

파이썬(을 이용한 데이터 분석) 기초

디지털혁신실 이현창

2021.11

주요 내용

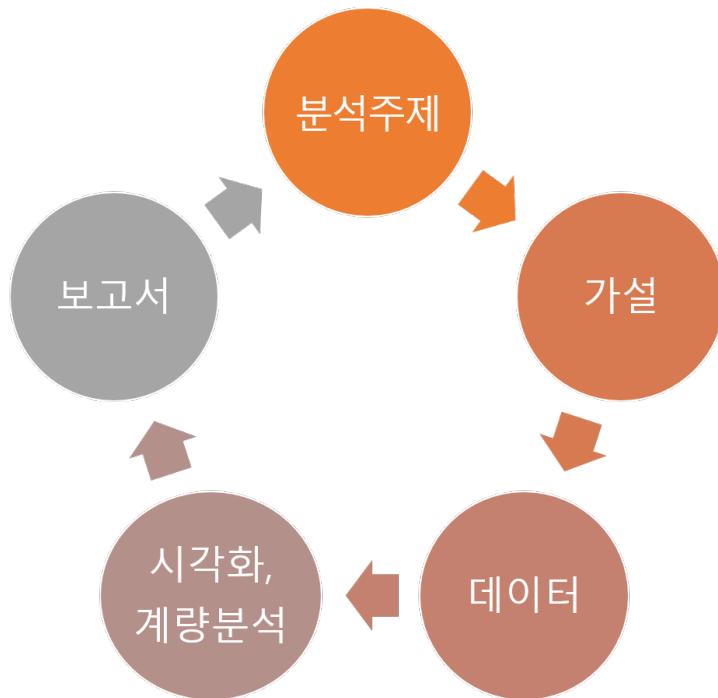
1	파이썬을 이용한 데이터 분석	2
2	파이썬 익숙해지기	11
2.1	주피터노트북	12
2.2	파이썬, 이 것만 알아두자	15
3	데이터 분석 라이브러리, 판다스	30
3.1	인덱스와 칼럼	30
3.2	데이터 분석	38
4	파이썬 활용하기	54
4.1	작업 환경 갖추기	55
4.2	데이터 입수, 전처리, 시각화	56
5	파이썬과 스타타	62
5.1	파이썬에서 스타타로	62
5.2	스타타에서 파이썬으로	63

1 파이썬을 이용한 데이터 분석

대규모 데이터에서 유의미한 정보를 추출하여 가설을 설정하고 검정할 때, 데이터를 수시로 입수하여 기존 보고서를 업데이트해야 할 때 파이썬을 이용하여 생산성을 높일 수 있다.

데이터 분석에 기반한 동향분석, 조사연구 업무는 크게 분석주제 선정, 가설 설정, 데이터 입수, 시각화 및 계량분석, 보고서 작성 등으로 구성되며, 각 단계는 끊임없이 피드백을 주고 받는다. 분석주제와 가설은 입수 가능한 데이터의 종류와 분석 결과에 따라 바뀌며, 보고서 작성 과정에서 어떤 데이터와 분석 방법을 사용할지도 계속해서 바뀌게 된다.

일반적인 데이터 분석 업무 흐름



애자일 업무 흐름

이처럼 데이터 분석 업무는 한 단계에서 다음 단계로 단선적으로 넘어가는 것이 아니라, 가설 설정과 데이터 작업, 보고서 작성의 각 단계를 끊임없이 오가는

과정을 반복하게 된다. 따라서, 대규모 데이터를 이용한 분석 보고서를 빠르게 완료하기 위해서는 가설 설정과 데이터 분석 사이의 피드백이 빠르게 이루어져야 한다(애자일 업무흐름).¹

일반적으로 업무 흐름의 각 단계 사이의 피드백을 저해하는 주 요인은 데이터 입수, 시각화, 계량분석에 많은 시간이 소요된다는 것이다. 특히, 디지털 혁신으로 다양한 형태의 대규모 미시자료를 분석에 활용하게 되면서 데이터 처리에 소요되는 시간을 줄이는 것이 보다 중요해지고 있다.²

어떻게?

데이터 분석은 어떤 데이터를 어떻게 분석할 것인가에 대한 여러 단계의 선택을 통해 이루어진다. 몇 안되는 변수의 짧은 시계열이나 적은 수의 개체(국가, 기업, 개인 등)로 구성된 데이터를 분석할 경우에는 모든 데이터를 엑셀 파일 시트에 모아두고 데이터가 어떻게 처리되는지 눈으로 보아가며 작업하는 것이 효율적이다(데이터 중심). 그러나, 변수 및 개체 수, 시계열 길이가 열배, 백배 늘어나면 연구자가 들여야하는 시간과 노력은 더 빠른 속도로 늘어난다.

단계별 데이터 작업(예시)



데이터 v. 프로세스

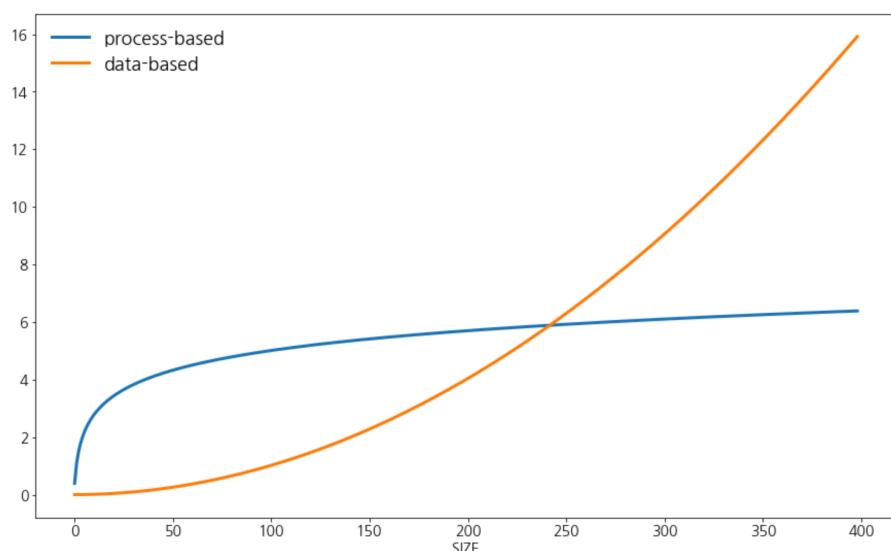
연구자의 가설이 데이터에 부합하는지 빠르게 점검하고 데이터에 기반한 가설을 설정하기 위해서는 데이터 분석에 들이는 시간을 줄여야 한다. 이를 위해서는

¹Agile workflow is an iterative method of delivering a project. In Agile, multiple individual teams work on particular tasks for a certain duration of time termed as ‘Sprints’. “Agile Workflow can be defined as the set of stages involved in developing an application, from ideation to sprints completion.”

²디지털 혁신에 중앙은행이 어떻게 대응해야 하는지는 2019년 1월 한은소식 [디지털 혁신과 중앙은행](#)을 참고하라.

첫째, 데이터 분석 과정을 세부 단위작업/프로세스로 나누어야 한다(프로세스 중심). 전체 데이터 작업을 컴퓨터가 바로 실행할 수 있는 단위 작업들로 나눈 다음, 컴퓨터가 단위 작업들을 각 변수, 시점, 개체에 대해 반복적으로 수행하도록 하면 데이터 규모가 증가하더라도 연구자가 들여야 하는 시간은 증가하지 않는다.

분석 소요 시간 비교: 프로세스 v 데이터 기반



이와 같이 데이터 분석 업무가 세부 단위 작업들로 나누어져 있으면, 특정 단계의 오류 수정이나 처리 방법 변경, 데이터 업데이트시 연구자가 해당 작업만 수행하면 나머지 작업들은 컴퓨터가 일괄적으로 처리하여 연구자의 시간을 효율적으로 사용할 수 있게 된다.

```
In [1]: # 자료 입수
for year in range(2000, 2020):
    df = import(http://data.com//data_year.xlsx)
    dfs = merge(dfs, df)
```

```
In [2]: # 자료 처리  
for var in [var1, var2, var3]:  
    dfs.var = function(dfs.var)
```

```
In [3]: # 계량 분석  
for period in [period1, period2]:  
    result_period = analysis(dfs, period)
```

데이터 구조

둘째, 연구자는 데이터 구조를 적절히 설정함으로써 다각적인 측면에서 데이터 속성을 빠르게 파악할 준비가 되어 있어야 한다. 데이터 규모가 작은 경우에는 컴퓨터 화면에 보여지는 엑셀 시트를 통해 데이터를 빠르게 파악하고 필요한 정보를 추출할 수 있다. 그러나, 데이터 규모가 커질수록 특정 변수, 시점, 개체의 속성을 파악하고, 분석하는데 터무니없이 오랜 시간이 걸리게 된다.

데이터에 구조를 부여하는 간단한 방법은 데이터의 각 항목에 이름을 붙이는 것이다. 테이블 형태로 주어진 데이터의 각 항목은 행과 열 위치로 특정된다. 만약 테이블 형태 데이터의 각 행과 열에 연구자가 이해할 수 있는 의미있는 이름이 붙어 있다면, 연구자가 원하는 데이터를 손쉽게 조회하고 분석할 수 있다. 예를 들어, 전세계 국가의 다양한 국민계정 항목으로 구성된 테이블에서 ‘2015년 이후 한국, 미국, 중국의 명목 경제성장을’과 같은 직관적인 방법으로 원하는 데이터를 조회하고 분석에 활용할 수 있다.

파이썬!

파이썬은 1989년 크리스마스 연휴를 앞둔 네덜란드 개발자 귀도 반 로섬(Guido van Rossum)의 취미 활동으로 시작되었다. 이후 버전이 올라갈 때마다 파이썬 형태도 크게 변하였다. 현재 가장 널리 사용되는 Python 3은 2008년 공개되었으며 2021년 12월 현재 3.8버전이 공개되어 있다. Python 3와 Python 2는 문법이 서로 다르므로 새롭게 파이썬을 시작할 경우 Python 2는 관심을 둘 이유가 없다.³

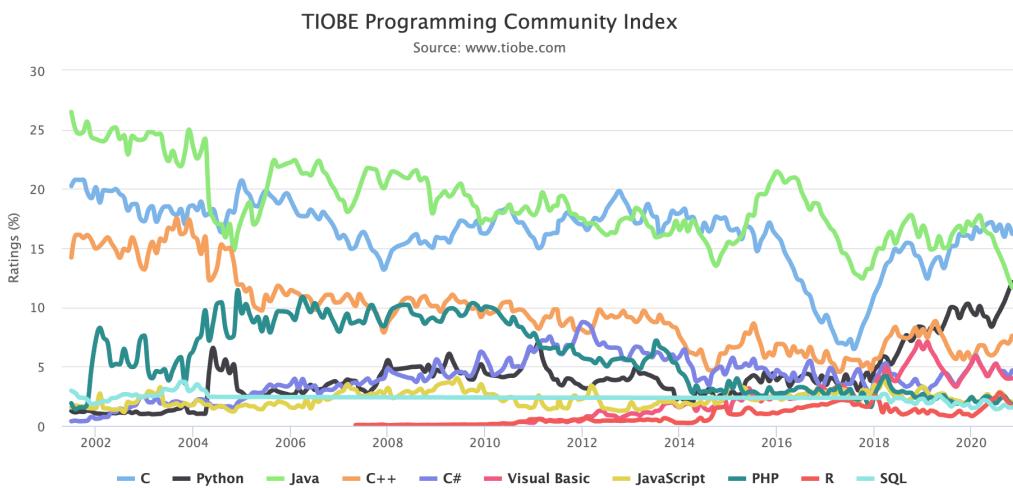
³불과 2, 3년 전까지만 해도 파이썬 라이브러리의 버전이 올라갈 때마다 각 라이브러리의

접근성

파이썬의 가장 큰 장점은 엄격한 문법 제약으로 파이썬 코드를 읽고 쓰기 쉽다는 것이다. 이를 파이썬 추종자들은 ‘가장 아름다운 하나의 답이 존재한다’고 표현한다. 하루 정도 시간을 투자하면 누구나 파이썬 코드를 읽을 수 있으며, Matlab 사용 경험이 있다면 1주일 이내에 본인이 작성한 매틀랩 코드를 파이썬으로 작성할 수 있다.

범용성

파이썬의 두번째 장점은 데이터 분석, 웹개발, 인공지능 등 다양한 분야에서 손쉽게 사용할 수 있다는 점이다. 사용자 선호도, 사용자수 등 조사에서 가장 인기있는 프로그래밍 언어로 꼽히고 있으며, 다양한 분야의 라이브러리들이 파이썬으로 먼저 개발되고 있다.



다양한 형태의 자료를 손쉽게 입수, 분석, 시각화할 수 있다는 장점으로 인해, 경제학, 핀테크, 빅데이터, 머신러닝 등 많은 영역에서 사용자 층이 넓어지고 있다. 폭넓고 깊은 사용자 커뮤니티는 온라인 질의응답을 통해 어떤 문제든 쉽고 빠르게 답을 얻게 해준다.⁴

주요 문법이 바뀌면서 기존에 작성된 코드를 수정해야 할 일이 많았으나, 최근에는 이러한 일이 거의 없다.

⁴파이썬을 처음 시작할 때 참고할 자료는 굉장히 많다. [QuantEcon](#)은 파이썬 기초부터

라이브러리

파이썬의 세번째 장점은 다양한 라이브러리, 개발 환경을 무료로 사용할 수 있다는 것이다. 대부분 라이브러리는 인터넷이 연결된 환경에서 간단한 명령어로 설치할 수 있으며, 사용자가 직접 작성한 코드를 라이브러리 형태로 저장하거나 배포할 수도 있다. 특히, 아나콘다(Anaconda)라는 기업이 배포하는 파일(Individual Edition)⁵을 설치하면 본 연수에서 다루는 모든 내용을 직접 실행할 수 있다.

인터페이스

파이썬의 네번째 장점은 훌륭한 개발 환경(IDE) 또는 인터페이스가 준비되어 있다는 것이다. 대표적인 개발 환경인 주피터노트북 Jupyter notebook/Jupyterlab⁶은 대화형식 인터페이스를 통해 직관적인 파이썬 활용을 돋는다. 개인 컴퓨터에 아나콘다 파이썬을 설치하면 주피터노트북을 통해 파이썬 코드를 작성하고 실행할 수 있다. 개인 컴퓨터 외에 한국은행 조사연구플랫폼(BReiT) 고급 데이터 분석환경 또는 구글 Colab에서 파이썬 코드를 작성하고 실행할 수 있으며, 모두 주피터노트북 환경에서 실행된다.

파이썬으로 할 수 있는 일

파이썬과 주피터노트북은 대화형 인터페이스를 이용하여 대규모 데이터에서 빠르게 의미있는 정보를 추출할 수 있게 해준다. 파이썬을 업무에 활용할 수 있는 사례를 몇가지 소개한다.

객체지향 프로그래밍

본 연수는 파이썬을 이용한 데이터 분석에 초점을 맞추고 있다. 그러나, 데이터 분석 이외에도 파이썬을 활용할 수 있는 영역이 많다. 첫째, 파이썬의 객체지향

경제학 분야에서의 활용까지 다양한 사례와 실제 코드를 제공한다.

⁵윈도우즈, 맥, 리눅스 각 운영체제에 설치할 수 있는 파일을 다음 링크(<https://www.anaconda.com/products/individual>)에서 다운로드 받을 수 있다. 다운로드한 파일을 실행하여 디폴트 설정에 따라 설치한다. 맥 터미널에서 주피터노트북이 실행되지 않을 경우 [installing on macOS](#)을 참고하여 Miniconda를 설치하면 문제가 해결될 수 있다.

⁶Jupyterlab은 Jupyter notebook에 비해 보다 대규모 프로젝트 개발에 용이한 기능들이 추가되었으나 기본적인 사용방법은 Jupyter notebook과 동일하다.

프로그래밍(object-oriented programming)을 이용하여 이질적 경제주체, 에이전트 기반 모형을 효율적으로 프로그래밍할 수 있다.

