14장. 데이터 분석 및 시각화

데이터 분석

- **데이터 분석을 위한 IDE(통합개발환경)**

  1. 주피터 노트북(Jupyter Notebook)

  아나콘다 플랫폼 다운로드 후 설치> Anaconda Navigater 실행 > 주피터 노트북 실행

  아나콘다(Anaconda)는 파이썬 기반의 오픈소스 플랫폼으로 데이터분석, 머신러닝등을 할 수있는 다양한 애플리케이션과 라이브러리를 제공한다.

  2. 코랩(Colaboratory)

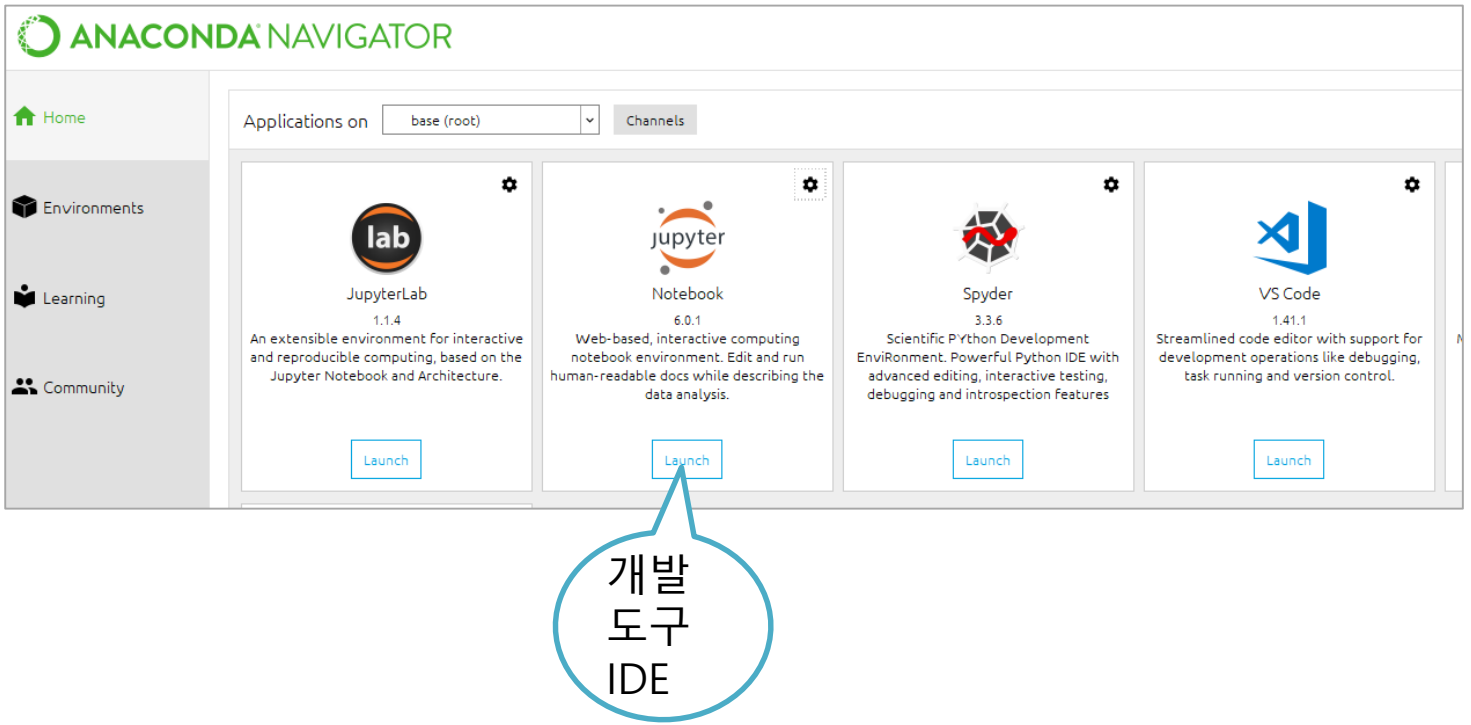
  - **구글 드라이브 – Colab 실행**

※ 1, 2 모두 브라우저 내에서 Python 스크립트를 작성하고 실행할 수 있는 소스 편집 도구이며, 웹 애플리케이션이다.

데이터 분석

▶ ***아나콘다 설치 -> anaconda 네비게이터 -> jupyter notebook***

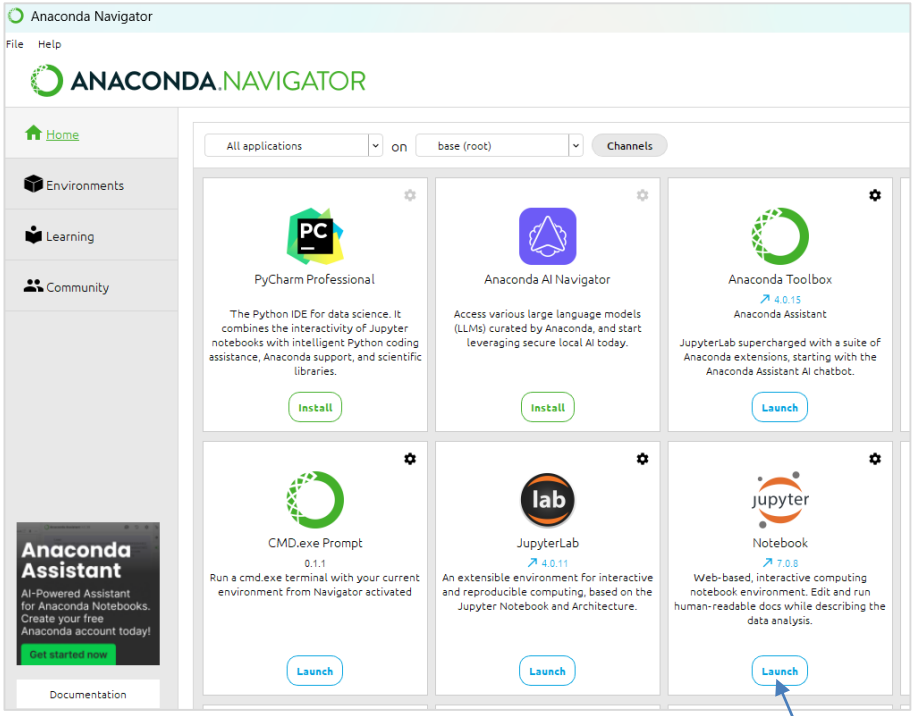
www.anaconda.com

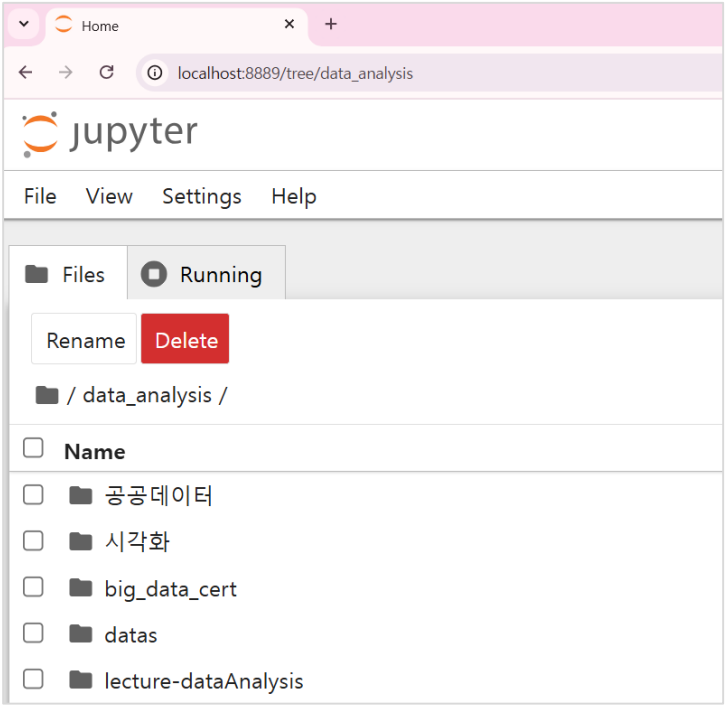


데이터 분석

- **파이썬 프로그래밍을 위한 IDE(통합개발환경)**

1. 주피터 노트북(Jupyter Notebook)



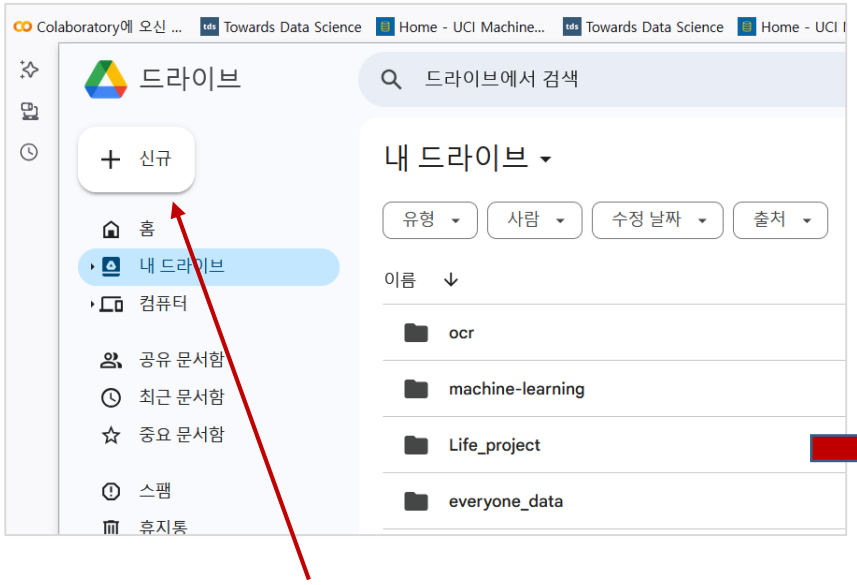


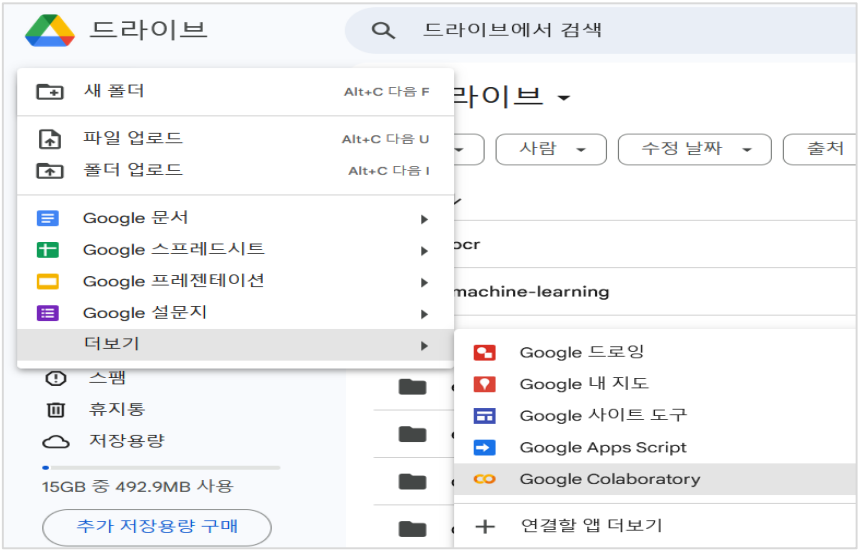
launch 클릭

데이터 분석

- **파이썬 프로그래밍을 위한 IDE(통합개발환경)**

2. 구글 코랩(Colaboratory)



