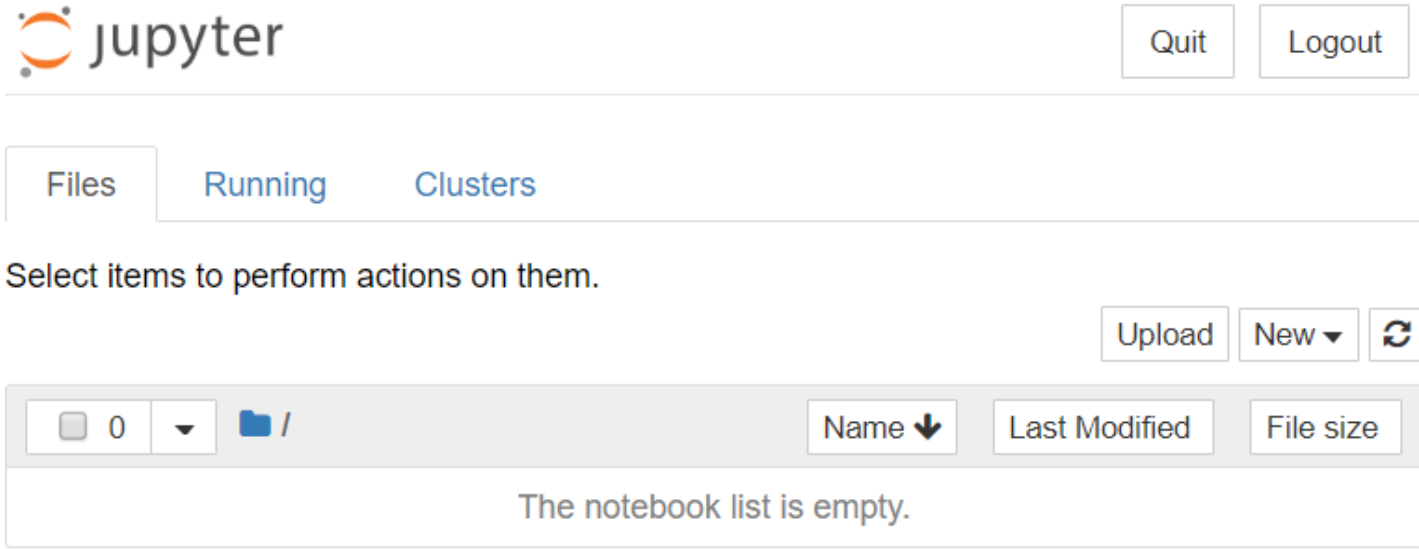
데이터 사이언스를 위한 파이썬

Pandas와 Matplotlib

도구의 준비
: Jupyter Notebook

- 파이썬 인터프리터
  - Python : https://www.python.org/
  - Anaconda : https://www.anaconda.com/



Jupyter Notebook

- Jupyter Notebook
  - Install Jupyter

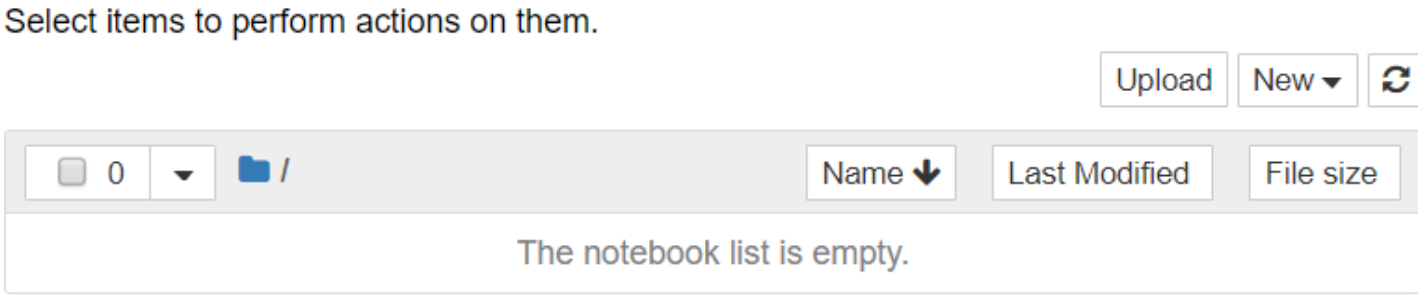
```
# Jupyter Install
pip install jupyter
```