예를 들어, 이 들 모형에서 가계는 연령, 자산규모 등 속성property과 노동 공급, 저축, 소비 등 행위method로 정의된다. 기업은 생산성 수준 속성과 산출물의 가격을 결정하는 행위로 정의된다. 이렇게 각 경제 주체의 속성과 행위가 명확히 정의되면, 경제 주체 간 상호작용을 보다 정확하고 이해하기 쉬운 방식으로 코딩하고 분석할 수 있다.

```
In [1]: # 가계
def consumer(age, wealth, productivity, ...):
    age, ... # properties
    supply_labor() # method
    save() # method

# 기업
def consumer(productivity, capital, ...):
    productivity, ... # properties
    invest() # method
    price() # method
```

데이터 분석에서도 객체지향 프로그래밍의 장점을 활용할 수 있다. 파이썬에서 다루어지는 각 자료형 역시 각각의 속성(property)과 행위/함수(method)가 정의되어 있다. 예를 들어, 뒤에서 다룰 넘파이 라이브러리의 대표적 자료형인 ndarray가 [1, 2, 3]으로 주어져 있다고 하자. 이 자료형 객체는 다양한 통계연산을 기본 함수로 갖고 있기 때문에 [1, 2, 3].sum() 또는 [1, 2, 3].mean()과 같은 방식으로 원하는 통계량을 빠르게 계산할 수 있다.

매틀랩 대체

둘째, 매틀랩을 대체할 수 있다. 최적화, 시뮬레이션 등 매틀랩을 이용한 작업을 모두 파이썬으로 수행할 수 있다. 매틀랩에 비해 파이썬은 무료이면서 빠르게 실행환경을 열고 원하는 작업을 수행할 수 있다는 장점이 있다. 매틀랩과 같은 수준의 매트릭스 연산을 보다 가볍게 수행할 수 있으며, 칼만필터(Kalman

filter) 추정, MCMC(Markov Chain Monte Carlo) 시뮬레이션 등과 같이 빠른 연산이 필요한 경우 C언어로 작성한 코드를 파이썬으로 컴파일하여 큰폭의 속도향상을 얻을 수 있다.⁷ 이 외에 데이터 분석, 문자열 처리 등에 특화된 다양한 라이브러리가 제공되어 데이터 입수부터 전처리, 분석에 이르는 전과정을 효율적으로 수행할 수 있다는 장점이 있다.

머신러닝

셋째, 최신 머신러닝 라이브러리를 사용할 수 있다. 최근 구글, 페이스북 등에서 개발된 머신러닝 라이브러리(Tensorflow, PyTorch)는 파이썬만을 지원하는 경우가 많다.

데이터 분석

마지막으로 데이터 분석이다. 파이썬의 강력한 데이터 분석 기능의 강점은 자료 구조가 복잡할수록 더욱 두드러진다. 복잡한 자료 구조를 가진 대표적 데이터로 국제투입산출표가 있다. 국제투입산출표(WIOT)는 개별 국가의 투입산출표를 여러 국가로 확장한 것으로, A국-a산업이 한해 동안 생산한 총 산출(x)을 모든 국가, 산업의 중간투입(z)과 A국-a산업의 부가가치(v)로 분해한 것이다.⁸

매틀랩에서 국제투입산출표를 분석할 경우 컴퓨터 메모리가 충분히 크지 않으면 컴퓨터 연산에 오랜 시간이 걸리며, 연구자가 매번 데이터 구조를 파악하는데 들이는 수고가 많아 비효율적이다. 데이터가 커 역행렬과 같은 컴퓨터 연산에 오랜 시간이 걸릴뿐만 아니라, 매트릭스로 정리된 표의 각 셀이 수십개 국가, 수십개 산업의 상호 연관관계중 무엇을 나타내는지 추적하는데 많은 수고를 들여야 하기 때문이다.

이에 반해, 파이썬의 데이터 분석 라이브러리인 판다스는 속도가 빠를 뿐만

⁷파이썬에서 DSGE 모형을 설정하고 분석하기 위한 여러 프로젝트가 진행되고 있으나, 아직 Matlab + Dynare 조합을 대체할 정도에 이르지는 못한 것으로 보인다. 시간 여유가 있다면, 파이썬에 어느 정도 익숙해진 다음 간단한 수준에서 Dynare를 대체하는 파이썬 코드를 작성해 볼 것을 추천한다.

⁸예를 들어, A-a열, B-b행에 해당하는 셀은 A국-a산업의 생산과정에 중간투입된 B국-b산업의 산출량, A-a열, v행은 A국-a산업의 생산과정에서 창출된 부가가치, A-a열, x행은 A국-a산업의 총 산출량을 나타낸다. A국-a산업의 총 산출중 투자(i), 소비(c) 등으로 최종사용된 것은 각각 i, c열의 A-a행에 표시된다. 이 표를 이용하여 특정국가의 소비 또는 투자 증가가 각 국 산업별 부가가치, 무역수지에 미치는 영향을 계산할 수 있다.

국제투입산출표 노트북

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** python2020 국제투입산출표
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help
- In [95]:**

```
import pandas as pd
import numpy as np
from IPython.display import IFrame, Image
idx = pd.IndexSlice
```
- Text:** 레온티에프 역행렬 함수
- Text:** 투입산출표
- Equation:** 중간투입행렬 Z , 최종수요행렬 F , 부가가치벡터 V , 충산출액터 X
$$\begin{bmatrix} Z \\ V' \\ X' \end{bmatrix} \text{ where } Z = \begin{bmatrix} Z_{c-m,c-m} & Z_{c-m,c-i} & Z_{c-m,k-m} & Z_{c-m,k-i} \\ Z_{c-i,c-m} & Z_{c-i,c-i} & Z_{c-i,k-m} & Z_{c-i,k-i} \\ Z_{k-m,c-m} & Z_{k-m,c-i} & Z_{k-m,k-m} & Z_{k-m,k-i} \\ Z_{k-i,c-m} & Z_{k-i,c-i} & Z_{k-i,k-m} & Z_{k-i,k-i} \end{bmatrix} \text{ and } F = \begin{bmatrix} F_{c-m,c-c} & F_{c-m,c-i} & F_{c-m,k-c} & F_{c-m,k-i} \\ F_{c-i,c-c} & F_{c-i,c-i} & F_{c-i,k-c} & F_{c-i,k-i} \\ F_{k-m,c-c} & F_{k-m,c-i} & F_{k-m,k-c} & F_{k-m,k-i} \\ F_{k-i,c-c} & F_{k-i,c-i} & F_{k-i,k-c} & F_{k-i,k-i} \end{bmatrix}$$
- Text:** 레온티에프 역행렬: $(I - A)^{-1}$
- Equation:** $X = AX + f$ with $A_{i,j} = Z_{i,j}/X_j$ and $f_i = \sum_j F_{i,j}$
- Equation:** $X = (I - A)^{-1}f$
- In [96]:**

```
def Leon(A):
    return pd.DataFrame(np.linalg.inv(np.identity(A.shape[0]) - A),
                        index=A.index, columns=A.columns)
```
- Text:** 주피터노트북에서 수식 편집
- In [97]:**

```
url = ('http://jupyter-notebook.readthedocs.io/en/latest/examples/Notebook/Typesetting%20Equations.html')
IFrame(url, width=900, height=450)
```
- Out[97]:** Motivating Examples
The Lorenz Equations
- Bottom:** Docs > Notebook Examples > Motivating Examples > Edit on GitHub

아니라⁹ 연구자가 생각대로 데이터 분석이 가능하다는 장점이 있다. 예를 들어, 매틀랩에서 한국 건설업이 미국 석유정제업에 중간 투입된 금액을 확인하기 위해서는 해당하는 행과 열 위치를 찾아야 한다. 반면, 판다스에서는 데이터의 각 행과 열에 이름이 붙어있기 때문에 데이터프레임(한국, 건설업; 미국, 석유정제업)과 같이 원하는 데이터의 이름을 특정하여 원하는 값을 찾을 수 있다. 이러한 이점은 데이터의 크기가 커지고, 자료 구조가 계속 변경될 때 더욱 중요해진다.

⁹판다스의 데이터 연산은 수치해석에 특화된 넘파이numpy 라이브러리에 기반하였으며, 많은 연산 알고리즘이 C언어 수준에서 작성되어 매우 빠르게 실행된다.

2 파이썬 익숙해지기

앞서 설명한 것처럼 데이터 분석을 목적으로 하는 경우 대화 형식의 인터페이스를 제공하는 주피터노트북 환경에서 파이썬을 이용하는 것이 좋다. 물론, 주어진 데이터에서 어떤 변수들을 선택하여 어떤 그래프를 그릴지 또는 어떤 계량 도구를 이용하여 분석할지가 모두 정해진 다음에는 데이터 입수부터 보고서 작성까지의 모든 단계를 하나의 파이썬 스크립트 파일(script file)¹⁰을 작성하여 실행하는 것이 효율적이다.

파이썬 실행

본 연수는 구글 Colab 환경에서 진행된다. 조사연구플랫폼(BReiT) – Model Hub의 고급 데이터 분석환경중 python 분석환경(BReiT Python)과 아나콘드 파이썬을 설치한 개인 컴퓨터에서 파이썬을 활용하는 방법도 간단히 소개한다.

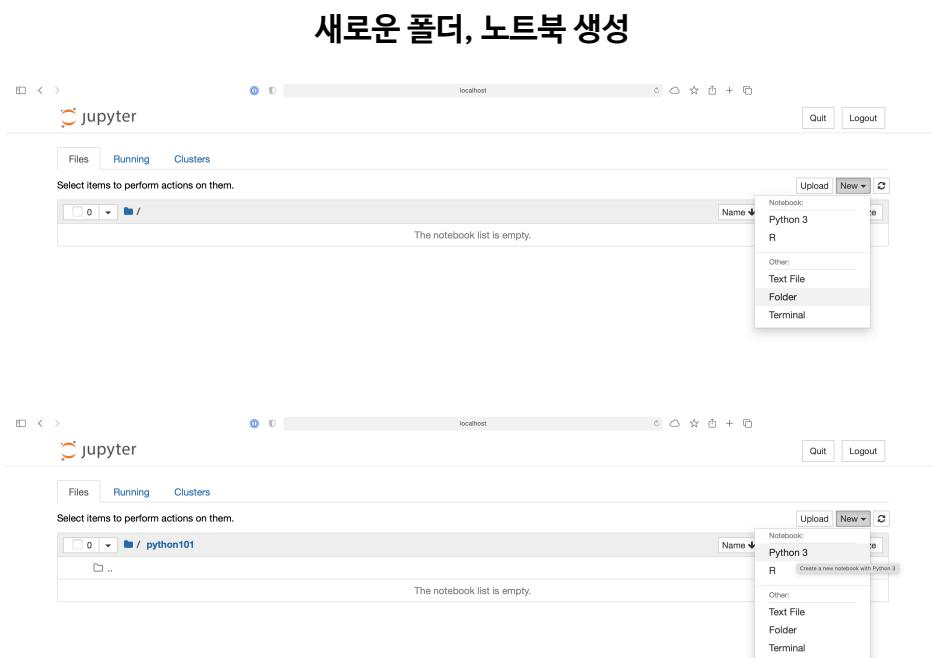
BReiT Python은 인터넷에 있는 데이터를 바로 입수할 수 없다는 단점이 있으나, 따로 파이썬 실행환경 설치를 신경쓸 필요가 없고 BReiT 데이터를 바로 입수할 수 있다는 장점이 있다. 대부분 파이썬 라이브러리는 이미 설치되어 있으며, 설치가 필요한 경우에도 간단한 명령어(pip)로 바로 설치할 수 있다. 오류가 발생할 경우 데이터서비스팀에 라이브러리 입수를 요청해야 한다.

구글 Colab 또는 BReiT Python과 다르게 외부망 PC나 개인 PC에 Anaconda의 파이썬 패키지를 설치하였다면 주피터노트북을 사용하기 위해 몇가지 준비를 해야한다. 먼저 설치된 응용프로그램중 Anaconda3 폴더 아래에 있는 Jupyter Notebook을 선택하거나, 윈도우의 Anaconda Prompt 또는 맥의 Terminal에서 jupyter notebook을 입력하고 엔터키를 입력하여 실행한다.

주피터노트북이 실행되면 기본 웹브라우저 창(또는 탭)이 열리면서 홈폴더(예, 'Users\bok') 내용을 표시한다. 가급적 웹 브라우저는 Google Chrome, 폰트는 Consolas를 추천한다. 본 연수에서 작업할 파일을 모아 두기 위해 홈폴더 아래에 폴더 하나를 만들어 두자. 새로운 폴더는 화면 오른쪽 위에서 New 버튼을 클릭한 다음 Folder 버튼을 선택한다. Untitled Folder가 생성되면 왼쪽 체크박스를 선택하고 화면 왼쪽 위 Rename 버튼을 클릭하여 원하는 이름(예,

¹⁰여러 파이썬 명령어, 함수 정의 등을 텍스트 파일 형식(확장자 .py)으로 저장한 것으로, 파일 실행시 파일내에 있는 모든 명령어가 순차적으로 실행된다.

python2021)으로 변경한다.



이제 새로운 폴더 이름을 선택하여 이동한 다음 새로운 주피터노트북 파일을 생성한다. 화면 오른쪽 위에서 New 버튼을 클릭한 다음 Python 3을 선택하여 노트북(확장자 ipynb)을 생성한다. 화면 왼쪽 위 메뉴 가운데 File - Rename... 을 선택하여 노트북 이름을 변경한다. 이제 주피터노트북에서 파이썬을 사용할 준비가 끝났다.

2.1 주피터노트북

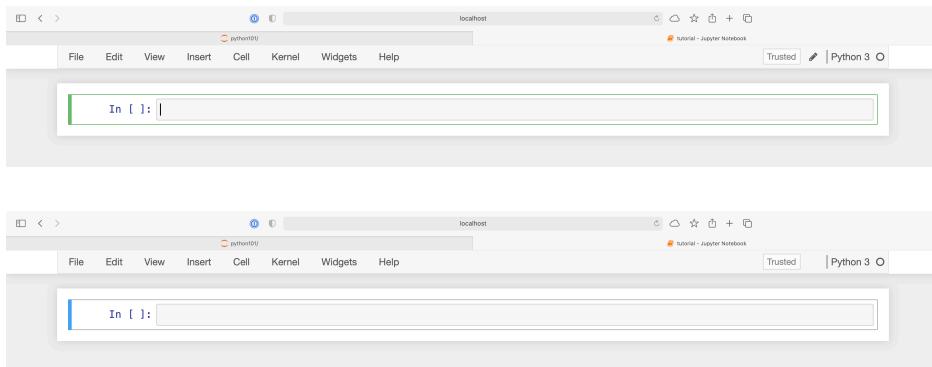
주피터노트북(이하, 노트북)¹¹은 대화창 형식으로 사용자와 파이썬을 연결하는 인터페이스이다. 노트북 사용법을 파악하고 파이썬 명령어를 실행하는 과정에서 파이썬과 노트북 모두 익숙해지게 된다.

¹¹여기서는 개인 컴퓨터에 설치한 아나콘다 파이썬을 기준으로 설명한다. Colab, BReiT Python에서 실행한 노트북은 기본적인 구조, 사용방법은 같으나 모양과 색, 단축키 등에 차이가 있다.

노트북은 여러 개의 셀(cell)로 이루어진다. 각 셀은 파이썬 명령어가 실행되는 Code 셀과 텍스트를 표시하는 Markdown/Raw NBconvert 셀 두 가지 타입으로 구분된다. 새로운 셀을 하나 생성하면 기본적으로 Code 타입이다. 셀을 다른 타입으로 변경하려면 노트북 단축키나 화면 위 메뉴(Cell - Cell Type)를 이용한다.

셀 모드

현재 선택된 셀은 녹색 테두리가 있는 편집모드(edit mode) 또는 파란색 테두리가 있는 명령모드(command mode)로 구분된다. 셀 내부에 커서가 깜박이는 편집모드에서는 셀 내에 파이썬 명령어나 텍스트를 입력하고 편집할 수 있다. 셀 내부에 커서가 보이지 않는 명령모드에서는 하나 또는 두개 이상의 셀을 대상으로 셀 복사하기/잘라내기, 셀 붙이기, 여러 셀 병합하기와 같은 셀 편집작업을 수행할 수 있다.



현재 선택된 셀 모드를 편집모드에서 명령모드로 전환하기 위해서는 현재 셀 입력창 밖을 마우스로 클릭하거나 `esc` 키를 누른다. 명령모드에서 편집모드로 전환하기 위해서는 현재 셀 입력창 안을 마우스로 클릭하거나 `Enter` 키를 누른다.