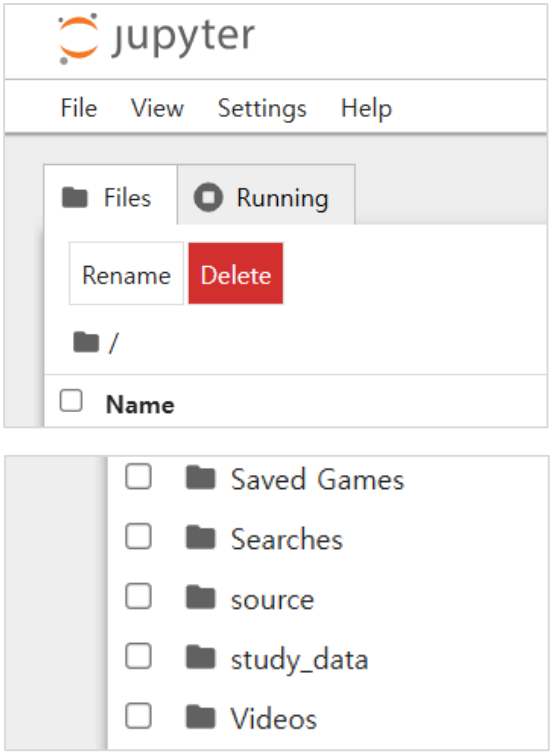


신규 > 더보기 > Google Colaboratory

jupyter notebook 사용

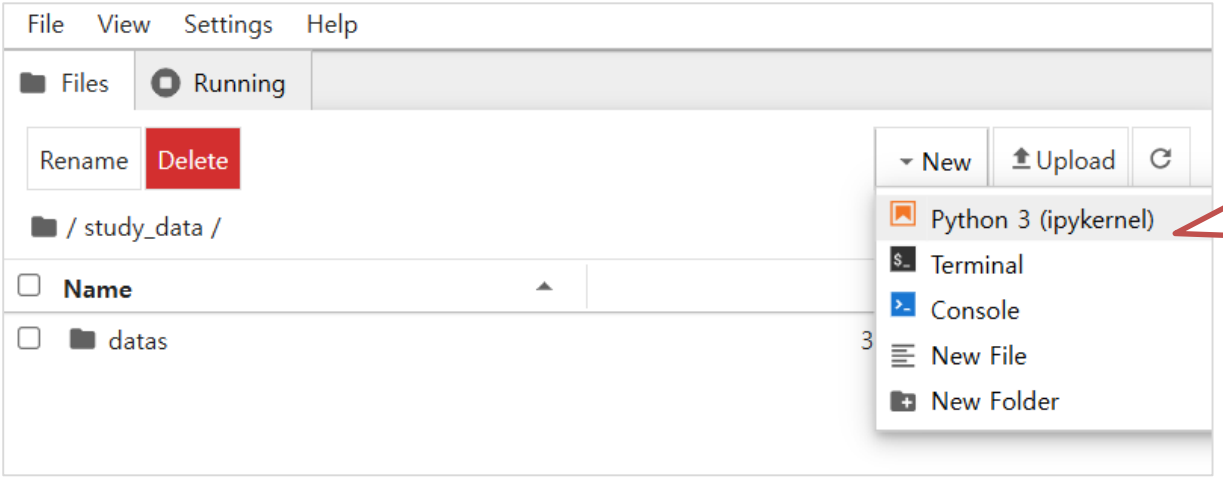
❖ 작업 디렉터리 생성

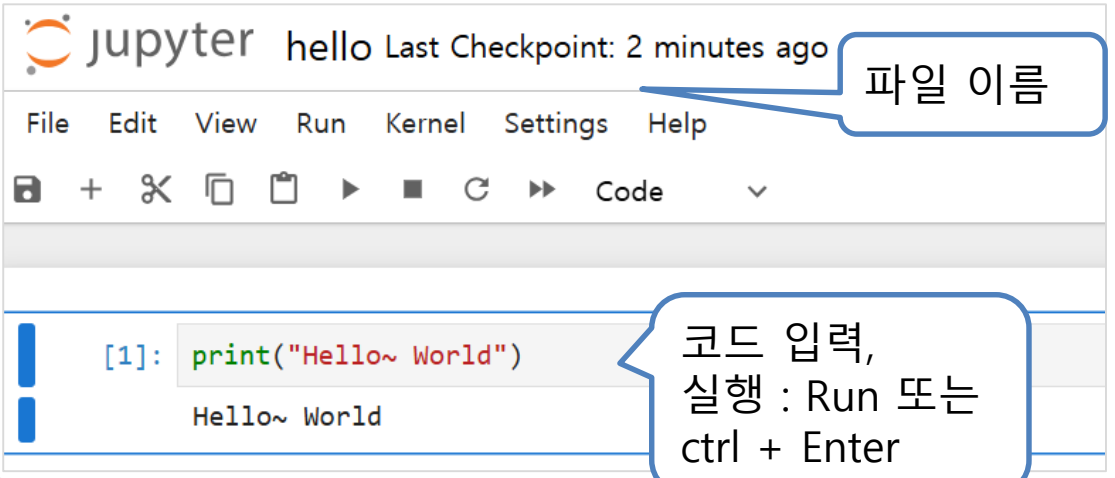
- User > Administrator 하위에
  study_data 폴더 생성 > 하위에 data 폴더 생성



jupyter notebook 사용

- 파일 만들기(확장자 - ipynb)





Markdown 문서 이해하기

- 파이썬 코딩

```
[1]: print("Hello~ World")
```

```
Hello~ World
```

```
[2]: print("안녕~ 세계")
```

```
안녕~ 세계
```

```
[7]: 10 + 20
     10 - 20
```

```
[7]: -10
```

```
[6]: n1 = 20
     n2 = 10

     print(n1 + n2)
     print(n1 - n2)
     print(n1 * n2)
     print(n1 / n2)
     print(n1 % n2)
```

```
30
10
200
2.0
0
```

```
[10]: carts = ["계란", "라면", "콩나물"]
      print(carts)

      for cart in carts:
          print(cart)
```

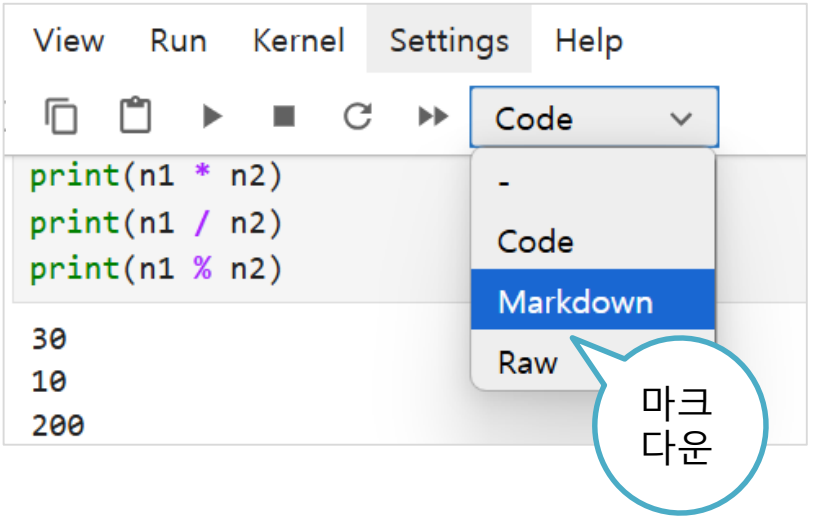
```
['계란', '라면', '콩나물']
계란
라면
콩나물
```

Markdown 문서 이해하기

- 마크다운(Markdown)

  마크다운(Markdown)은 일반 텍스트 기반의 마크업 언어이다.

  - 기호 종류
    - 제목: #의 개수가 클수록 작은 제목(## 텍스트)
    - 목록 – '*' 사용



Markdown 문서 이해하기

- 마크다운(Markdown)

```
### 함수 만들기
* 기능 - 절대값 계산
* 설명 - 음수는 양수로 양수는 양수로 반환
```

```
def my_abs(x):
    if x < 0:
        return -x
    else:
        return x

print(my_abs(-3))
```

```
3
```

함수 만들기

- 기능 - 절대값 계산
- 설명 - 음수는 양수로 양수는 양수로 반환

```
def my_abs(x):
    if x < 0:
        return -x
    else:
        return x

print(my_abs(-3))
```

```
3
```

판다스(Pandas)