  - Jupyter Notebook 실행

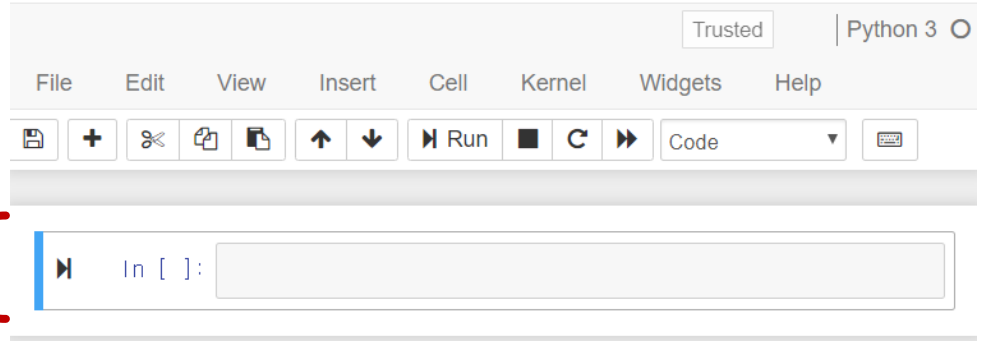
```
# Run Jupyter
jupyter notebook {notebook-folder}
```

    - 노트북 폴더는 생략 가능

도구의 준비
: Jupyter Notebook



- 새 노트북 생성
  - New > Python 3



- Cell 안에 파이썬 코드를 입력
  - Shift + Enter : Cell 내의 코드를 실행한 후 새 Cell 생성
  - Ctrl + Enter : Cell 내의 코드를 실행

- 활성화된 Cell은 사각형 테두리로 표시
  - 편집 모드 : Python 코드를 입력하고 실행할 수 있는 상태
  - 명령 모드 : 셀 자체의 추가, 삭제, 이동 등을 수행할 수 있는 모드
    - 편집 모드에서 명령 모드로 전환하려면 ESC 키

도구의 준비
: PIP

- PIP : 파이썬 라이브러리(패키지)를 관리해주는 프로그램

- 주요 명령어
  - pip list : 현재 파이썬 환경에 설치된 파이썬 라이브러리 목록

```
# 설치된 패키지 목록 확인
pip list
```

  - pip install {패키지명} : 지정한 패키지명의 파이썬 라이브러리를 현재 환경에 설치

```
# 지정한 패키지를 설치
pip install jupyter
```

  - pip uninstall {패키지명} : 지정한 패키지를 현재 파이썬 환경으로부터 제거

```
# 지정한 패키지를 삭제
pip uninstall jupyter
```

도구의 준비
: Pandas 설치

- Pandas : 데이터 분석을 위한 파이썬 라이브러리
  - 추가 제공 자료형 : DataFrame, Series
  - 데이터 분석을 위한 다양한 기능 제공

- Pandas 설치

```
# 판다스 설치
pip install pandas
```

- Pandas 사용을 위한 패키지 임포트
  - 일반적으로 Pandas는 pd라는 별칭으로 임포트

```
# Pandas Inport
import pandas as pd
```

Pandas 활용 EDA 따라 해보기
: 데이터 불러오기

- 데이터 셋 불러오기

```
# Pandas Inport
import pandas as pd

df = pd.read_csv("./data/gapminder.tsv", sep="\t")
print(df)
```

Pandas 입출력 지원 Data Source
- 텍스트 파일
- CSV 파일
- 엑셀 파일
- SQL 데이터베이스
- HDF5

- CSV(Comma Seperated Values)
  - 구조가 간단하여 널리 사용되는 데이터 포맷
  - 기본적으로는 콤마(,)를 기준으로 값을 구분함
  - sep 키워드 인자를 부여하여 구분자를 변경할 수 있음

```
          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
...           ...       ...   ...      ...       ...          ...
1701     Zimbabwe    Africa  1997   46.809  11404948   792.449960
1702     Zimbabwe    Africa  2002   39.989  11926563   672.038623
1703     Zimbabwe    Africa  2007   43.487  12311143   469.709298

[1704 rows x 6 columns]
```