단축키¹²

셀 편집모드에서 셀 내부의 내용을 편집할 때는 일반적인 텍스트 편집기능 이용할 수 있다. 셀 명령모드에서 셀 잘라내기, 붙이기, 삭제 등 셀을 대상으로

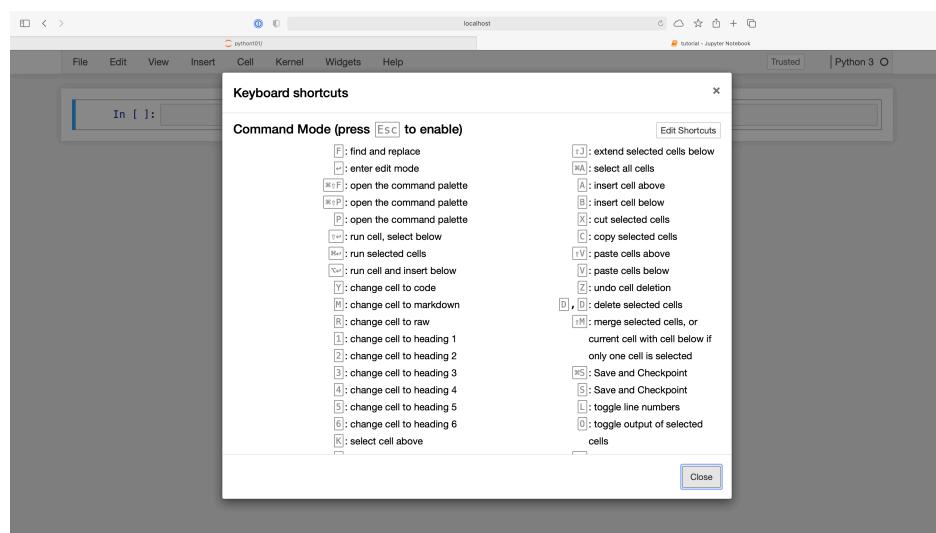
¹²컴퓨터 운영체제에 따라 단축키가 다르므로, 운영체제에 맞는 단축키는 노트북의 전체 단축키 목록을 확인한다. 본 자료에서는 윈도우 pc를 기준으로 설명한다.

편집할 때는 노트북 단축키나 화면 위의 Edit/Insert 메뉴를 이용한다. 주요 노트북 단축기는 아래와 같다.

- 현재 셀 잘라내기/복사하기: **x/c**
- 잘라낸/복사된 셀을 현재 선택된 셀 아래에 붙이기: **v**
- 현재 셀 삭제: **d,d**
- 현재 셀 위/아래에 새로운 셀 만들기: **a/b**
- 현재 셀 타입을 Code/Markdown으로 전환: **y/m**
- 실행취소 Undo: **z**
- 명령모드에서 전체 단축기 목록 확인: **h**
- 선택된 두 개 이상의 셀을 병합: **m**

한편, 셀 편집모드에서 현재 커서 위치를 기준으로 셀을 두 개로 분할하고자 할 경우 **shift + control + -(minus)**를 입력한다.

주피터노트북 단축기(명령모드에서 **h**키 입력)



셀 실행

현재 셀을 실행하기 위해서는 `shift + Enter` 또는 `control + Enter` 키를 입력한다. `shift + Enter`는 현재 셀을 실행한 다음 바로 아래 셀이 선택된 상태에 놓이게 되며, 다음 셀이 없으면 새로운 셀 생성된다. `control + Enter`는 현재 셀을 실행한 후 다음 셀로 선택이 넘어가지 않고 실행된 셀 선택을 유지한다.

Code 셀을 실행하면 셀 내부에 입력된 명령어들이 파이썬 문법에 따라 실행되며, Markdown 셀을 실행하면 마크다운 문법에 따라 텍스트가 보여진다. 선택된 셀 타입, 셀 내부에 입력된 내용에 관계없이, 명령모드에서 숫자키 `1, ..., 6`을 누르면 현재 셀이 Markdown 타입으로 바뀌면서 마크다운 `headline` 서식이 적용된다.

2.2 파이썬, 이 것만 알아두자

이제 노트북 인터페이스를 이용하여 파이썬을 시작할 준비가 끝났다. 다른 프로그래밍 언어와 같이 파이썬도 1. 자료형, 2. 함수/클래스 정의, `for-loop`, `if-else` 등 문법, 3. 함수, 클래스 등을 모아놓은 라이브러리 등을 이용하여 원하는 기능을 자유롭게 구현할 수 있다. 파이썬은 다른 언어에 비해 외워야 할 것이 많지 않고 직관적으로 이해되는 부분이 많아, 업무에 사용할 수 있는 코드를 빠르게 작성할 수 있다.

1. 자료형: `int`, `float`, `bool`, `string`, `list`, `dict`
2. 문법: `def`, `for`, `if-else`, `list comprehension`
3. 라이브러리: `numpy`, `pandas`, `matplotlib`, `pytorch`, ...

자료형

대부분 프로그래밍 언어와 같이 파이썬도 정수, 실수, 참/거짓, 리스트, 딕셔너리 등의 자료형을 갖는다. 그러나, 변수 자료형은 연산 과정에서 언제든지 바뀔 수 있기 때문에 사용자가 어떤 변수의 자료형이 무엇인지 굳이 신경쓸 일이 없으며, 필요에 따라 원하는 자료형으로 변경하면 된다.

숫자, 참/거짓, 문자열

파이썬은 수 `number`를 정수 `int`와 실수 `float` 자료형으로 나누어 인식한다. 실수

자료형은 소수점 이하 값이 있는 숫자와 빈값null, 무한값inf 등을 포함한다. 일반적 연산에서는 자료형을 신경쓸 필요가 없지만, 연산 결과가 뒤에서 다를 리스트나 다른 자료형의 인덱스로 사용될 경우 int() 함수를 이용하여 정수 자료형으로 변경한다.

참/거짓 자료형bool은 True 또는 False값을 갖는다. 이 들 변수를 서로 더하거나 곱하면 True는 1, False는 0으로 변환된다. 이들 자료형은 대부분 논리연산의 결과로 생성된다.

파이썬의 가장 큰 장점은 문자열 자료형을 손쉽게 다룰 수 있다는 점이다. 문자열은 'a', "KOR", '2021'과 같이 하나 이상의 문자를 따옴표(' ' 또는 " ")로 둘러쌓아 정의한다.

파이썬에서는 정수, 실수, 부울변수, 문자열 등의 값을 갖는 변수를 아래와 같이 정의한다.

```
In [1]: a = 5
        b = (a == 5.0) # 논리 연산
        c = 'KOR'
        print(a, b, c)
```

5 True KOR

파이썬은 print() 함수를 이용하여, 다양한 자료형의 값을 표시한다. 문자열의 경우 문자열 내용을 그대로 표시하면 되지만, 숫자의 경우 천단위마다 ,(comma) 표시를 하거나 소수점 이하 몇자리까지 표시할지 등을 선택할 수 있다. 숫자와 문자열을 특정 형식에 맞추어 표시하고자 할때 f-string을 이용한다.

```
In [2]: a = 2
        b = 'two'
        print(f'{a} is {b}')
        print(f'{a:.2f} is {b}') # a를 소수점 이하 2자리까지 표시
        print(f'{one thousand} is {1000:.2f}') # 천단위(,) 표시
```

```
2 is two  
2.00 is two  
one thousand is 1,000.00
```

리스트

리스트list는 파이썬을 이용한 데이터 분석에서 가장 기초가 되는 자료형이며, 하나 이상의 항목을 [] 안에 두어 정의된다. 리스트는 이후 소개할 array, dictionary, DataFrame 등 다양한 데이터 자료형을 생성하고 이용하는데 사용된다.

- ['KOR', 'USA', 'CHN']
- [1, 2, 3, 4, 5]
- ['KOR', 1, nan, False]

다른 프로그래밍 언어와 구별되는 파이썬의 특징 중 하나는 여러 항목으로 구성된 자료형에서 첫번째 위치를 가리키는 인덱스가 0이고 이후 인덱스는 1씩 커지며, 전체 항목의 개수가 n일때 마지막 항목의 인덱스는 n-1 또는 -1이다. 연속된 여러 항목을 나타내고자 할 경우 콜론(:)을 이용한다. [0:2] 또는 [:2]는 처음 두 개의 항목을, [-2:]은 마지막 두 개의 항목을 가리킨다.

```
In [1]: a = ['KOR', 'USA', 'CHN']  
print(a[0], a[-1], a[:2], a[-2:])  
  
KOR CHN ['KOR', 'USA'] ['USA', 'CHN']
```

노트북에서 리스트 변수명을 입력하고 실행하면 리스트 길이에 따라 표시 방법이 달라진다. 리스트 길이가 짧으면 위에서와 같이 가로로 각 항목을 표시하지만, 리스트가 너무 길면 가독성을 위해 세로로 표시된다. 길이에 관계없이 항상 가로로 표시되길 원한다면, 노트북 명령어 %pprint를 실행하여 설정을 바꿀 수 있다. 원래 설정으로 돌아가려면 %pprint 명령어를 다시 실행한다.

문자열 리스트를 print() 명령어로 표시하면 각 문자열이 따옴표로 둘러싸여 괄호안에 표시된다. 따옴표와 괄호 없이 문자열만 표시하기 위해서는 asterisk(*)

기호 뒤에 리스트 변수명을 입력한다. 각 항목을 특정 기호로 구분하여 표시하기 위해서는 `sep`인자를 지정한다. 두 개 `print()` 함수를 실행하면 첫번째 결과 다음에 줄을 바꾸어 두번째 결과가 표시된다. 두 `print()` 함수 간 구분 표시를 바꾸려면 `end`인자를 지정한다.

In [2]: `%pprint`

Pretty printing has been turned OFF

In [3]: `a * 10`

Out [3]: `['KOR', 'USA', 'CHN', 'KOR', 'USA', 'CHN', ...]`

In [4]: `%pprint`

Pretty printing has been turned ON

In [5]: `a * 10`

Out [5]: `['KOR',
 'USA',
 'CHN',
 ...]`

In [6]: `print(*a, sep = ', ')`

KOR, USA, CHN

In [7]: `print(*a, end = ', ')
print(*a,) # end='\n'
print(*a,)`

KOR USA CHN, KOR USA CHN

KOR USA CHN

두 개의 리스트를 합하거나 하나의 리스트에서 다른 리스트에 속한 항목을 빼는 연산을 생각해보자. 파이썬은 두 개 리스트의 항목을 각각 합하여 새로운 리스트를 만드는 연산(+)을 지원한다. 이때 특정 항목이 양 리스트에 모두 속한 경우 새로운 리스트에는 동일한 항목이 중복되어 존재하므로 유의해야 한다. 반면, 파이썬은 하나의 리스트에서 다른 리스트에 속한 항목을 제외하는 연산이 따로 정의되어 있지 않지만, list comprehension 또는 set() 함수를 이용하여 원하는 작업을 수행할 수 있다.

```
In [8]: ['KOR', 'USA', 'CHN'] + ['CHN', 'AUS']
```

```
Out [8]: ['KOR', 'USA', 'CHN', 'CHN', 'AUS']
```

```
In [9]: # list comprehension  
[s for s in ['KOR', 'USA', 'CHN'] if s not in ['CHN',  
'AUS']]
```

```
Out [9]: ['KOR', 'USA']
```

또한, list comprehension으로 여러 리스트를 결합하여 새로운 리스트를 만들 수 있다. 이를 통해, 여러 층위의 구조를 갖는 데이터에 대해 손쉽게 행과 열 이름을 생성하여 지정할 수 있다.

```
In [10]: print([n + 4 for n in [1, 2, 3]])  
print([s + ':g' for s in ['KOR', 'USA', 'CHN']])
```

```
[5, 6, 7]  
['KOR:g', 'USA:g', 'CHN:g']
```

위에서 입력셀 실행시 파이썬 명령어에 따라 출력셀이 생기거나 출력셀 없이 결과만 표시되었다. 일반적으로 새로운 숫자나 문자열, 리스트를 만들거나 이전에 정의된 변수의 변수명을 입력하고 실행하면 새로운 출력셀이 생기면서 숫자, 문자열, 리스트를 표시한다. 반면, 변수에 값을 지정하거나, 변수값을 print() 함수로 표시하거나, %pprint와 같은 특수 명령어를 실행할 경우 출력셀 없이 결과만 표시된다.

주피터노트북은 이전에 실행된 출력셀의 내용을 손쉽게 이용하는 간단한 방법을 제공한다. 9번 출력셀의 값을 확인하고자 한다면 Out[9] 또는 _9를 입력하고 실행한다. 그리고, 가장 최근, 또는 그 이전 출력셀의 값은 _ 또는 __을 입력하여 확인할 수 있다.

In [10]: _

Out [10]: ['KOR', 'USA']

In [11]: Out[8]

Out [11]: ['KOR', 'USA', 'CHN', 'CHN', 'AUS']

딕셔너리

딕셔너리 dictionary는 {key1:value1, key2:value2, ...} 형태로 정의된다. key는 숫자 또는 문자열, value는 임의의 자료형을 지정할 수 있다. 값 대체 (replace), 데이터프레임 생성 등에 이용된다. 길이가 같은 두 개의 리스트에 대해 zip, dict 함수를 차례대로 적용하여 딕셔너리를 생성할 수 있다.

In [1]: iso = {'US': 'USA', 'KR': 'KOR', 'JP': 'JPN', 'CN': 'CHN'}
print(iso['US'], iso['KR'])

USA KOR

In [2]: iso['AU'] = 'AUS' # 새로운 key와 value 추가
print(iso)

{'US': 'USA', 'KR': 'KOR', 'JP': 'JPN', 'CN': 'CHN', 'AU': 'AUS'}

In [3]: dict(zip(['한국', '미국', '일본'], ['KR', 'US', 'JP']))

Out [3]: {'한국': 'KR', '미국': 'US', '일본': 'JP'}

딕셔너리는 데이터 소스에 따라 서로 달리 쓰이는 용어를 통일하거나 이름이 너무 길때 단축이름으로 대체하기 위해 주로 활용된다. 예를 들어, 국가별 연간

경제성장률과 가계신용 증가율을 각각 IMF와 BIS에서 입수한다고 하자. BIS 데이터는 ISO 기준 국가코드를 제공하므로 IMF 데이터의 국가명을 ISO 2 또는 ISO 3 코드로 변환하면¹³ 두 개 데이터를 국가명과 연도를 기준으로 결합할 수 있다.

```
In [1]: import pandas as pd  
countries = pd.Series(['South Korea', 'United States'])  
print(countries)
```

```
0 South Korea  
1 United States  
dtype: object
```

```
In [2]: cdict = {'South Korea': 'KR', 'United States': 'US'}  
countries.replace(cdict)
```

```
Out [2]: 0 KR  
1 US  
dtype: object
```

터플

터플tuple은 (var1, var2, ...) 형태로 정의되며, 리스트와 같은 방식으로 각 항목을 나타낼 수 있다. 리스트나 딕셔너리에 비해서는 사용할 일이 많지 않다. 뒤에서 다룰 판다스의 데이터프레임에서 행이나 열 이름이 터플 자료형

¹³country_converter 라이브러리는 다양한 국가명을 ISO 2, ISO 3 코드로 변환하는 함수를 제공한다.

```
In [1]: import country_converter as coco  
print(coco.convert('Republic of Korea', to='ISO2'))  
  
print(coco.convert('South Korea', to='ISO2'))
```

```
Out [1]: KR  
KR
```

이면, 손쉽게 여러 계층구조를 가진 자료구조를 만들 수 있다.