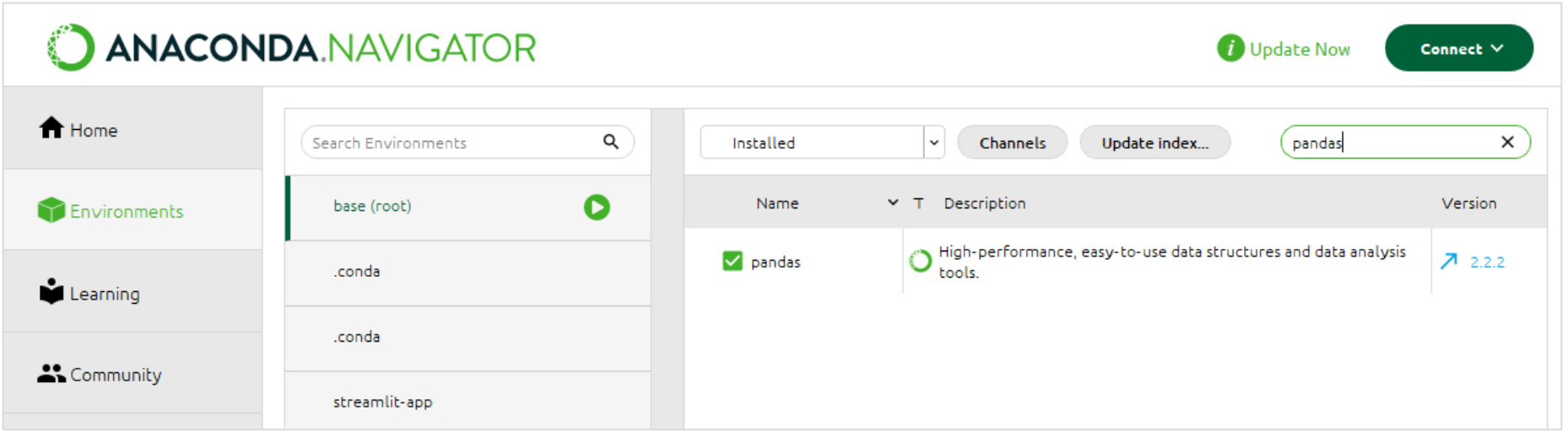
- **판다스**
  - 판다스(pandas) 라이브러리는 데이터를 수집하고 정리하는 데 최적화된 도구이다.
  - 가장 많이 사용되는 프로그래밍 언어인 **파이썬(python)**을 기반으로 한다.
  - **시리즈(Series)**와 **데이터프레임(DataFrame)**이라는 구조화된 데이터 형식을 제공한다.
  - 판다스 설치 - 아나콘다 플랫폼에 이미 설치됨

```
- pip install pandas
```

판다스(Pandas)

- **판다스**

  아나콘다 네비게이터 > Environments에서 설치 검색



판다스(Pandas)

- **판다스 자료구조**
  - **시리즈(Series)** – 데이터가 순차적으로 나열된 1차원 배열의 형태를 가지며, 인덱스(Index)는 데이터 값(value)과 일대일 대응이 된다.
  - **데이터프레임(DataFrame)** – 행(Row)과 열(Column)로 만들어지는 2차원 배열 구조로 표(Table) 형태이다.

	제품명
0	키보드
1	마우스
2	USB 메모리

인덱스 (Index)

[시리즈]

	제품명	가격
0	키보드	20,000
1	마우스	32,000
2	USB 메모리	10,000

칼럼 (column)

[데이터프레임]

판다스(Pandas)

- **시리즈 만들기**

import pandas as pd(별칭)

pd.Series(데이터, 인덱스)

```
import pandas as pd

# 데이터 - 리스트, 딕셔너리, 튜플
# 인덱스 - 생략하면 기본(0, 1, 2...)

data = ['키보드', '마우스', 'USB 메모리']

# 시리즈 객체 생성
s1 = pd.Series(data, index=[1, 2, 3])

print(s1)
print(type(s1))  #자료형
```

```
1         키보드
2         마우스
3    USB 메모리
dtype: object
```

판다스(Pandas)

- **시리즈 만들기**

```
# 시리즈의 속성
print(s1.index)  #인덱스
print(s1.values) #값
print(s1.shape)  # 시리즈의 크기(튜플)

# 요소 선택
print(s1[1])
print(s1[2])
print(s1[0:3])
```

```
idx = [1, 2, 3]

s2 = pd.Series([20000, 32000, 10000], index = idx)

print(s2)
print(s2.shape)
print(s2.dtype)  #데이터 타입 - int(숫자), object(문자)
```

```
1    20000
2    32000
3    10000
dtype: int64
```

판다스(Pandas)

- **데이터프레임 만들기 – 시리즈 결합**

```
import pandas as pd(별칭)

pd.DataFrame(데이터, 인덱스)
```

속성	기능
index	- 데이터프레임의 인덱스 (생략하면 기본 0, 1, 2)
columns	- 데이터프레임의 칼럼
shape	- 데이터프레임의 크기 (행, 열)

판다스(Pandas)

- **데이터프레임 만들기 – 시리즈 결합**

```
import pandas as pd

df = pd.DataFrame({
    "name": s1,
    "price": s2,
})
print(df)
print(type(df))

print(df.columns)  # 칼럼 속성
print(df.shape)
```

	name	price
1	키보드	20000
2	마우스	32000
3	USB 메모리	10000

데이터프레임

- **데이터 프레임 만들기 – 직접 생성**

```python
import pandas as pd
# 행과 열 - 2차원 리스트
data = [
    ['키보드', 20000],
    ['마우스', 32000],
    ['USB 메모리', 11000]
]

df = pd.DataFrame(data, columns=["name", "price"])
print(df)

# 칼럼 검색 - 검색([]-대괄호)
print(df["name"])
print(df[["name", "price"]])
```

	name	price
0	키보드	20000
1	마우스	32000
2	USB 메모리	11000

데이터프레임

- **데이터 프레임 만들기 – 직접 생성**

```
df = pd.DataFrame({
    "name" : ['키보드', '마우스', 'USB 메모리'],
    "price" : [20000, 32000, 10000]
})
print(df)

# 칼럼 추가
df["spec"] =  ['유선', '무선', '128GB']

# 칼럼(파생칼럼) 추가
df["할인가"] = df["price"] * 0.8
print(df)

# 칼럼명 변경
df = df.rename(columns={"name": "품명", "price": "가격", "spec": "제품상세"})
print(df)
```

	품명	가격	제품상세	할인가
0	키보드	20000	유선	16000.0
1	마우스	32000	무선	25600.0
2	USB 메모리	11000	128GB	8800.0

데이터프레임

- **데이터 프레임 - 행/열 선택**

속성	범위	예시
loc[인덱스명, 칼럼명] (location)	끝 인덱스 포함	[0:2] 일때 2 포함 (0, 1, 2)
iloc[인덱스번호, 칼럼번호] (integer location)	끝 인덱스 - 1	[0:2] 일때 2 제외 (0, 1)

```
# 행 선택 - loc, iloc 속성
# loc[행(행값)]
print(df.loc[0])
print(df.loc[1])
print(df.loc[[0, 1]]) #인덱싱(리스트)
print(df.loc[0:1])    #슬라이싱(범위연산자)
```

데이터프레임

- **데이터 프레임 - 행/열 선택**

```
# loc[행(행값), 열(칼럼명)]
print(df.loc[0, "품명"])
print(df.loc[1, "가격"])
print(df.loc[0:1, "품명"])
print(df.loc[0:1, ["품명", "가격"]])
print(df.loc[0:1, "품명" : "제품상세"])
print(df.loc[:, "품명":"할인가"])  #행, 열 전체

# iloc[행(인덱스), 열(인덱스)], 인덱스 이므로 종료값-1로 함
print(df.iloc[0, 0])
print(df.iloc[1, 1])
print(df.iloc[0:2, 0])
print(df.iloc[0:2, 0:2])
print(df.iloc[:, :]) #행, 열 전체
```

데이터프레임

- 데이터 프레임 - 행/열 선택

df.copy() – 데이터프레임 복사

df.set_index(“칼럼명”) – 인덱스 설정

```
df2 = df.copy()
print(df2)

# 인덱스 설정 - set_index(칼럼명)
df2 = df2.set_index("품명")
print(df2)

print("==========================================")
# 행 선택(loc)
print(df2.loc["키보드"])
print(df2.loc[["키보드","마우스"]])
print(df2.loc["키보드":"USB 메모리"])
```

데이터프레임

- **데이터 프레임 - 행/열 선택**

```
# loc[행(행값), 열(칼럼명)]
print(df2.loc["키보드", "가격"])
print(df2.loc["키보드", "가격":"제품상세"])
print(df2.loc["키보드", ["가격", "제품상세"]])

# iloc[행(인덱스), 열(인덱스)]
print(df.iloc[0, 0])
print(df.iloc[1, 1])
```

데이터프레임

- 데이터 프레임 - 요소값 변경(업데이트)

df.loc[행이름, 열이름] = “변경할 값 "

df.iloc[행번호, 열번호] = “변경할 값 "

```
# 데이터프레임 복사
df3 = df.copy()
df3

# 'USB 메모리'의 가격을 12000으로 수정
# df3.loc[2, "가격"] = 12000
# df3.loc[2, "할인가"] = df3.loc[2, "가격"] * 0.8
df3

df3.iloc[2, 1] = 12000
df3.iloc[2, 3] = df3.iloc[2, 1] * 0.8
df3
```

데이터프레임

- 데이터 프레임 - 행/열 추가

df[열] = “추가할 칼럼” => **열 추가**

df.loc[행] = “추가할 값” => **행 추가**

```
# 열(칼럼) 추가
df3["수량"] = [100, 50, 100]
df3

# 행 추가
df3.loc[3] = ["키보드", 35000, "무선", 0.0, 50]
df3.loc[3, "할인가"] = df3.loc[3, "가격"] * 0.8
df3
```

데이터프레임

- 데이터 프레임 - 행/열 삭제

df.drop(행, axis=0) => **행 삭제**

df.drop(열, axis=1) => **열 삭제**

```
# 행 삭제 - drop() axis=0은 행삭제, axis=1은 열 삭제
df3 = df3.drop(3, axis=0)
print(df3)

# 열 삭제
# df3 = df3.drop("할인가", axis=1)
df3 = df3.drop(["제품상세","할인가"], axis=1)
print(df3)
```

데이터 탐색

- **데이터 프레임을 CSV 파일로 변환**
  - CSV(Comma Seperated Value) 파일: 콤머로 구분된 파일로 확장자가 .csv인 파일이다.