Pandas 활용 EDA 따라 해보기
: 데이터 확인

- head 메서드 : 불러온 데이터프레임의 앞쪽 데이터를 확인

```
# 데이터 확인
print(df.head())
```

  - 기본적으로 5개의 앞쪽 행을 출력
  - 출력하고자 하는 행의 개수 지정 가능

```
       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
```

- tail 메서드 : 불러온 데이터프레임의 뒤쪽 데이터를 확인

```
# 데이터 확인
print(df.tail())
```

  - 기본적으로는 5개의 뒤쪽 행을 출력
  - 출력하고자 하는 행의 개수 지정 가능

```
       country continent  year  lifeExp       pop   gdpPercap
1699  Zimbabwe    Africa  1987   62.351   9216418  706.157306
1700  Zimbabwe    Africa  1992   60.377  10704340  693.420786
1701  Zimbabwe    Africa  1997   46.809  11404948  792.449960
1702  Zimbabwe    Africa  2002   39.989  11926563  672.038623
1703  Zimbabwe    Africa  2007   43.487  12311143  469.709298
```

Pandas 활용 EDA 따라 해보기
: 데이터 형식의 확인

- type 함수를 이용, 데이터프레임의 자료형을 확인해 봅시다.

```
# 데이터 형식의 확인
print(type(df))
```

```
<class 'pandas.core.frame.DataFrame'>
```

- 데이터 프레임의 형태를 확인해 봅시다.

```
# 데이터 프레임의 형태 확인
print(df.shape)
```

```
(1704, 6)
```

- 데이터 프레임에 포함된 정보를 확인해 봅시다

```
# 포함된 데이터(컬럼) 확인
print(df.columns)
```

```
Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'], dtype='object')
```

Pandas 활용 EDA 따라 해보기
: 데이터 프레임 구성 정보 확인

- dtypes : 각 컬럼의 자료형 확인

```
# 각 컬럼의 자료형 확인
print(df.dtypes)
```

```
country        object
continent      object
year            int64
lifeExp       float64
pop             int64
gdpPercap     float64
dtype: object
```

- 데이터프레임의 전반적 정보 확인

```
# 데이터 프레임의 전반적 정보 확인
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1704 entries, 0 to 1703
Data columns (total 6 columns):
country      1704 non-null object
continent    1704 non-null object
year         1704 non-null int64
lifeExp      1704 non-null float64
pop          1704 non-null int64
gdpPercap    1704 non-null float64
dtypes: float64(2), int64(2), object(2)
memory usage: 80.0+ KB
None
```

판다스 자료형	파이썬 자료형	설명
object	string	문자열
int64	int	정수
float64	float	실수
datetime64	datetime	파이썬 표준 datetime

Pandas 활용 EDA 따라 해보기
: 컬럼 단위 데이터의 추출

- 단일 컬럼의 추출

```
# 단일 컬럼 데이터 얻기
countries = df['country']
print(countries.head())
```

```
0    Afghanistan
1    Afghanistan
2    Afghanistan
3    Afghanistan
4    Afghanistan
Name: country, dtype: object
```

- 추출한 컬럼 데이터 형식의 확인

```
# 타입의 확인
print(type(countries))
```

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

- Series와 DataFrame : Pandas의 두 가지 기본 데이터 구조

데이터 타입	설명
Series	1차원 데이터를 다루는 자료 구조. 단일 자료형으로만 이루어짐
DataFrame	복수 개의 Series로 이루어진 자료 구조. 2차원 데이터를 다루는 데 효과적

Pandas 활용 EDA 따라 해보기
: 컬럼 단위 데이터의 추출

- 복수 컬럼 데이터의 추출
  - 컬럼 명을 리스트로 전달하면 복수 컬럼 데이터 추출 가능

```
# 복수 개 컬럼 데이터 얻기
subset = df[['country', 'continent', 'year']]
print(subset.head())
```

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

- 추출한 컬럼 데이터 형식의 확인

```
# 타입의 확인
print(type(subset))
```

```
<class 'pandas.core.frame.DataFrame'>
```

