
y_test = np.array(df_test['income-level'])
x = np.array(df_train.drop(['income-level'], 1))
y = np.array(df_train['income-level'])
```

np.array(df_test.drop(['income-level'], 1)) : df_test에서 타겟변수(income-level)열을 제외하고 numpy 배열 형태로 만듦.

SECTION 1.5

모델링

- `tree.DecisionTreeClassifier` 함수를 사용하여 decision tree 모델을 만든다.

실행 코드

```
# Decisiontree 모델링
```

```
dt = tree.DecisionTreeClassifier(criterion='entropy')  
dt.fit(x, y)
```

출력 결과

```
>>> # Decisiontree 모델링
```

```
dt = tree.DecisionTreeClassifier(criterion = 'entropy')  
dt.fit(x, y)
```

`dt = tree.DecisionTreeClassifier(criterion = 'entropy')` : Decisiontree 모델을 생성하며 노드를 나누는 기준으로 'entropy'를 사용한다. 이외의 파라미터들은 default값이 적용되었고, `tree.DecisionTreeClassifier` 함수의 나머지 파라미터들은

<http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html> 에 자세히 설명되어 있다.

SECTION 1.6

Decision tree 모델 평가

- 모델 사용 결과를 Confusion Matrix로 만들어 출력해보고 Accuracy를 계산한다.

실행 코드

모델 평가

```
pred = dt.predict(x_test)
```

```
confusion_matrix = pd.crosstab(y_test, pred, rownames=['True'],
                               colnames=['Predicted'], margins=True) ## crosstab
print(confusion_matrix)
```

```
total = confusion_matrix[0][0] + confusion_matrix[0][1] + confusion_matrix[1][0] + confusion_matrix[1][1]
accuracy = (confusion_matrix[0][0]+confusion_matrix[1][1]) / total
print(accuracy)
```

출력 결과

```
print(confusion_matrix)
```

```
Predicted   0   1  All
True
```

```
0      10878 1557 12435
```

```
1      1490 2356  3846
```

```
All     12368 3913 16281 >>> total = confusion_matrix[0][0] + confusion_matrix[0][1] + confusion_matrix[1][0] + confusion_matrix[1][1]
```

