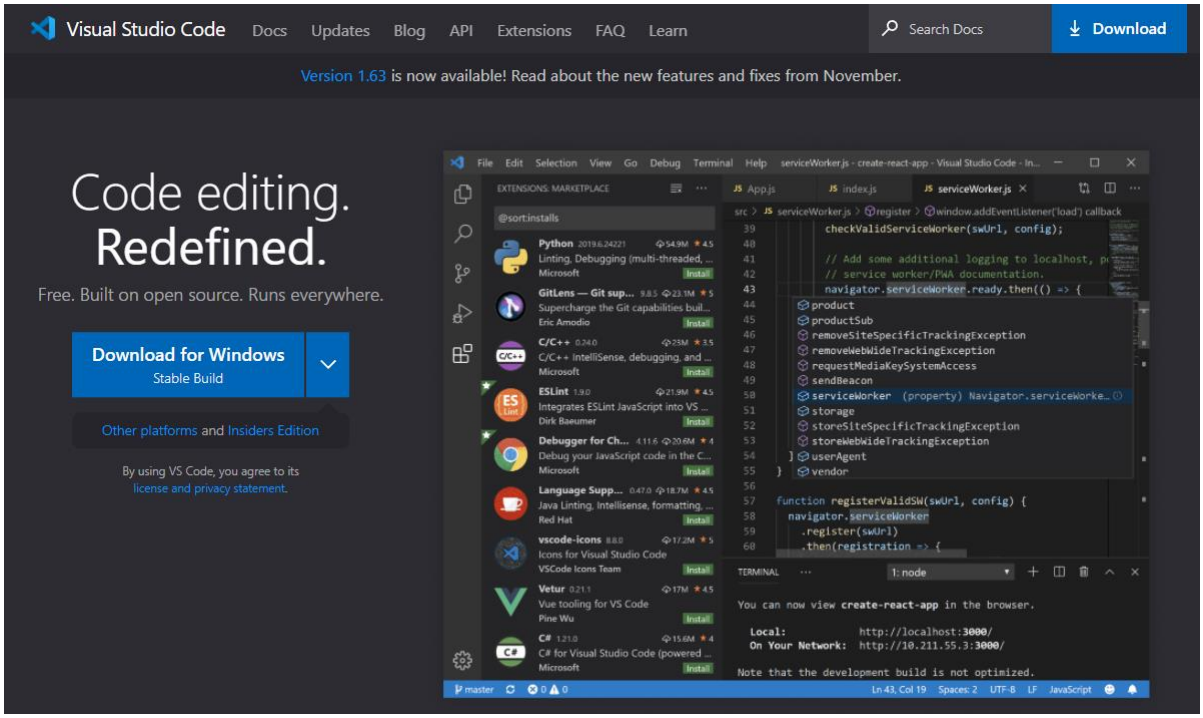
목차

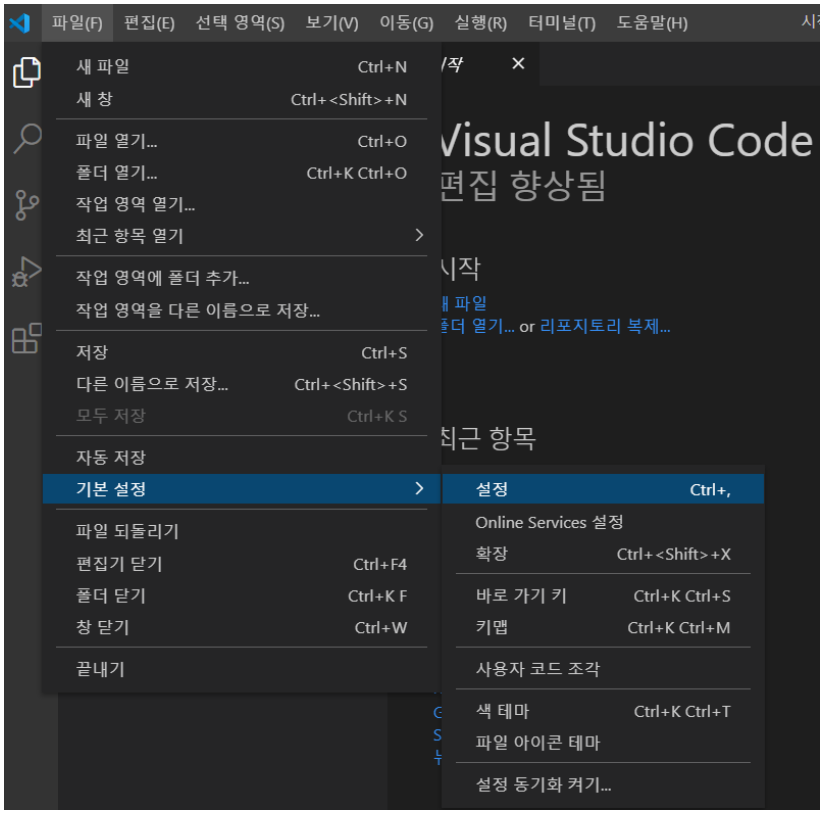
1장 파이썬 개발을 위한 Visual Studio Code 설치와 셋팅

꾸준하게 파이썬 커뮤니티에서 인기를 끌고 있는 개발툴은 Visual Studio Code 입니다. 다양한 개발환경을 지원하면서 윈도우, 맥, 리눅스를 모두 지원하는 만능의 통합 에디터입니다. python IDLE 를 사용하면서 추가로 설치해서 사용하면 복잡한 코드나 파일로 구성된 스크립트를 쉽게 디버깅할 수 있는 장점이 있습니다.

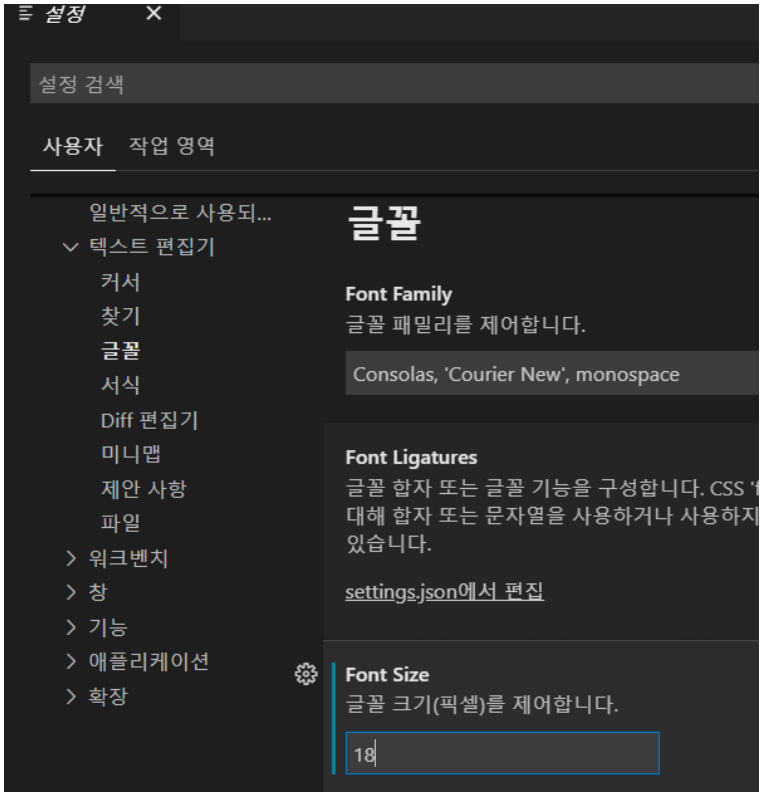
설치를 하려면 https://code.visualstudio.com 사이트에 접속해서 최신 버전을 받으면 됩니다. 업데이트가 빠르게 진행되서 책에 추가된 버전보다 높은 버전들이 보여도 전혀 문제 없습니다. 최신 버전을 설치하면 됩니다. 윈도우, 맥, 리눅스등의 주요 플랫폼을 모두 지원합니다.



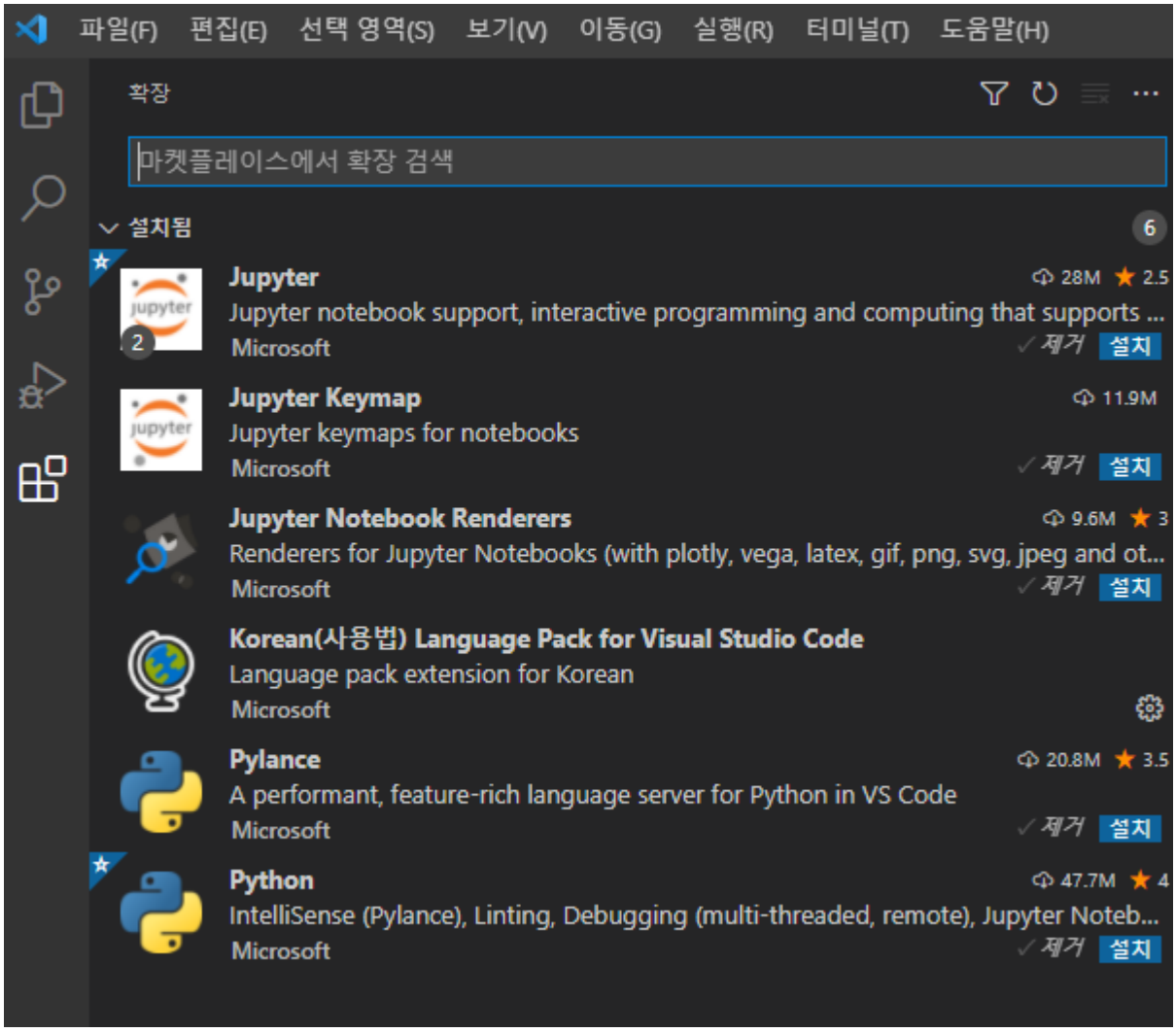
약간의 환경 셋팅이 필요합니다. 기존 통합툴과는 달리 VisualStudio Code 는 툴바가 없습니다. 메뉴와 단축키를 통해 사용하면 됩니다. 파일(File) => 기본설정(Preferences) => 설정(Settings)을 클릭합니다. 실습을 위해 글자체나 글자크기등을 변경할 수 있습니다.



항목중에 글꼴에서 Font Size 를 변경할 수 있습니다. 10 포인트가 너무 작다면 12~16 포인트로 변경하면 됩니다.

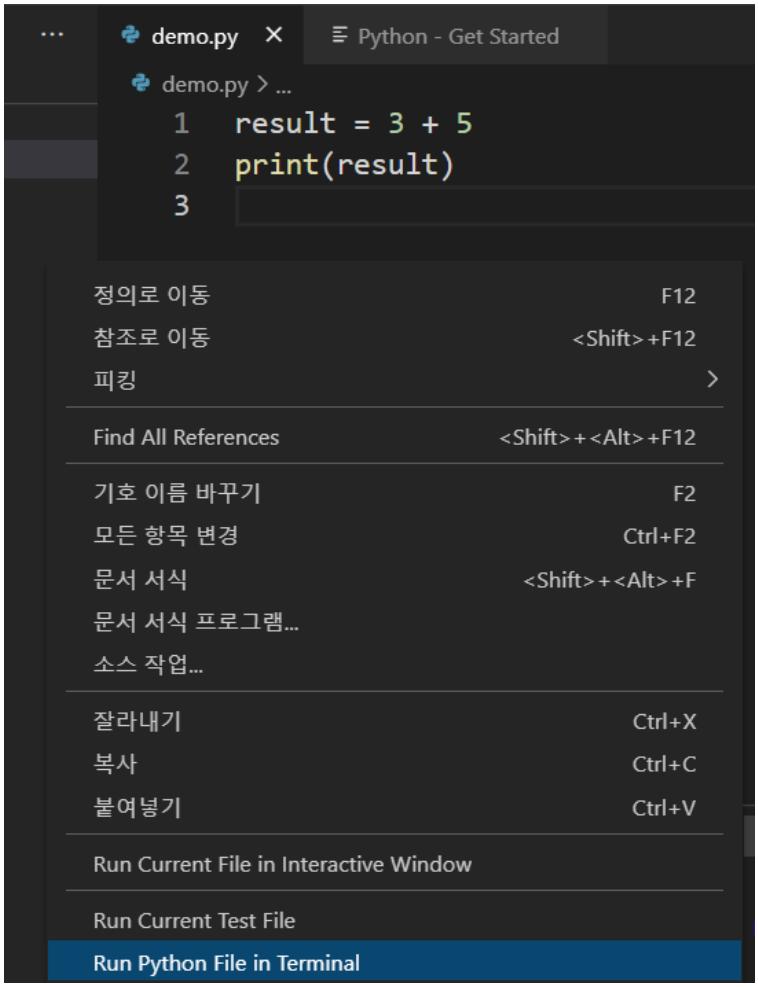


이제 익스텐션(Extension)을 설치하면 파이썬을 위한 개발툴로 활용할 수 있습니다. Visual Studio Code 에서 왼쪽에 있는 탭에서 Extension 을 클릭합니다. 다섯번째에 있는 버튼입니다. 마켓플레이스에서 Python 을 입력합니다. 가장 위쪽에 올라오는 Python 을 설치하면 추가로 Pylance, Jupyter Notebook Renderers, Jupyter Keymap, Jupter 등이 설치됩니다. 나중에 사용하게 되는 Extension 이기 때문에 그대로 두면 됩니다.