- 복수 컬럼 데이터를 추출하면 Pandas는 복수 개의 Series로 구성된 데이터 프레임을 반환

Pandas 활용 EDA 따라 해보기
: 행 단위 데이터의 추출

속성	설명
loc	인덱스를 기준으로 행 데이터 추출
iloc	행 번호를 기준으로 행 데이터 추출

- loc과 iloc
  - 행 단위 데이터 추출을 위해서는 loc, iloc 속성을 사용

- 인덱스와 행 번호
  - 인덱스: 데이터 식별을 위한 식별자. 숫자 이외에 문자열을 지정할 수도 있음
  - 행 번호: 실제 데이터가 데이터 프레임 내에 위치한 행의 번호. 정수만으로 데이터를 조회, 추출할 수 있으며 역 행번호를 이용한 조회도 가능

```
       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
```

인덱스

Pandas 활용 EDA 따라 해보기
: 행 단위 데이터의 추출

- loc과 iloc을 이용하여 특정 행의 데이터를 추출해 봅니다.

```
# 특정 행의 데이터 추출
print(df.loc[3])
print(df.iloc[3])
```

```
country      Afghanistan
continent           Asia
year                1967
lifeExp            34.02
pop             11537966
gdpPercap        836.197
Name: 3, dtype: object
country      Afghanistan
continent           Asia
year                1967
lifeExp            34.02
pop             11537966
gdpPercap        836.197
Name: 3, dtype: object
```

- 마지막 행의 출력
  - shape 는 (행 수, 열 수) 튜플을 반환하므로 행 수를 얻어올 수 있음
  - iloc은 역 행번호 조회가 가능하므로 -1을 이용
  - tail 메서드를 이용, 마지막 한 개의 행을 얻어올 수 있음

```
# 마지막 행의 출력
print(df.loc[df.shape[0] - 1])
print(df.iloc[-1]) # iloc은 역 행번호 조회 가능
print(df.tail(n=1))
```

단, **tail**로 얻어온 자료형은 **DataFrame**으로 **Series** 형식이 아님에 유의

Pandas 활용 EDA 따라 해보기
: 행 단위 데이터의 추출

- loc, iloc의 인자 값으로 행 정보 리스트를 전달하면 복수 개의 행을 추출할 수 있음

```
# 복수 개 행의 출력
print(df.loc[[0, 99, 999]]) # 인덱스 0, 99, 999 행 출력
```

```
          country continent  year  lifeExp       pop   gdpPercap
0     Afghanistan      Asia  1952   28.801   8425333  779.445314
99     Bangladesh      Asia  1967   43.453  62821884  721.186086
999       Mongolia      Asia  1967   51.253   1149500 1226.041130
```

Pandas 활용 EDA 따라 해보기
: 행 단위 데이터의 추출 - 슬라이싱 구문

- loc, iloc에 컬럼 목록을 함께 지정하면 특정 행과 열에 해당하는 데이터를 추출할 수 있다
  - 파이썬 리스트에서 범위 지정을 이용한 슬라이싱과 비슷

```
# 슬라이싱 구문을 이용한 데이터 추출
# 전체 행 선택, year, pop 컬럼 선택
subset2 = df.loc[:, ['year', 'pop']]
print(subset2.head())
```

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

  - 전체 행 데이터 중 1, 4, -1(마지막) 컬럼 데이터 추출
    - 음수 인덱싱은 iloc만 지원

```
# 전체 행 데이터, 2, 4, -1(마지막) 컬럼 데이터 추출
subset3 = df.iloc[:, [2, 4, -1]]
print(subset3.head())
```

```
   year       pop   gdpPercap
0  1952   8425333  779.445314
1  1957   9240934  820.853030
2  1962  10267083  853.100710
3  1967  11537966  836.197138
4  1972  13079460  739.981106
```