```
accuracy = (confusion_matrix[0][0]+confusion_matrix[1][1]) / total
```

```
print(accuracy)
```

```
0.812849333579
```

```
# dt.predict(x_test) : 생성된 Decisiontree 모델에 타겟변수를 제외한 테스트 데이터(x_test)를 적용한다.  
# pd.crosstab(y_test, pred, rownames=['True'], colnames=['Predicted'], margins=True) : pandas의 crosstab함수를 사용하여 Confusion Matrix를 생성한다.  
# 생성된 Confusion Matrix에서 accuracy를 계산한다.
```

SECTION 2

German credit data 분석 실습

Scikit-learn는 분석을 위한 패키지로서 Clustering, Model Selection, Ensemble, SVM 등 다변량 분석부터 머신러닝 기법까지 많은 부분의 분석을 제공한다. 이번 파트에서는 German credit data 를 통해 Logistic Regression, Decision Tree, Random Forest 세 가지 분석을 실시한 후 비교해 보겠다.

SECTION 2.1

German credit 데이터 설명

- German credit 데이터 설명

| Attribute | Type | Domain |
|-----------------------------------|-----------|---|
| Creditability | Binay | target |
| Account Balnace | Category | 1 : ... < 0 DM 2 : 0 <= ... < 200 DM 3 : ... >= 200 DM /alary assignments for at least 1 year 4 : no checking accoun |
| Duration of Credit (month) | Numerical | |
| Payment Status of Previous Credit | Category | 0 : no credits taken/ all credits paid back duly 1 : all credits at this bank paid back duly 2 : existing credits paid back duly till now 3 : delay in paying off in the past 4 : critical account/ other credits existing (not at this bank) |

| Attribute | Type | Domain |
|------------------------------|-----------|--|
| Purpose | Category | 0 : car (new) 1 : car (used) 2 : furniture/equipment 3 : radio/television 4 : domestic appliances 5 : repairs 6 : education 7 : (vacation - does not exist?) 8 : retraining 9 : business 10 : others |
| Credit Amount | Numerical | |
| Value Savings/Stocks | Category | 1 : ... < 100 DM 2 : 100 <= ... < 500 DM 3 : 500 <= ... < 1000 DM 4 : .. >= 1000 DM 5 : unknown/ no savings account |
| Length of current employment | Category | 1 : unemployed 2 : ... < 1 year 3 : 1 <= ... < 4 years 4 : 4 <= ... < 7 years 5 : .. >= 7 years |
| Instalment per cent | Numerical | |
| Sex & Marital Status | Category | 1 : male : divorced/separated 2 : female : divorced/separated/married 3 : male : single 4 : male : married/widowed 5 : female : single |

| Attribute | Type | Domain |
|-------------------------------|-----------|---|
| Guarantors | Category | 1 : none 2 : co-applicant 3 : guarantor |
| Duration in Current address | Numerical | |
| Most valuable available asset | Category | 1 : real estate 2 : if not A121 : building society savings agreement/ life insurance 3 : if not A121/A122 : car or other, not in attribute 6 4 : unknown / no property |
| Age (years) | Numerical | |
| Concurrent Credits | | 1 : other banks 2 : Dept Stores 3 : None |
| Type of apartment | | 1 : rent 2 : own 3 : for free |
| No of Credits at this Bank | Numerical | |
| Occupation | | 1 : unemployed/ unskilled - non-resident 2 : unskilled – resident 3 : skilled employee / official 4 : management/ self-employed/ highly qualified employee/ officer |
| No of dependents | Numerical | |
| Telephone | | 1 : none 2 : yes, registered under the customers name |
| Foreign Worker | | 1 : yes 2 : no |

SECTION 2.2

German credit 데이터 불러오기

▪ Pandas 패키지를 활용한 데이터 불러오기

실행 코드

```
import pandas as pd
import random

german=pd.read_csv("http://freakonometrics.free.fr/german_credit.csv") ## 파일불러오기
random.seed(1234) ## seed고정

subset_full=list(range(0,1000))
subset=random.sample(subset_full,600) ## train row 추출

train=german.loc[subset]
train_x=train.iloc[:, :-1] ## 트레이닝 셋 input data
train_y=train.iloc[:, -1] ## 트레이닝 셋 target data

test=german.drop(subset)
test_x=test.iloc[:, :-1] ## 테스트 셋 input data
test_y=test.iloc[:, -1] ## 테스트 셋 target data
```

```
# pd.read_csv : 판다스 패키지의 read_csv함수를 사용하여 web에서 csv파일을 불러온다.
# random.sample : train 과 test 로 나누기 위해 train 에 사용할 row를 추출한다.
# train_x = train.iloc[:, :-1] : 마지막 컬럼(type, 타겟변수)을 제외한 데이터를 train_x에 할당한다.
# train_y = train.iloc[:, -1] : 마지막 컬럼(type, 타겟변수)만 train_y에 할당한다.
# 테스트 데이터셋에도 동일하게 적용한다.
```

SECTION 2.3

LogisticRegression 함수를 사용한 로지스틱 회귀 모델링

실행 코드

```
##### logistic
from sklearn.linear_model import LogisticRegression
logistic=LogisticRegression()
logistic.fit(train_x,train_y)
logistic.score(train_x,train_y)
```

출력 결과

```
##### logistic.fit
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                    verbose=0, warm_start=False)

##### logistic.score
0.95499999999999996
```

logistic = LogisticRegression(): 특별한 옵션을 주지 않고 기본 default 값으로 binomial regression 실시 만약 multinomial regression 을 원할 때는 옵션을 지정하면 된다. LogisticRegression 함수의 파라미터들은 http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression 에 자세히 설명되어 있다.

SECTION 2.4

DecisionTreeClassifier 함수를 사용한 의사결정 모델링

실행 코드