아무래도 개발툴이 한글 지원이 되면 좀 더 편합니다. 익스텐션에서 "Korean Language Pack for Visual Studio Code"을 검색해서 Install 을 클릭하면 한글로 변경됩니다. 혹시 Visual Studio Code 가 영문으로 계속 나오면 툴을 한번 종료했다가 다시 시작하면 됩니다.

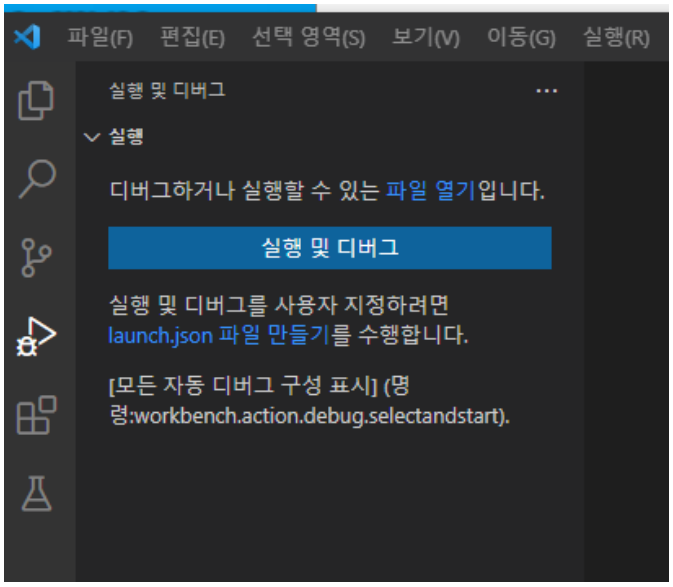
테스트를 위해 파일메뉴에서 "새 파일"을 클릭합니다. 파일 이름에 "demo"을 입력하고 파일 형식에서 "Python"을 찾아서 클릭합니다.



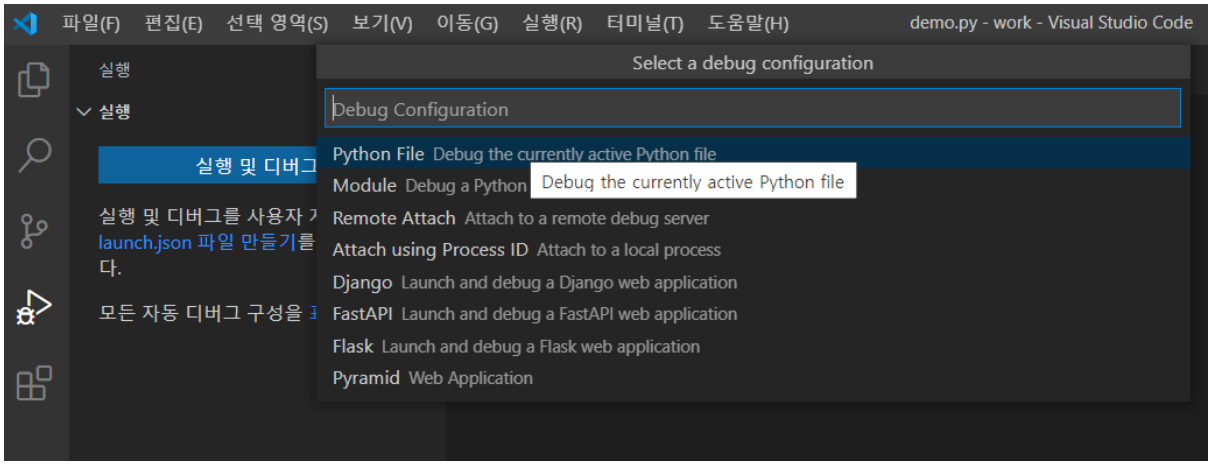
위와 같이 코드를 작성해 봅니다. 간단한 디버깅을 위해 함수를 하나 정의하고 함수를 호출하고 반복구문을 돌려봅니다. 작성한 코드를 실행할 경우 마우스 오른쪽 버튼을 클릭해서 "Run Python File in Terminal"을 클릭하면 됩니다. 하단에 터미널이 오픈되서 결과가 출력됩니다.

이번에는 디버깅하는 방법을 연습해 봅니다. 작성중인 demo.py 파일에 보면 라인번호 바로 앞에 마우스로 클릭해서 중단점(Break Point)를 추가할 수 있는 공간이 있습니다. 여기를 클릭해서 적색 점이 추가되면 디버깅하는 모드에서는 중단점 역할을 수행합니다. 적색점은 토글이 가능합니다. 한번 더 클릭하면 삭제가 되고, 다시 클릭하면 중단점이 추가됩니다. 왼쪽상단에 위치한 버튼들

중에 네번째 있는 디버그 버튼을 클릭합니다. 처음 디버깅을 하는 경우에는 환경값을 구성해야 합니다. launch.json 파일을 생성해야 합니다. 비주얼 스튜디오 코드에서 왼쪽의 버튼중에 삼각형으로 되어 있는 플레이 버튼이 "실행 및 디버그"버튼입니다. "launch.json 파일 만들기"를 클릭하면 됩니다. 처음 한번만 클릭하면 됩니다.

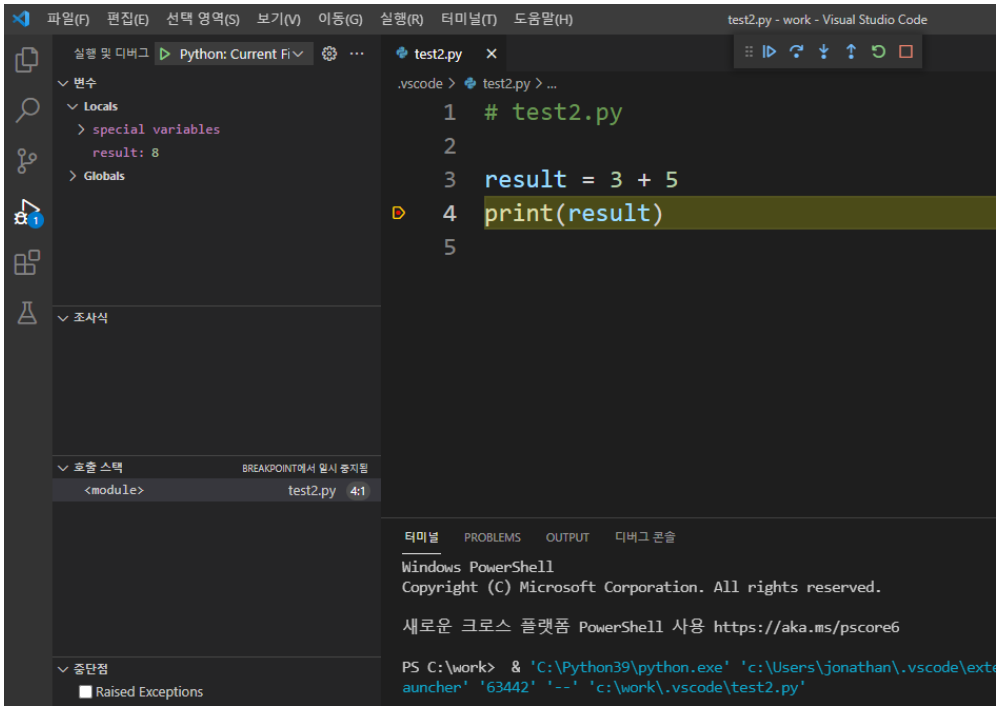


아래의 화면과 같이 어떤 파이썬 Debug Configuration 을 물어보면 가장 상단에 있는 "Python File"을 클릭하면 됩니다. 우리는 웹 환경이 아닌 기본 파이썬 개발 환경에서 작업을 하기 때문입니다.



상단에 플레이 버튼을 클릭하면 디버깅이 시작됩니다(단축키는 F5 입니다). 중지점에서 단축키로 F11 을 누르면 라인단위(Step Into)로 디버깅되는 것을 볼 수 있습니다. Visual Studio Code 는 코드 자동완성과 디버깅하는 용도로 멋진

도구입니다! 다른 개발자가 작성한 코드를 분석하거나 혹은 내가 작성한 코드에 문제점이 있는지를 체크하고 검사할 경우 디버깅하는 도구가 있다면 편하게 작업을 할 수 있습니다. 여러 번 사용하면서 익숙해지도록 연습을 하면 됩니다.



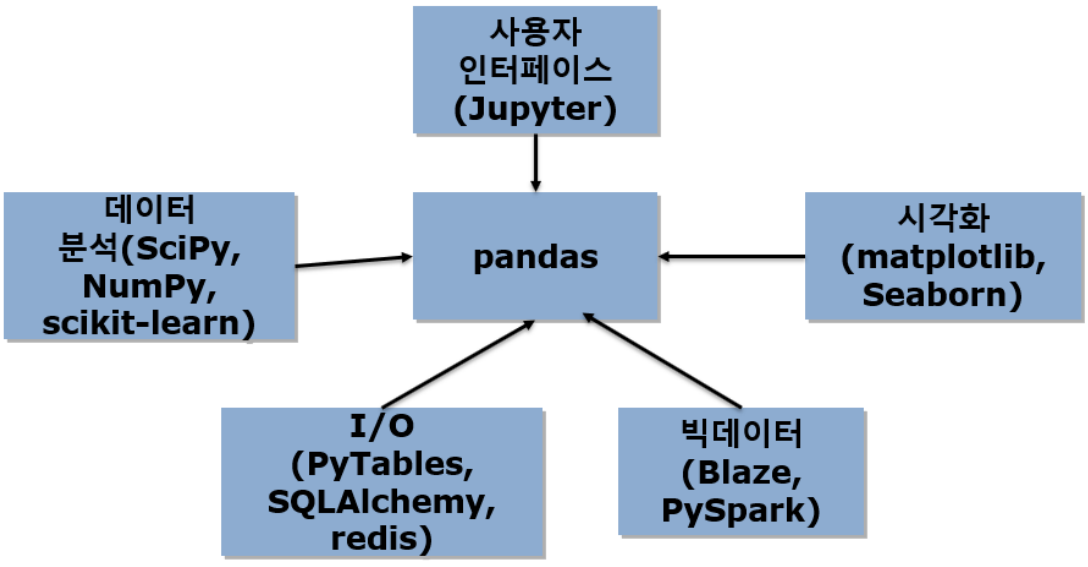
2장 데이터 분석을 위한 pandas, matplotlib 연습

1) 데이터 사이언스 분야 소개

파이썬은 통계에 특화된 R언어와 비교해도 데이터 분석이나 시각화 분야에서 매우 빠르게 인기가 올라가고 있습니다. 시중 서점의 IT코너를 살펴보면 파이썬이란 언어가 국내에서도 자바나 .NET만큼 인기를 끌면서 다양한 분야의 파이썬 책들이 출판되어 있습니다. 파이썬은 앞에서 살펴본 것 처럼 문자열 처리, 웹크롤링, GUI, 데이터베이스 연계 작업 뿐만 아니라 데이터 분석과 시각화에도 편리하게 사용할 수 있는 개발툴과 모듈들이 제공됩니다.

파이썬의 인기있는 활용 분야 중에 하나가 데이터 사이언스 분야입니다. 인공지능 분야와 데이터 사이언스 분야는 파이썬의 핫한 분야들 중에 하나입니다. 파이썬을 이용한 데이터 관리 및 분석 도구들을 아래와 같이 모아서 PyData Stack이라고 부릅니다.

도토리를 줍듯이 데이터를 인터넷상에서 크롤링하고, 회사 내부에 있는 CSV, 텍스트 파일, 엑셀 파일을 읽어서 하나의 거대한 스프레드시트(또는 관계형데이터베이스에서 말하는 테이블)로 만드는 것은 pandas에서 제공하는 DataFrame클래스를 통해서 처리합니다. 이렇게 수집하고 분석한 DataFrame객체를 시각화 하는 것은 matplotlib을 통해 처리할 수 있습니다. 이런 데이터를 분석해서 머신 러닝 분야에서 활용할 수 있는 접근하기 쉬운 프레임워크는 scikit-learn입니다.



수업에서는 주로 pandas와 matplotlib을 살펴보려고 합니다. pandas는 구조화된 데이터를 빠르고 쉬우면서도 다양한 형식으로 가공할 수 있는 풍부한 자료 구조와 함수를 제공합니다. pandas는 NumPy의 고성능 배열 계산 기능과 스프레드시트, SQL 같은 관계형 데이터베이스의 유연한 데이터 조작 기능을 조합한 것입니다.

추가로 사용되는 라이브러리는 아래와 같이 설치하면 됩니다. pandas와 matplotlib이 필요하고 추가로 필요한 라이브러리들이 있어서 커맨드창(층)을 오픈한 상태에서 아래와 같이 설치하면 됩니다.

```
pip install numpy
pip install scipy
pip install matplotlib
pip install pandas
pip install seaborn
pip install xlrd
pip install openpyxl
```

Pandas로 할 수 있는 일은 다음과 같습니다.

- 데이터 입출력 기능(csv, excel, RDB, HDF5)
- 데이터를 처리에 효율적인 포맷으로 저장
- 데이터의 NaN(누락값)처리
- 데이터를 일부 분리 혹은 결합
- 데이터의 유연한 변경을 위한 피봇 처리
- 데이터에 대한 통계 처리 및 회귀 처리
- 데이터 요약 및 그룹 연산

matplotlib는 그래프나 2차원 데이터 시각화를 생성하는 유명한 파이썬 라이브러리입니다. IPython에서 matplotlib으로 생성한 그래프는 그래프 윈도우에 있는 툴바로 특정부분을 확대하거나 그래프의 여기저기를 인터랙티브하게 살펴볼 수 있습니다.

Jupyter qtconsole은 계산용 파이썬 도구 모음에 포함된 컴포넌트이며 인터랙티브하고 강력한 생산적인 환경을 제공합니다. Jupyter qtconsole은 파이썬 코드를 작

성하고 테스트하며 디버깅을 할 수 있는 생산적인 향상된 파이썬 쉘을 제공합니다.

- Jupyter qtconsole은 웹브라우져와 연결할 수 있는 메스메티카 스타일의 HTML 노트북 기능이 제공됩니다.
- 그래프를 즉시 그려보거나 여러 줄을 편집할 수 있는 기능 그리고 문법 강조 기능을 가진 Qt 프레임워크 기반의 GUI콘솔입니다.