Pandas 활용 EDA 따라 해보기
: 행 단위 데이터의 추출 - range 활용

- Python의 내장 메서드 range를 이용하여 컬럼의 목록을 지정해 줄 수 있음
  - range 메서드 : 지정한 구간의 정수 리스트를 반환

```
# range를 이용한 데이터의 추출
u3_range = range(4)
print(u3_range)

print(df.iloc[:, u3_range].head())
```

```
range(0, 4)
       country continent  year  lifeExp
0  Afghanistan      Asia  1952   28.801
1  Afghanistan      Asia  1957   30.332
2  Afghanistan      Asia  1962   31.997
3  Afghanistan      Asia  1967   34.020
4  Afghanistan      Asia  1972   36.088
```

  - 전체 행 데이터 중 1, 4, -1(마지막) 컬럼 데이터 추출
    - 음수 인덱싱은 iloc만 지원

```
# 짝수번째 컬럼만 출력
subset4 = df.iloc[:, range(0, df.shape[1], 2)]
print(subset4.head())
```

```
range(0, 6, 2)
       country  year       pop
0  Afghanistan  1952   8425333
1  Afghanistan  1957   9240934
2  Afghanistan  1962  10267083
3  Afghanistan  1967  11537966
4  Afghanistan  1972  13079460
```

Pandas 활용 EDA 따라 해보기
: 기초 통계의 계산

- 열을 연도별로 그룹화

```
# df를 연도별로 그룹화
grouped_year_df = df.groupby('year')
print(grouped_year_df.head())
print(type(grouped_year_df))
```

```
       country continent  year  lifeExp       pop     gdpPercap
0  Afghanistan      Asia  1952   28.801   8425333    779.445314
......
59   Argentina  Americas  2007   75.320  40301927  12779.379640
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
```

Pandas 활용 EDA 따라 해보기
: 기초 통계의 계산

- 그룹화된 데이터프레임 컬럼의 산술 평균 (mean)

```
# 그룹화한 데이터프레임의 산술 평균 구하기
print(grouped_year_df['lifeExp'].mean())
```

```
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
```

- 계산 가능한 다양한 계산 값의 출력 (describe)

```
# 데이터프레임에서 활용 가능한 기초 통계량들
print(grouped_year_df.describe())
```

Pandas 활용 EDA 따라 해보기
: 기초 통계의 계산

- 그룹화된 데이터프레임 다중 컬럼의 통계 계산
  : 통계량을 추출할 컬럼 리스트 전달

```
# 여러 컬럼의 산술 평균 구하기
print(grouped_year_df[['lifeExp', 'gdpPercap']].mean())
```

- 그룹화된 데이터 개수(빈도수) 세기 : nunique

```
# 그룹화된 데이터 개수(빈도) 세기
print(df.groupby('continent')['country'].nunique())
```

```
continent
Africa      52
Americas    25
Asia        33
Europe      30
Oceania      2
Name: country, dtype: int64
```

```
            lifeExp     gdpPercap
year
1952  49.057620   3725.276046
1957  51.507401   4299.408345
1962  53.609249   4725.812342
1967  55.678290   5483.653047
1972  57.647386   6770.082815
1977  59.570157   7313.166421
1982  61.533197   7518.901673
1987  63.212613   7900.920218
1992  64.160338   8158.608521
1997  65.014676   9090.175363
2002  65.694923   9917.848365
2007  67.007423  11680.071820
```

Pandas 활용 EDA 따라 해보기
: 시각화를 활용한 데이터의 확인

- 통계량은 데이터를 대표하는 값으로 해당 데이터 셋을 설명해줄 수는 있지만,
  실제 데이터가 어떻게 분포되어 있는지 전체적인 그림은 보여줄 수 없음

- 데이트를 시각화하면 데이터를 전반적으로 이해하거나 추이를 파악하고자 할 때 많은 도움을
  얻을 수 있음

- Matplotlib는 파이썬에서 가장 널리 사용되는 시각화 라이브러리
  - Matplotlib의 설치

```
# Matplotlib 설치
pip install matplotlib
```