```
##### Decision Tree
from sklearn.tree import DecisionTreeClassifier
DT=DecisionTreeClassifier(max_depth=10,min_samples_split=3)
DT.fit(train_x,train_y)
DT.score(train_x,train_y)
```

출력 결과

```
##### DT.fit
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,
                        max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
                        min_samples_split=3, min_weight_fraction_leaf=0.0,
                        presort=False, random_state=None, splitter='best')
```

```
##### DT.score
0.9916666666666667
```

DT = DecisionTreeClassifier(max_depth=10,min_saples_split=3): 최대 가지깊이는 10, 샘플은 최소 3개로 나뉘서 실시하는 의사결정 나무를 실시하였다 만약 DecisionTree 를 이용하여 연속형 변수를 모델링 하고 싶다면 DecisionTreeRegressor 함수를 사용해야 한다.
DecisionTreeClassifier 함수의 파라미터들은 <http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier> 에 자세히 설명되어 있다.

SECTION 2.5

RandomForestClassifier 함수를 사용한 랜덤포레스트 모델링

실행 코드

```
##### Radom Forest
from sklearn.ensemble import RandomForestClassifier
RF=RandomForestClassifier(n_estimators=5000)
RF.fit(train_x,train_y)
RF.score(train_x,train_y)
```

출력 결과

```
##### RF.fit
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=None, max_features='auto', max_leaf_nodes=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=5000, n_jobs=1,
                        oob_score=False, random_state=None, verbose=0,
                        warm_start=False)

##### RF.score
1.0
```

RF = RandomForestClassifier(n_estimators=5000) : tree수 5000개로 실시하는 랜덤포레스트 모델을 실시하였다 tree수가많을수록 예측력은 증가하지만 overfitting 될 가능성과 시간이 오래걸리는 단점이 있다. RandomForestClassifier 함수의 파라미터들은 <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier> 에 자세히 설명되어 있다.

SECTION 2.6

여러 모델 평가

▪ Confusion Matrix

실행 코드

```
### predict
pred_logit=logistic.predict(test_x)
pred_DT=logistic.predict(test_x)
pred_RF=RF.predict(test_x)

#### confusion_matrix
confusion_matrix_logit=pd.crosstab(test_y, pred_logit, rownames=['True'],
                                   colnames=['Predicted'], margins=True)
confusion_matrix_DT=pd.crosstab(test_y, pred_DT, rownames=['True'],
                                 colnames=['Predicted'], margins=True)
confusion_matrix_RF=pd.crosstab(test_y, pred_RF, rownames=['True'],
                                 colnames=['Predicted'], margins=True)
print(confusion_matrix_logit)
print(confusion_matrix_DT)
print(confusion_matrix_RF)
```

출력 결과

```
#### print(confusion_matrix_logit)
```

```
Predicted   1   2 All
```

True

```
1           385   4  389
```

```
2           11   0   11
```

```
All         396   4  400
```

```
#### print(confusion_matrix_DT)
```

```
Predicted   1   2 All
```

True

```
1           385   4  389
```

```
2           11   0   11
```

```
All         396   4  400
```

```
#### print(confusion_matrix_RF)
```

```
Predicted   1 All
```

True

```
1           389  389
```

```
2           11   11
```

```
All         400  400
```

- Accuracy

실행 코드

```
##### accuracy
accuracy_logit=(confusion_matrix_logit[1][1]+confusion_matrix_logit[2][2]) / 400
accuracy_DT=(confusion_matrix_DT[1][1]+confusion_matrix_DT[2][2]) / 400
accuracy_RF=(confusion_matrix_RF[1][1]) / 400

accuracy=Series([accuracy_logit,accuracy_DT,accuracy_RF],index=["logistic","DecisionTree","RandomForest"])
print(accuracy)
```

출력 결과

```
logistic      0.9625
DecisionTree  0.9625
RandomForest  0.9725
```