2)pandas의 Series와 DataFrame학습

pandas에서 제공하는 Series클래스와 DataFrame클래스는 다양한 방식으로 임포트를 할 수 있습니다.

from pandas import Series, DataFrame

Series

Out[13]: pandas.core.series.Series

import pandas

pandas.Series

Out[15]: pandas.core.series.Series

멀티캠퍼스의 각 날짜별 주식의 종가를 입력해 봅니다.

multicampus = Series([31500, 31300, 31000, 31200, 31300, 31400])

multicampus

Out[17]:

0 31500

1 31300

2 31000

3 31200

4 31300

5 31400

dtype: int64

날짜와 같이 종가를 보여주면 편합니다. 그래서 아래와 같이 인덱스를 추가합니다.

multicampus2 = Series([31500, 31300, 31000, 31200, 31300, 31400], index=['02-06', '02-05', '02-04', '02-03', '02-02', '02-01'])
multicampus2
Out[19]:
02-06 31500
02-05 31300
02-04 31000
02-03 31200
02-02 31300
02-01 31400
dtype: int64

앞에서 배운 사전구조처럼 날짜를 키로 주면 종가를 리턴합니다.

multicampus2['02-05']
Out[20]: 31300

이번에는 pandas에서 제공하는 DataFrame을 학습해 봅니다. 아래는 네이버에서 검색한 삼성전자시세입니다.

일별시세

날짜	종가	전일비	시가	고가	저가	거래량
2021.12.27	80,200	▼ 300	80,600	80,600	79,800	7,412,235
2021.12.24	80,500	▲ 600	80,200	80,800	80,200	12,086,380
2021.12.23	79,900	▲ 500	79,800	80,000	79,300	13,577,498
2021.12.22	79,400	▲ 1,300	78,900	79,400	78,800	17,105,892
2021.12.21	78,100	▲ 1,000	77,900	78,300	77,500	14,245,298
2021.12.20	77,100	▼ 900	77,600	77,800	76,800	11,264,375
2021.12.17	78,000	▲ 200	76,800	78,000	76,800	13,108,479
2021.12.16	77,800	▲ 200	78,500	78,500	77,400	11,996,128
2021.12.15	77,600	▲ 600	76,400	77,600	76,300	9,584,939
2021.12.14	77,000	▲ 200	76,500	77,200	76,200	10,976,660

위와 같은 구조를 저장하려면 엑셀처럼 저장해야 합니다. 각 일자별로 외국인 거래량, 지분율, 기관 거래량, 일별 주가, 등락률이 있습니다. 키 이름을 주고 리스트를 저장합니다. 입력을 쉽게 할 수 있도록 간단한 숫자들을 사용해서 딕셔너리에 저장합니다. 아직은 실제 데이터가 아니기 때문에 입력이 간단한 숫자들을 입력해 둡니다.

```
data = { 'foreigner':[1,2,3,4,5,6], 'sratio':[10,20,30,40,50,60],
         'org':[100,200,300,400,500,600], 'sprice':[1,2,3,4,5,6],
         'private':[10,20,30,40,50,60] }
```

```
data
```

```
Out[2]:
{'foreigner': [1, 2, 3, 4, 5, 6],
 'org': [100, 200, 300, 400, 500, 600],
 'private': [10, 20, 30, 40, 50, 60],
 'sprice': [1, 2, 3, 4, 5, 6],
 'sratio': [10, 20, 30, 40, 50, 60]}
```

딕셔너리를 인자로 하여 DataFrame이라는 클래스 생성자를 다음과 같이 호출하면 DataFrame객체가 생성됩니다.

```
frame = DataFrame(data)
```

```
type(frame)
```

```
Out[27]: pandas.core.frame.DataFrame
```

```
frame
```

```
Out[6]:
```

	foreigner	org	private	sprice	sratio
0	1	100	10	1	10
1	2	200	20	2	20
2	3	300	30	3	30

3	4	400	40	4	40
4	5	500	50	5	50
5	6	600	60	6	60

Pandas의 DataFrame도 각 행에 대해서 숫자로 구성된 인덱스 값을 자동으로 할당해 줍니다. 이것은 Series 자료구조와 동일합니다. 출력된 컬럼의 순서를 변경하려면 다음과 같이 간단하게 변경할 수 있습니다.

frame2 = DataFrame(data, columns=['foreigner', 'sratio', 'org', 'sprice', 'private'])

frame2

Out[9]:

	foreigner	sratio	org	sprice	private
0	1	10	100	1	10
1	2	20	200	2	20
2	3	30	300	3	30
3	4	40	400	4	40
4	5	50	500	5	50
5	6	60	600	6	60

DataFrame은 Series와 마찬가지로 인덱스 값을 자동으로 할당되는 숫자가 아닌 개발자가 원하는 명시적인 값으로 지정할 수 있습니다. 일자를 인덱스로 사용하도록 변경해 봅니다.

frame3 = DataFrame(data, columns=['foreigner', 'sratio', 'org', 'sprice', 'private'], index=['02-06', '02-05', '02-04', '02-03', '02-02', '01-30'])

frame3

Out[12]:

	foreigner	sratio	org	sprice	private
02-06	1	10	100	1	10

02-05	2	20	200	2	20
02-04	3	30	300	3	30
02-03	4	40	400	4	40
02-02	5	50	500	5	50
01-30	6	60	600	6	60

DataFrame의 컬럼은 Column의 이름을 통해서 쉽게 열 단위로 데이터에 접근할 수 있습니다.

frame3['sratio']

Out[13]:

02-06 10
02-05 20
02-04 30
02-03 40
02-02 50
01-30 60
Name: sratio, dtype: int64

2차원의 데이터에서 행단위로 접근하는 것은 에러가 납니다. 그래서 loc로 접근해야 합니다.

frame3.loc['02-05']

Out[14]:

foreigner 2
sratio 20
org 200
sprice 2
private 20
Name: 02-05, dtype: int64

DataFrame을 이용하면 행과 열을 쉽게 바꿀 수 있습니다. 분석하는 관점을 변경

한다면 이렇게 데이터를 행과 열을 뒤집어서 접근할 수 있습니다.

frame3.T

Out[15]:

	02-06	02-05	02-04	02-03	02-02	01-30
foreigner	1	2	3	4	5	6
sratio	10	20	30	40	50	60
org	100	200	300	400	500	600
sprice	1	2	3	4	5	6
private	10	20	30	40	50	60

첫번째 단계에서는 먼저 데이터를 수집해야 합니다. 주위를 둘러보면서 도토리를 줍듯이 데이터를 수집하는 단계입니다. Pandas로 파일을 읽어서 DataFrame객체를 생성하는 경우 몇가지 함수들이 제공됩니다.

함수	설명
read_csv	파일에서 구분된 데이터를 읽어옵니다. 데이터 구분자는 쉼표(,)를 기본으로 합니다.
read_table	파일, URL 또는 파일과 유사한 객체로부터 구분된 데이터를 읽어옵니다. 데이터 구분자는 탭(₩t)를 기본으로 합니다.
read_fwf	고정폭 컬럼 형식에서 데이터를 읽어옵니다(구분자가 없는 데이터).
read_excel	엑셀에 있는 데이터를 읽어옵니다.

첫번째는 pandas에 전역 함수인 read_csv()를 사용해서 구분자가 ","로 되어 있는 ex1.csv파일의 내용을 읽어서 바로 DataFrame객체를 리턴받는 방법입니다. Jupyter 노트북 개발환경에서는 운영체제에 있는 명령을 직접 내부에서 실행할 수 있도록 !명령어를 지원합니다. 텍스트 파일의 내용을 출력하도록 !type을 실행할 수 있습니다.

!type c:₩work₩ex1.csv

id, name, price, description

1, iphone, 890000, iphone 6s 7 8 x

2, android, 990000, samsung phone

3, winphone, 450000, microsoft winphone

import pandas as pd

df = pd.read_csv('c:₩₩work₩₩ex1.csv')

df

Out[13]:

	id	name	price	description
0	1	iphone	890000	iphone 6s 7 8 x
1	2	android	990000	samsung phone
2	3	winphone	450000	microsoft winphone

pd.read_csv('c:₩work₩ex2.csv', header=None)

Out[21]:

	0	1	2	3
0	1	iphone	890000	iphone 6s 7 8 x
1	2	android	990000	samsung phone
2	3	winphone	450000	microsoft winphone

컬럼이름을 부여하는 경우라면 아래와 같이 실행합니다.

pd.read_csv('c:₩work₩ex2.csv', names=['id','name','price','description'])

Out[23]:

	id	name	price	description
0	1	iphone	890000	iphone 6s 7 8 x
1	2	android	990000	samsung phone
2	3	winphone	450000	microsoft winphone

계층적 색인을 지정하고 싶다면 컬럼 번호나 이름의 리스트를 넘기면 됩니다.

!type c:₩work₩csv_mindex.csv

```
key1, key2, value1, value2
mobile, ios, 100, 200
mobile, android, 200, 500
mobile, bada, 5, 10
mobile, tizen, 150, 250
notebook, ios, 11, 12
notebook, android, 22, 33
notebook, bada, 50, 60
notebook, tizen, 300, 400
```

pd.read_csv('c:₩work₩csv_mindex.csv', index_col=['key1', 'key2'])

Out[33]:

		value1	value2
key1	key2		
mobile	ios	100	200
	android	200	500
	bada	5	10
	tizen	150	250
notebook	ios	11	12
	android	22	33
	bada	50	60
	tizen	300	400

별도의 구분자가 없고 공백문자만 있는 경우라면 read_table()함수로 읽어서 처리합니다. 앞에서 학습한 정규표현식에서 정리한 것처럼 ₩s는 공백문자를 의미하고 +는 1개 또는 그 이상의 공백문자가 출현할 수 있음을 의미합니다.

result = pd.read_table('c:₩work₩ex3.txt', sep='₩s+')

result

Out[37]:

	data1	data2	data3
0	1.2	2.3	1.3
1	0.1	1.5	2.5
2	3.1	3.2	3.3

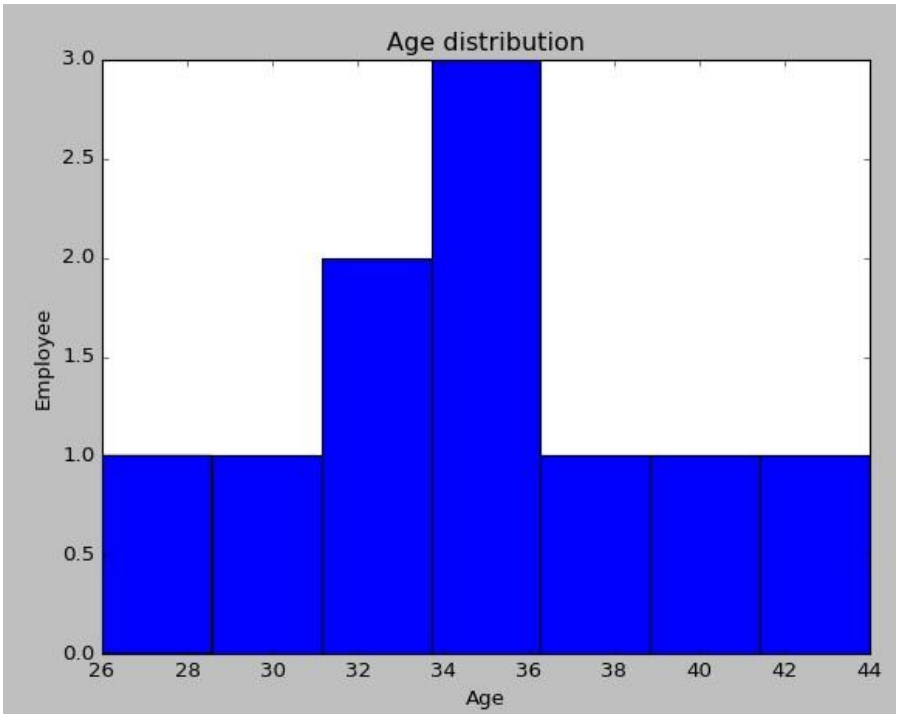
엑셀파일을 아래와 같이 작성합니다. 강사가 제공하는 Demo.xlsx를 사용하면 됩니다.

EMPID	Gender	Age	Sales	BMI	Income
E001	M	34	123	Normal	350
E002	F	40	114	Overweight	450
E003	F	37	135	Obesity	169
E004	M	30	139	Underweight	189
E005	F	44	117	Underweight	183
E006	M	36	121	Normal	90
E007	M	32	133	Obesity	166
E008	F	26	140	Normal	120
E009	M	32	133	Normal	75
E010	M	36	133	Underweight	40

아래의 %pylab이라는 매직 명령어는 그래프를 출력하는 경우에 사용합니다.
주피터 노트북인 경우는 %matplotlib inline 명령어를 사용합니다.

```
import matplotlib.pyplot as plt
import pandas as pd
%pylab
Using matplotlib backend: Qt4Agg
Populating the interactive namespace from numpy and matplotlib

df = pd.read_excel('c:₩₩work₩₩demo.xlsx', 'Sheet1')
```

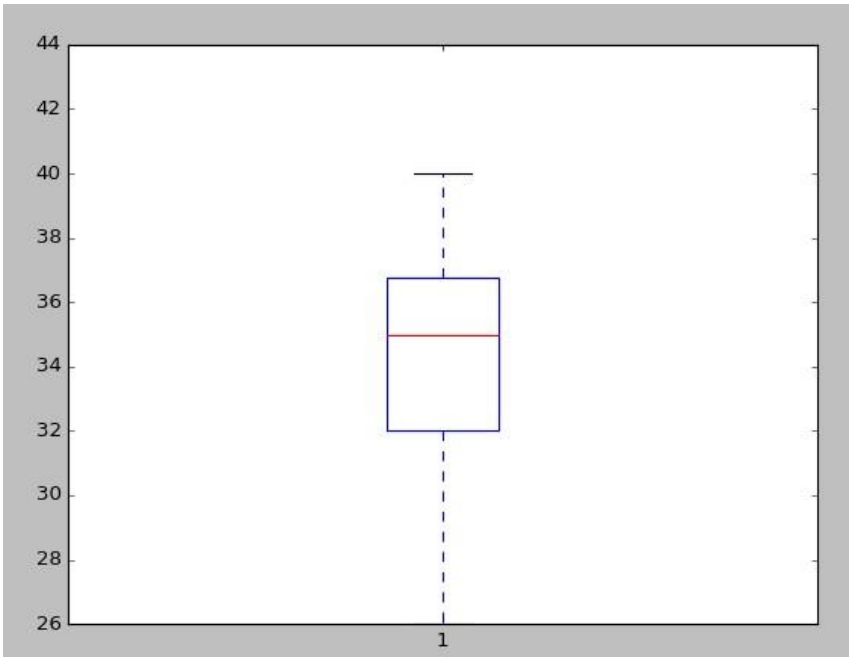
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.hist(df['Age'], bins = 7)
Out[8]:
(array([ 1.,  1.,  2.,  3.,  1.,  1.,  1.]),
 array([ 26.        ,  28.57142857,  31.14285714,  33.71428571,
         36.28571429,  38.85714286,  41.42857143,  44.        ]),
 <a list of 7 Patch objects>)

plt.title('Age distribution')
Out[9]: <matplotlib.text.Text at 0x6c741d0>

plt.xlabel('Age')
Out[10]: <matplotlib.text.Text at 0xf0eff0>

plt.ylabel('Employee')
Out[11]: <matplotlib.text.Text at 0x6c6d8f0>
```



(왼쪽의 y축에는 사람 숫자, 하단의 X축은 나이별로 출력됩니다)

(박스 차트)

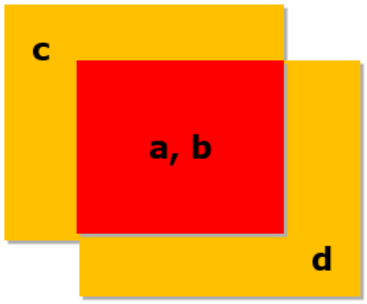
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.boxplot(df['Age'])
```



(나이가 밀집되어 있는 구간이 32살에서 36살임을 알 수 있습니다.)