  - Matplotlib 임포트

```
# Matplotlib 임포트
import matplotlib.pyplot as plt
```

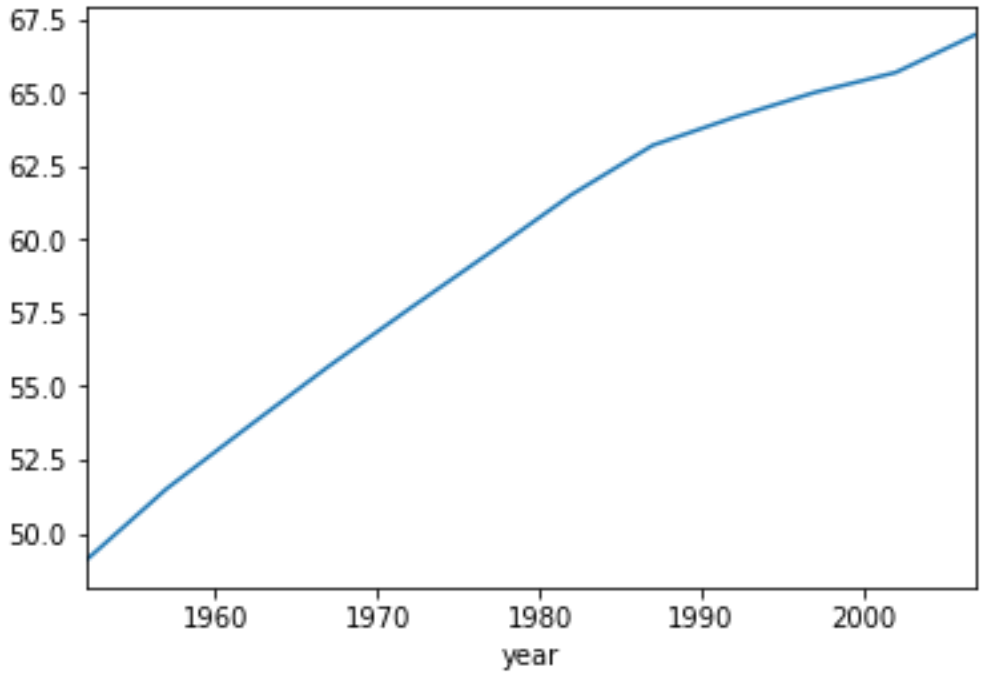
Pandas 활용 EDA 따라 해보기
: 시각화를 활용한 데이터의 확인

- year로 그룹화한 데이터프레임에서 lifeExp만 추출하여 평균값을 구해봅니다.

```
life_expectancy = df.groupby('year')['lifeExp'].mean()
print(life_expectancy)
```

- Matplotlib의 pyplot을 이용하여 그래프를 그려봅니다.

```
import matplotlib.pyplot as plt
life_expectancy.plot()
```



데이터 프레임과 시리즈

Pandas 기초 자료형

Series

- 시리즈 만들기 : Pandas의 Series 클래스에 리스트를 전달하여 시리즈를 생성

```
import pandas as pd
s = pd.Series(['홍길동', 28])
print(s)
```

```
0    홍길동
1     28
dtype: object
```

Pandas 문자열

  - Series는 여러 개의 단일 데이터타입을 저장하는 자료형

- 인덱스 부여 : 시리즈를 생성할 때 인덱스를 부여하지 않으면 0부터 자동 지정, 임의로 인덱스를 부여하고자 할 때는 Index 인자를 통해 인덱스로 사용할 키의 인덱스를 지정

```
# 인덱스의 지정
s = pd.Series(['홍길동', 28],
index=['Name', 'Age'])
print(s)
```

```
Name    홍길동
Age      28
dtype: object
```

- 이미 만들어진 Series의 경우, index 리스트에 직접 접근, 인덱스를 새로 부여할 수 있음

Series
: 기초 통계 메서드

- 시리즈를 구성하고 있는 데이터가 수치형이라면 다양한 기초 통계 메서드를 활용할 수 있음

```
kor = [80, 75, 90, 100, 65] # 데이터 리스트 생성
kor_s = pd.Series(kor) # 시리즈 생성
kor_s.describe() # 통계 요약 메서드
```

```
count      5.000000
mean      82.000000
std       13.509256
min       65.000000
25%       75.000000
50%       80.000000
75%       90.000000
max      100.000000
dtype: float64
```

- 시리즈에서 사용 가능한 통계 메서드

시리즈 메서드	설명
describe()	요약 통계량 계산
min()	최솟값
max()	최댓값
mean()	산술평균
median()	중앙값
isin()	시리즈에 포함된 값이 있는지 확인
drop_duplicate()	중복 없는 시리즈 반환