터플 자료형을 이용하여 두 개 이상 변수를 한줄에서 생성하거나, 두 개 변수의 이름을 서로 바꿀 수 있다.

```
In [1]: a, b, c = 5, 10, 15  
print(a, b, c)
```

Out [1]: 5, 10, 15

```
In [2]: a, b = b, a  
print(a, b)
```

Out [2]: 10, 5

문법

다른 프로그래밍 언어와 파이썬의 가장 큰 차이점은 함수 정의, for-loop, if-else 구문 등의 시작과 끝을 들여쓰기(스페이스 4개)로 구분한다는 것이다. 이는 파이썬에서 코드 작성의 자유도를 크게 제약하지만, 동시에 코드 작성의 효율성과 가독성을 높이는 장점으로 평가된다.

```
In [1]: def sum_two_numbers(a, b):  
    return a + b
```

```
In [2]: c = sum_two_numbers(1, 2)  
print(c)
```

3

```
In [3]: countries = ['KOR', 'USA', 'CHN']  
for c in countries:  
    if c == 'KOR':  
        print(c, 'is my country.')  
    else:  
        print(c)
```

```
KOR is my country.  
USA  
CHN
```

앞에서 설명한 list comprehension의 장점은 일반적인 for-loop 구문과 비교할 때 두드러진다. 앞 리스트에서 뒤 리스트에 속한 항목을 빼는 작업을 list comprehension 및 일반적인 for-loop 구문으로 구현하면 아래와 같다.

```
In [1]: # list comprehension  
[s for s in ['KOR', 'USA', 'CHN'] if s not in ['CHN', 'AUS']]
```

```
Out [1]: ['KOR', 'USA']
```

```
In [2]: # for-loop  
clist = []  
for c in ['KOR', 'USA', 'CHN']:  
    if c not in ['CHN', 'AUS']:  
        clist = clist + [c] # clist.append(c)  
clist
```

```
Out [2]: ['KOR', 'USA']
```

사용자는 필요에 따라 파이썬 명령어를 적절히 조합하여 쉽게 함수를 정의하고 이용할 수 있다. 반복되는 작업을 함수로 작성하여 두고 필요할 때마다 호출하여 사용하면 많은 시간을 절약할 수 있다. 간결한 코드 작성은 지향하는 파이썬 문법의 특징으로 인해, 사용자 정의 함수도 몇 줄 이내에 작성되는 경우가 많다. 그러나, 너무 짧고 압축적인 함수는 시간이 지나 다시 사용할 때나 다른 사용자와 공유할 때 이해하기 어렵다는 단점이 있다.

데이터 분석 과정에서 여러 형태의 for-loop 구문을 사용하게 된다. 하나의 리스트에 대해 리스트의 첫째 항목부터 마지막 항목까지 순서대로 정해진 작업을 반복하여 수행하는 것이 가장 대표적인 사용법이다. 다음으로 많이 사용되는 형태는 리스트의 각 항목과 그 항목의 리스트 내 위치를 같이 순환 변수로 이용하는 것이다. `for i, c in enumerate(list):` 와 같이 실행한다. 마지막은 두 개 리스트에서 각 항목을 순서대로 추출하는 것으로 `for i, j in zip(list1,`

list2):와 같이 실행한다.

```
In [3]: for i, c in enumerate(['KOR', 'USA', 'CHN']):  
    print(i, c)
```

```
0 KOR  
1 USA  
2 CHN
```

```
In [4]: for i, c in zip(['KOR', 'USA', 'CHN'], ['한국', '미국', '중국']):  
    print(i, c)
```

```
KOR 한국  
USA 미국  
CHN 중국
```

`if-else`와 유사한 구문으로 `try-except`가 있다. `try`: 이후 명령어 실행과정에서 오류가 발생하면 `except`: 이후 명령어를 실행한다. `try`: 이후 실행문에 오류가 발생할 가능성이 있으나, 정확히 어떤 오류인지 특정하지 않고 조건문을 사용할 수 있다는 장점이 있으나, 예상하지 못한 방식으로 동작할 수 있다는 단점이 있다.

`for-loop`과 `if-else` 구문을 조합하여 코드를 작성할 때 조건에 따라 `for-loop`를 중단하거나 다음 항목으로 넘어가야 할 때 `break` 또는 `continue` 명령어를 사용한다. `if-else` 구문에서 특별히 실행할 작업이 없을때는 `pass` 명령어를 사용한다.

앞에서 소개한 이질적 경제주체 모형, 에이전트 기반 모형 등을 파이썬을 이용하여 프로그래밍할 경우 여러 클래스Class를 정의하여 사용한다. 또한, 이들 모형이 아니더라도 대부분 파이썬 자료형, 함수 등은 클래스로 정의되어 직관적으로 사용하기 편리하다는 이점이 있다. 이러한 이점을 누리는데 굳이 어떻게 클래스를 정의하고 사용하는지를 공부할 필요는 없으므로 본 연수에서 클래스와 관련된 내용은 따로 다루지 않는다.

라이브러리

파이썬은 분석 목적에 따라 다양한 라이브러리를 설치하여 사용할 수 있다. 분야별로 가장 널리 쓰이는 라이브러리는 다음과 같다.

데이터 분석: Pandas (built upon Numpy)

매틀랩 대체: Numpy, Scipy

그래프: Matplotlib, Seaborn, Bokeh

계량 분석: Statsmodels

머신러닝: Scikit-learn, TensorFlow, PyTorch

시스템: pickle, os, glob, platform

데이터 입수: Fred, World Bank

기타: country_converter, IPython, urllib 등

위 라이브러리중 numpy, matplotlib, pickle 등의 기능은 판다스에 이미 구현되어 있어 판다스만 사용해도 충분하다. 그러나, 판다스의 기본 기능이 넘파이를 기반으로 하고 있어, 넘파이 이용이 익숙해지면, 판다스를 이용한 데이터 분석도 요령이 생기게 된다. 아래에서는 넘파이를 이용하여 파이썬에서 라이브러리를 어떻게 설치, 호출, 사용하는지 설명한다.

넘파이

넘파이numpy는 파이썬에서 과학적 컴퓨터 연산(NUMerical + PYthon)을 수행하기 위한 라이브러리이다. 숫자와 관련된 대부분 분석은 넘파이 라이브러리를 이용하여 수행할 수 있다. 파이썬으로 매틀랩을 대체하고자 할 경우, 대부분 작업에서 넘파이 라이브러리면 충분하다.

다른 기본 라이브러리와 마찬가지로 넘파이 라이브러리도 Anaconda 설치파일에 포함되어 설치된다.¹⁴ 만약, 설치되어 있지 않다면 노트북에서 아래와 같이 설치할 수 있다. 다른 라이브러리도 같은 방식으로 설치한다.

¹⁴BReiT Python에도 기본 설치되어 있다.

```
In [1]: !pip install numpy  
...  
...  
Successfully built numpy  
Installing collected packages: numpy  
Successfully installed numpy-1.18.5
```

파이썬에서 기본 제공하는 명령어를 제외하고, 특정 라이브러리에서 제공하는 명령어들은 해당 라이브러리에서 속한 명령어임을 명시해야 의도한대로 실행된다. `numpy`를 ‘특정 이름’으로 호출하고 `numpy`에 속한 ‘특정 함수’를 실행할 경우 `특정_이름.특정_함수`와 같은 방식으로 실행한다.

```
In [1]: import numpy  
import numpy as np  
print(numpy.sin(0), np.sin(0))
```

0.0 0.0

```
In [2]: print(sin(0))
```

NameError: name 'sin' is not defined

```
In [3]: print(np.pi)
```

3.141592653589793

넘파이의 기초 자료형 `ndarray`는 리스트와 비슷하게 생겼으며, 리스트나 넘파이 명령어를 이용하여 생성할 수 있다. 1차 `ndarray`은 벡터, 2차원 `ndarray`은 매트릭스와 같이 활용되며¹⁵, 매틀랩에서와 같이 다양한 연산 작업을 수행할 수 있다.

¹⁵넘파이는 매트릭스 자료형도 지원하나, `ndarray` 대비 제한된 활용성 때문에 매트릭스 연산이 필요한 경우에도 2차원 `ndarray`를 이용하는 것을 추천한다.

```
In [1]: v = np.array([1, 2, 3]) # 1-dim array  
print(v)  
type(v)
```

```
[1 2 3]
```

Out [1]: numpy.ndarray

```
In [2]: m = np.array([[1, 2], [3, 4]]) # 2-dim array  
print(m)
```

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

```
In [3]: print(v.mean(), v.std(), v.max())
```

```
2.0 0.816496580927726 3
```

```
In [4]: print(np.sin([0, np.pi / 2, np.pi]))
```

```
[0.000000e+00 1.000000e+00 1.2246468e-16]
```

리스트와 같이 인덱스를 이용하여 ndarray의 특정 항목 값을 찾거나 변경할 수 있다. 2차원 ndarray의 경우 첫번째 인덱스는 행, 두번째 인덱스는 열을 표시한다.

```
In [1]: a = np.array([[1, 2], [3, 4]])  
a[0, 0] = -1 # a의 첫 행, 첫 열 값을 변경  
print(a)
```

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

```
In [2]: a[0] = 0 # a의 첫 행 값을 모두 0으로 변경, a[0] = [0, 0]과 동일  
print(a)
```

```
[[0 0]
 [3 4]]
```

```
In [3]: a[:, 1] = [5, 6] # 두 번째 열의 값을 5, 6으로 변경
print(a)
```

```
[[0 5]
 [3 6]]
```

1차원 및 2차원 ndarray에 대해 일반적인 벡터, 매트릭스 연산을 수행할 수 있다. 1차원 및 2차원 ndarray에 숫자 하나를 더하거나 곱하면 ndarray의 각 항목별로 숫자가 더해지거나 곱해진다. 또한, 크기가 같은 두 개의 ndarray를 더하거나 곱하면 같은 위치에 있는 항목끼리 값이 더해지거나 곱해진다. 두 개의 2차원 ndarray가 있을 때 앞 ndarray의 가로 길이와 뒤 ndarray의 세로 길이가 일치하면 매트릭스 곱하기(dot product)를 할 수 있다.

```
In [1]: v1 = np.array([1, 2, 3])
v2 = np.array([-1, 0, 1])
```

```
In [2]: print(v1 + 1)
print(v1 * 2)
print(v1 + v2)
```

```
[2 3 4]
[2 4 6]
[0 2 4]
```

```
In [3]: v1 * v2 # element-wise product
v1.dot(v2) # inner product
```

```
[-1 0 3]
2
```

```
In [4]: print(m.T) # transpose
```

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

```
In [5]: v = np.array([1, 0])
print(m.dot(v), v.dot(m)) # matrix multiplication
# 1-dim array는 dim에 맞춰 행/열벡터 전환
```

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

파이썬에서 확률변수 추출은 넘파이 random을 이용한다.

```
In [6]: print(np.random.randn(3)) # 표준정규분포
# np.random.randint(1, 5, 3)는 1과 5 사이 정수
```

```
[1.6705317 0.82114883 1.71938099]
```

3 데이터 분석 라이브러리, 판다스

판다스(PANDAS: Python Data Analysis Library)는 데이터 분석에 특화된 파이썬 라이브러리이다. 판다스의 기본 자료형은 1차원 데이터인 시리즈 Series, 2차원 데이터프레임 DataFrame이다. 데이터프레임은 각 행과 열에 라벨이 붙어있으며, 행 이름을 인덱스 index, 열 이름을 칼럼 columns 이라고 한다.

데이터프레임의 특정 열 또는 행을 뽑으면 시리즈이고, 대부분 데이터 분석은 2차원 데이터를 대상으로 이루어지므로 본 연수는 데이터프레임 자료형을 중심으로 판다스의 주요 기능을 설명한다.

3.1 인덱스와 칼럼

데이터프레임의 인덱스와 칼럼은 데이터프레임 생성시 지정할 수 있고, 생성후에 변경할 수도 있다. 대부분의 경우 데이터 입수 단계에서 인덱스와 칼럼으로 사용할 행과 열을 지정한다.

```
In [1]: df = pd.DataFrame([[1,2],[3,4],[5,6]])
df.columns = ['KOR', 'USA']
df.index = [2017, 2018, 2019]
# pd.DataFrame(..., columns = ..., index = ...)
# pd.DataFrame({'KOR':[1,3,5], 'USA':[2,4,6]}, index = ...)
df
```

```
Out [1]:
```

	KOR	USA
2017	1	2
2018	3	4
2019	5	6

데이터프레임의 기초 통계량(mean, std, sum, median, etc.)은 기본적으로 각 열(axis = 0)에 대해 계산되며, 필요한 경우 행(axis = 1)에 대해 계산 할 수 있다. 각 열에 대해 하나의 통계량이 계산되므로, 계산 결과는 1차원 자료인 시리즈가 되고 열 이름은 시리즈의 인덱스가 된다. 시리즈를 데이터프레임

자료형으로 바꾸기 위해서는 `.to_frame()`를 이용한다.

In [2]: `df.mean() # df.mean(axis = 0)과 동일`

Out [2]:
KOR 3.0
USA 4.0
dtype: float64

In [3]: `df.mean().to_frame('mean')`

mean

KOR 3
USA 4

데이터프레임의 특정 열을 인덱스로 지정하기 위해서는 `set_index()`를, 인덱스를 데이터프레임의 열로 되돌리기 위해서는 `reset_index()`를 이용한다. 두 개 이상 열을 인덱스로 지정할 때는 해당하는 열 이름의 리스트를 지정한다. 만약, 열 이름이 너무 길거나 복잡한 경우 해당 열의 위치를 이용하여 열 이름 리스트(예, 두번째부터 네번째 열: `columns[1:4].tolist()`)를 생성하여 지정 할 수 있다.¹⁶ 두 개 이상의 열을 인덱스로 지정하면, 뒤에 설명할 멀티인덱스가 된다.

In [1]: `import pandas as pd
df = pd.DataFrame([[1, 2], [3, 4]])
df # 행과 열 이름이 주어지지 않은 경우 0, 1, 2, ...로 지정`

0 1

0 1 2
1 3 4

¹⁶이 때, 인덱스로 지정된 열의 이름은 인덱스의 이름이 된다. 인덱스와 칼럼의 이름을 변경할 때는 `df.index.names = ['index1', 'index2', ...]`와 같이 인덱스 또는 칼럼의 `names` 속성에 값을 지정한다. 인덱스 이름을 없애고 싶으면 `[None, None, ...]` 값을 지정한다. 이름이 없는 인덱스를 데이터프레임의 열로 되돌리면, 열 이름이 `Unnamed:0, Unnamed:1, ...`과 같이 지정되므로, 가급적 의미있는 인덱스 이름을 지정해두는 편이 좋다.

```
In [2]: df.set_index(0) # 첫번째 열(열 이름 0)을 index로 지정
# 인덱스 지정 결과 저장은 df = df.set_index(0) 또는
# df.set_index(0, inplace=True)
# 인덱스를 데이터프레임 열로 되돌리기는 df = df.reset_index()
```

Out [2]:

	0	1
1	2	
3	4	

슬라이스

데이터프레임의 특정 영역을 잘라내어(slice) 값을 보여주거나 값을 변경하려는 경우 list, ndarray와 같이 원하는 영역의 상대적 위치를 알면 df.iloc[,]을 이용한다. 원하는 항목의 행과 열 이름을 알고 있는 경우 df.loc[,]을 이용하여 보다 빠르고 정확하게 원하는 영역을 찾을 수 있다.

```
In [3]: df.loc[2018:, :] # df.loc[2018:], df.iloc[1:, :]과 동일
```

Out [3]:

	KOR	USA
2018	3	4
2019	5	6

```
In [4]: df.loc[2017, :] # Series 반환. df.iloc[0, :]과 동일
```

Out [4]:

KOR	1
USA	2
Name:	2017, dtype: int64

```
In [5]: df.loc[[2017], :] # 하나의 행/열을 []으로 선택시 df 자료형 유지
```

Out [5]:

	KOR	USA
2017	1	2

```
In [6]: df = df.loc[2018:, :].copy()
# 데이터프레임의 일부를 잘라내 새로운 데이터프레임 생성
# 복사본(copy())을 이용하지 않을 경우 값 변경시 경고
# 데이터프레임의 일부를 잘라내 값을 변경할 경우,
# 원본 데이터프레임 값을 변경하려는 건지 아닌 건지
# 파이썬이 혼란스러워하기 때문
df
```

Out [6]:

	KOR	USA
2018	3	4
2019	5	6

```
In [7]: df.loc[2018, 'KOR'] = 0
df.loc[:, 'USA'] = 1 # 또는 [1, 1]
df
```

Out [7]:

	KOR	USA
2018	0	1
2019	5	1

```
In [8]: df.loc[:, :] = [[1, 2], [3, 4], [5, 6]]
df
```

Out [8]:

	KOR	USA
2018	1	2
2019	3	4

멀티인덱스

판다스의 멀티인덱스 MultiIndex는 데이터프레임의 행과 열 라벨에 계층을 두어 보다 효과적으로 데이터를 관리하도록 한다. 국제투입산출표의 경우 열 라벨을 한국-제조업, 한국-서비스업, 미국-제조업, ...와 같이 관리하는 것보다 열의 첫번째 단계는 국가(한국, 미국, ...), 두번째 단계는 산업(제조업, 서비스업, ...)으로 나누어 지정하면, 보다 체계적이고 효율적으로 데이터를 관리할

수 있다.

```
In [1]: df = pd.DataFrame('country':['USA', 'KOR', 'USA', 'KOR'],
                           'pop':[1, 2, 3, 4], 'gdp':[3, 4, 5, 6],
                           index = [2016, 2016, 2017, 2017])
```

```
In [2]: df = df.set_index('country', append = True)
df
```

```
Out [2]:
```

	country	pop	gdp
2016	USA	1	3
	KOR	2	4
2017	USA	3	5
	KOR	4	6

MultiIndex를 이용하여 행이나 열 라벨을 직접 멀티인덱스로 지정할 수 있다. 이 때 주어진 라벨 유형에 따라 from_product, from_arrays, from_tuples 등을 이용한다. 예를 들어 국가, 연도의 2단계로 이루어진 멀티인덱스에서 국가별로 동일한 연도에 대한 데이터가 준비되어 있다면 국가명 리스트(['US', 'KR']), 연도 리스트([2016, 2017])에 대해 from_product 명령으로 멀티인덱스를 생성한다. 멀티인덱스 단계별로 데이터 개수가 다르면 from_arrays, from_tuples를 이용한다.

```
In [3]: from pandas import MultiIndex as MI
df.index = MI.from_product([[16, 17], ['미국', '한국']])
# MI.from_arrays([[16, 16, 17, 17], ['미국', '한국',
# '미국', '한국']])
```

Out [3]:

		pop	gdp
16	미국	1	3
	한국	2	4
17	미국	3	5
	한국	4	6

행이나 열 라벨이 ‘16:미국’, ‘17:미국’과 같이 두 개 이상 계층에 속한 라벨들이 특정 기호로 연결되어 있는 경우 손쉽게 멀티인덱스로 변경할 수 있다. 또한, 반대로 멀티인덱스를 단일 계층의 인덱스로 변경할 수 있다.

```
In [4]: df.index = [':'.join(c).strip() for c in df.index.values]
df
```

Out [4]:

		pop	gdp
16:미국		1	3
16:한국		2	4
17:미국		3	5
17:한국		4	6

```
In [5]: df.index = MI.from_tuples(df.index.str.split(':').tolist())
df
```

Out [5]:

		pop	gdp
16	미국	1	3
	한국	2	4
17	미국	3	5
	한국	4	6

데이터 크기가 커지면, 데이터 속성을 파악하기 위해 행과 열 라벨에 어떤 값들이 있는지 파악할 필요가 있다. 멀티인덱스가 아닌 경우 `df.index.unique()`, `df.columns.unique()`는 중복되는 이름을 제거하고, 행과 열에 어떤 라벨들이

있는지 리스트로 보여준다. 멀티인덱스인 경우에는 레벨을 특정하여 어떤 값들이 있는지 확인할 수 있다. 어떤 데이터프레임의 열이 멀티인덱스이고, 레벨 0, 1, 2 까지 있는 경우 레벨 1 열 이름에 있는 값은 `df.columns.get_level_values(1)` 으로 확인할 수 있다. 중복되는 라벨이 많은 경우 `.unique()`을 덧붙여 중복을 제외한 라벨만 표시되도록 한다.

만약, 멀티인덱스중 특정 레벨에 값이 하나만 있다면, 해당 레벨은 제거하는 것이 효율적이다. 멀티인덱스인 열의 두번째 레벨(레벨 1)을 제거하기 위해서는 `df.columns.droplevel(1)`을 실행한다. 멀티인덱스의 레벨 순서를 바꾸기 위해서는 `reorder_levels()` 함수를 이용한다.

다양한 층위에서 데이터 선택

멀티인덱스가 정의된 데이터프레임에서 인덱스의 각 층위에서 데이터 범위를 지정하여 원하는 데이터만 조회할 수 있다. 이 경우 판다스의 `IndexSlice`를 이용한다.

```
In [1]: df = pd.DataFrame('country':[ 'USA', 'KOR', 'USA', 'KOR'],
                         'pop':[1, 2, 3, 4],
                         'gdp':[3, 4, 5, 6],
                         index = [2016, 2016, 2017, 2017])
df = df.set_index('country', append=True)

idx = pd.IndexSlice
df.loc[idx[:, 'USA'], :]
```

```
Out [1]:
```

	country	pop	gdp
2016	USA	1	3
2017	USA	3	5

```
In [2]: df.loc[idx[2016, 'USA'], :]
```

```
Out [2]: pop 1  
gdp 3  
Name: (2016, USA), dtype: int64
```

특정 열의 값을 이용한 데이터 선택

데이터프레임 특정 열의 값을 기준으로 원하는 데이터를 선택할 수 있다. 열의 값이 숫자인 경우 ge, gt, le, lt, between, 문자열인 경우 isin, contains 등을 이용한다.

```
In [1]: df
```

	country	pop	gdp
2016	USA	1	3
Out [1]:	2016	KOR	2
	2017	USA	3
	2018	KOR	4

```
In [2]: df.loc[df['pop'].ge(3)]  
# df.loc[df['pop'].gt(2)], df.loc[df['pop'].between(3, 4)]
```

	country	pop	gdp
Out [2]:	2017	USA	3
	2018	KOR	4

```
In [3]: df.loc[df.country.isin(['USA', 'JPN'])]
```

	country	pop	gdp
Out [3]:	2016	USA	1
	2017	USA	3

3.2 데이터 분석

데이터프레임의 인덱스와 칼럼을 이용하여 어떻게 데이터 내용을 선택할 수 있는지 알아 보았다. 다음으로 데이터프레임 연산 및 결합, 새로운 변수 생성, 통계량 계산 등 실제 데이터를 분석하는데 활용되는 판다스 기능들을 알아본다.

데이터프레임 연산

데이터프레임 간의 연산은 라벨이 동일한 행과 열을 기준으로 이루어진다. 두 개의 데이터프레임을 더할 경우 행과 열 라벨이 동일한 항목의 경우에만 더해지고, 한 데이터프레임에만 존재하는 행 또는 열의 경우 NaN가 채워진다. 두 개의 데이터프레임에 대해 매트릭스 곱하기(dot product)는 앞 데이터프레임의 열 라벨과 뒤 데이터프레임의 행 라벨이 일치하는 경우 오류없이 수행된다.

```
In [9]: def display_by_side(*args):
    # 두 개 데이터프레임을 옆으로 나란히 표시하는 함수
    html_str = ''
    for df in args:
        html_str += df.to_html()
    display_html(html_str.replace('table', 'table style="display:inline"'), raw=True)

df = pd.DataFrame('KOR':[1, 3], 'USA':[2, 4])
df1 = df + 10
df1.columns = ['USA', 'CHN']
display_by_side(df1, df + df1)
```

```
Out [9]:
```

	USA	CHN		CHN	KOR	USA
2018	11	12	2018	NaN	NaN	13
2019	13	14	2019	NaN	NaN	17

```
In [10]: display_by_side(df, df.T)
```

```
Out [10]:
```

	KOR	USA	2018	2019
2018	1	2	KOR	1
2019	3	4	USA	2
				4

```
In [11]: df.dot(df.T)
```

```
Out [11]:
```

	2018	2018
2018	5	11
2019	11	25

데이터프레임 결합

비슷한 데이터프레임 여러 개를 결합하거나, 서로 다른 데이터프레임을 결합하여 데이터를 분석할 경우 concat나 merge 함수를 사용한다. concat는 동일한 열 또는 행 라벨을 가진 여러 데이터프레임들을 세로 또는 가로로 결합할 때 사용한다. merge는 두 개의 데이터프레임이 하나(국가명) 혹은 두개 이상(국가명, 연도)의 공통된 열을 가지고 있을 때, 해당 열을 기준으로 데이터 프레임을 결합한다.

```
In [1]: df1 = pd.DataFrame({'USA':[1, 3], 'CHN':[2, 4])
df2 = pd.DataFrame({'JPN':[2, 4], 'KOR':[6, 8])
display_by_side(df1, df2)
```

```
Out [1]:
```

	USA	CHN	JPN	KOR
2018	1	2	2018	6
2019	3	4	2019	8

```
In [2]: pd.concat([df1, df2], axis = 1)
# pd.merge(df1, df2, left_index = True, right_index = True)
```

```
Out [2]:
```

	USA	CHN	JPN	KOR
2018	1	2	2	6
2019	3	4	4	8

일반적으로 concat는 연도, 국가, 기업 등 새로운 관측치가 추가될 때 주로 이용된다. 이 때 기존 관측치와 새로운 관측치는 동일한 변수들을 포함한다. merge는 동일한 시점, 국가, 기업에 대해 새로운 변수가 추가될 때 시점, 국가명 등을 기준으로 변수들을 결합할 때 주로 이용된다.

새로운 열 만들기

데이터프레임 구조는 스타타와 같이 각 열은 변수, 각 행은 시점/국가/개인별 관측치를 나타내는 것으로 구성하는 것이 효율적이다. 판다스에서는 아래와 같이 새로운 변수를 생성할 수 있다.

```
In [12]: df['Total'] = df.sum(axis = 1) # 열을 합하여 새로운 열 만들기  
df
```

```
Out [12]:  
      KOR   USA  Total  
2018     3     4      7  
2019     5     6     11
```

```
In [13]: df.div(df['Total'], axis = 0) * 100 # 'Total'대비 각 열의 비중
```

```
Out [13]:  
      KOR      USA  Total  
2018  0.4285  0.5714    100  
2019  0.4545  0.5454    100
```

숫자 데이터를 범주형 데이터로 변환

데이터프레임의 정수, 실수 등 숫자 데이터를 수의 크기에 따라 범주형 데이터로 전환하여 히스토그램을 그리거나 범주형 데이터 분석을 수행할 수 있다.

값을 미리 정의된 범위 ranges에 따라 나눌 경우 pd.cut(df.var, ranges), 전체 값을 n개의 균등한 범위로 나눌 경우 pd.qcut(pd.var, n)를 이용한다.

```
In [1]: from numpy.random import randint  
df = pd.DataFrame({'a':range(4), 'b':randint(1,10,4)})  
df
```

	a	b
0	0	5
Out [1]:	1	1
	2	2
	3	3
	2	2

```
In [2]: pd.cut(df.b, [0, 5, 10])
# 데이터프레임의 'b' 열 값을 (0, 5], (5, 10]으로 구분
# 가장 작은 값(0)을 포함하기 위해서는 include_lowest=True
# 작은 값 포함, 큰 값 제외는 right=False
```

```
Out [2]: 0 (0, 5]
          1 (5, 10]
          2 (0, 5]
          3 (0, 5]
Name: b, dtype: category
Categories (2, interval[int64]): [(0, 5] < (5, 10]]
```

```
In [3]: df.loc[:, 'c'] = pd.cut(df.b, [0, 5, 10], labels=False)
# labels=False은 범주 대신 숫자(순서) 반환
```

	a	b	c
0	0	5	0
Out [3]:	1	1	8
	2	2	3
	3	3	2
	2	2	0

```
In [4]: pd.qcut(df.b, 2)
```

```
Out [4]: 0 (4.0, 8.0]
          1 (4.0, 8.0]
          2 (1.999, 4.0]
          3 (1.999, 4.0]
Name: b, dtype: category
```

```
Categories (2, interval[float64]): [(1.999, 4.0] < (4.0, 8.0)]
```

특정 열 기준 그룹 지정

groupby는 데이터프레임의 특정 열을 기준으로 데이터프레임을 나누어 원하는 작업을 수행할 때 사용한다. 예를 들어, 연도 및 국가별 인구수와 경제규모GDP 데이터가 있을 때, 국가별 인구수, 경제규모의 합, 평균 등을 계산할 수 있다.

```
In [1]: df
```

```
Out [1]:
```

	country	pop	gdp
2016	USA	1	3
2016	KOR	2	4
2017	USA	3	5
2018	KOR	4	6

```
In [2]: df.groupby('country').sum()
```

```
Out [2]:
```

country	pop	gdp
USA	4	8
KOR	6	10

데이터프레임 특정 열의 값이 아니라, 값이 속한 범주에 따라 그룹을 지정하여 분석할 경우 pd.cut() 함수를 이용한다.

```
In [1]: df
```

	pop	gdp
0	1	3
Out [1]:	1	4
	2	5
	3	6

```
In [2]: df.groupby(pd.cut(df['pop'], [0, 2, 4]))['gdp'].mean()  
# pop이 0과 2사이, 2와 4 사이에 속한 국가들의 평균 gdp
```

```
Out [2]: pop  
(0, 2] 3.5  
(2, 4] 5.5  
Name: gdp, dtype: float64
```

lambda 함수

파이썬에서는 lambda 명령어를 이용하여 한줄 짜리 함수를 정의할 수 있다.

```
In [1]: square = lambda x: x * x  
square(2)
```

```
Out [1]: 4
```

데이터프레임에 lambda 함수 적용

데이터프레임 값을 원하는 포맷으로 표시하거나 반복적인 계산을 간단하게 하기 위해 lambda 함수를 이용한다. 데이터프레임의 각 행 또는 열을 대상으로 lambda 함수를 적용하려는 경우 apply, 데이터프레임의 각 항목을 대상으로 함수를 적용하려는 경우 applymap, 시리즈의 각 항목을 대상으로 함수를 적용하려는 경우 map 함수를 사용한다.

```
In [1]: df
```

	country	pop	gdp
2016	USA	1	3
Out [1]:	2016	KOR	2
	2017	USA	3
	2018	KOR	4

```
In [2]: df[['pop', 'gdp']].apply(lambda x: x.div(x.sum()))
```

	pop	gdp
0	0.1	0.166667
Out [2]:	1	0.2
	2	0.3
	3	0.4

```
In [3]: df = df[['pop', 'gdp']].apply(lambda x: x.div(x.sum()))
df.applymap(lambda x: f'{x:.2f}'')
```

	pop	gdp
0	0.10	0.17
Out [3]:	1	0.20
	2	0.30
	3	0.40

스택/언스택

데이터프레임의 행 또는 열을 다른 축(axis)으로 이동시키려는 경우 스택/언스택 stack/unstack을 이용한다. 만약, 행 또는 열이 멀티인덱스인 경우 이동하려는 행 또는 열의 레벨을 지정한다.

```
In [1]: df
```

	country		pop	gdp
Out [1]:	2016	USA	1	3
	2016	KOR	2	4
	2017	USA	3	5
	2017	KOR	4	6

In [2]: df.unstack(level = 1)

Out [2]:	pop		gdp	
	KOR	USA	KOR	USA
2016	2	1	4	3
2017	4	3	6	5

값 변경

외부에서 입수한 문자열 형식의 데이터는 지나치게 길거나 의미를 파악하기 어려운 용어, 일관되지 않은 표기 등의 이유로 바로 사용하기 어려운 경우가 많다. 이 경우 `replace()` 또는 `rename()` 함수를 이용하여 데이터프레임의 값, 행과 열 라벨 등을 일괄적으로 변경할 수 있다.¹⁷

먼저, `replace()`는 데이터프레임의 숫자, 문자열을 변경하는데 사용된다.¹⁸ 데이터프레임 df의 여러 문자열('pat1', 'pat2', ...)을 다른 문자열('repl1', 'repl2', ...)로 변경하고자 할 경우 문자열 변경 내역을 딕셔너리로 정리하여 `df = df.replace({'pat1': 'repl1', 'pat2': 'repl2', ...})` 명령을 수행한다. 만약, 'KOR:GDP'에서 'KOR_GDP'으로 데이터프레임의 문자열중 일부분만 변경할 경우 `df.replace(':', '_', regex=True)`와 같이 변경하려는 문자열과 `regex=True` 옵션을 지정한다.