데이터베이스 스타일로 DataFrame을 합칠 수 있습니다. merge()함수를 사용해 봅니다. 우리가 사용하는 텍스트파일, CSV파일, 엑셀파일들이 혹시 관계형 데이터베이스에 저장된 내용을 추출(Export)한 것이라면 관계성을 가지고 있는 경우도 많습니다. 예를 들면 제품 테이블의 데이터를 products.csv로 저장하고, 분류테이블의 데이터를 categories.csv로 저장했다면 분류테이블의 기본키(행 단위로 입력되는 데이터의 유일성을 체크하는 제약조건) 컬럼인 categoryID가 제품테이블에는 categoryID 참조키(다른 테이블의 기본키 컬럼을 복사해서 사용) 컬럼으로 복사되어 있는 경우입니다. 보통 1:N(일대다)의 관계성을 가진다고 말합니다. 혹시 이런 경우의 단편적인 데이터라면 데이터를 합쳐야 원하는 관점의 분석을 할 수 있습니다.

아래의 코드는 1:N(일대다)의 예제입니다. df1 데이터는 key컬럼은 정리하면 a, b, c를 가지고 있고 df2의 key컬럼은 a, b, d를 가지고 있습니다. 두 개의 DataFrame을 key컬럼 기반으로 합치면 a, b가 교집합으로 처리될 수 있습니다.




```
from pandas import Series, DataFrame
df1 = DataFrame({'key':['b','b','a','c','a','a','b'], 'data1':range(7)})

df2 = DataFrame({'key':['a','b','d'], 'data2':range(3)})
```

df1

Out[29]:

	data1	key
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	a
6	6	b

df2

Out[30]:

	data2	key
0	0	a
1	1	b
2	2	d

어떤 컬럼을 병합할지를 명시하지 않으면 merge()함수는 겹치는 컬럼의 이름을

키로 사용합니다. 여기서는 key컬럼입니다. 기존 df1에 df2에 있던 data2컬럼이 병합이 됩니다. 그러면 a라는 key에는 0이 추가되고, b라는 key에는 1이 추가됩니다.

pd.merge(df1, df2, on='key')

Out[31]:

	data1	key	data2
0	0	b	1
1	1	b	1
2	6	b	1
3	2	a	0
4	4	a	0
5	5	a	0

만약 두 객체에 공통되는 컬럼 이름이 하나도 없다면 따로 지정해 주면 됩니다.

df3 = DataFrame({'lkey':['b','b','a','c','a','a','b'], 'data1':range(7)})
df4 = DataFrame({'rkey':['a','b','d'], 'data2':range(3)})

pd.merge(df3, df4, left_on='lkey', right_on='rkey')

Out[34]:

	data1	lkey	data2	rkey
0	0	b	1	b
1	1	b	1	b
2	6	b	1	b
3	2	a	0	a
4	4	a	0	a
5	5	a	0	a

수평으로 데이터를 병합하는 것이 아닌 수직으로 데이터를 병합할 때는 concat() 함수를 사용할 수도 있습니다. 동일한 형식의 행 데이터들을 결합할 경우는 concat()함수를 사용하면 됩니다.

```
s1 = Series([0,1], index=['a','b'])
s2 = Series([2,3,4], index=['c','d','e'])
s3 = Series([5,6], index=['f','g'])
pd.concat([s1, s2, s3])
```

Out[38]:

```
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
```

3)matplotlib소개와 활용

pandas와 같이 사용할 라이브러리 중에 matplotlib가 있습니다. 이 라이브러리를 살펴보도록 합니다. matplotlib는 주로 2D 도표를 위한 데스크탑 패키지로 출판물 수준의 도표를 만들 수 있도록 설계되었습니다. 2002년 존 헌터는 파이썬에서 MATLAB과 유사한 인터페이스를 지원하고자 matplotlib프로젝트를 시작했습니다. 그 후로 다른 많은 개발자들이 수년간 협력해서 Jupyter qtconsole, notebook과 matplotlib를 통합해서 과학계산 컴퓨팅을 위한 다양한 기능을 겸비한 생산적인 환경을 구축했습니다. Jupyter에서 GUI툴킷과 함께 matplotlib를 사용하면 도표의 확대와 회전과 같은 인터랙티브한 기능을 사용할 수 있습니다.

matplotlib에서 그래프는 figure 객체 내에 존재합니다. 그래프를 위한 새로운 그림판은 plt.figure를 사용해서 생성할 수 있습니다. 아래와 같은 매직 명령어 %pylab을 실행하면 matplotlib과 numpy를 별도로 임포트하지 않아도 바로 사용할 수 있습니다. 그림판에 차트를 출력하고자 할 때 미리 실행하는 특수 명령어 입니다.

%pylab

Using matplotlib backend: Qt5Agg

Populating the interactive namespace from numpy and matplotlib

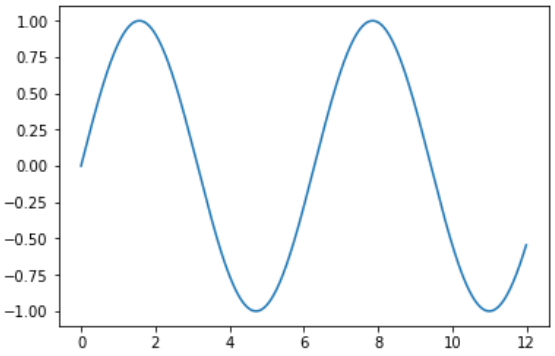
또는 아래와 같이 직접 matplotlib.pyplot와 numpy를 임포트해도 됩니다. 그림판이 현재 작업하는 콘솔 내부에 출력되기를 원하면 %matplotlib inline을 실행하면 됩니다. np.arange는 0부터 12까지 0.01 간격으로 데이터를 만들고 그 리스트를 np.sin에 입력하면 sin값이 나타납니다. t라는 시간 혹은 그래프상에서 x축을 의미하는 데이터를 0부터 12까지 만들고 사인 함수(np.sin)에 입력해서 그 출력을 y로 저장했습니다. t는 약 1200개 정도의 값을 가진 일종의 배열입니다.

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

t = np.arange(0, 12, 0.01)

y = np.sin(t)

plt.figure(figsize=(10,6))
```

Out[43]: <matplotlib.figure.Figure at 0x2d9380759b0><matplotlib.figure.Figure at 0x2d9380759b0>