DataFrame

- 데이터프레임 만들기 : Pandas의 DataFrame 클래스에 딕셔너리를 전달

```
# 데이터 프레임의 생성
scores_df = pd.DataFrame({
    "KOR": [80, 90, 75],
    "ENG": [90, 80, 70],
    "MATH": [80, 90, 85]},
    index = ["홍길동", "김철수", "이영희"]
)
print(scores_df)
```

```
      KOR  ENG  MATH
홍길동   80   90    80
김철수   90   80    90
이영희   75   70    85
```

  - Series와 마찬가지로 인덱스를 부여하지 않으면 0부터 자동 부여

- 향후, 인덱스를 변경하고자 하면 DataFrame의 index 속성을 이용하여 확인, 변경

DataFrame

- loc을 이용한 관측치의 확인

```
# loc을 이용한 관측치의 확인
print(scores_df.loc['김철수'])
print(type(scores_df.loc['김철수']))
# 단일 관측치는 Series로 반환
```

```
KOR     90
ENG     80
MATH    90
Name: 김철수, dtype: int64
<class 'pandas.core.series.Series'>
```

- 동적 컬럼의 추가
  - 추가 컬럼이 필요하면 새 인덱스를 이용하여 컬럼을 추가할 수 있음

```
# 동적 컬럼의 추가
scores_df['TOTAL'] = scores_df['KOR'] + scores_df['ENG'] + scores_df['MATH']
print(scores_df)
```

```
      KOR  ENG  MATH  TOTAL
홍길동   80   90    80    250
김철수   90   80    90    260
이영희   75   70    85    230
```

Series와 불린 추출

- 원하는 데이터를 추출할 때 보통은 추출할 데이터의 인덱스를 정확히 모르는 경우가 많음
  - 인덱스에 특정 조건을 부여하여 해당 조건에 만족하는 값만 추출할 수 있음

```
# 불린 추출
# 시리즈 내 값이 80 이상인지 확인
scores_df['AVERAGE'] > 80
```

```
홍길동     True
김철수     True
이영희    False
Name: AVERAGE, dtype: bool
```

  - 추출된 각 행의 불린 결과를 바탕으로 조건을 만족하는(True) 행만 추출할 수 있다

```
# 불린 추출
# 데이터 프레임 내에서 평균이 80 이상인 학생만 추출
filtered_df = scores_df[scores_df['AVERAGE'] >= 80]
print(filtered_df)
```

```
      KOR  ENG  MATH  TOTAL     AVERAGE
홍길동   80   90    80    250  83.333333
김철수   90   80    90    260  86.666667
```

외부 파일 불러오기
: read_csv

- Pandas의 read_ 계열 메서드를 이용하면 다양한 형식의 파일을 DataFrame으로 변환할 수 있음

```
# thieves.txt 파일로부터 데이터를 불러들여 DataFrame으로 변환
thieves_df = pd.read_csv("./data/thieves.txt", sep="\t")
print(thieves_df)
```

```
    홍길동   175.8   73.2
0   전우치   170.2   66.3
1   임꺽정   186.7   88.2
2   장길산   188.3   90.0
```

- 기본적으로 read_csv 메서드는 첫 번째 행을 컬럼 헤더로 활용한다.
  따라서 첫 번째 행이 컬럼 헤더가 아닐 경우에는 header=None 인자값을 주어야 한다

```
thieves_df2 = pd.read_csv("./data/thieves.txt",
                          sep="\t", header=None)
print(thieves_df2)
```

```
       0      1     2
0   홍길동   175.8   73.2
1   전우치   170.2   66.3
2   임꺽정   186.7   88.2
3   장길산   188.3   90.0
```

외부 파일 불러오기
: read_csv

- 데이터를 불러올 때 특정 컬럼을 인덱스로 사용하고자 할 때, index_col 인자값으로 인덱스로 사용할 컬럼을 지정할 수 있다

```
thieves_df3 = pd.read_csv("./data/thieves.txt",
                          sep="\t", header=None, index_col=0)
print(thieves_df3)
```

```
            1     2
0
홍길동   175.8  73.2
전우치   170.2  66.3
임꺽정   186.7  88.2
장길산   188.3  90.0
```