☞ CSV 파일로 변환

df.to_csv("food.csv", index=False)

☞ CSV 파일 읽기

pd.read_csv("food.csv")

데이터 탐색

- 데이터 프레임을 CSV 파일로 변환

```python
# 데이터프레임 만들기
food = pd.DataFrame({
    "메뉴" : ["김밥", "비빔밥", "자장면", "우동", "김밥", "자장면", "김밥"],
    "가격" : [3000, 7000, 5500, 5500, 3500, 6000, 4000],
    "분류" : ["한식", "한식", "중식", "일식", "한식", "중식", "한식"]
})
# print(food)

# csv 파일로 변환
food.to_csv("food.csv", index=False)

# csv 파일 읽어오기
df = pd.read_csv("food.csv")
print(df)
```

데이터 탐색

- **데이터 탐색을 위한 주요 함수**

함수	기능
head(n) / tail(n)	- 상위, 하위 n개의 행을 보여줌 (생략되면 기본 5개)
info()	- 데이터의 정보 (데이터 개수, 칼럼, 자료형 등)
value_counts()	- 칼럼의 고유한 값의 개수
describe(include='number') describe(include='object')	- 수치형 칼럼 통계 요약 - 문자형 칼럼 통계 요약
astype(자료형)	- 칼럼의 자료형 변환 (int – 정수형, float-실수형)

데이터 탐색

- 데이터 탐색을 위한 주요 함수

```python
# 데이터 출력 - head(), tail() -> 기본 5개
print(df.head())
print(df.tail())

# 데이터 정보 - info()
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   메뉴      7 non-null      object
 1   가격      7 non-null      int64 
 2   분류      7 non-null      object
dtypes: int64(1), object(2)
memory usage: 300.0+ bytes
```

데이터 탐색

- 데이터 탐색을 위한 주요 함수

```
# 고유한 값의 개수 - value_counts()
print(df["메뉴"].value_counts())
print(df["분류"].value_counts())

# 통계 요약 - describe()
# print(df.describe())
print(df.describe(include='number'))
print(df.describe(include='object'))

# 칼럼 추가 - "할인가"(10%)
discount = 0.1
df["할인가"] = df["가격"] * (1 - discount)
print(df)
print(df.info())

# 자료형 변환 - astype()
df["할인가"] = df["할인가"].astype("int64")
print(df)
print(df.info())
```

데이터 탐색

- **조건 검색(필터링)**

비교 연산 기호	설명
>, >=	크다, 크거나 같다.
<, <=	작다, 작거나 같다
==	두 항이 같다.
!=	두 항이 같지 않다.

논리 연산	연산 기호	설명
교집합(AND)	&	두 조건이 모두 참일때 True 반환
합집합(OR)	|	두 조건중 하나 이상이 참일때 True 반환
부정(NOT)	~	조건이 참이면 False, 거짓이면 True를 반환

데이터 탐색

- **데이터 탐색 – 조건 검색(필터링)**

```python
# 조건 검색(필터링)
# 메뉴가 '자장면'인 음식의 정보
result1 = (df["메뉴"] == '자장면')
print(df[result1])

# 메뉴가 '김밥'이 아닌 음식의 정보
result2 = (df["메뉴"] != '김밥')
# result2 = ~(df["메뉴"] == '김밥')
print(df[result2])

# 가격이 4000원 미만인 음식의 정보
result3 = (df["가격"] < 4000)
print(df[result3])

# 메뉴가 '자장면'이고, 가격이 6000원 이상인 음식의 정보
# result4 = (df["메뉴"] == '자장면') & (df["가격"] >= 6000)
result4 = (df["메뉴"] == '자장면') | (df["가격"] >= 6000)
print(df[result4])
```

데이터 탐색

● 데이터 탐색 - 정렬

정렬 분류	함수	파라미터
칼럼 기준	sort_values(칼럼, 파라미터)	오름차순: ascending=True 내림차순: ascending=False
인덱스 기준	sort_index(파라미터)	

```
# 칼럼 기준 정렬 - sort_values(칼럼명)
# 메뉴를 오름차순 정렬하기 - 생략하면 오름차순(ascending=True)
df.sort_values("메뉴", ascending=True)

# 가격을 내림차순 정렬하기
df.sort_values("가격", ascending=False)

# 가격을 내림차순 정렬, 가격이 같으면 메뉴를 오름차순 정렬
sort = df.sort_values(["가격", "메뉴"], ascending=[False, True])
sort

# 인덱스 기준 정렬
df.sort_index(ascending=True)
```

데이터 전처리

- **판다스의 통계, 수학 , 그룹핑 함수**

함수	기능
count() / sum() / mean()	개수 / 합계 / 평균
var() / std()	분산 / 표준편차
max() / min()	최대값 / 최소값
idxmax() / idxmin()	최대값 위치 / 최소값 위치
quantile(.25) / quantile(.75)	1사분위수 / 3사분위수
mode()	빈도값
nlargest() / nsmallest()	몇번째로 큰값/작은값
round(수, 자리수)	반올림
groupby(칼럼명) agg([칼럼리스트])	그룹화 여러 개의 열에 대해 집계 연산

데이터 전처리

- 판다스의 통계, 수학 함수

```
print("개수:", df["가격"].count())
print("합계:", df["가격"].sum())
print("평균:", round(df["가격"].mean(), 2))
print("분산:", round(df["가격"].var(), 2))
print("표준편차:", round(df["가격"].std(), 2))
print("최대값:", df["가격"].max())
print("최소값:", df["가격"].min())
print("최대값의 위치:", df["가격"].idxmax())
print("최소값의 위치:", df["가격"].idxmin())
```

데이터 전처리

- 판다스의 통계, 수학 함수

```
# 사분위수 - quantile()
print("1사분위수:", df["가격"].quantile(.25))
print("2사분위수:", df["가격"].quantile(.50))
print("3사분위수:", df["가격"].quantile(.75))
print("4사분위수:", df["가격"].quantile(1.0))

# 조건 - 1사분위수 보다 작은 가격을 출력
result = df["가격"] < df["가격"].quantile(.25)
print(df[result])

# 최빈값 - mode()
print("최빈값:", df["가격"].mode()[0])

# nlargest(), nsmallest()
print(df.nlargest(2, "가격"))
print(df.nsmallest(2, "가격"))
```

데이터 전처리

- 그룹핑 - groupby()

```
import pandas as pd

df = pd.read_csv("food.csv")
df

df["수량"] = [10, 10, 15, 5, 7, 7, 8]
df

# csv 파일로 변환
df.to_csv("food2.csv", index=False)
```

데이터 전처리

- **그룹핑 - groupby()**

```
food = pd.read_csv("food2.csv")
# print(food)

# 그룹화 - groupby()
df["분류"].value_counts()

food.groupby("메뉴").mean(numeric_only=True)
food.groupby("분류").mean(numeric_only=True) #모든 수치 칼럼 평균

food.groupby("분류")["가격"].mean()  #분류별 가격의 평균
food.groupby("분류")["수량"].mean()  #분류별 수량의 평균
```

데이터 전처리

- **데이터프레임 합치기**

함수	방식	기능
concat()	단순 병합	위+아래, 왼쪽+오른쪽으로 합치기
merge	키(Key) 기준 병합	특정 키를 기준으로 합치기

```
# 단순 병합 - concat()
lunch = pd.DataFrame({
    'menu': ["피자", "치킨", "햄버거"],
    'price': [20000, 15000, 8000]
})
lunch

drink = pd.DataFrame({
    'menu': ["주스", "커피", "콜라"],
    'price': [3000, 4000, 2000]
})
drink
```

데이터 전처리

- **데이터프레임 합치기 – 단순 병합**

```
# 위 + 아래
# ignore_index -> 기존 인덱스를 새롭게 변경, axis=0 생략 가능
# pd.concat?
full_menu = pd.concat([lunch, drink], ignore_index=True, axis=0)

# 왼쪽 + 오른쪽
# full_menu = pd.concat([lunch, drink], axis=1)
full menu
```

데이터 전처리

- 데이터프레임 합치기 – 키(Key) 기준 병합

```
가격 = pd.DataFrame({
    'menu': ["피자", "치킨", "햄버거"],
    'price': [20000, 15000, 8000]
})
가격

주문수량 = pd.DataFrame({
    'menu': ["피자", "치킨", "햄버거"],
    'quantity': [2, 3, 5]
})
주문수량

# pd.merge?
메뉴정보 = pd.merge(가격, 주문수량, on="menu")
```

데이터 전처리

- **결측치 삭제 및 대체**

함수	설명
drop(행, axis=0) drop(열, axis=1)	NaN이 포함된 행 삭제 NaN이 포함된 열 삭제
dropna()	NaN이 포함된 모든 행 삭제
fillna()	- 수치형: 평균(mean()), 중간값(median()) - 문자형: 빈도값(mode())
isnull() isnull().sum()	- 결측치 확인(True / False로 반환) - 결측치 개수(True(1))

데이터 전처리

- **결측치 삭제**

```python
import pandas as pd
import numpy as np
# numpy - 수치 계산을 효율적으로 처리하는 파이썬 라이브러리.

df = pd.DataFrame({
    'A': [1, None, 3],
    'B': [4, 5, None],
})
print(df)

# csv 파일 생성
df.to_csv("data_nan.csv", index=False)
```

data_nan.csv

	A	B
1	1.0	4.0
2		5.0
3	3.0	

✓ **None**은 파이썬에서 NULL을 의미(공백)

✓ **np.nan**은 넘파이가 제공하는 데이터 누락을 의미함

데이터 전처리

- **결측치 삭제**

```
# 칼럼 추가 - 결측치 적용
df['C'] = np.nan
print(df)
# df.info()
print(df.isnull().sum())

# 결측치가 있는 모든 행 삭제
# df.dropna(axis=0, inplace=True)
print(df)
print(df.isnull().sum())

# 1행 삭제
df = df.drop(1, axis=0)
df

# 열 삭제
df_new = df.drop(['A', 'C'], axis=1)
print(df_new)
```

데이터 전처리

- **결측치 대체**

```
data = {
    "메뉴": ["김밥", "비빔밥", "자장면", "우동", None, "자장면", "김밥"],
    "가격": [3000, 7000, 5500, None, 3500, 6000, 4000],
    "분류": ["한식", "한식", "중식", "일식", "한식", "중식", "한식"]
}

food = pd.DataFrame(data)
food
food.info()

# 결측치 확인 - isnull()
print(food.isnull())
print(food.isnull().sum()) #True(1) / False(0)
```

데이터 전처리

- **결측치 대체**

```
# 수치형 - 중간값으로 대체
median = food["가격"].median()
median

# fillna()
food["가격"] = food["가격"].fillna(median)
food

# 문자형 - 최빈값
frequency = food["메뉴"].mode()[0]
frequency
food["메뉴"] = food["메뉴"].fillna(frequency)
food

print(food.isnull().sum())
```

데이터 전처리

- **문자열 함수**
  - 판다스에서는 **str** 접근자를 사용해 문자열을 다룬다.