```
plt.plot(t, y)
```

Out[44]: [<matplotlib.lines.Line2D at 0x2d9381d6978>]



다시 아래의 코드를 실행하면 새로운 그림판을 출력하고 여기에 격자를 추가합니다. xlabel()메서드로 x축 라벨을, ylabel()메서드로 y축 라벨을 출력합니다. title()메서드로 제목도 출력할 수 있습니다.

plt.figure(figsize=(10,6))

Out[8]: <matplotlib.figure.Figure at 0x1e1170d9080>

plt.plot(t, y)

Out[9]: [<matplotlib.lines.Line2D at 0x1e117138cc0>]

plt.grid()

plt.xlabel('time')

Out[11]: Text(0.5,36.6122,'time')

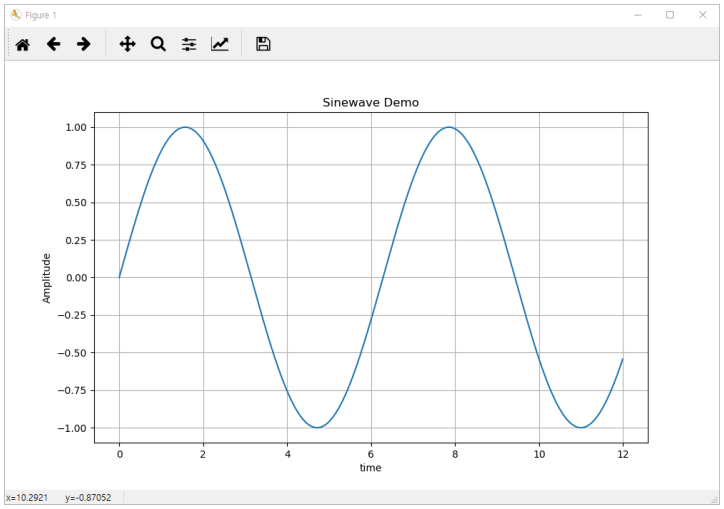
plt.ylabel('Amplitude')

Out[12]: Text(67.3472,0.5,'Amplitude')

plt.title('Sinewave Demo')

Out[13]: Text(0.5,1,'Sinewave Demo')

plt.show()



이번에는 plot()메서드를 2번 사용해서 한 차트에 그래프를 2개 그려봅니다. 범례(legend)를 추가할 때는 plt.legend()메서드를 사용하면 됩니다.

plt.figure(figsize=(10,6))

Out[15]: <matplotlib.figure.Figure at 0x1e117146470>

plt.plot(t, np.sin(t))

Out[16]: [<matplotlib.lines.Line2D at 0x1e1172cc358>]

plt.plot(t, np.cos(t))

Out[17]: [<matplotlib.lines.Line2D at 0x1e118716d30>]

plt.grid()

plt.xlabel('time')

Out[19]: Text(0.5,36.6122,'time')

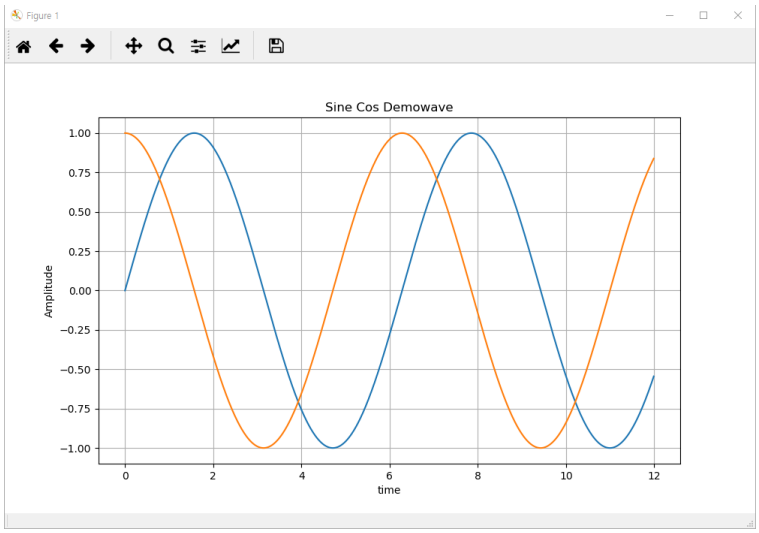
plt.ylabel('Amplitude')

Out[20]: Text(67.3472,0.5,'Amplitude')

plt.title('Sine Cos Demowave')

Out[21]: Text(0.5,1,'Sine Cos Demowave')

plt.show()



차트를 그릴 때 lw 옵션으로 선의 굵기를 지정할 수 있으며, color 옵션으로 색상을 지정할 수 있습니다.

plt.figure(figsize=(10,6))

Out[2]: <matplotlib.figure.Figure at 0x204b72083c8>

t = [0, 1, 2, 3, 4, 5, 6]

y = [10, 20, 5, 10, 20, 30, 25]

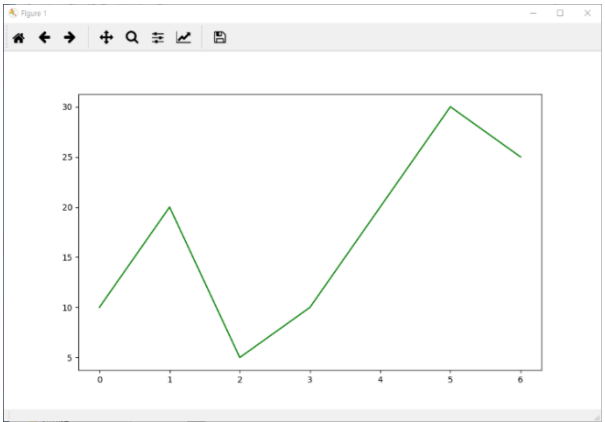
plt.figure(figsize=(10,6))

Out[5]: <matplotlib.figure.Figure at 0x204b737f048>

plt.plot(t, y, color='green')

Out[6]: [<matplotlib.lines.Line2D at 0x204b82b2a90>]

plt.show()

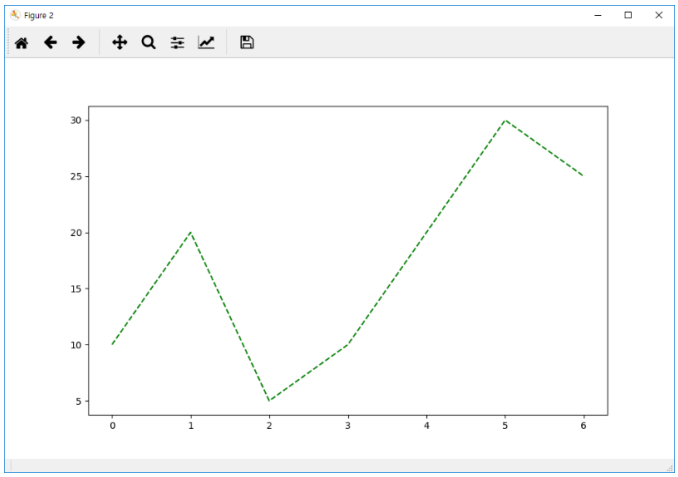


또는 linestyle='dashed'옵션으로 선 스타일을 지정할 수 있습니다.

plt.figure(figsize=(10,6))
Out[8]: <matplotlib.figure.Figure at 0x204b82bdda0>

plt.plot(t, y, color='green', linestyle='dashed')
Out[9]: [<matplotlib.lines.Line2D at 0x204b788c1d0>]

plt.show()



marker 옵션을 사용하면 데이터가 있는 곳에 마킹할 수 있습니다. 여기에 markerfacecolor 옵션과 markersizer 옵션으로 마커의 크기와 색상을 지정할 수 있습니다.

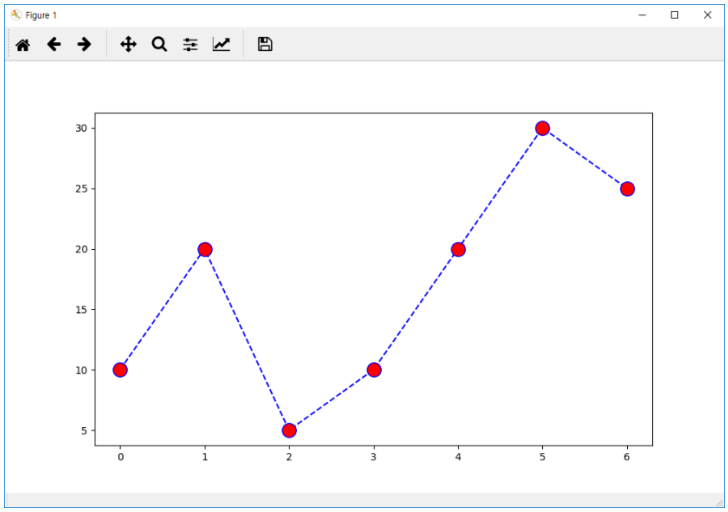
plt.figure(figsize=(10,6))

Out[11]: <matplotlib.figure.Figure at 0x204b7894710>

plt.plot(t, y, color='blue', linestyle='dashed', marker='o', markerfacecolor='red', markersize=14)

Out[12]: [<matplotlib.lines.Line2D at 0x204b78cfcc0>]

plt.show()



선을 그리는 plot()메서드 외에도 scatter()메서드를 사용하면 산포도(분포도)를 그릴 수 있습니다.

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

y = np.array([2,3,5,7,9,5,2,1,7,9])

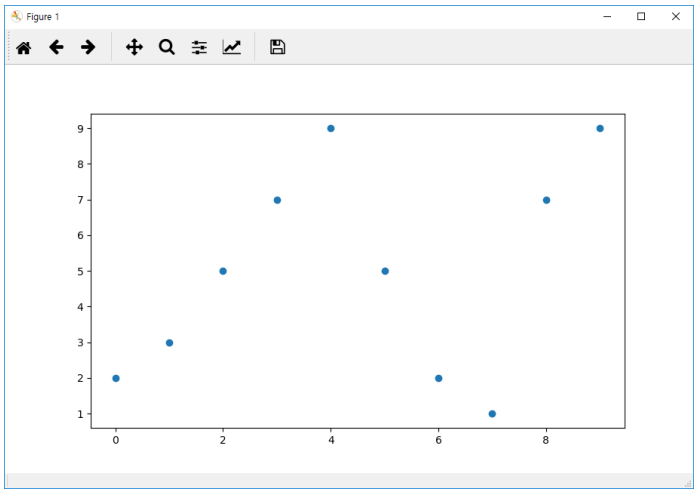
plt.figure(figsize=(10,6))

Out[17]: <matplotlib.figure.Figure at 0x204b78b57b8>

plt.scatter(t,y)

Out[18]: <matplotlib.collections.PathCollection at 0x204b7952470>

plt.show()



여기에 marker 를 '>'로 지정하면 삼각형 모양으로 출력할 수 있습니다.

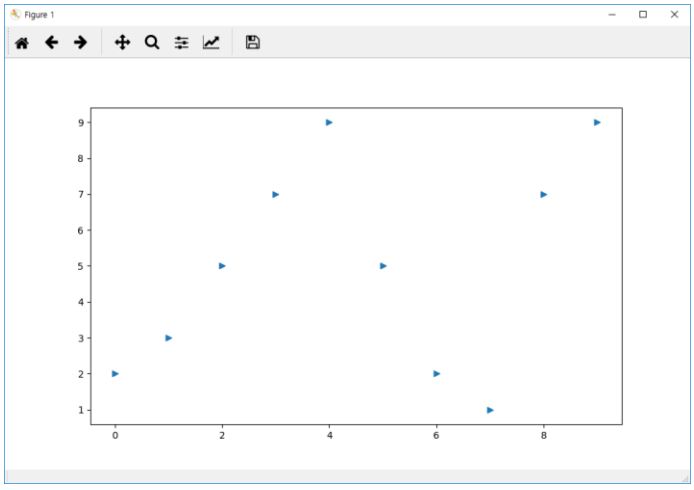
plt.figure(figsize=(10,6))

Out[20]: <matplotlib.figure.Figure at 0x204b79372b0>

plt.scatter(t, y, marker='>')

Out[21]: <matplotlib.collections.PathCollection at 0x204b7a29e80>

plt.show()



x 축의 값인 r 에 따라 색상을 바꾸는 color map 을 지정할 수 있습니다. 이때 s 옵션은 마커의 크기입니다.

colormap = t

plt.figure(figsize=(10,6))

Out[24]: <matplotlib.figure.Figure at 0x204bbe16438>

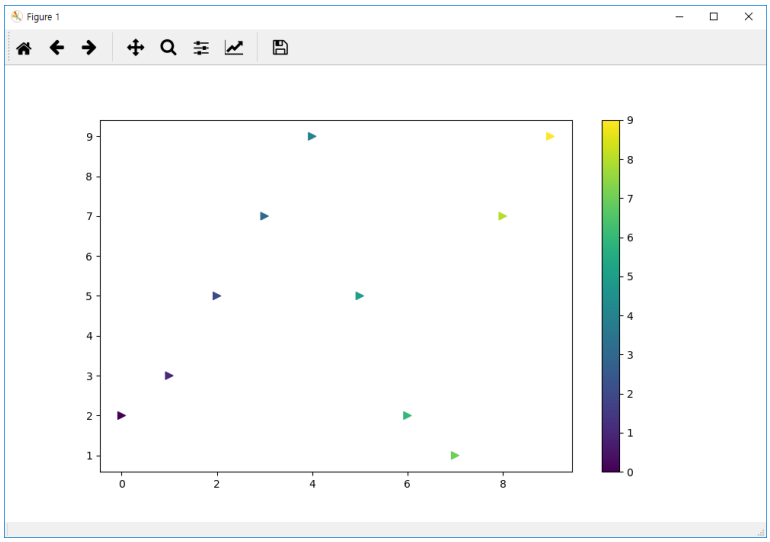
plt.scatter(t, y, s=50, c=colormap, marker='>')

Out[25]: <matplotlib.collections.PathCollection at 0x204b83254a8>

plt.colorbar()

Out[26]: <matplotlib.colorbar.Colorbar at 0x204b9247e48>

plt.show()



numpy 의 램덤변수 함수를 이용해서 데이터 세개를 생성합니다. 이때 loc 옵션으로 평균값과 scale 옵션으로 표준편차를 지정합니다.

s1 = np.random.normal(loc=0, scale=1, size=1000)

s2 = np.random.normal(loc=5, scale=0.5, size=1000)

s3 = np.random.normal(loc=10, scale=2, size=1000)

plt.figure(figsize=(10,6))

Out[31]: <matplotlib.figure.Figure at 0x204b8348be0>

plt.plot(s1, label='s1')

Out[32]: [<matplotlib.lines.Line2D at 0x204bb094cc0>]

plt.plot(s2, label='s2')

Out[33]: [<matplotlib.lines.Line2D at 0x204bbc83358>]

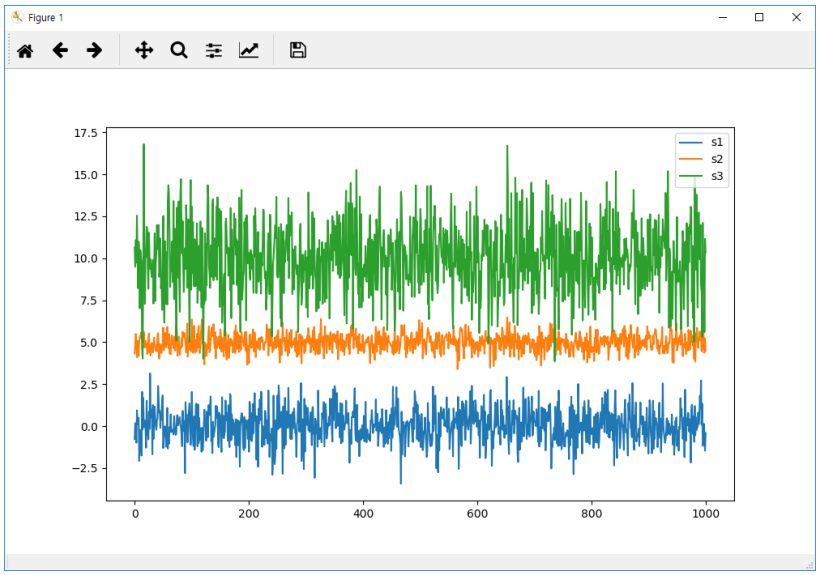
plt.plot(s3, label='s3')

Out[34]: [<matplotlib.lines.Line2D at 0x204bbc839b0>]

plt.legend()

Out[35]: <matplotlib.legend.Legend at 0x204bbca1ac8>

plt.show()



차트에 필요한 정보를 출력할 때 기본적으로 한글이 깨져서 출력됩니다. 이 부분을 수정하려면 그림판을 만들기 전에 폰트를 윈도우인 경우는 '맑은고딕' 또는 macOS 인 경우는 '애플고딕'으로 변경해 주면 됩니다. matplotlib 에서

사용하는 동적인 rc 셋팅은 영구적으로 local copy 로 저장해 둘 수 있습니다. 자세한 셋팅은 아래의 주소에서 확인해 볼 수 있습니다.

https://matplotlib.org/users/customizing.html

우리는 한글이 출력되도록 변경하려고 합니다. 윈도우 플랫폼인지 mac 플랫폼인지를 체크하기 위해서 platform 모듈을 임포트 합니다. matplotlib 에서 제공하는 font_manager, rc 를 임포트 합니다. platform.system()함수를 통해 'Darwin'이라는 문자열을 사용하면 macOS 로 판단하면 됩니다. 그렇지 않고 'Windows'면 윈도우입니다. 각 플랫폼의 폰트를 구해서 rc 의 'font'항목에 셋팅해 주면 됩니다.

```
import platform
from matplotlib import font_manager, rc

plt.rcParams['axes.unicode_minus'] = False

if platform.system() == 'Darwin':
    rc('font', family='AppleGothic')
elif platform.system() == 'Windows':
    path = 'c:/Windows/Fonts/malgun.ttf'
    font_name = font_manager.FontProperties(fname=path).get_name()
    rc('font', family=font_name)
else:
    print('Unknown system')
```

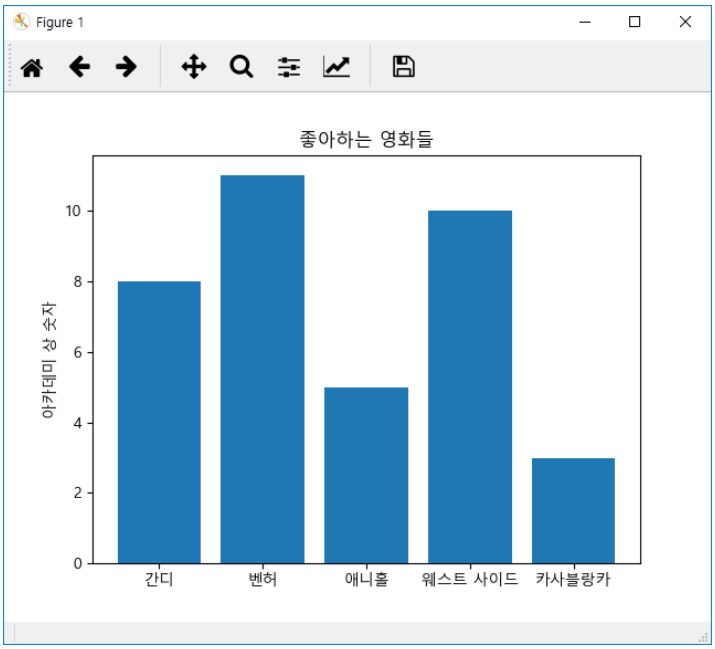
이번에는 한글도 정상적으로 출력되는지 확인해 봅니다. 영화 정보를 입력해서 막대 그래프로 출력해 봅니다. movies 에 리스트로 영화제목을 입력하고 num_oscars 에 오스카상을 받은 횟수를 리스트로 입력해 둡니다.

movies = ["애니홀", "벤허", "카사블랑카", "간디", "웨스트 사이드"]
num_oscars = [5, 11, 3, 8, 10]

plt.bar(movies, num_oscars)
Out[46]: <Container object of 5 artists>

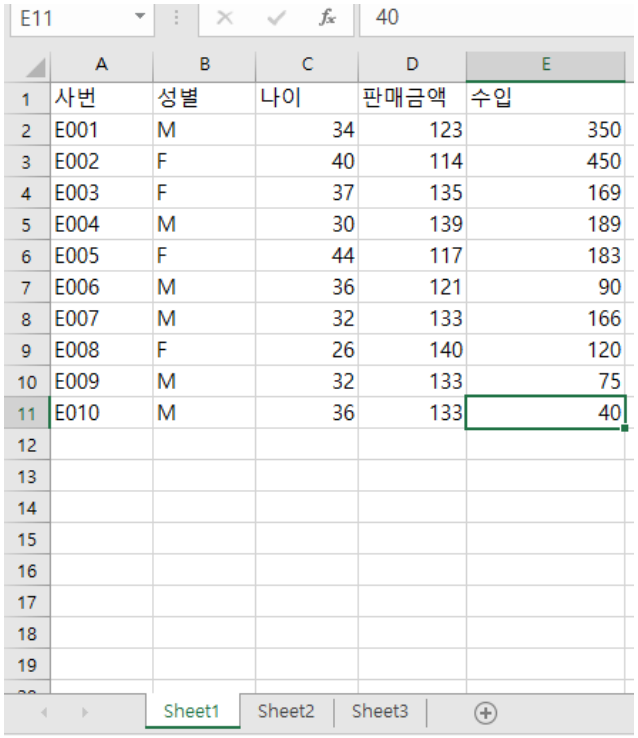
plt.ylabel("아카데미 상 숫자")
Out[48]: Text(49.4722,0.5,'아카데미 상 숫자')

plt.title("좋아하는 영화들")
Out[49]: Text(0.5,1,'좋아하는 영화들')



이번에는 엑셀 파일을 읽어들여서 해당 데이터를 집계를 하고 차트로 출력하는 경우를 실습으로 진행합니다. 아래와 같은 내용이 demo.xlsx 파일로 저장되어 있습니다. "Sheet1"로 된 시트에 저장된 내용입니다. 직원들의 사번, 성별, 나이, 판매금액, 수입이 저장되어 있는 엑셀 파일입니다. pandas 에서 제공하는

read_excel()함수를 통해 파일 이름과 시트 이름을 지정하면 읽어서 바로 DataFrame 객체로 리턴합니다. 히스토그램 형태로 출력할 때 hist()에 bins 를 7 로 지정하면 7 개의 수직 막대 그래프로 출력해 줍니다.



	A	B	C	D	E
1	사번	성별	나이	판매금액	수입
2	E001	M	34	123	350
3	E002	F	40	114	450
4	E003	F	37	135	169
5	E004	M	30	139	189
6	E005	F	44	117	183
7	E006	M	36	121	90
8	E007	M	32	133	166
9	E008	F	26	140	120
10	E009	M	32	133	75
11	E010	M	36	133	40