데이터프레임의 한 개 열, 또는 행이나 열 라벨에 포함된 문자열을 변경할 때는 문자열 처리에 특화된 `str.replace()`을 이용한다.¹⁹ `str.replace()`은 기본

¹⁷ 특정 위치에 있는 항목 값을 변경하고자 할 경우 `.loc[]` 또는 `.iloc[]`을 이용한다.

¹⁸ 숫자의 경우 문자열에 비해 변경 과정이 간단하므로, 문자열 변경을 중심으로 설명한다.

¹⁹ `Series.str()`, `Index.str()`은 시리즈, 인덱스의 각 항목에 대한 문자열 함수 벡터이다(vectorized string functions for Series and Index). 문자열 함수는 `replace`, `upper`,

적으로 문자열의 일부만 변경하는 것이 가능하며, 다양한 문자열 처리 기능을 사용하도록 되어 있다(regex=True). 데이터프레임의 'country' 열에 있는 문자열에서 'KOR:GDP'를 'KOR_GDP'와 같이 변경하려는 경우 df['country'] = df['country'].str.replace(':', '_') 명령을 실행한다.

데이터프레임의 행이나 열 라벨에 포함된 문자열을 변경하고자 할 때 rename()을 이용할 수도 있다. 열 라벨에 있는 문자열('pat1', 'pat2', ...)을 ('repl1', 'repl2', ...)으로 변경할 때는 df = df.rename(columns = dict)와 같이 한다. 만약, 데이터프레임이 멀티인덱스이고 첫 번째 레벨의 열 라벨을 앞에서 와 같이 변경할 때는 df = df.rename(columns = dict, level = 0)과 같이 변경하려는 열 레벨을 지정한다.

시계열

판다스에서 데이터프레임의 인덱스를 데이트타임인덱스 DatetimeIndex 자료형으로 지정하면 시계열 빈도 변경 resample, 증가율/차분 pct_change() / diff(), 특정 연도/월 선택 등 시계열 분석에 유용한 여러 기능을 이용할 수 있다.

In [1]: df # 분기별 자료

	consumption	gdp
2017-03-31	1	2
Out [1]: 2017-06-30	2	4
	:	:
2018-12-31	8	16

In [2]: type(df.index)

Out [2]: DatetimeIndex

In [3]: df.resample('A').sum() # 'M' 월, 'Q' 분기, '2A' 2년, ...
first(), last(), mean(), max(), min(), interpolate(), ...

lower, isdigit, count, strip, split 등이 있다.

```
Out [3]:   consumption    gdp
           2017-12-31      10      20
           2018-12-31      26      52
```

엑셀, csv 파일 등으로 입수한 데이터에 날짜를 나타내는 행이나 열이 있으면 판다스의 `to_datetime()`를 이용하여 데이트타임 자료형으로 지정할 수 있다. 이 때, 날짜는 '2020-12-31', '20201231', '31 Dec 2020', '2020Q4' 등의 문자열이어야 한다. 입수한 파일의 날짜 정보를 `to_datetime()`가 데이트타임 자료형으로 지정하지 못할 경우 위와 같은 형태의 문자열로 변경한 다음 다시 시도한다. 데이트타임 자료형인 데이터프레임 열을 `set_index()`를 이용하여 인덱스로 지정하면 데이트타임인덱스가 된다.

```
In [1]: df
```

```
Out [1]:   value      date
           1     2  20201201
           2     4  20201202
           3     5  20201203
```

```
In [2]: df.date = pd.to_datetime(df.date)
df.date
```

```
Out [2]: 0 2020-12-01
1 2020-12-02
2 2020-12-03
Name: date, dtype: datetime64[ns]
```

```
In [3]: df = df.set_index('date')
df
```

		value
	date	
Out [3]:	2020-12-01	2
	2020-12-02	4
	2020-12-03	5

여러 출처에서 입수한 데이터를 결합하여 분석할 경우 자주 발생하는 오류는 같은 연도별/분기별/월별 데이터인데 데이터 출처에 따라 서로 다른 월이나 일 기준으로 날짜가 표기된 경우이다. 이 경우 판다스의 YearEnd(), YearBegin(), QuarterEnd(), QuarterBegin(), MonthEnd(), MonthBegin() 등을 이용하여 날짜 표기를 일치시킬 수 있다.

```
In [4]: from pandas.tseries.offsets import YearEnd, YearBegin
df.index + YearEnd()
```

		value
	date	
Out [4]:	2020-12-31	2
	2020-12-31	4
	2020-12-31	5

```
In [5]: df.index + YearBegin()
```

		value
	date	
Out [5]:	2020-01-01	2
	2020-01-01	4
	2020-01-01	5

시계열 데이터의 연도나 분기, 월별 속성을 파악하거나, 스타타에서 데이터를 분석하기 위해 각 행의 연도, 분기, 월을 나타내는 열을 새로 생성하는 경우가 있다. 이 때 데이트타임 자료형의 열(예, date)로부터 df.date.dt.year, df.date.dt.month와 같이 연도, 분기, 월 등을 추출할 수 있다.

데이터프레임 쓰기/읽기

판다스에서는 다양한 형태로 입수한 데이터를 데이터프레임 자료형으로 변환하여 저장하고, 분석 결과 역시 데이터프레임 형태로 저장한다. 데이터프레임을 파일로 쓰기는 `df.to_pickle()`, 저장된 데이터프레임 읽기는 `pd.read_pickle()` 함수를 이용한다.²⁰

```
In [1]: import os  
os.mkdir('pickle') # 작업 폴더 아래 'pickle' 폴더 생성  
df.to_pickle('pickle/df1.pkl')
```

```
In [2]: df2 = pd.read_pickle('pickle/df1.pkl')
```

판다스는 엑셀, 스타타, CSV, TXT 등 대부분 파일 포맷을 데이터프레임으로 읽고 쓰는 함수를 제공한다.

```
In [1]: df1 = pd.read_excel('input/f1.xlsx', sheet_name='Sheet1')  
# skiprows, header, dtype  
df2 = pd.read_csv('input/f2.csv')  
df3 = pd.read_stata('input/f3.dta')
```

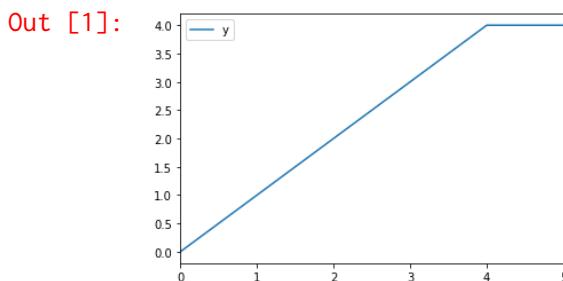
하나의 데이터프레임 `df`을 `df1.to_excel('file')`을 이용하여 excel 파일로 저장하였다가 다시 `df2=pd.read_excel('file')`으로 불러들이면 두 데이터프레임 `df1`, `df2`은 서로 일치하지 않는다. 이는 데이터프레임을 엑셀파일로 저장할 때 데이터프레임의 인덱스가 엑셀 파일의 첫번째 열로 저장되고 이후 다시 불러올 때 새로운 데이터프레임 열로 인식되기 때문이다. 원 데이터프레임의 자료구조를 그대로 갖도록 하기 위해서는 `df2=pd.read_excel('file', index_col=0)`와 같이 엑셀파일의 첫번째 열이 인덱스임을 명시한다.

그래프

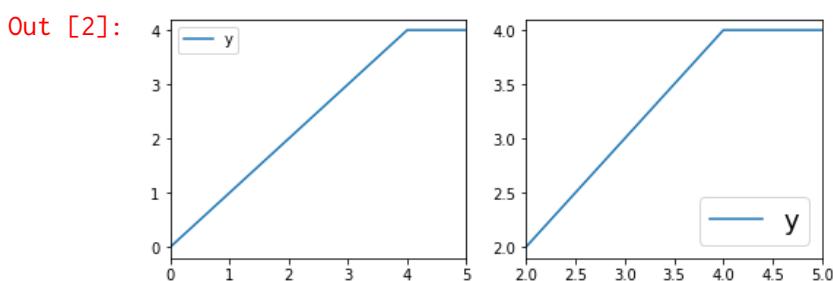
²⁰파일을 읽거나 저장할 때 우분투 운영체제인 BReiT Python에서는 'folder1/folder2/filename.abc'와 같이 파일경로를 지정하면 되지만, 일부 운영체제에서는 오류가 발생한다. 운영체제와 관계없이 파일경로를 지정하기 위해서는 `os` 라이브러리를 이용하여 `os.path.join('folder1', 'folder2', 'filename.abc')`와 같이 지정한다.

판다스는 그래프 그리기 라이브러리인 `matplotlib`에 기반하여 작성되어 있기 때문에, 자체적으로 그래프 그리기 기능을 제공한다. 단순한 선 그래프 뿐만 아니라 다양한 형식의 그래프를 지원하며, 여러 그래프를 나란히 두고 비교할 수 있도록 `subplot`을 지원한다. 그래프 크기와 범례, 축서식, 범위 등 그래프의 모든 요소를 사용자가 원하는대로 조정할 수 있다.

```
In [1]: %matplotlib inline  
# 그래프 확대, 크기 조정 기능은 %matplotlib notebook  
df = pd.DataFrame([0, 1, 2, 3, 4, 4], columns=['y'])  
df.plot();
```



```
In [2]: import matplotlib.pyplot as plt  
# subplot, legend, axis 관리를 위해 matplotlib.pyplot 이용  
  
fig, ax = plt.subplots(1, 2, figsize=(8, 3))  
df.plot(ax=ax[0])  
df.loc[2: ].plot(ax=ax[1])  
ax[1].legend(loc=4, fontsize=18);
```



f-string을 이용하여 그래프 x 축과 y 축 값의 표시 형식을 지정할 수 있다.

```
In [3]: from matplotlib.ticker import FuncFormatter as FF

fig, ax = plt.subplots(1, 2, figsize=(8, 3))
df.plot(ax=ax[0])
df.loc[2:].plot(ax=ax[1])
ax[1].legend(loc=4, fontsize=18)
dp2 = lambda x, p: f'{x:.2f}' # p is not used
ax[1].yaxis.set_major_formatter(FF(dp2))
ax[1].xaxis.set_major_formatter(FF(dp2))
```

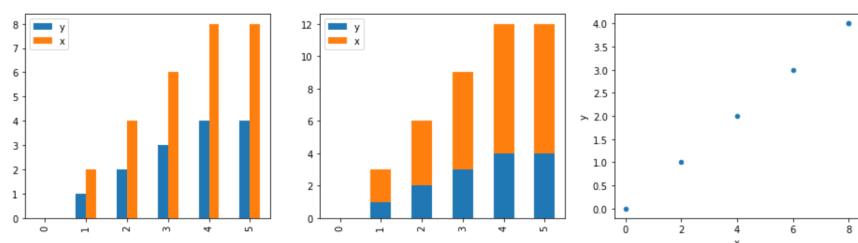
막대 및 누적 막대 그래프(bar chart), 산포도(scatter plot)는 `df.plot.bar()`, `df.plot.bar(stacked=True)`, `df.plot.scatter(x='var1', y='var2')` 등 명령어를 이용한다.

```
In [4]: df['x'] = df['y'] * 2

fig, ax = plt.subplots(1, 3, figsize=(16, 4))

df.plot.bar(ax=ax[0])
df.plot.bar(stacked=True, ax=ax[1])
df.plot.scatter(x='x', y='y', ax=ax[2])
```

Out [4]:



시계열 데이터프레임(데이터타임인덱스)의 경우 선 그래프는 x축의 날짜가 보기 좋게 표시되지만, 막대 그래프는 x축의 날짜가 데이터타임 자료형 그대로 (연도-월-일 시간:분:초) 표시되어 가독성이 떨어진다. 이 경우 x축 눈금이름

`ticklabels`의 표시 형식을 직접 지정할 수 있다. 시계열 자료를 막대 그래프로 표시하는 경우가 많다면 함수로 정의하여 편하게 사용할 수 있다.

```
In [5]: fig, ax = plt.subplots(1, 1)
df.plot.bar(ax=ax)

ticklabels = [pd.to_datetime(t.get_text()).strftime('%y')
             for t in ax.get_xticklabels()]
ax.set_xticklabels(ticklabels, rotation=0)
```

```
In [6]: def xaxis_date_format(ax, fmt = '%y'):
    '''fmt= '%b \n %y', '%y-%m-%d', etc '''
    ticklabels = [pd.to_datetime(t.get_text()).strftime('%y')
                  for t in ax.get_xticklabels()]
    ax.set_xticklabels(ticklabels, rotation=0)

fig, ax = plt.subplots(1, 1)
df.plot.bar(ax=ax)
xaxis_date_format(ax, '%y')
```

한글

파이썬에서 변수명이 한글인 경우 그래프 범례가 제대로 표시되지 않는 경우가 있다. 이 때 시스템의 한글폰트 파일경로와 `matplotlib`의 `font_manager`, `rc`를 이용하여 한글폰트를 지정한다. 또한, 한글폰트를 지정하고 나면, 데이터에 마이너스 값이 있는 경우 축 레이블에서 마이너스('-) 부호가 깨져 보인다. 이를 방지하기 위해 '`axes.unicode_minus`' 옵션을 `False`로 지정한다.

```
In [1]: import matplotlib.pyplot as plt

plt.rc('font',family='NanumGothic')
plt.rcParams['axes.unicode_minus'] = False
```

```
# import numpy as np  
# pd.DataFrame(np.random.randn(100), columns=['소득']).plot()
```

파이썬 matplotlib에서 지정할 수 있는 폰트중 나눔글꼴의 이름과 파일 경로는 다음과 같이 확인할 수 있다.

```
In [2]: from matplotlib import font_manager as fm  
  
for f in fm.fontManager.ttflist:  
    if 'Nanum' in f.name:  
        print(f.name, '\t', f.fname)
```

```
NanumBarunpen C:\Windows\Fonts\NanumBarunpenR.ttf  
NanumMyeongjo C:\Windows\Fonts\NanumMyeongjoBold.ttf  
NanumGothic C:\Windows\Fonts\NanumGothicBold.ttf  
...
```

한글 폰트가 없는 경우 원하는 글꼴을 직접 설치하고 matplotlib에게 알려주어야 한다. BRiT Python의 경우 나눔글꼴이 설치되어 있다. Google Colab 또는 리눅스의 경우 아래와 같이 나눔글꼴을 설치한 다음 matplotlib의 글꼴 리스트를 다시 구성(rebuild)하여²¹ matplotlib이 새로운 폰트를 사용할 수 있도록 한다. 그 다음 주피터노트북을 다시 시작한다. 윈도우, 맥 컴퓨터의 경우 나눔글꼴 설치 방법을 쉽게 찾을 수 있다.

```
In [2]: !sudo apt-get install -y fonts-nanum  
  
from matplotlib import font_manager as fm  
fm._rebuild()
```

²¹동 명령어는 matplotlib 캐시폴더(matplotlib.get_cachedir())를 실행하여 확인)내에 있는 모든 파일을 삭제하고 다시 생성하는 역할을 수행한다.

4 파이썬 활용하기

이제까지 파이썬을 이용하여 데이터를 분석하기 위한 준비를 마쳤다. 주피터노트북을 실행한 다음 필요한 라이브러리를 불러오고, 그래프가 적절히 표시되도록 한글폰트 등을 지정하는 명령어를 실행한다. 만약, 불러와야 할 라이브러리나 자주 쓰는 사용자 정의 함수가 있으면 해당 파이썬 코드를 파이썬 스크립트(확장자 .py)로 저장해둔 다음 노트북 첫번째 셀에서 스크립트 파일을 실행하여 수고를 덜 수 있다.