함수	설명
문자열.str[:4]	- 문자열 추출
문자열.str.replace(변경전, 변경후) replace()	- 변경전 문자를 변경후로 치환(숫자불가) - 요소 단위 변경(숫자도 변경가능)
문자열.str.split()	문자열 분리
str.contains(문자열)	특정 문자열 검색
str.len()	문자열 길이
str.lower() / str.upper()	소문자로 변경 / 대문자로 변경
str.strip() / str.lstrip() / str.rstrip()	공백문자 양쪽 / 왼쪽 / 오른쪽 제거

데이터 전처리

- **문자열 함수**

```
df = pd.DataFrame({
    'A': ["데이터 분석", "머신러닝 분류", "카이제곱 검정"],
    'B': [10, 20, 30],
    'C': ['ab cd', 'AB CD', 'ab cd ']
})
# print(df)

# str 속성 - 문자 추출
# df['A추출'] = df['A'].str[:4]
df['A추출'] = df['A'].str[-2:]
print(df)

# replace() - 요소 단위, str.replace()는 특정 문자열
# df['A'] = df['A'].replace("머신러닝 분류", "머신러닝 회귀")
df['A'] = df['A'].str.replace("분류", "회귀")
# print(df)
```

데이터 전처리

- **문자열 함수**

```
# replace() - 숫자 변경 가능, str.replace()는 숫자 변경 불가
df['B'] = df['B'].replace(20, 40)
# df['B'] = df['B'].str.replace(20, 40)
print(df)

print(df['A'].str.split()) #공백 문자로 분리
df['A'].str.split()[0]
df['D'] = df['A'].str.split().str[0]
print(df)

# contains() - 특정 문자열 포함
df["검정포함여부"] = df['A'].str.contains("검정")
print(df)

# 소문자로 변경
df['C'] = df['C'].str.lower()
print(df['C'])

# 공백 제거
df['C'] = df['C'].str.strip()
print(df['C'])
```

데이터 전처리

- **시계열 데이터(Time Series Data)**

  시간의 흐름에 따라 수집된 데이터로, 특정 시간 간격을 두고 연속적으로 관측된 값을 의미한다.

함수	설명
pd.to_datetime(df["날짜"])	문자형(object)을 날짜형(datatime)으로 변환
df["날짜"].**dt.year** df["날짜"].**dt.month** df["날짜"].**dt.day**	년도 월 일
df["날짜"].**dt.hour** df["날짜"].**dt.minute** df["날짜"].**dt.second**	시 분 초
df["날짜"].**dt.dayofweek**	요일 추출(0-월, 1-화, 2-수, 3-목, 4-금, 5-토, 6-일)
df["시간"].**dt.total_seconds()**	총초 / 60 -> 분으로 환산

데이터 전처리

- **시계열 데이터**

```
data = {
    "날짜": ["2025-3-1", "2025-3-2", "2025-3-3"],
    "날짜와시간": ["2025-3-1 10:20:50",
            "2025-3-2 11:25:10", "2025-3-3 12:40:15"]
}

df = pd.DataFrame(data)
df.info()

# 자료형 변환(parsing) : object -> to_datetime()
df["날짜"] = pd.to_datetime(df["날짜"])
df["날짜와시간"] = pd.to_datetime(df["날짜와시간"])
df
df.info()
```

데이터 전처리

- **시계열 데이터**

```
# 요일(0-월, 1-화, 2-수, 3-목, 4-금, 5-토, 6-일)
weeks = df["날짜와시간"].dt.dayofweek
print(type(weeks)) # 시리즈
print(list(weeks)) # 리스트로 변환
for week in weeks:
    if week == 0:
        print("월")
    if week == 1:
        print("화")
    if week == 2:
        print("수")
    if week == 3:
        print("목")
    if week == 4:
        print("금")
    if week == 5:
        print("토")
    if week == 6:
        print("일")
```

데이터 전처리

- 시계열 데이터

```
data = {
    "출발시간": ["2025-3-10 10:20:50"],
    "도착시간": ["2025-3-10 10:30:10"]
}

df = pd.DataFrame(data)

df["출발시간"] = pd.to_datetime(df["출발시간"])
df["도착시간"] = pd.to_datetime(df["도착시간"])

df.info()

# df["시간차이(분)"] = (df["도착시간"] - df["출발시간"])
df["시간차이(분)"] = (df["도착시간"] - df["출발시간"]).dt.total_seconds() / 60
df
```

Numpy(넘파이)

- **Numpy(Numeric Python)**
  - ✓ 행렬이나 다차원 배열을 쉽게 처리할 수 있도록 지원하는 파이썬의 라이브러리이다.
  - ✓ NumPy는 데이터 구조 외에도 수치 계산을 위해 효율적으로 구현된 기능을 제공한다

```
import numpy as np
```

수학 관련 함수	설명
np.sum()	합계(리스트, 튜플)
np.mean()	평균(리스트, 튜플)
np.square(n)	n의 제곱수
np.sqrt(n)	n의 제곱근

Numpy(넘파이)

- **수학 관련 함수**

```
import numpy as np

# 수학 관련 함수
n1 = np.sum([1, 2, 3])  #합계
print(n1)
n2 = np.mean([1, 2, 3, 4]) #평균
print(n2)
n3 = np.square(9) #제곱수
print(n3)
n4 = np.sqrt(4) #제곱근
print(n4)
```

Numpy(넘파이)

- **Numpy(Numeric Python)**

배열 관련 함수	설명
np.array([1, 2, 3])	1차원 배열
np.array([[1, 2], [3,4]])	2차원 배열
np.arange(n)	1차원 배열로 0부터 n-1까지 생성
a.reshape(x, y)	1차원 배열(a)을 2차원 배열로 변환
b.flatten()	2차원 배열(b)을 1차원 배열로 변환
np.zeros()	0으로 만들어진 배열 생성
np.ones()	1로 만들어진 배열 생성

Numpy(넘파이)

- **Numpy(Numeric Python)**

```python
# 1차원 배열
x = np.array([1, 2, 3])
print(x)
print(type(x)) # 타입
print(x.shape) # 크기

# 인덱싱 및 슬라이싱
print(x[0])
print(x[2])
print(x[-1])
print(x[0:3])
print(x[:])
print(x[:-1])
```

```python
# 2차원 배열
d2 = np.array([
    [1, 2, 3],
    [4, 5, 6]
])
print(d2)
print(d2.shape)

# 인덱싱 및 슬라이싱
print(d2[0, 0])
print(d2[1, 1])
print(d2[0:, 0:])
print(d2[1:, 1:])
```

Numpy(넘파이)

- **Numpy(Numeric Python)**

```
# arange(시작값, 종료값, 증감값) - (종료값-1)
a = np.arange(10)
print(a)
b = np.arange(0, 10, 1)
print(b)
c = np.arange(0, 10, 2)
print(c)

# 1차원 배열을 2차원 배열
a2 = a.reshape(2,5)
print(a2)

# 2차원을 1차원 배열로
a3 = a2.flatten()
print(a3)
```

Numpy(넘파이)

- **Numpy(Numeric Python)**

```
# 결측치(nan)
a = np.nan
print(a)
print(type(a))

# 0 생성
a1 = np.zeros((3, 3))
print(a1)

# 1 생성
a2 = np.ones((2, 3))
print(a2)

a3 = np.ones((2, 3), dtype=int)
print(a3)
```

numpy 모듈 응용 예제

- 2019년 프로야구 성적표

2017kbo.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

==========2019 한국 프로야구 성적표==========

순위	팀	승	패	무	승률
1	두산	88	55	1	0.615
2	키움	86	57	1	0.601
3	SK	88	55	1	0.615
4	LG	79	64	1	0.552
5	NC	73	69	2	0.514
6	KT	71	71	2	0.500
7	기아	62	80	2	0.437
8	삼성	60	83	1	0.420
9	한화	58	96	0	0.403
10	롯데	48	93	3	0.340

numpy 모듈 응용 예제

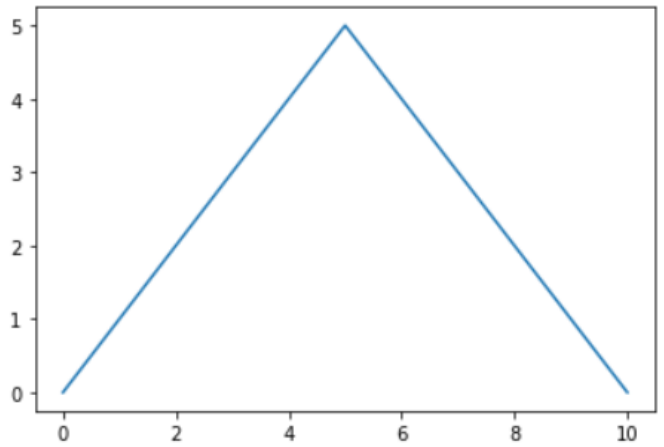
- 2019년 프로야구 성적표

```
import numpy as np

rank = 1 + np.arange(10)

team = np.array(["두산", "키움", "SK", "LG", "NC", "KT", "기아", "삼성", "한화", "롯데"])
win = np.array([88, 86, 88, 79, 73, 71, 62, 60, 58, 48])
lose = np.array([55, 57, 55, 64, 69, 71, 80, 83, 96, 93])
draw = np.array([1, 1, 1, 1, 2, 2, 2, 1, 0, 3])
win_rate = np.array([0.615, 0.601, 0.615, 0.552, 0.514, 0.500, 0.437, 0.420, 0.403, 0.340])

with open("2019kbo.txt", 'w') as f:
    f.write("==============2019 한국 프로야구 성적표==============\n")
    head = "순위\t팀\t승\t패\t무\t승률"
    f.write(head)
    f.write('\n')
    for i in range(len(team)):
        txt = "%d\t" % (rank[i]) + team[i] + '\t' + "%d\t" % (win[i]) + \
              "%d\t" % (lose[i]) + "%d\t" % (draw[i]) + "%.3f\t" % (win_rate[i]) +'\n'
        f.write(txt)
```