```
import pandas as pd

dfExcel = pd.read_excel('c:₩₩work₩₩demo.xlsx', 'Sheet1')

dfExcel
Out[85]:
```

	사번	성별	나이	판매금액	수입
0	E001	M	34	123	350
1	E002	F	40	114	450
2	E003	F	37	135	169
3	E004	M	30	139	189
4	E005	F	44	117	183
5	E006	M	36	121	90
6	E007	M	32	133	166
7	E008	F	26	140	120

8 E009 M 32 133 75
9 E010 M 36 133 40

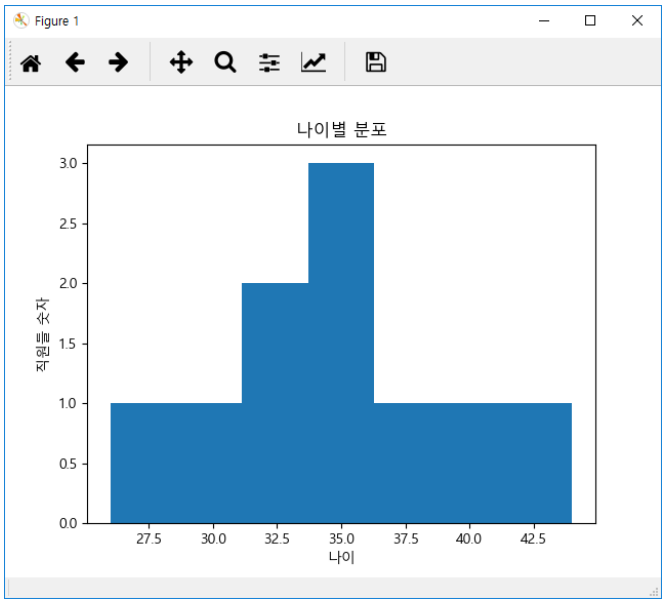
plt.figure(figsize=(10, 8))
Out[56]: <matplotlib.figure.Figure at 0x204befc8438>

plt.hist(dfExcel['나이'], bins=7)

plt.title('나이별 분포')
Out[61]: Text(0.5,1,'나이별 분포')

plt.xlabel('나이')
Out[62]: Text(0.5,24.4122,'나이')

plt.ylabel('직원들 숫자')
Out[63]: Text(46.3472,0.5,'직원들 숫자')

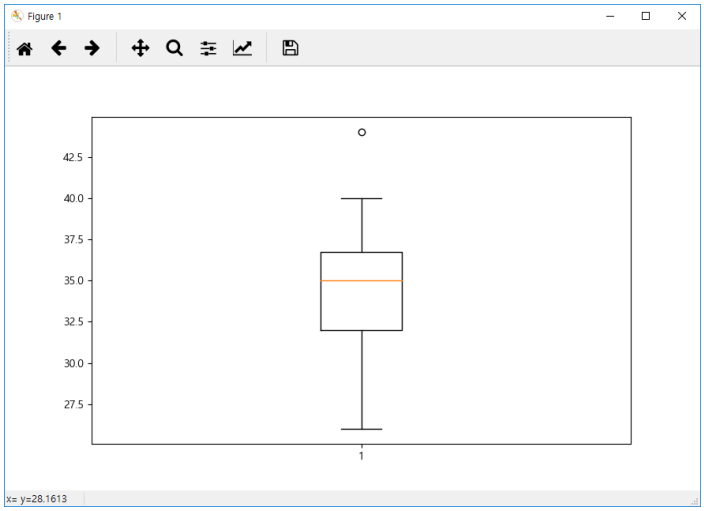


이번에는 동일한 데이터를 박스 차트로 출력해 봅니다. 나이가 밀집된 구간이 32 살에서 36 살임을 볼 수 있습니다.

plt.figure(figsize=(10,6))

Out[67]: <matplotlib.figure.Figure at 0x204bf275be0>

plt.boxplot(dfExcel['나이'])



이번에는 남녀별로 2 개의 그룹으로 묶고 각 판매금액의 합계를 구한 결과를 가지고 차트를 그려봅니다.

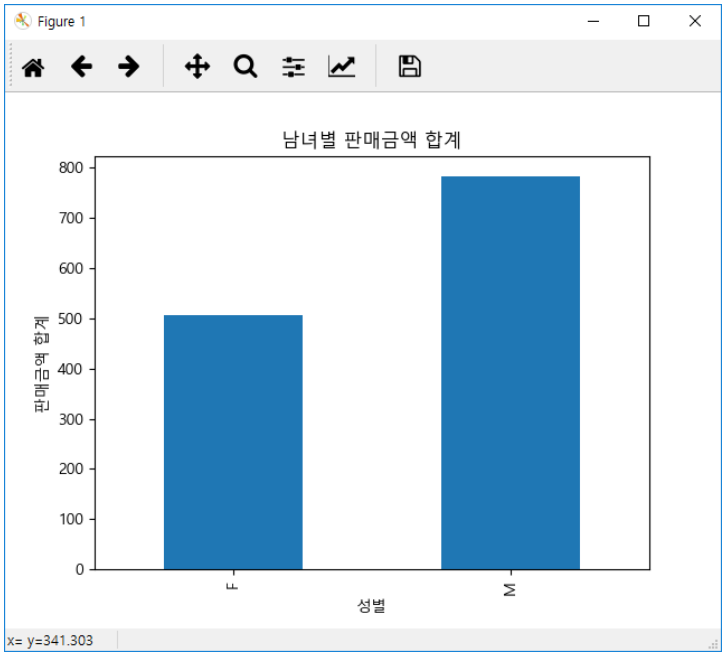
var = dfExcel.groupby('성별').판매금액.sum()
fig = plt.figure()
ax1 = fig.add_subplot(1, 1, 1)

ax1.set_xlabel('성별')
Out[91]: Text(0.5,24.4122,'성별')

ax1.set_ylabel('판매금액 합계')
Out[92]: Text(46.3472,0.5,'판매금액 합계')

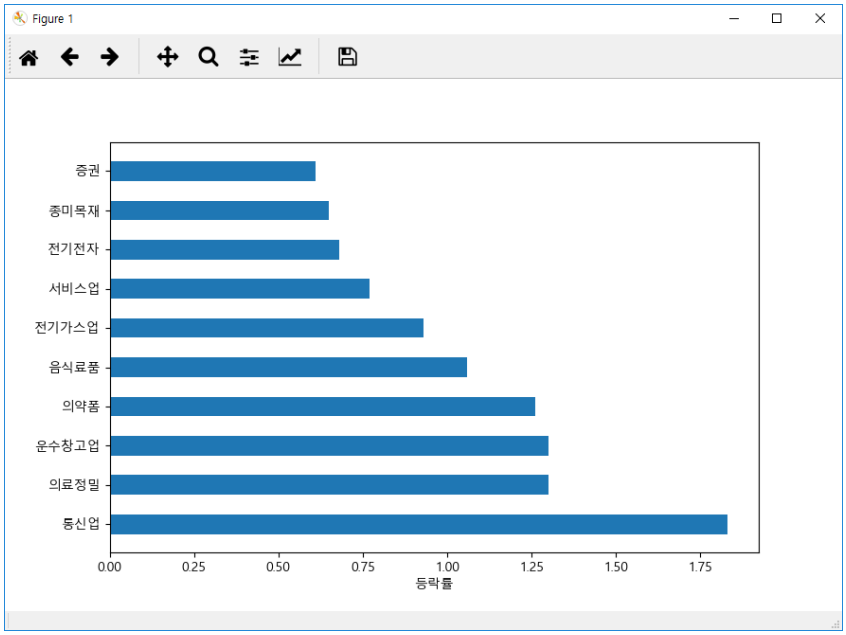
ax1.set_title('남녀별 판매금액 합계')
Out[93]: Text(0.5,1,'남녀별 판매금액 합계')

var.plot(kind='bar')
Out[94]: <matplotlib.axes._subplots.AxesSubplot at 0x204bfa36b70>



이번에는 수평으로 차트를 출력해 봅니다. 각 업종별로 주식의 등락률을 리스트로 담아서 출력합니다. plt.figure()로 그림판을 미리 만들고, 여기에 1 개의 차트를 추가합니다. plt.barh()로 지정하면 수평 막대 그래프로 출력할 수 있습니다.

```
industry = ['통신업', '의료정밀', '운수창고업', '의약품', '음식료품', '전기가스업',
'서비스업', '전기전자', '종미목재', '증권']
fluctuations = [1.83, 1.30, 1.30, 1.26, 1.06, 0.93, 0.77, 0.68, 0.65, 0.61]
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(1, 1, 1)
ypos = np.arange(10)

plt.barh(ypos, fluctuations, align='center', height=0.5)
Out[100]: <Container object of 10 artists>

plt.yticks(ypos, industry)
...
plt.xlabel('등락률')
Out[102]: Text(0.5,59.6122,'등락률')
```



pyplot인터페이스는 인터랙티브 사용에 맞추어 설계되었으며 xlim(), xticks(), xticklabels() 같은 메서드로 이루어져 있습니다. 이런 메서드를 가지고 표의 범위를 지정하거나 눈금의 위치, 눈금의 이름을 각각 조절할 수 있습니다.
아무런 인자도 없이 호출하면 현재 설정되어 있는 매개변수의 값을 반환합니다. plt.xlim()메서드는 현재 x축의 범위를 반환합니다. 축을 꾸미는 방법을 설명하기 위해 무작위 값으로 간단한 그래프를 하나 생성해 봅니다.
x축의 눈금을 변경하기 가장 쉬운 방법은 set_xticks()와 set_xticklabels()메서드를 사용하는 것입니다. set_xticks()메서드는 전체 데이터 범위에 맞춰 눈금을 어디에 배치할지 지정합니다. 기본적으로 이 위치에 눈금 이름이 들어갑니다. 하지만 다른 눈금 이름을 지정하고 싶다면 set_xticklabels()를 사용해도 됩니다.
범례를 추가하는 몇가지 방법이 있는데 가장 쉬운 방법은 각 그래프에 label인자를 넘기는 것입니다. loc인자는 범례를 그래프에서 어디에 위치시킬지 지정해주는 인자입니다. 최대한 방해가 되지 않는 곳에 두는 'best'옵션만으로도 충분합니다. 다른 옵션으로는 'ceter right', 'upper left', 'upper right', 'lower right'등을 사용할 수 있습니다.

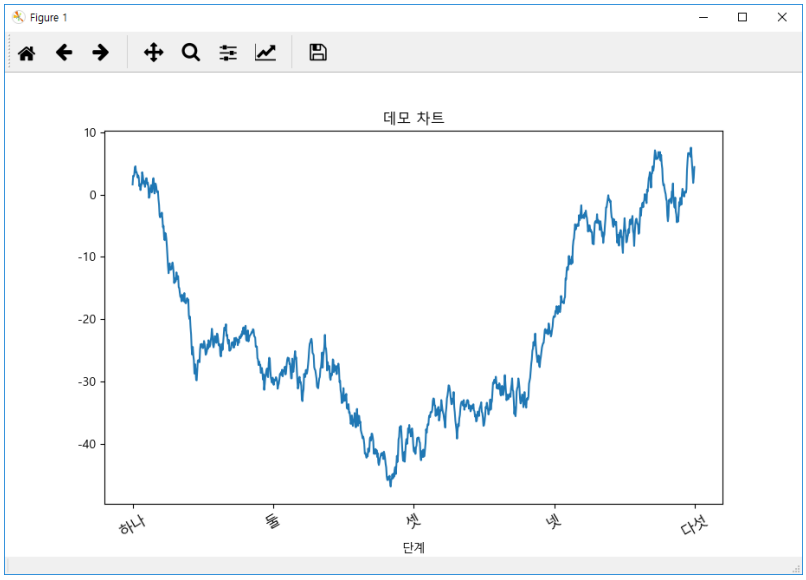
```
fig = plt.figure(figsize(10,6))
ax = fig.add_subplot(1, 1, 1)
ax.plot(randn(1000).cumsum())
```

Out[109]: [<matplotlib.lines.Line2D at 0x204c052c208>]

ticks = ax.set_xticks([0, 250, 500, 750, 1000])
labels = ax.set_xticklabels(['하나', '둘', '셋', '넷', '다섯'], rotation=30, fontsize='large')

ax.set_title('데모 차트')
Out[107]: Text(0.5,1,'데모 차트')

ax.set_xlabel('단계')
Out[108]: Text(0.5,19.2648,'단계')



데이터 집합을 분류하고 각 그룹별로 집계나 변형 같은 어떤 함수를 적용하는 것은 데이터 분석 과정에서 매우 중요한 일입니다. 데이터를 불러오고 취합해서 하나의 데이터 집합을 준비하고 나면 그룹 통계를 구하거나 가능하다면 피벗 테이블을 구해서 보고서를 만들거나 시각화하게 됩니다. pandas는 데이터 집합을 자연스럽게 나누고 요약할 수 있는 groupby라는 유연한 방법을 제공합니다. 먼저 다음과 같은 DataFrame으로 표현되는 간단한 표 형식의 데이터가 있습니다.

from pandas import Series, DataFrame

df = DataFrame({'key1':['a','a','b','b','a'], 'key2':['one','two','one','two','one'], 'data1':np.random.randn(5), 'data2':np.random.randn(5)})

df

Out[113]:

	data1	data2	key1	key2
0	-0.269427	0.152798	a	one
1	-0.089356	0.134447	a	two
2	-0.228561	0.324110	b	one
3	1.278906	-0.082617	b	two
4	0.086400	0.001231	a	one

이 데이터를 key1으로 묶고 각 그룹에서 data1에 대한 집계된 결과를 구합니다. 여러가지 방법이 있지만 그중 하나는 data1에 대해 groupby 메서드를 호출하고 그룹핑하는 기준이 되는 컬럼으로 key1칼럼을 넘깁니다.

grouped = df['data1'].groupby(df['key1'])

grouped

Out[115]: <pandas.core.groupby.SeriesGroupBy object at 0x00000204C0D58C88>

이 grouped변수는 groupby객체입니다. 이 객체는 그룹 연산을 위해 필요한 모든 정보를 가지고 있기 때문에 각 그룹에 어떤 연산을 적용할 수 있게 해줍니다. 평균값을 구하려면 mean()메서드를 사용하면 됩니다. 합계를 구하는 경우는 sum() 메서드를 호출하고, 그룹으로 묶은 개수를 알고 싶은 경우라면 count()메서드를 사용합니다. SQL에서 일반적으로 사용하는 집계함수들과 이름이 비슷하게 되어 있습니다. SQL구문을 사용하지 않고도 데이터를 분석할 때 사용할 수 있는 고마운 메서드들 입니다.

grouped.mean()

Out[116]:

key1

a -0.090794

b 0.525173
Name: data1, dtype: float64

grouped.sum()
Out[117]:
key1
a -0.272383
b 1.050345
Name: data1, dtype: float64

grouped.count()
Out[118]:
key1
a 3
b 2
Name: data1, dtype: int64

만약 여러 개의 컬럼을 리스트로 넘겼다면 좀 더 다른 결과를 얻을 수 있습니다. 예를 들면 groupby()메서드에 리스트로 [df['key1'], df['key2']]로 넘길 수 있습니다.

means = df['data1'].groupby([df['key1'], df['key2']]).mean()
means
Out[120]:
key1 key2
a one -0.091513
two -0.089356
b one -0.228561
two 1.278906

이번에는 gapminder파일을 사용해서 데이터셋을 사용하는 방법을 연습해 봅니다. 제공된 파일을 아래와 같이 로딩합니다.

In [1]:
```
import pandas as pd

df = pd.read_csv("c:₩₩work₩₩gapminder.tsv", sep="₩t")
```

In [2]:
```
df.head()
```

Out[2]:

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106

파이썬의 내장 함수 type()으로 형식을 확인하거나 df.shape로 행과 열의 크기를 볼 수 있습니다. 또는 컬럼 이름들을 확인하는 경우에 df.columns와 같이 실행하면 됩니다.

In [3]:
```
#형식 확인
type(df)
```

Out[3]:
```
pandas.core.frame.DataFrame
```

In [4]:
```
df.shape
```

Out[4]:
```
(1704, 6)
```

In [5]:
```
#컬럼 이름 나열
df.columns
```

Out[5]:
```
Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'], dtype='objec
t')
```

열단위로 데이터를 추출할 경우 아래와 같이 실행합니다.