- 데이터 프레임의 컬럼 명을 바꿔 봅니다

```
# 컬럼의 변경
thieves_df3.columns = ['Height', 'Weight']
print(thieves_df3)
```

```
        Height  Weight
0
홍길동   175.8    73.2
전우치   170.2    66.3
임꺽정   186.7    88.2
장길산   188.3    90.0
```

- 인덱스의 이름을 바꾸고자 할 때는 index.name 속성을 변경

```
# 인덱스 이름을 바꿔 봅니다.
thieves_df3.index.name = "Name"
```

외부 파일로 저장하기
: to_csv

- 데이터프레임을 파일로 저장하고자 할 때는 to_ 계열 메서드를 사용

```
# 데이터 프레임을 csv로 저장
thieves_df3.to_csv("./data/thieves.csv")
```

- 한글 윈도에서 내용이 깨질 경우, encoding을 MS949로 변경

```
thieves_df3.to_csv("./data/thieves.csv", encoding="MS949")
```

결측치와 이상치

- 결측치(Missing Value) : 누락된 값, 비어 있는 값
  - 누락값이라고도 하며 NaN, NAN, nan 등으로 표기 가능
  - 결측치 이용을 위해서는 numpy 모듈 필요

```
# 결측치
from numpy import nan, NaN, NAN
```

- 이상치(Outlier) : 측정된 값은 있으나 정상 범주에서 벗어나 극단적으로 크거나 작은 경우
  - 이상치가 포함되어 있으면 분석 결과가 왜곡되므로 분석에 앞서 이상치를 제거하는 작업이 필요
  - 이상치 제거를 위해서는 정상 범위에 대한 명확한 규정이 필요
    1. 논리적 판단
    2. 통계적인 기준을 이용 : 상자 그림(Box Plot) 등을 이용

결측치와 이상치

- 결측치(Missing Value)가 생기는 이유
  - 처음부터 측정되지 않은 경우(원시 데이터에 값이 비어 있는 경우)
  - 정상적인 데이터를 연결, 입력하는 과정에서 발생 가능

- 결측치와 이상치의 예

```
exam_scores = pd.Series([90, 80, 120, nan, 95, 80, -10])
print(exam_scores)
```

```
0      90.0
1      80.0
2     120.0
3       NaN
4      95.0
5      80.0
6     -10.0
dtype: float64
```

결측치

- 결측치는 Pandas의 isnull 메서드로 확인 가능
  - 결측치가 아님을 확인하고자 한다면 notnull 메서드

```
# 결측치의 확인
print(pd.isnull(exam_scores))
```

```
0    False
1    False
2    False
3     True
4    False
5    False
6    False
dtype: bool
```

결측치

결측치와 이상치

- 결측치는 sum, count, mean 등 기초 통계 함수로부터 배제됨

```
# 결측 빈도 확인
import numpy as np
print("결측치 수:", np.count_nonzero(exam_scores.isnull()))
```

```
# 결측 빈도 확인
num_rows = exam_scores.shape[0]
num_missing = num_rows - exam_scores.count()
print("결측치 수:", num_missing)
```

- 이상치는 통계 결과를 왜곡할 수 있으므로 적절한 조치 수행
  - 예: 이상치로 판단되는 범위의 값을 결측치로 처리

```
# 이상치를 결측치로 변환
~exam_scores.isin(range(0, 101))
exam_scores[~exam_scores.isin(range(0, 101))] = nan
print(exam_scores)
```

```
0    90.0
1    80.0
2     NaN
3     NaN
4    95.0
5    80.0
6     NaN
dtype: float64
```

결측치의 대체

- 결측치가 많은 데이터의 경우, 결측치를 무작정 제거하면 너무 많은 데이터의 손실로 분석 결과가 왜곡
  - 평균, 최빈값 등 대표값을 구해 결측치를 하나의 값으로 일괄 대체

```
# 결측치를 중앙값으로 대체
med_score = exam_scores[exam_scores.notnull()].median()
print(med_score)
exam_scores[exam_scores.isnull()] = med_score
print("평균:", exam_scores.mean())
```