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

        import matplotlib.pyplot as plt

        plt.rc('font',family='NanumGothic')
        plt.rcParams['axes.unicode_minus'] = False

        def display_by_side(*args):
            # 두 개 데이터프레임을 옆으로 나란히 표시하는 함수
            html_str = ''
            for df in args:
                html_str += df.to_html()
            display_html(html_str.replace('table','table style="display:inline"'), raw=True)

        %matplotlib inline
```

```
In [1]: %run "prep.py" # tutorial.ipynb와 같은 폴더(python2021)에 저장

        %matplotlib inline
```

4.1 작업 환경 갖추기

데이터 분석 과정에서 동일한 작업을 반복하여 수행할 경우 동 작업을 함수로 정의하여 필요할 때마다 호출하거나, `for-loop` 구문을 작성하여 일괄 실행하는 것이 효율적이다. 또한, 문자열 형태의 자료를 기억하기 쉬운 단축명으로 대체하는 딕셔너리를 미리 저장해두면 분석 과정에서 수시로 활용할 수 있다.

사용자 정의 함수와 딕셔너리를 스크립트 파일을 관리하기 위한 에디터 프로그램으로 마이크로소프트의 Visual Studio Code(VS Code)를 추천한다. 무료 프로그램이며, 파이썬 문법에 따라 파이썬 명령어와 변수, 숫자들이 하이라이트되어 읽기 편하다.

VS Code

```
파일(F) 편집(E) 선택(M) 보기(V) 이동(I) 실행(R) 터미널(T) 도움말(H)
basic.py | labor here - Visual Studio Code

상위 폴더
... + basicspx x
modules + basicspx.py
1 # 아래 파일은 경이보라인을 IMPORT
2 import numpy as np
3 import pandas as pd
4 # last updated : 2019-09-27
5
6 import pandas as pd
7 import numpy as np
8 import re
9 import math
10 import json
11 import pickle
12 import platform
13 import webbrowser
14
15 from pandas import Timestamp as stamp
16
17 import matplotlib
18 import matplotlib.pyplot as plt
19 import matplotlib.ticker as tickier
20 import matplotlib.dates as ndates
21
22 from urllib.request import urlopen
23 from IPython.display import HTML, IFrame, Image, display, display_html
24 from pandas.tseries.offsets import YearEnd, QuarterEnd, MonthEnd, YearBegin, QuarterBegin, MonthBegin
25
26
27 #####
28 ##### Pre-Imports #####
29 #####
30
31 # pandas
32 idx = pd.IndexSlice
33 pd.set_option('precision=4')
34 pd.set_option('display.max_rows', 80)
35 pd.set_option('display.max_colwidth', 100)
36 pd.set_option('display.float_format', lambda x: '{:.2f}'.format(x))
37 # pd.set_option('display.float_format', lambda x: '{:.0f}'.format(x))
38 # if abs(x) < 100 and x<0 then else '{:.0f}'.format(x))
39 pd.set_option('precision', 4)
40
41
42 # colors
43 colors = list(matplotlib.colors.cnames.keys())
44
45
46 # 같은 맵핑과 스타일은 그대로 써 두세요
47 colors = [(85/255, 142/255, 213/255), (110/255, 110/255, 110/255),
48 (175/255, 175/255, 195/255), (250/255, 192/255, 145/255),
49 (150/255, 115/255, 170/255)]
```

데이터를 여러 단계에 걸쳐 검토하고 분석하는 과정에서 기초 데이터, 파이썬 스크립트 파일, 데이터프레임 pickle 파일, 그래프 등을 수시로 읽어오고 변경하고 저장하게 된다. 이들 데이터를 체계적으로 관리하는 하나의 방법은 작업 폴더 아래 노트북 파일을 두고, 아래와 같이 파일 종류별로 여러 개의 폴더를 생성하여 파일들을 저장하는 것이다.

- input: 데이터 소스에서 입수한 xlsx, csv, dta, txt 파일

- `script`: 라이브러리, 사용자 정의 함수 등의 파이썬 스크립트 파일
- `pickle`: 데이터프레임 피클 파일
- `output`: 분석 결과를 정리한 엑셀, 이미지 파일
- `stata`: 스타타 데이터 포맷 파일, do-파일

노트북 파일, 데이터 폴더, 스크립트 폴더, 피클 폴더 등이 저장된 작업 폴더(예, python2021)를 압축하여 다른 컴퓨터에 옮기거나 다른 연구자에게 공유하면 언제 어디서나 동일한 작업을 수행할 수 있다. 또한, 데이터 분석 업무가 끝난 다음 자료를 업데이트하거나 진행중인 작업을 새로운 연구자가 이어받을 때 위와 같이 정리된 작업 폴더가 있으면, 노트북 파일을 통해 데이터 입수 및 처리, 계량 분석이 어떻게 이루어졌는지 손쉽게 파악할 수 있다.

4.2 데이터 입수, 전처리, 시각화

본 절은 BIS, 한국부동산원, 한국은행 ECOS 등에서 데이터를 입수하고 분석하는 과정을 다룬다.²² Fred, World Bank 등의 경우 open API를 이용하여 데이터 조회, 다운로드 기능을 제공하는 파이썬 라이브러리가 있다. 동 라이브러리 설치와 사용법은 본 연수에서 다루지 않지만, 인터넷 검색을 통해 쉽게 찾아볼 수 있다.

국제결제은행

BIS 웹페이지는 GDP 대비 신용 비율, 환율, 자산 가격, DSR 등 다양한 데이터를 엑셀 파일 형태로 제공하고 있다. 국가별, 부문별(민간, 기업, 가계 등) 신용 규모 엑셀 파일을 통 파일의 url 주소를 이용하여 데이터프레임으로 내려받고 정리한다. BIS의 주제별 엑셀 파일 주소는 항상 동일하므로, 시간이 지나 엑셀 파일에 새로운 관측치가 추가된 경우 주피터노트북의 엑셀 파일 다운로드 셀을 실행하면 업데이트된 파일을 바로 다운로드할 수 있다.

엑셀 파일 형태로 제공되는 BIS 데이터는 파일의 'Quarterly Series' 시트에 있으며(`sheet_name='Quarterly Series'`), 각 시계열 항목의 이름이 주기: 국가:항목1:항목2:... 와 같은 형식으로 주어져 있다. 내려받은 엑셀 파일을

²²노트북 파일은 [깃허브](#)에서 다운로드 받을 수 있다.

BIS statistics 웹페이지

The screenshot shows the BIS statistics website. At the top, there's a navigation bar with links for Home, Statistics - About, Central bank hub, Statistics, Banking services, and Media & speeches. Below the navigation is a search bar labeled 'Search the website' with a magnifying glass icon. The main content area has a sidebar on the left with categories like Statistics, About, Tools and support, International banking, Debt securities, Credit, Global liquidity, Derivatives, Foreign exchange, Property prices, Payment systems, and Other indicators. Each category has a red arrow icon to its right. The main content area contains sections for 'About BIS statistics' (with a brief description), 'Access our data' (links to BIS Statistics Explorer, BIS Statistics Warehouse, CSV files, and Terms of permitted use of BIS statistics), and 'Latest data releases' (a table with rows for various statistics releases from December 2020). A small note at the bottom of the sidebar says 'Open "https://www.bis.org/statistics/totcredit.htm" in a new tab'.

판다스 데이터프레임으로 불러올 때, 첫 3개 행은 제외하여(`skiprows=3`) 4번째 행이 데이터프레임 칼럼 라벨로 지정되도록 한다. 칼럼 라벨을 `str.split('::')`으로 나누어 멀티인덱스로 지정하면 원하는 시계열 데이터에 빠르게 접근할 수 있다.

파일의 첫번째 칼럼은 날짜`Period`를 나타낸다. 날짜가 `dd.mm.yyyy`형태로 표기되어 있어, 첫번째 열을 데이터프레임의 인덱스로 지정하여(`index_col=0`) 불러오면 자동으로 판다스 데이터타임인덱스 자료형으로 변환된다.

데이터프레임 칼럼은 3개 레벨로 구성되어 있다. 각 레벨에 속한 라벨은 `df.columns.get_level_values()`를 이용하여 확인할 수 있다. 예를 들어, 칼럼의 세번째 레벨에는 민간, 비금융법인, 가계에 해당하는 P, N, H 라벨이 있다.

BIS DSR 엑셀파일

	A	B	C	D	E	F	G	H	I	J
1		Australia - Households and NPISHs	Australia - Non-financial corporations	Australia - Private non-financial sector	Belgium - Households and NPISHs	Belgium - Non-financial corporations	Belgium - Private non-financial sector	Brazil - Private non-financial sector	Canada - Households and NPISHs	Canada - Non-financial corporations
2		Per Cent (Units)	Per Cent (Units)	Per Cent (Units)	Per Cent (Units)	Per Cent (Units)	Per Cent (Units)	Per Cent (Units)	Per Cent (Units)	Per Cent (Units)
3		Australia	Australia	Australia	Belgium	Belgium	Brazil	Canada	Canada	Canada
4	Period	Q:AU:H	Q:AU:N	Q:AU:P	Q:BE:H	Q:BE:N	Q:BE:P	Q:BR:P	Q:CA:H	Q:CA:N
43	30.09.2008	17.2	52.9	23.8	6.5	44.7	18.7	16.7	13.0	38.6
44	31.12.2008	16.4	51.2	23.0	6.5	47.3	19.3	19.9	12.7	40.3
45	31.03.2009	15.9	48.1	22.1	6.6	49.2	19.4	17.9	12.4	41.1
46	30.06.2009	15.2	45.2	21.0	6.6	53.1	19.9	16.0	12.4	44.7
47	30.09.2009	15.4	44.4	20.9	6.7	53.3	20.0	14.9	12.5	46.9
48	31.12.2009	15.6	44.3	21.0	6.7	52.0	19.9	14.6	12.5	46.8
49	31.03.2010	16.0	44.7	21.3	6.8	50.3	19.7	13.7	12.4	46.2
50	30.06.2010	16.6	43.8	21.7	6.8	50.7	19.8	13.3	12.6	44.5
51	30.09.2010	16.4	43.3	21.4	6.8	48.0	19.3	13.8	12.5	43.7
52	31.12.2010	16.8	42.2	21.6	6.8	46.3	19.0	13.9	12.6	42.0
53	31.03.2011	16.5	42.0	21.2	6.8	46.5	19.2	15.9	12.6	41.3
54	30.06.2011	16.6	41.7	21.2	6.9	47.1	19.5	16.7	12.5	41.2
55	30.09.2011	16.0	41.0	21.1	6.9	47.4	19.0	16.0	12.0	41.7

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

```
url = 'https://www.bis.org/statistics/dsr/dsr.xlsx'
df = pd.read_excel(url, sheet_name='Quarterly Series',
                    skiprows=3, index_col=0)
multiindex = df.columns.str.split(':').tolist()
df.columns = pd.MultiIndex.from_tuples(multiindex)

df.columns.get_level_values(2).unique()
```

```
Index(['H', 'N', 'P'], dtype='object')
```

한국부동산원

한국부동산원은 매월 광역시도별 공동주택 실거래가와 전국주택가격동향조사 결과를 공개한다. 전국주택가격 동향조사는 시군구별 주택유형별(아파트, 연립다세대, 단독 등) 전월세 및 매매가격지수, 전월세 및 매매 중위가격 등을 포함한다.

한국부동산원 부동산통계정보 웹페이지

번호	분류	제작	등록일	조회	첨부
10	지가변동률	전국지가변동률 조사 (2021년 09월)	2021.10.25.	12961	[파일]
9	전국주택가격동향조사	전국주택가격동향조사_월간 (2021년 10월)	2021.11.15.	17981	[파일]
8	주민이파드가격동향조사	주택가격동향조사 주간 이파드동향 (2021년 11월 12일)	2021.11.15.	12827	[파일]
7	월세가격동향조사	(구)월세가격동향조사 (2019년 06월)	2015.08.24.	1830	[파일]
6	실거래가지수	공동주택 실거래가지수 (2021년 05월)	2021.07.29.	5308	[파일]
5	임대료지수	상업용부동산 임대료지수 (2021년 3분기)	2021.10.28.	12501	[파일]

한국부동산원 웹페이지에서 전국주택가격동향조사 엑셀 파일을 다운로드한다음, 주피터노트북에서 데이터프레임으로 읽어온다. 한국부동산원의 주택가격통계 엑셀 파일은 시군구 지역 이름이 지역에 따라 서로 다른 방식으로 표기되어 있어, 판다스에서 바로 활용하기 어렵다. 이와 같이 주어진 지역 이름을 분석에 용이한 방식으로 정리하는 방법을 소개한다.²³

엑셀파일을 보면 첫번째 열은 전국, 수도권, 지방권, 5/6대광역시, 8/9개도 등 주요 권역과 서울, 부산, 경기도 등 17개 광역시도가 라벨로 주어져 있다. 두번째 열부터 네번째 열까지는 광역시도의 하위 지역분류 및 시군구가 있다. 네개 열에 걸쳐 표기된 지역이름을 주요 권역 또는 광역시도-시군구와 같이 단일 층위로 변경할 경우 데이터를 보다 손쉽게 분석에 활용할 수 있다. 이를

²³엑셀 파일로 다운로드받은 데이터는 데이터 제공 기관에 따라 다양한 형태로 데이터가 정리되어 있다. 계층이 있는 행과 열 라벨을 표시하는 방식, 행과 열에 해당하는 셀의 위치, 시계열 방향(가로/세로), 빈 값을 나타내는 문자열 등이 모두 다르다. 데이터 형태에 따라 다르겠지만, 가급적 데이터 제공 기관에서 기본 제공하는 방식으로 엑셀 파일을 내려받고, 내려받은 파일 그대로 파이썬에 불러오는 것이 좋다.

만약, 데이터를 다운로드받을 때 데이터 형태를 수정하거나, 다운로드받은 엑셀 파일은 수작업으로 수정할 경우 지금 바로 파이썬으로 들여야 할 수고는 많이 줄어들 수 있지만, 다음에 새로 업데이트된 파일을 다운로드받아 작업할 경우 매번 수작업을 반복해야하는 단점이 있다. 또한, 이 경우 데이터를 어떻게 정리했는지에 대한 기록을 남기기 어렵기 때문에, 이전에 했던 작업을 다시 재현하지 못할 위험도 있다.

위해 데이터프레임 빈항목 채우기(fillna()), 여러 칼럼에 있는 문자열 결합(.agg(' - '.join, axis=1)) 등 함수를 이용한다. 또한, '2000년 1월'과 같이 주어진 날짜 라벨을 값 대체를 통해 'yyyy-mm' 형태로 바꾸고, 데이트타임인덱스 자료형으로 변경한다.

월간 매매가격지수 종합

The screenshot shows an Excel spreadsheet titled "월간 매매가격지수_종합". The top row contains various menu options like Home, Insert, Draw, Page Layout, Formulas, Data, Review, View, and Tell me. Below the menu is a toolbar with icons for Cut, Copy, Paste, Format, etc. The main content starts with a header row containing parameters such as "통계표명: 월간 매매가격지수_종합", "수록기간: 2003년 11월 ~ 2020년 11월", "조회기간: 2003년 11월 ~ 2020년 11월", "출처: 한국감정원", "자료다운일자: 2020.12.01 10:41:57", and "단위: 지수". The data is presented in a table with columns for the year and month (e.g., 2003년 11월) and various regional and sectoral breakdowns. The table spans from row 11 to 21, with the first column being "지역" and the last column being "2004년 08월". The data values range from approximately 60 to 70.

한국은행 경제통계시스템

한국은행 경제통계시스템 데이터를 파이썬으로 입수하는 방법은 두 가지가 있다. 첫째 방법은 ECOS 웹페이지에서 원하는 통계를 찾은 다음, 엑셀 파일 형태로 다운로드 받는 것이다. 이렇게 다운로드 받은 파일을 판다스의 데이터프레임으로 불러온 다음 시계열 형식으로 정리한다.

두번째 방법은 ECOS open API를 이용하는 것이다. ECOS open API를 통해 ECOS의 거의 대부분 경제통계 데이터를 자동으로 입수할 수 있다. ECOS open API를 통해 데이터를 입수해보고, ECOS에서 원하는 통계 자료를 자동으로 입수하는 파이썬 함수`ecos()`를 설명한다.²⁴

²⁴한국은행 ECOS open API를 이용하여 자동으로 통계자료를 입수하기 위해서는 먼저 API

한국은행 ECOS OpenAPI 웹페이지

The screenshot shows the main interface of the ECOS OpenAPI website. At the top, there's a blue header bar with 'HOME | ECOS'. Below it is a navigation menu with tabs: '서비스 소개', '서비스 이용', '개발 가이드', '참여 소통', '이용 현황', and 'My Page'. The main content area features a large banner with a world map background. On the left, text reads: '경제통계 OPEN API는 한국은행이 작성·수집한 경제통계 정보를 보다 편리하게 활용할 수 있도록 외부에 공개한 인터페이스입니다.' To the right of the text is an illustration of an open book with charts and graphs. Below the banner are five circular icons representing different services: '사이트접속' (with a laptop icon), 'OPEN API 인증키 신청' (with a lock icon), 'OPEN API 검색 및 이용방법 확인' (with a globe icon), 'OPEN API를 이용 어플리케이션 제작' (with a smartphone icon), and '어플리케이션 등록' (with a smartphone icon). Further down, there are sections for 'OPEN API 기능 소개' (with a 'GO' button) and 'OPEN API 제공 목록' (with a 'GO' button). At the bottom, there are three tables: '공지사항', 'Q&A', and '변동내역알림', each with a '+' sign to expand more details.

다음으로, `ecos()` 함수를 이용하여 자금순환표의 부문별 금융부채 자료, 명목 GDP를 입수하고, credit/GDP 비율과 HP filter를 통한 추세와 순환변동을 계산하고, 보고서 수준의 그래프 그리는 과정을 설명한다. 보고서 수준의 그래프 그리기는 예시를 위한 것이며, 실제 데이터 분석 과정에서는 그래프 꾸미기에 너무 많은 신경을 쓰지 않는 것이 좋다. 어차피, 최종 보고서에 들어갈 그래프는 엑셀로 그린다고 생각하는 것이 편하다.

국제투입산출표

마지막으로 첫 시간에 소개한 국제투입산출표를 보다 자세히 살펴본다. 엑셀 자료를 읽어들여 자료 구조에 맞는 형태로 정리하고, 다양한 판다스 및 주피터 노트북 기능을 이용하여 데이터를 분석하고, 데이터 분석 과정을 문서화하는 방법을 알아본다.

접근을 위한 인증키를 받아야 한다. 인증키 신청 및 open API 이용 방법은 [ECOS open API](#)에서 확인할 수 있다.

5 파이썬과 스타타

여러 분야에서 파이썬의 활용도가 높아지고 있지만, 아직 계량 분석에 있어서는 파이썬보다 스타타가 월등히 간편하고 빠르다. 물론, 스타타의 계량 분석 도구들을 파이썬에서 구현할 수 있겠지만, 그 결과물이 스타타보다 빠를 것 같지 않고 신뢰성도 확보하기 어렵다. 굳이 바퀴를 다시 발명한 필요는 없다.²⁵

파이썬과 스타타는 각각의 장단점을 갖고 있다. 계량 분석은 스타타를 이용하고, 스타타 계량 분석 이전 및 이후의 데이터 관리는 파이썬을 이용하는 것이 효율적이다. 파이썬과 스타타를 어떻게 함께 사용하는지 간단히 설명하고, 실제 분석 사례를 살펴본다.

5.1 파이썬에서 스타타로

판다스는 데이터프레임을 스타타 파일 형식으로 저장하고(.to_stata()), 다시 데이터프레임으로 읽을 수 있다(.read_stata()). 이때 데이터프레임에 한글 문자열이 있으면 오류가 발생하므로 한글 문자열은 모두 영문이나 숫자로 바꾸어준다. 또한, 스타타는 판다스의 데이트타임 자료형을 인식하지 못하므로 연도, 월, 일 등을 데이터프레임의 새로운 컬럼으로 생성해둔다.

스타타가 제공하는 함수는 대체로 같은 기능을 하는 판다스 보다 빠르다. 그러므로, 파이썬내에서 너무 오랜 시간이 걸리거나 계량분석 이후 사용하지 않는 변수를 생성하는 일은 스타타내에서 수행하는 것이 좋다. 만약, 스타타에서 제공하는 함수인데, 시간이 오래 걸린다면, 파이썬에서 시도해보는 것도 좋다. 그룹내 분위값 생성²⁶과 같은 작업은 파이썬이 더 빠르다.

한편, 스타타는 한번에 하나의 작업 파일만을 대상으로 작업을 수행한다. 이에 반해, 파이썬 판다스는 데이터프레임의 갯수에 제한이 없다. 이 때문에 여러 기간, 국가, 변수 데이터를 결합하고 정리하는 작업은 판다스가 월등히 편하다.

²⁵주피터노트북내에서 같은 컴퓨터에 설치된 스타타에 스타타 명령어를 보내고, 스타타 분석 결과를 다시 주피터노트북내에서 확인하는 방법도 있다. 그러나, 본격적인 계량 분석을 수행하기에는 비효율적이고 이용 가능한 명령어도 제한이 많아 추천하지 않는다.

²⁶스타타의 xtile()과 같은 함수는 변수의 그룹내 분위값을 계산한다. 판다스의 qcut()을 이용하여 동일한 작업을 수행할 수 있다.

df.to_stata()