```
In [6]: #열단위로 추출
country_df = df["country"]
type(country_df)
```

```
Out[6]: pandas.core.series.Series
```

```
In [7]: country_df.head()
```

```
Out[7]: 0    Afghanistan
1    Afghanistan
2    Afghanistan
3    Afghanistan
4    Afghanistan
Name: country, dtype: object
```

리스트에 열 이름을 전달하면 여러 개의 열을 추출할 수 있습니다. (continent는 대륙이라는 뜻 20/3/11)

```
In [8]: #리스트에 열 이름을 전달하면 여러 개의 열을 추출할 수 있다.
subset = df[["country","continent","year"]]
type(subset)
```

```
Out[8]: pandas.core.frame.DataFrame
```

```
In [9]: subset.head()
```

Out[9]:

	country	continent	year
0	Afghanistan	Asia	1952
1	Afghanistan	Asia	1957
2	Afghanistan	Asia	1962
3	Afghanistan	Asia	1967
4	Afghanistan	Asia	1972

행 단위로 데이터를 추출하는 경우에는 loc속성과 iloc속성을 사용할 수 있습니다. loc속성은 인덱스를 기준으로 행 데이터를 추출할 수 있습니다.

```
In [10]: #loc속성으로 행 데이터 추출하기
         df.loc[0]
```

```
Out[10]: country      Afghanistan
         continent          Asia
         year               1952
         lifeExp          28.801
         pop             8425333
         gdpPercap        779.445
         Name: 0, dtype: object
```

```
In [11]: df.loc[99]
```

```
Out[11]: country      Bangladesh
         continent          Asia
         year               1967
         lifeExp          43.453
         pop             62821884
         gdpPercap        721.186
         Name: 99, dtype: object
```

한번에 여러 행의 데이터를 추출하려면 리스트에 원하는 인덱스를 담아서 전달하면 됩니다.

```
In [13]: #인덱스가 1, 100, 200을 한번에 추출하려면 리스트에 원하는 인덱스를 담아 전달하면 된다.
         df.loc[[1,100,200]]
```

Out[13]:

	country	continent	year	lifeExp	pop	gdpPercap
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
100	Bangladesh	Asia	1972	45.252	70759295	630.233627
200	Burkina Faso	Africa	1992	50.260	8878303	931.752773

이번에는 iloc속성으로 행 데이터를 추출해 봅니다. loc속성은 데이터프레임의 인덱스를 사용하여 데이터를 추출했지만 iloc속성은 데이터 순서를 의미하는 행 번호를 사용하여 데이터를 추출합니다. 지금은 인덱스와 행 번호가 동일하여 동일한 결과값이 출력됩니다.

In [14]:
```
#이번에는 iloc속성으로 행 데이터를 추출한다. loc속성은 데이터프레임의 인덱스를 사용하여
#데이터를 추출했지만 iloc속성은 데이터 순서를 의미하는 행 번호를 사용하여
#데이터를 추출한다.
#지금은 인덱스와 행 번호가 동일하여 동일한 결과값이 출력된다.
df.iloc[1]
```

Out[14]:
```
country      Afghanistan
continent           Asia
year                1957
lifeExp           30.332
pop              9240934
gdpPercap        820.853
Name: 1, dtype: object
```

iloc속성은 음수를 사용해도 데이터를 추출할 수 있습니다. iloc속성도 여러 데이터를 한번에 추출할 수 있습니다.

In [16]:
```
#iloc속성은 음수를 사용해도 데이터를 추출할 수 있다.
df.iloc[-1]
```

Out[16]:
```
country      Zimbabwe
continent      Africa
year             2007
lifeExp         43.487
pop           12311143
gdpPercap      469.709
Name: 1703, dtype: object
```

In [18]:
```
# iloc속성도 여러 데이터를 한번에 추출할 수 있다.
df.iloc[[1,100,200]]
```

Out[18]:

	country	continent	year	lifeExp	pop	gdpPercap
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
100	Bangladesh	Asia	1972	45.252	70759295	630.233627
200	Burkina Faso	Africa	1992	50.260	8878303	931.752773

loc, iloc속성을 좀 더 자유롭게 사용하려면 추출할 데이터의 행과 열을 지정하는 방법을 알아야 합니다. Df.loc[[행],[열]]이나 df.iloc[[행],[열]]과 같이 지정하면 됩니다. 다음은 모든 행(:)의 데이터에 대해 year, pop열을 추출하는 방법입니다.

In [20]:
```
#슬라이싱 구문으로 데이터 추출하기
#다음은 모든 행(:)의 데이터에 대해 year, pop열을 추출하는 방법입니다.
subset = df.loc[:, ["year","pop"]]
subset.head()
```

Out[20]:

	year	pop
0	1952	8425333
1	1957	9240934
2	1962	10267083
3	1967	11537966
4	1972	13079460

수업에서 배운 range()함수를 사용하면 해당 열에 해당하는 데이터를 추출할 수 있습니다. 전체 데이터에 대해 5개의 컬럼을 추출해서 사용할 수 있습니다.

In [22]:
```
#range메서드로 데이터 추출하기
small_range = list(range(5))
df.iloc[:, small_range]
```

Out[22]:

	country	continent	year	lifeExp	pop
0	Afghanistan	Asia	1952	28.801	8425333
1	Afghanistan	Asia	1957	30.332	9240934
2	Afghanistan	Asia	1962	31.997	10267083
3	Afghanistan	Asia	1967	34.020	11537966
4	Afghanistan	Asia	1972	36.088	13079460
...	...	...	...	...	...
1699	Zimbabwe	Africa	1987	62.351	9216418
1700	Zimbabwe	Africa	1992	60.377	10704340
1701	Zimbabwe	Africa	1997	46.809	11404948
1702	Zimbabwe	Africa	2002	39.989	11926563
1703	Zimbabwe	Africa	2007	43.487	12311143

1704 rows × 5 columns

아래와 같이 작성하면 특정행과 특정 컬럼을 지정해서 자유롭게 데이터를 추출할

수 있습니다. 리스트 형태로 1,100,200번째 행을 지정하고 컬럼으로 "country", "lifeExp", "gdpPercap"을 지정해서 슬라이싱을 한 결과입니다.

```
In [29]: #iloc속성의 열 지정값으로 정수 리스트를 전달하는 것이 간편해 보이지만 컬럼이름을 직접
         #부여하는 것이 사용하기에는 더 편리하다.
         df.loc[[1,100,200], ["country","lifeExp","gdpPercap"]]
```

Out[29]:

	country	lifeExp	gdpPercap
1	Afghanistan	30.332	820.853030
100	Bangladesh	45.252	630.233627
200	Burkina Faso	50.260	931.752773

기본적인 통계는 아래와 같이 작성할 수 있습니다. 연도별 lifeExp열의 평균을 계산합니다. 데이터를 year열로 그룹화하고
lifeExp열의 평균을 구하면 됩니다.

```
In [30]: #기본적은 통계 사용하기
         #연도별 lifeExp열의 평균을 계산한다. 데이터를 year열로 그룹화하고
         #lifeExp열의 평균을 구하면 된다.
         df.groupby("year")["lifeExp"].mean()
```

```
Out[30]: year
         1952    49.057620
         1957    51.507401
         1962    53.609249
         1967    55.678290
         1972    57.647386
         1977    59.570157
         1982    61.533197
         1987    63.212613
         1992    64.160338
         1997    65.014676
         2002    65.694923
         2007    67.007423
         Name: lifeExp, dtype: float64
```

lifeExp, gdpPercap열의 평균값을 연도, 지역별로 그룹화하여 한번에 계산할 수 있다. 아래와 같이 ["year","continent"]을 묶어서 그룹에 대한 기준을 지정하고, 두번째 리스트로 ["lifeExp","gdpPercap"]을 지정해서 mean()메서드로 평균을 구하면 됩니다.

In [31]:
```
#lifeExp, gdpPercap열의 평균값을 연도, 지역별로 그룹화하여 한번에 계산하기
multi_group_var = df.groupby(["year","continent"])[["lifeExp","gdpPercap"]].mean()
multi_group_var
```

Out[31]:

		lifeExp	gdpPercap
year	continent		
1952	Africa	39.135500	1252.572466
	Americas	53.279840	4079.062552
	Asia	46.314394	5195.484004
	Europe	64.408500	5661.057435
	Oceania	69.255000	10298.085650
1957	Africa	41.266346	1385.236062
	Americas	55.960280	4616.043733
	Asia	49.318544	5787.732940
	Europe	66.703067	6963.012816
	Oceania	70.295000	11598.522455
	Africa	43.319442	1598.078825

이번에는 앞에서 보았던 갭마인더 파일을 통해 챠트를 그려봅니다.

%matplotlib inline

import matplotlib.pyplot as plt

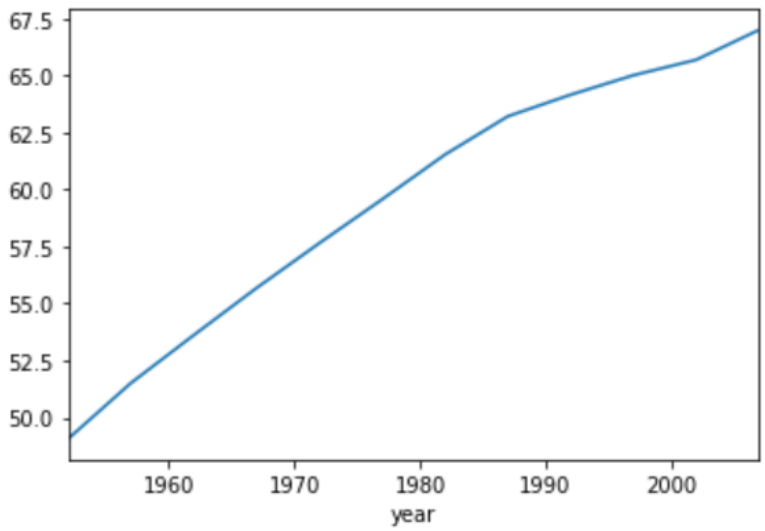
year열을 기준으로 그룹화한 데이터프레임에서 lifeExp열만 추출해서 평균값을 구한 것으로 차트를 그립니다.

```
In [33]: #year열을 기준으로 그룹화한 데이터프레임에서 lifeExp열만 추출하여 평균값을 구한다.
global_yearly_life_expectancy = df.groupby("year")["lifeExp"].mean()
global_yearly_life_expectancy
```

```
Out[33]: year
1952    49.057620
1957    51.507401
1962    53.609249
1967    55.678290
1972    57.647386
1977    59.570157
1982    61.533197
1987    63.212613
1992    64.160338
1997    65.014676
2002    65.694923
2007    67.007423
Name: lifeExp, dtype: float64
```

```
In [36]: global_yearly_life_expectancy.plot()
```

```
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x211645cdb08>
```



데이터 시각화를 보여주는 전형적인 사례로 앤스콤 4분할 그래프(Anscombe's quartet)가 있습니다. 이 그래프는 영국의 프랭크 앤스콤이 데이터를 시각화하지 않고 수치만 확인할 때 발생할 수 있는 함정을 보여주기 위해 만든 그래프입니다. 앤스콤 4분할 그래프를 구성하는 데이터 집합은 4개의 그룹으로 구성되어 있으며 모든 데이터 그룹은 x,y열을 가지고 있습니다. 그런데 이 4개의 데이터 그룹은 각각 평균, 분산과 같은 수치값이나 상관관계, 회귀선이 같다는 특징이 있습니다. 그래서 이런 결과만 보고 "데이터 그룹 I,II,III,IV의 데이터는 모두 같을 것이다"라고 착각할 수 있습니다. 바로 이것이 앤스콤이 지적한 "함정"입니다. 하지만 각

데이터 그룹을 시각화하면 데이터 그룹이 서로 다른 데이터 패턴을 가지고 있다는 점을 금방 알 수 있습니다.
앤스콤 데이터 집합은 seaborn라이브러리에 포함되어 있습니다. seaborn라이브러리의 load_datase메서드에 문자열 anscombe를 전달하면 앤스콤 데이터 집합을 불러올 수 있습니다.

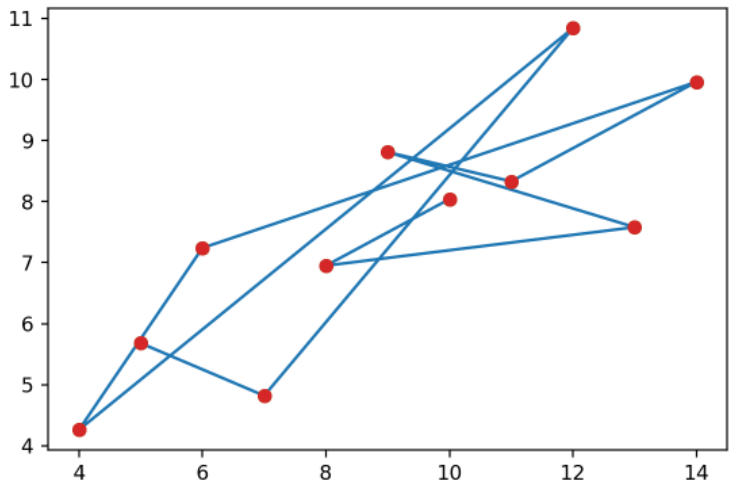
In [37]:
```
#앤스콤 데이터 집합으로 그리기
import seaborn as sns

anscombe = sns.load_dataset("anscombe")
anscombe
```

Out[37]:

	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33
5	I	14.0	9.96
6	I	6.0	7.24
7	I	4.0	4.26

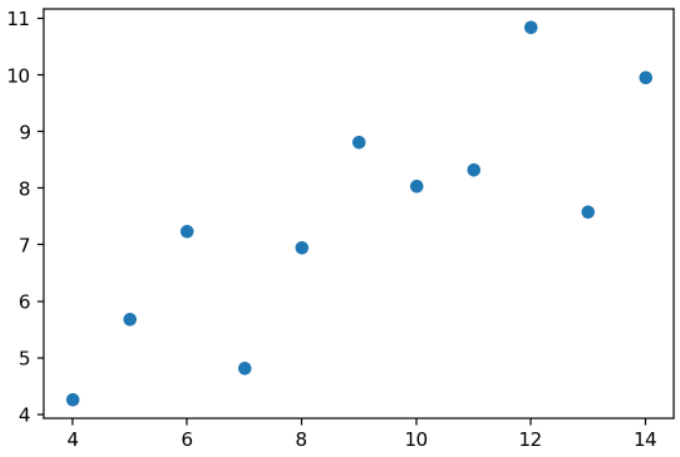
다음은 anscombe데이터프레임의 dataset열에서 데이터 값이 I인 것만 추출한 것입니다. 즉, 첫번째 데이터그룹을 추출한 것입니다. 선 그래프는 plot메서드로 그릴 수 있습니다.