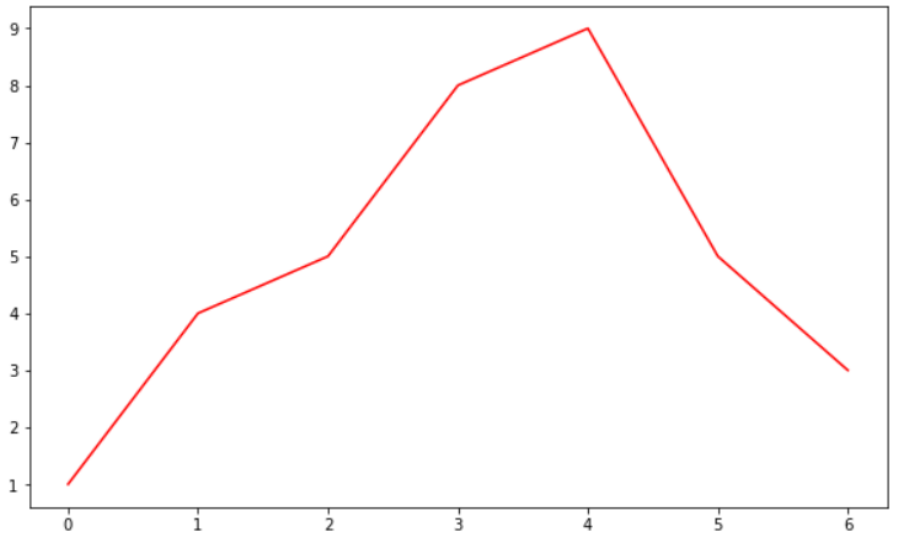
Matplotlib 모듈

데이터를 시각적으로 표현하기 - matplotlib

In [2]:
```
import matplotlib.pyplot as plt

x = [0,1,2,3,4,5,6,7,8,9,10]
y = [0,1,2,3,4,5,4,3,2,1,0]

plt.plot(x, y)
plt.show()
```



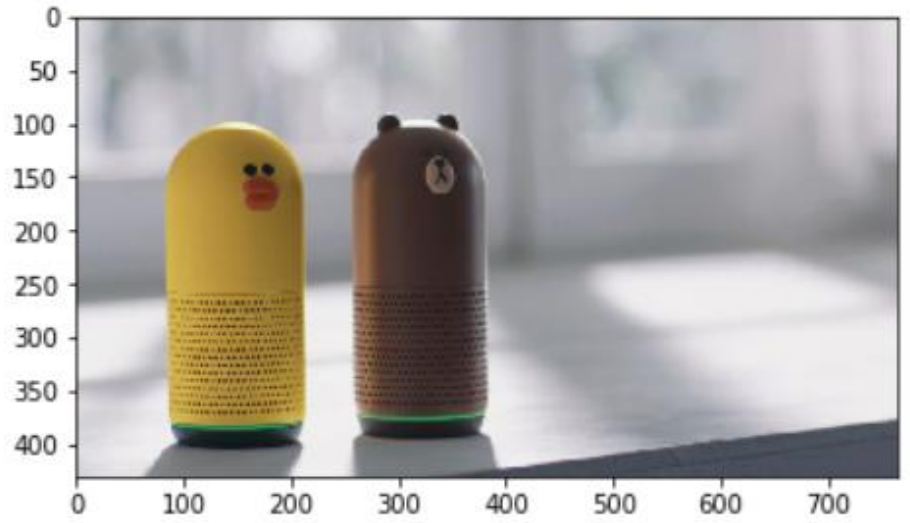

```
t = [0, 1, 2, 3, 4, 5, 6]
y = [1, 4, 5, 8, 9, 5, 3]
plt.figure(figsize=(10,6))
plt.plot(t, y, color="red")
plt.show()
```



Matplotlib – 이미지 표시

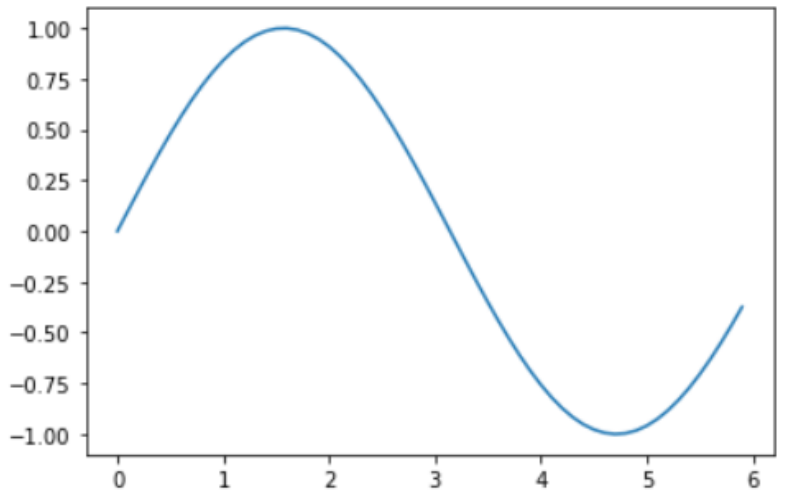
이미지 표시하기

```
from matplotlib.image import imread

img = imread("C:/Python/pynum_pd/friends.jpg")
plt.imshow(img)
plt.show()
```

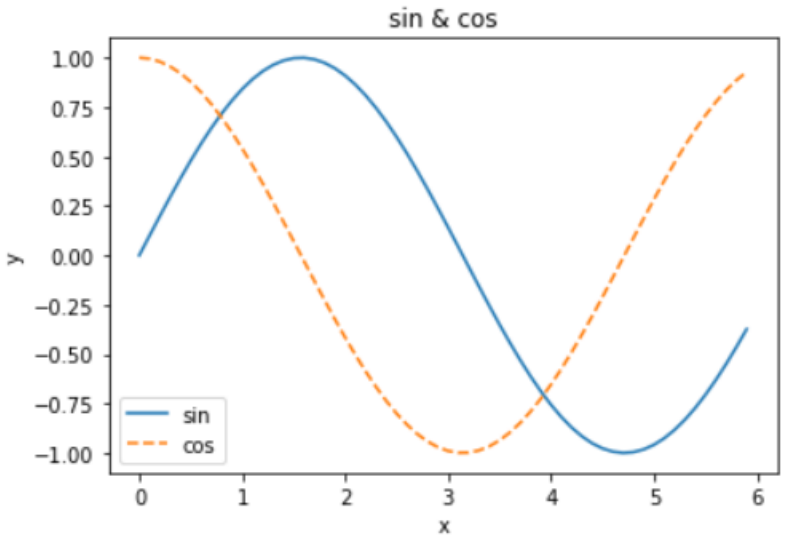


Matplotlib + Numpy 모듈

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

x = np.arange(0,6,0.1)
y = np.sin(x)

plt.plot(x, y)
plt.show()
```

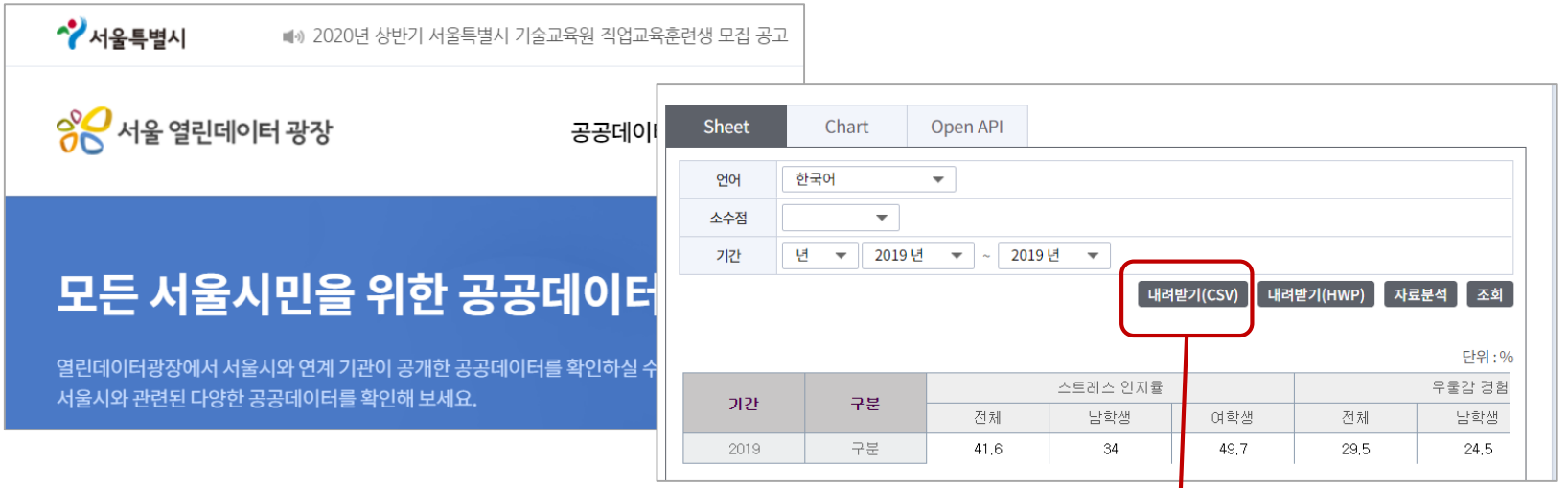



```
x = np.arange(0,6,0.1)
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y1, label="sin")
plt.plot(x, y2, linestyle="--", label="cos")
plt.xlabel("x")
plt.ylabel("y")
plt.title('sin & cos')
plt.legend()
plt.show()
```



서울시 청소년 정신건강 분석

- 자료 얻기
  - 열린 데이터 광장 > 청소년 정신건강 검색 –> 자료 다운로드



서울시 청소년 정신건강 분석

- 데이터 읽어 오기
  - pd.read_csv("./data/report.txt", delimiter='\t')

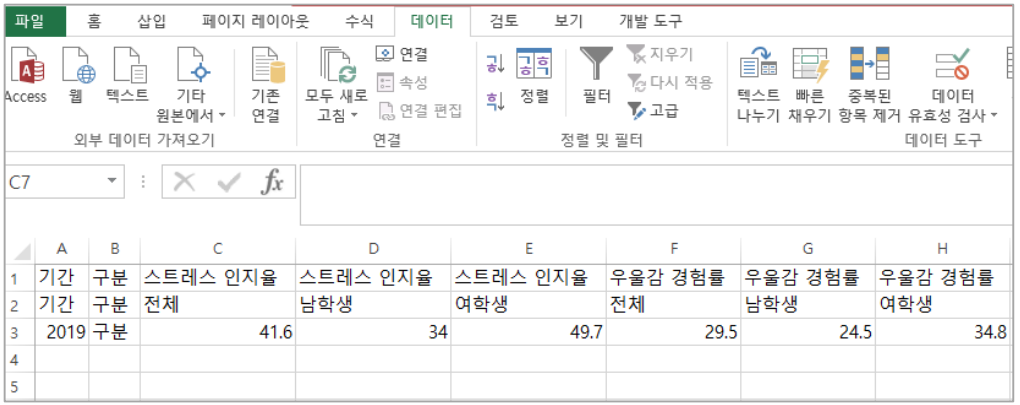
서울시 청소년들의 스트레스 데이터 읽어보기