```

STATA .dta 파일 export

In [124]: kis = kis[kis.자본총계 > 0]      # 2014년 기준 26808개 기업중 3056개 기업의 자본총계가マイナス
kis = kis[kis.매출액 > 0]      # 2014년 기준 10건이 < 0
kis = kis[kis.유동자산 > 0]      # 2014년 기준 4건이 < 0
kis = kis[kis.비유동자산 > 0]      # 2014년 기준 1건이 < 0

age_qvl = ['AGE', 'sa', 'ea', 'cash', 'liq', 'qasset', 'VL', 'qvl']
mf_prod = [(mf + str(i)) for i in range(0, 5)] + [prod + str(i) for i in range(0, 8)]
            + [qvl + str(i) for i in range(0, 6)]

dta = kis[[DATE, 'KIS', 'KSIC2', '주거래은행', '시장', '기업규모', 'DEBT', '자산총계', '감가상각비',
           '매출액', '유동자산'] + age_qvl + mf_prod].copy()

dta.columns = ['DATE', 'KIS', 'KSIC', 'KSIC2', 'BANK', 'MKT', 'SIZE', 'debt', 'asset', 'dep',
               'sales', 'TASSET'] + age_qvl + mf_prod

dta = dta.drop(['BANK', 'MKT'], axis = 1)

# # 소속시장, 현재 기업 상태를 영문으로 변경
# mkt_k = ['코스닥상장', '외국인', '코스닥판권', '상장', '코넥스', '상장관리', '폐업', '파출수합병', '일반',
#           '공모관', '기타', '개인', '비영리단체']
# mkt_e = ['KO', 'EA', 'KOI', 'KT', 'KK', 'KII', 'CL', 'MA', 'OD', 'PU', 'OT', 'PR', 'NP']
# dta.loc[:, 'MKT'].replace(dict(zip(mkt_k, mkt_e)))

# # 14년 취약성 수준(BIS 비율)에 따라 국민·신한·하나·우리·산업·기업:1-2-3-4-5-6 으로 지정
# dta['BANK'] = dta['BANK'].replace(dict(zip(['국민', '신한', '하나', '우리', '산업', '기업'], [0, 1, 2, 3, 4, 5])))
# dta.loc[~dta.BANK.isin([0, 1, 2, 3, 4, 5]), 'BANK'] = 6

dta.loc[:, ['SIZE']] = dta.loc[:, ['SIZE']].replace({'대기업':0, '중소기업':1, '기타':2})

dta.DATE = pd.DatetimeIndex(dta.DATE).year.astype(str)
dta.DATE = dta.DATE.str[2:]

dta = dta.set_index(['DATE', 'KIS', 'KSIC', 'KSIC2', 'SIZE'])

dta.to_stata('stata//ffdc.dta')

```

In [1]:

In [1]:

5.2 스타타에서 파이썬으로

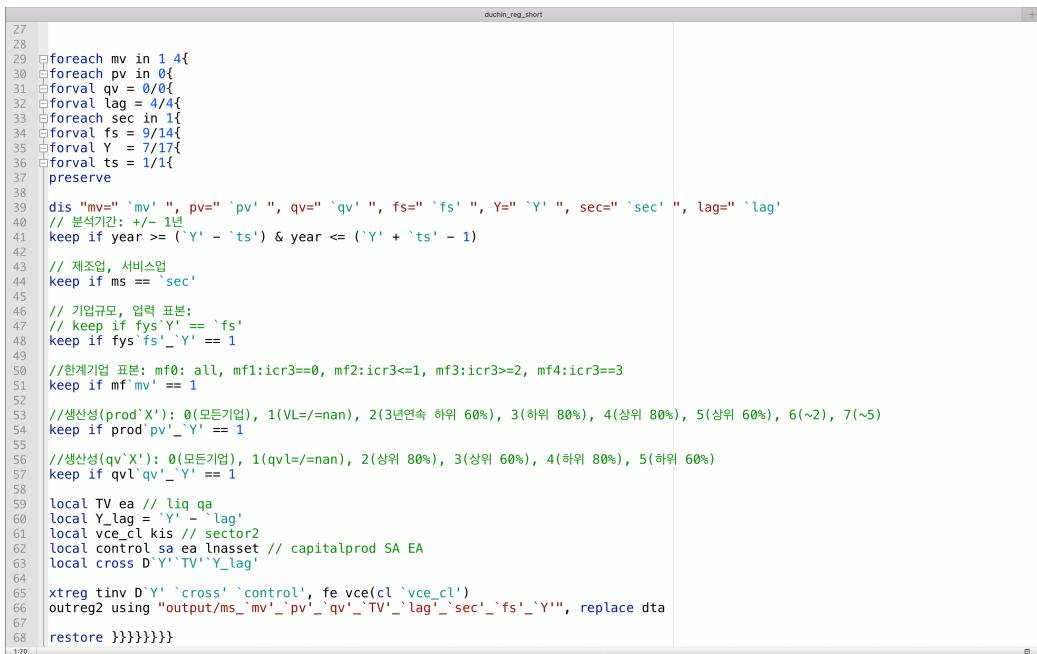
대규모 데이터를 이용한 분석에서는 다양한 분석 기간, 변수 선택, 모형 설정 등에 대한 계량 분석을 통해 분석 결과의 강건성을 검증한다. 또한, 다각적인 계량 분석 과정을 거치면서 가설 설정 단계에서 미처 생각하지 못한 새로운 가설 또는 분석 주제에 대한 아이디어를 얻기도 한다. 이처럼 여러 계량 분석 결과들 사이의 관계나 패턴을 찾아내기 위해서는 분석 결과들이 한눈에 들어오도록 정리하는 것이 필수적이다.

스타타는 하나의 계량 모형을 빠르게 분석할 수 있지만, 분석 결과를 시각화하고 여러 분석 결과를 한 곳에 모아 보기 좋게 정리하기가 어렵다. 물론, 파이썬에도 스타타 분석 결과를 바로 불러와 정리하는 함수는 없다. 분석 모형, 데이터 범위에 따라 계량 분석 결과의 형태도 달라지므로 파이썬을 이용하더라도 분석 결과를 어떻게 취합하고 정리할지는 매번 연구자가 직접 고안하고 코딩하는 과정을 거쳐야한다.

그러나, 다행스럽게도 파이썬의 빠른 코딩 속도, 강력한 문자열 처리 기능은

스타타 분석 결과를 쉽고 빠르게 정리하도록 도와준다. 한 가지 방법은 스타타 분석 결과를 텍스트 파일로 저장한 다음²⁷ 파이썬의 문자열 처리 함수를 이용하여 정리하는 것이다. 파이썬에서의 작업을 고려하여, 스타타에서 변수명을 일관된 규칙에 따라 생성하면 전체 과정을 보다 효율적으로 진행할 수 있다.

스타타 do-file



```

27
28
29 foreach mv in 1 4{
30   foreach pv in 0{
31     foreach qv = 0/0{
32       foreach lag = 4/4{
33         foreach sec in 1{
34           foreach ts = 9/14{
35             foreach Y = 7/17{
36               foreach fs = 1/1{
37                 preserve
38
39                 dis "mv=""`mv'"", pv=""`pv'"", qv=""`qv'"", fs=""`fs'"", Y=""`Y'"", sec=""`sec'"", lag=""`lag'"
40 // 분석기간: +/- 1년
41 keep if year >= (`Y' - `ts') & year <= (`Y' + `ts' - 1)
42
43 // 제조업, 서비스업
44 keep if ms == `sec'
45
46 // 기업규모, 입력 표본:
47 // keep if fys`Y' == `fs'
48 // keep if fys`fs'_`Y' == 1
49
50 //한계기업 표본: mf0: all, mf1:icr3==0, mf2:icr3<=1, mf3:icr3>=2, mf4:icr3==3
51 keep if mf mv' == 1
52
53 //생산성(prod`X'): 0(모든기업), 1(VL=/nan), 2(3년연속 하위 60%), 3(하위 80%), 4(상위 80%), 5(상위 60%), 6(~2), 7(~5)
54 keep if prod`pv'_`Y' == 1
55
56 //생산성(qv`X'): 0(모든기업), 1(qv1=/nan), 2(상위 80%), 3(상위 60%), 4(하위 80%), 5(하위 60%)
57 keep if qv1`qv'_`Y' == 1
58
59 local TV ea // liq qa
60 local Y_lag = `Y' - `lag'
61 local vce_cl kis // sector2
62 local control sa ea lnasset // capitalprod SA EA
63 local cross D`Y'`TV``Y_lag'
64
65 xtreg tinv D`Y' `cross` `control', fe vce(cl `vce_cl')
66 outreg2 using "output/ms_`mv'_`pv'_`qv'_`TV'_`lag'_`sec'_`fs'_`Y'", replace dta
67
68 restore }}}}}}}}}
1:79

```

스타타 추정 결과가 다수의 파일에 저장되어 있으면, 파이썬의 for-loop 구문을 이용하여 각 파일에 저장된 추정 결과를 읽어와 분석 기간, 변수 선택, 모형 설정 등에 따른 추정 결과를 하나의 데이터프레임에 정리 할 수 있다. 만약, 추정 결과들이 하나의 파일에 같이 저장되어 있으면, 파일내 각 추정 결과의 시작을 알리는 구문(회귀 분석의 경우 Linear Regression)의 위치를 파악한 다음 각 추정 결과를 나누어 데이터프레임에 정리한다.

전체 추정 결과들이 정리되면, 동일한 설명 변수의 추정 계수와 유의수준을 여러

²⁷스타타의 outreg2 함수를 이용하면, 각 추정 결과를 .txt, .dta, .xlsx 등의 파일 형식으로 저장할 수 있다. 그러나, 모형에 포함된 설명변수 갯수가 많은 경우 실행속도가 굉장히 느려진다. 이 경우 스타타 result 창의 추정 결과를 텍스트 파일에 테이블 형식으로 저장한 다음 파이썬에서 불러와 추정결과를 나누는 것이 더 빠를 수 있다.

분석 기간, 모형에 대해 쉽게 비교할 수 있다.

스타타 분석 결과 비교

The screenshot shows a Jupyter Notebook interface with a Python 3 kernel. The code cell contains the following Python script:

```
df_stata = get_duchin(mod, sec = [1], smpl = fs, ys = range(8, 18))

be = df_stata.unstack().stack(level = 0)
pv = be.loc[idx[['X', 'id'], ['p']], idx[1, fs, range(8, 18)]]
be = be.loc[idx[['X', 'id'], ['b', 'e']], idx[1, fs, range(8, 18)]]
be = be.rename(index=reorder_dict, level = 0).sort_index(axis = 0)
pv = pv.rename(index=reorder_dict, level = 0).sort_index(axis = 0)
be = be.astype(float)
pv = pv.astype(float)

be = be.stack(level = 1)
be = be.reorder_levels([0, 2, 1], axis = 0).sort_index(axis = 0)
pv = pv.stack(level = 1)
pv = pv.reorder_levels([0, 2, 1], axis = 0).sort_index(axis = 0)

be.to_excel('output//' + mod + '_asset_age.xlsx')
display((df_stata['b'].loc['X'].applymap(dp4) * (df_stata['p'].loc['X'] >= 1)).T)
display_side_by_side(pv.loc['IX'])

year 8 9 10 11 12 13 14 15 16 17
sec smpl
0 0.0947 0.0731 0.0485 0.0416
9 0.1250 0.0621 0.0524 0.0486
1 10 0.1250
11 0.1060 0.1250
12 0.0835 0.0999 0.0481 0.0519

sec 1
year 8 9 10 11 12 13 14 15 16 17
smpl
0 p 3 3 2 0 0 0 0 0 3 0
9 p 3 2 2 0 0 0 0 0 2 0
10 p 0 3 0 0 0 0 0 0 0 0 0
11 p 3 0 0 0 0 0 3 0 0 0
12 p 3 3 2 0 0 0 0 0 0 3 0
```