```
In [46]: %matplotlib notebook
         import matplotlib.pyplot as plt

In [42]: #anscombe데이터프레임의 dataset열에서 데이터 값이 I인 것만 추출함
         #즉, 첫번째 데이터그룹을 추출한 것이다.
         dataset_1 = anscombe[anscombe["dataset"] == 'I']
         plt.plot(dataset_1["x"], dataset_1["y"])
```



만약 점으로 그래프를 그리려면 "o"를 세번째 인자로 전달하면 됩니다.

```
In [47]: #plot메서드는 기본적으로 선으로 그래프를 그린다. 만약 점으로 그래프를 그리려면 o를
         #세번째 인자로 전달하면 된다.
         plt.plot(dataset_1["x"], dataset_1["y"], "o")
```



앤스콤 데이터 집합은 4개의 데이터 그룹으로 구성되어 있으며 각 데이터 그룹의

차이를 파악하려면 그래프로 시각화해야 한다고 했습니다. 이번에는 모든 데이터 그룹에 대하여 그래프를 그려봅니다.

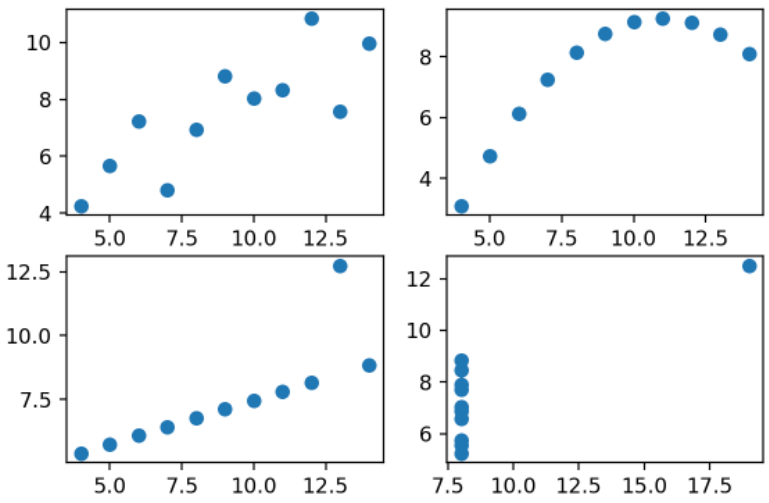
In [49]:
```
#앤스콤 데이터프레임의 dataset열의 값이 I,II,III,IV인 것을 불린 추출하여 저장한다.
dataset_2 = anscombe[anscombe["dataset"] == "II"]
dataset_3 = anscombe[anscombe["dataset"] == "III"]
dataset_4 = anscombe[anscombe["dataset"] == "IV"]
```

In [50]:
```
#먼저 그래프 격자가 위치할 기본 틀을 만든다.
fig = plt.figure()
```

In [51]:
```
#그래프 격자를 그린다. 2행 2열의 1번, 2번, 3번, 4번을 추가한다.
axes1 = fig.add_subplot(2,2,1)
axes2 = fig.add_subplot(2,2,2)
axes3 = fig.add_subplot(2,2,3)
axes4 = fig.add_subplot(2,2,4)
```

In [52]:
```
#이제 plot메서드에 데이터를 전달하여 그래프를 그린다.
axes1.plot(dataset_1["x"], dataset_1["y"], "o")
axes2.plot(dataset_2["x"], dataset_2["y"], "o")
axes3.plot(dataset_3["x"], dataset_3["y"], "o")
axes4.plot(dataset_4["x"], dataset_4["y"], "o")
```

Out[52]:
```
[<matplotlib.lines.Line2D at 0x211685856c8>]
```



기초 그래프 그리기

seaborn라이브러리에는 tips라는 데이터 집합이 있습니다. tips데이터집합은 어떤 식당에서 팁을 지불한 손님의 정보를 모아둔 것입니다. tips데이터집합을 불러와 변수 tips에 저장합니다. tips는 지불금액, 팁, 성별, 흡연 유무, 시간, 전체 인원을 담고 있습니다.

In [58]:
```
#tips데이터집합을 불러와 변수 tips에 저장한다.
#tips는 지불금액, 팁, 성별, 흡연 유무, 시간, 전체 인원을 담고 있다.
tips = sns.load_dataset("tips")
tips.head()
```

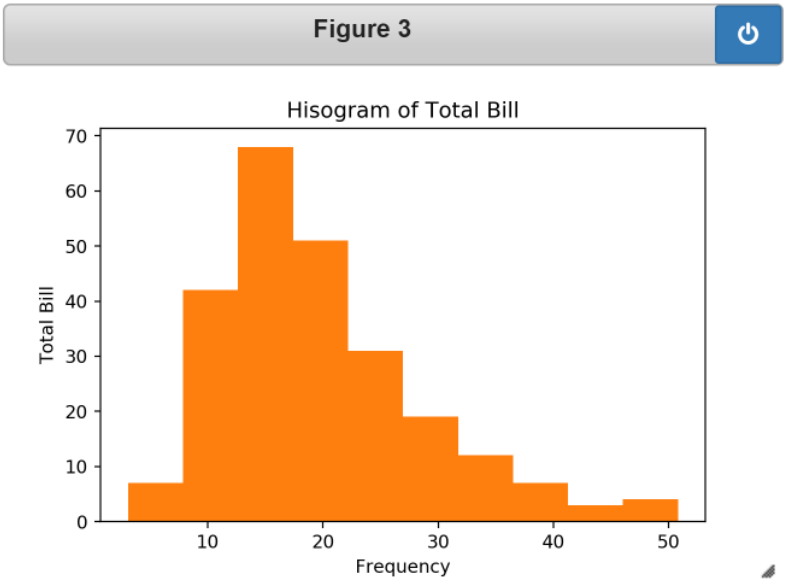
Out[58]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

히스토그램은 데이터프레임의 열 데이터 분포와 빈도를 살펴보는 용도로 자주 사용합니다.
이때 데이터프레임의 total_bill, tip등의 열을 변수라고 부르기도 한다.
그리고 변수를 하나만 사용해서 그린 그래프를 "일변량 그래프"라고 한다.

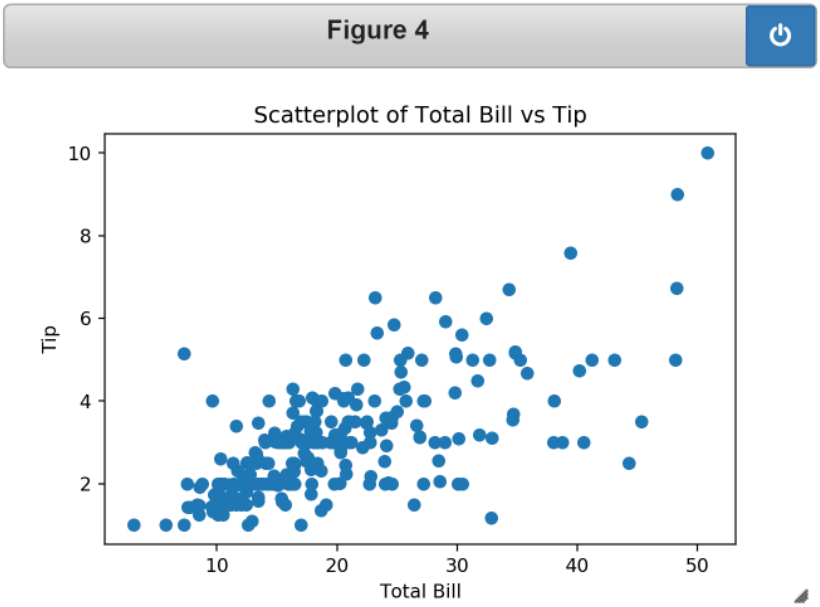
In [59]:
```
#히스토그램은 데이터프레임의 열 데이터 분포와 빈도를 살펴보는 용도로 자주 사용한다.
#이때 데이터프레임의 total_bill, tip등의 열을 변수라고 부르기도 한다.
#그리고 변수를 하나만 사용해서 그린 그래프를 "일변량 그래프"라고 한다.
fig = plt.figure()
axes1 = fig.add_subplot(1,1,1)
```

In [61]:
```
#hist메서드에 total_bill열을 전달하면 히스토그램이 만들어진다.
#이때 x축의 간격은 bins인자값으로 조정할 수 있다. bins인자값을 10을 지정하면
#x축의 간격을 10으로 조정할 수 있다.
#나머지는 그래프의 제목과 x,y축의 제목을 추가하는 코드이다.
axes1.hist(tips["total_bill"], bins=10)
axes1.set_title("Hisogram of Total Bill")
axes1.set_xlabel("Frequency")
axes1.set_ylabel("Total Bill")
fig
```



이번에는 산점도 그래프를 그려봅니다. 산점도 그래프는 변수 2개를 사용해서 만드는 그래프이며 변수 2개를 사용하기 때문에 통계 용어로 "이변량 그래프"라고 부릅니다. 다음은 total_bill열에 따른 tip열의 분포를 나타낸 산점도 그래프입니다.

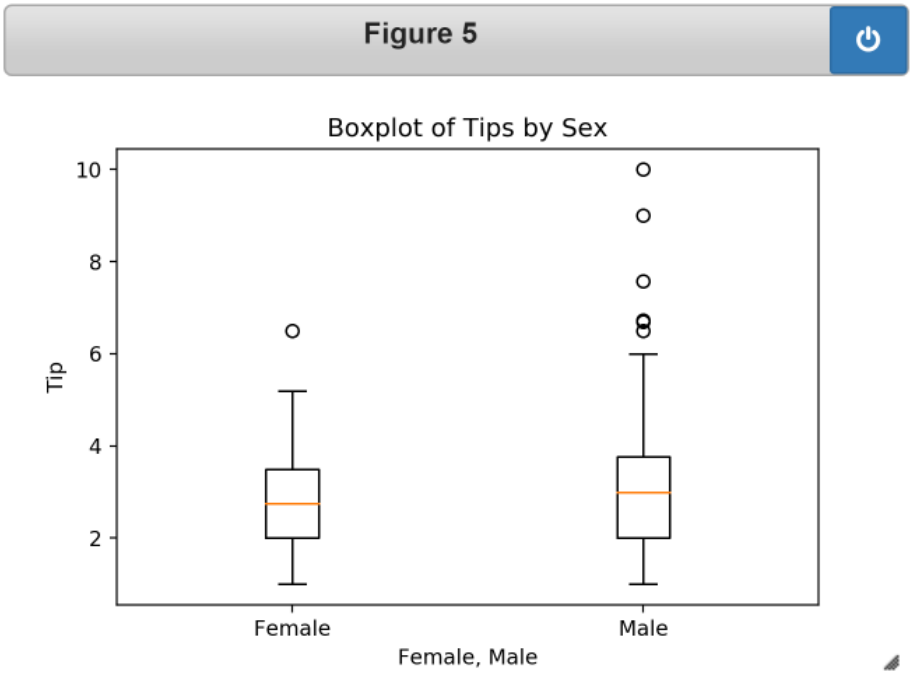
In [62]:
```
#이번에는 산점도 그래프를 그려본다.
scatter_plot = plt.figure()
axes1 = scatter_plot.add_subplot(1,1,1)
axes1.scatter(tips["total_bill"], tips["tip"])
axes1.set_title("Scatterplot of Total Bill vs Tip")
axes1.set_xlabel("Total Bill")
axes1.set_ylabel("Tip")
fig
```



박스그래프를 그려본다. 박스 그래프는 이산형 변수와 연속형 변수를 함께 사용하는 그래프입니다. 이산형 변수란 Female, Male과 같이 명확하게 구분되는 것을 의미하고, 연속형 변수란 Tip과 같이 명확하게 셀 수 없는 범위의 값을 의미합니다. boxplot메서드를 사용하면 박스 그래프를 그릴 수 있습니다.

첫번째 인자는 tips데이터프레임에서 성별이 Female인 데이터와 Male인 데이터에서 tip열 데이터만 추출하여 리스트에 담아 전달한 것입니다. 두번째 인자는 labels인자값으로 성별을 구분하기 위한 이름을 추가합니다.

```
boxplot = plt.figure()
axes1 = boxplot.add_subplot(1,1,1)
axes1.boxplot([tips[tips["sex"] == "Female"]["tip"],
               tips[tips["sex"] == "Male"]["tip"]],
               labels=["Female","Male"])
axes1.set_xlabel("Female, Male")
axes1.set_ylabel("Tip")
axes1.set_title("Boxplot of Tips by Sex")
```



다변량 그래프 그리기

앞에서는 지불 금액(total_bill)과 팁(tip)만을 사용하여 산점도 그래프를 그렸습니다. 만약 여기에 성별을 추가하여 산점도 그래프를 표현하려면 어떻게 해야 할까요? 점의 색상을 다르게 하면 됩니다. 만약 식사 비용을 추가한다면 점의 크기를 다르게 하는 방법으로 산점도 그래프를 표현할 수 있습니다. 이렇게 3개 이상의 변수를 사용하는 다변량 그래프는 적재적소에 맞는 그래프 요소를 추가하여 그래프를 표현하야 합니다.

아래는 문자열을 0과 1로 변환하는 함수입니다.

```
def recode_sex(sex):
    if sex == "Female":
        return 0
    else:
        return 1
```

이제 recode_sex메서드가 반환한 값을 데이터프레임에 추가하면 됩니다. 이때 sex 열에 recode_sex함수를 브로드캐스팅하기 위하여 apply메서드를 사용했습니다.

```
tips["sex_color"] = tips["sex"].apply(recode_sex)
```

그래프를 그리는 김에 테이블당 인원 수(size)도 산점도 그래프에 추가해 봅니다. 테이블당 인원 수는 점의 크기로 표현하면 적당합니다. 다음은 scatter메서드에 s,c인자값으로 테이블당 인원 수와 성별의 치환값을 전달한 것입니다. s는 점의 크기(size)를, c는 점의 색상(color)를 의미합니다.
alpha인자값을 0.5로 지정하여 점의 투명도를 조절했습니다.

```
scatter_plot = plt.figure()
axes1 = scatter_plot.add_subplot(1,1,1)
axes1.scatter(
    x = tips["total_bill"],
    y = tips["tip"],
    s = tips["size"] * 10,
    c = tips["sex_color"],
    alpha = 0.5)
axes1.set_title("Total Bill vs Tip Colored by Sex and Sized by Size")
axes1.set_xlabel("Total Bill")
axes1.set_ylabel("TIp")
```