```
import pandas as pd

mental = pd.read_csv("./data/report.txt", delimiter='₩t')
mental
```

	기간	구분	스트레스 인지율	스트레스 인지율.1	스트레스 인지율.2	우울감 경험률	우울감 경험률.1	우울감 경험률.2
0	기간	구분	전체	남학생	여학생	전체	남학생	여학생
1	2019	구분	41.6	34	49.7	29.5	24.5	34.8

서울시 청소년 정신건강 분석

◆ csv를 엑셀 파일로 변환하기
데이터 - > 외부데이터 가져오기(텍스트) -> 탭으로 구분



	A	B	C	D	E	F	G	H
1	기간	구분	스트레스 인지율	스트레스 인지율	스트레스 인지율	우울감 경험률	우울감 경험률	우울감 경험률
2	기간	구분	전체	남학생	여학생	전체	남학생	여학생
3	2019	구분	41.6	34	49.7	29.5	24.5	34.8
4								
5								

```
pd.read_excel("./data/teenage_mental.xlsx", usecols="C:K")
```

	스트레스 인지율	스트레스 인지율.1	스트레스 인지율.2	우울감 경험률	우울감 경험률.1	우울감 경험률.2
0	전체	남학생	여학생	전체	남학생	여학생
1	41.6	34	49.7	29.5	24.5	34.8

서울시 청소년 정신건강 분석

- 열 이름 바꾸기

```
col_names=['스트레스', '스트레스남학생', '스트레스여학생', '우울감경험률', '우울남학생', '우울여학생',
```

```
pd.read_excel("./data/teenage_mental.xlsx", header=1, usecols="C:K", names=col_names)
```

	스트레스	스트레스남학생	스트레스여학생	우울감경험률	우울남학생	우울여학생	자살생각율	자살남학생	자살여학생
0	41.6	34	49.7	29.5	24.5	34.8	14.2	10.8	17.9

- 변수에 담은후 스트레스 질문에 그렇지 않다 데이터 행 만들기

```
raw_data = pd.read_excel("./data/teenage_mental.xlsx", header=1, usecols="C:K", names=col_names)
raw_data
```

	스트레스	스트레스남학생	스트레스여학생	우울감경험률	우울남학생	우울여학생	자살생각율	자살남학생	자살여학생
0	41.6	34	49.7	29.5	24.5	34.8	14.2	10.8	17.9

```
raw_data.loc[1] = 100 - raw_data.loc[0]
raw_data
```

	스트레스	스트레스남학생	스트레스여학생	우울감경험률	우울남학생	우울여학생	자살생각율	자살남학생	자살여학생
0	41.6	34.0	49.7	29.5	24.5	34.8	14.2	10.8	17.9
1	58.4	66.0	50.3	70.5	75.5	65.2	85.8	89.2	82.1

서울시 청소년 정신건강 분석

➢ '응답' 열이름 만들기

```
raw_data["응답"] = ["그렇다", "아니다"]
raw_data
```

	스트레스	스트레스남학생	스트레스여학생	우울감경험률	우울남학생	우울여학생	자살생각율	자살남학생	자살여학생	응답
0	41.6	34.0	49.7	29.5	24.5	34.8	14.2	10.8	17.9	그렇다
1	58.4	66.0	50.3	70.5	75.5	65.2	85.8	89.2	82.1	아니다

➢ '응답' 인덱스 설정하기

```
raw_data.set_index('응답', drop=True, inplace=True)
raw_data
```

응답	스트레스	스트레스남학생	스트레스여학생	우울감경험률	우울남학생	우울여학생	자살생각율	자살남학생	자살여학생
그렇다	41.6	34.0	49.7	29.5	24.5	34.8	14.2	10.8	17.9
아니다	58.4	66.0	50.3	70.5	75.5	65.2	85.8	89.2	82.1

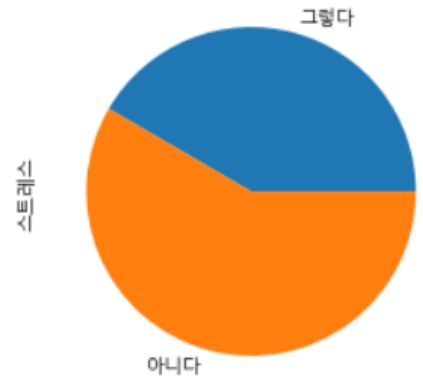
서울시 청소년 정신건강 분석

- 데이터를 그래프로 표현하기

```
import matplotlib.pyplot
from matplotlib import font_manager, rc
f_path = "C:/Windows/Fonts/malgun.ttf"
font_name = font_manager.FontProperties(fname=f_path).get_name()
rc('font', family=font_name)
```

```
raw_data["스트레스"].plot.pie()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dd343e81c8>
```



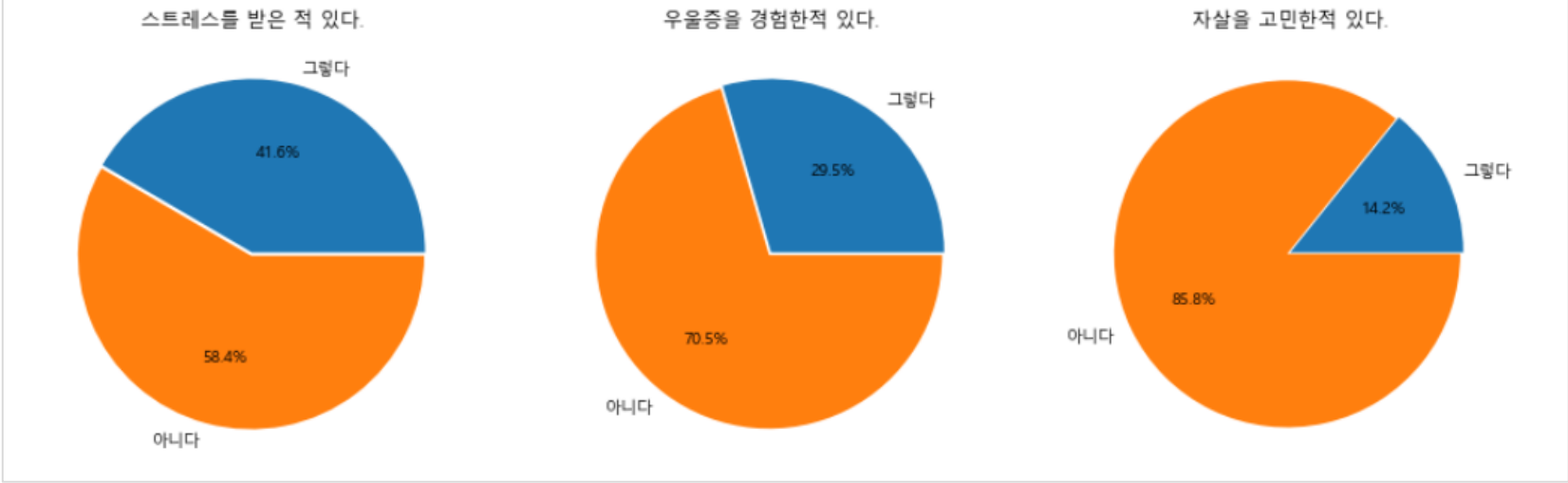
한글 글꼴로 변환

서울시 청소년 정신건강 분석

- 데이터를 그래프로 표현하기

```
f, ax = plt.subplots(1, 3, figsize=(16, 8))
raw_data['스트레스'].plot.pie(explode=[0, 0.02], ax=ax[0], autopct='%1.1f%%')
ax[0].set_title("스트레스를 받은 적 있다.")
ax[0].set_ylabel('')

raw_data['우울감경험률'].plot.pie(explode=[0, 0.02], ax=ax[1], autopct='%1.1f%%')
ax[1].set_title("우울증을 경험한적 있다.")
ax[1].set_ylabel('')

raw_data['자살생각율'].plot.pie(explode=[0, 0.02], ax=ax[2], autopct='%1.1f%%')
ax[2].set_title("자살을 고민한적 있다.")
ax[2].set_ylabel('')
```

```
Text(0, 0.5, '')
```



서울시 운동을 하지 않는 이유 통계

서울시 운동을 하지 않는 이유 통계

○ 통계개요
* 통계명 : 운동을 하지 않는 이유
* 통계종류 : 서울서베이의 서울시민 운동을 하지 않는 이유를 제공하는 일반・조사통계
* 작성목적 : 도시정책지표에 대한 포괄적인 자료구축 및 분석을 실시함으로써 서울의

Sheet | Chart | Open API

언어	한국어
소수점	
기간	년 2017 년 ~ 2017 년

내려받기(CSV) 내려받기(HWP) 자료분석 조회

단위 : %

기간	대분류	분류	운동을 할 충분한 시간이 없어서
	서울시		49.7
	성별	남자	55.2
		여자	45
	연령별	10대	55.7
		20대	54.8
		30대	58.1
		40대	57.7
		50대	50.1

서울시 운동을 하지 않는 이유 통계

```
not_exercise['대분류']=="성별"
```

```
0     False
1      True
2      True
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13    False
```

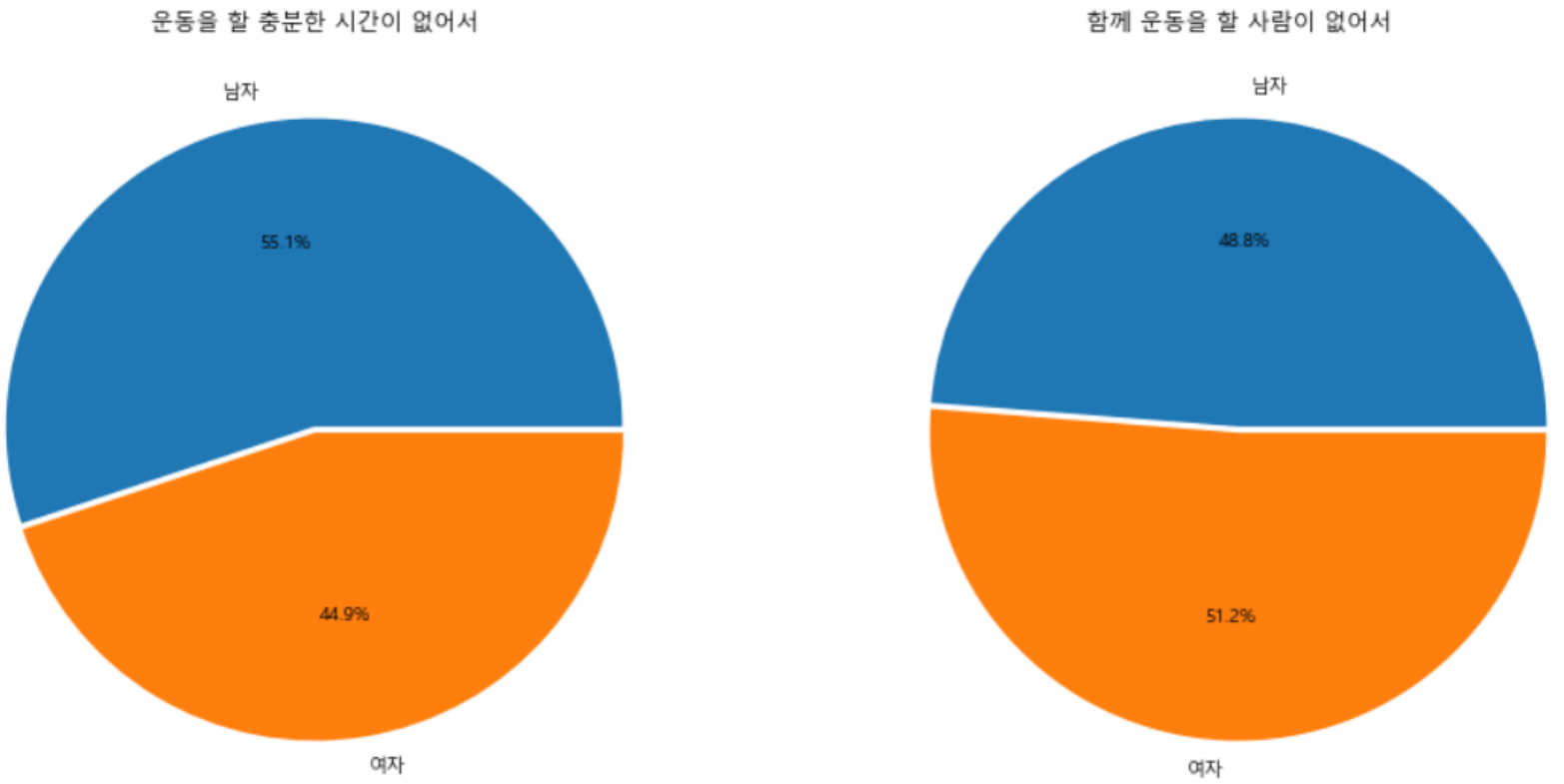
성별만 복사해서 표시

```
not_ex_sex = not_exercise[not_exercise["대분류"]=="성별"].copy()
not_ex_sex
```

	대분류	분류	운동을 할 충분한 시간이 없어서	함께 운동을 할 사람이 없어서	운동을 할 만한 장소가 없어서
1	성별	남자	55.2	7.9	5.5
2	성별	여자	45.0	8.3	6.0

서울시 운동을 하지 않는 이유 통계

```
f, ax = plt.subplots(1,2, figsize=(16, 8))
not_ex_sex["운동을 할 충분한 시간이 없어서"].plot.pie(explode=[0, 0.02], ax=ax[0], autopct='%1.1f%%')
ax[0].set_title("운동을 할 충분한 시간이 없어서")
ax[0].set_ylabel('')

not_ex_sex["함께 운동을 할 사람이 없어서"].plot.pie(explode=[0, 0.02], ax=ax[1], autopct='%1.1f%%')
ax[1].set_title("함께 운동을 할 사람이 없어서")
ax[1].set_ylabel('')
plt.show()
```



OpenCV

- **OpenCV 사용법**
  - 영상 처리와 컴퓨터 비전을 위한 오픈 소스 라이브러리이다.
  - 공장에서 제품 검사, 의료 영상 처리 및 보정, CCTV, 로보틱스 등에 활용
  - **설치**

    pip install opencv-python

  - **사용**

    import cv2

OpenCV

- **OpenCV 사용법**
  - 이미지 읽기.

    cv2.imread(file, flag)

  - 이미지 출력

    cv2.imshow(이미지제목, 이미지파일)

  - 키보드 입력을 처리하는 함수

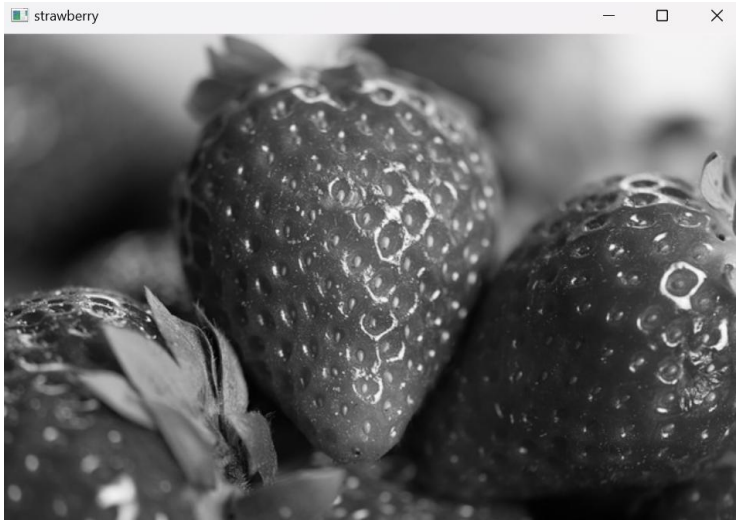
    cv2.waitkey(0)

    (0 - 계속 대기, 1000 – 1초)

  - 윈도우(창) 닫기

    cv2.destroyAllWindows()

이미지 출력



이미지 출력

- 실습 예제

```
import cv2

"""
# source 폴더에서 이미지 불러오기
- IMREAD_UNCHANGED(컬러읽고 투명부분도 읽어옴)
- IMREAD_COLOR(컬러읽고 투명부분은 무시)
- IMREAD_GRAYSCALE(회색으로 읽어옴)
"""
img = cv2.imread('source/strawberries.jpg', cv2.IMREAD_UNCHANGED)

cv2.imshow('strawberries',img) # 새 창으로 띄우기

cv2.waitKey(2000)  # 2초 대기후 종료

cv2.destroyAllWindows() # 모든 창 닫기
```

이미지 출력

- 실습 예제

```
import cv2

img = cv2.imread('source/strawberries.jpg', cv2.IMREAD_COLOR)
#이미지를 회색으로 변경
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

cv2.imshow('strawberries',img_gray)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

이미지 출력

- 실습 예제

```
import cv2

img = cv2.imread('source/strawberries.jpg', cv2.IMREAD_COLOR)
#이미지를 회색으로 변경
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

cv2.imshow('strawberries',img_gray)

key = cv2.waitKey(0)
if key == ord('q'):
    print(key)
    print(chr(key))

cv2.destroyAllWindows()
```

```
113
q
```

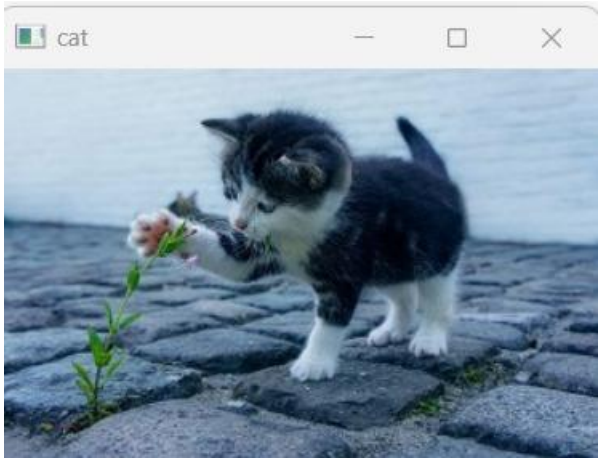
Opencv로 이미지를 파일로 쓰기

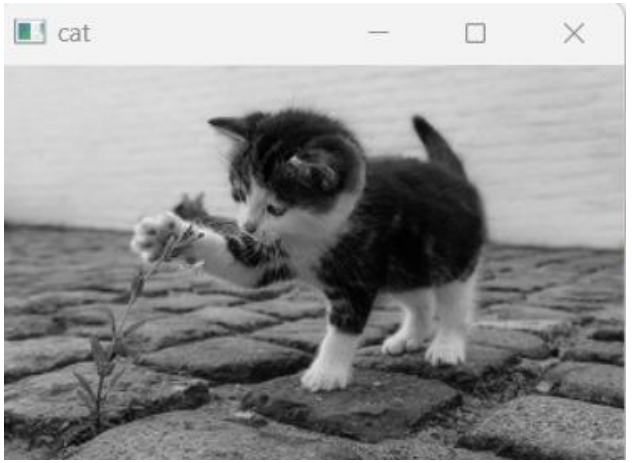
- 이미지를 파일에 쓰기

```
# 이미지를 흑백으로 불러오기
img_dog = cv2.imread('source/dog2.jpg', cv2.IMREAD_GRAYSCALE)

cv2.imshow('dog',img_dog) # 새 창으로 띄우기

cv2.waitKey(0)  # 무한 대기

# output 폴더에 저장하기
cv2.imwrite('output/dog2.jpg', img_dog)

cv2.destroyAllWindows()
```

OpenCV

- 이미지 처리 예제





OpenCV

- 이미지 처리 예제

```
import cv2

img = cv2.imread('./source/cat.jpg', cv2.IMREAD_COLOR)
cv2.imshow('cat', img)
#cv2.waitKey(2000)  # 2초간 대기, 0-계속 대기
cv2.waitKey(0)

# 파일 쓰기
cv2.imwrite('./source/cat_copy.jpg', img)

# gray 로 변경 - cvtColor() 함수 사용, RGB가 아닌 BGR임
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('cat', img_gray)
cv2.waitKey(0)
```

OpenCV

- **Jupyter notebook에서 구현**

opencv - 영상 처리 모듈

In [6]:
```
import cv2

img = cv2.imread("./source/cat.jpg", cv2.IMREAD_COLOR)
cv2.imshow('cute cat', img)
cv2.waitKey(0)
```

Out[6]: -1

파일 쓰기

In [5]:
```
cv2.imwrite("./source/cat2.jpg", img)
```

Out[5]: True

In [7]:
```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('cute cat', img_gray)
cv2.waitKey(0)
```

Out[7]: -1