12장. 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)

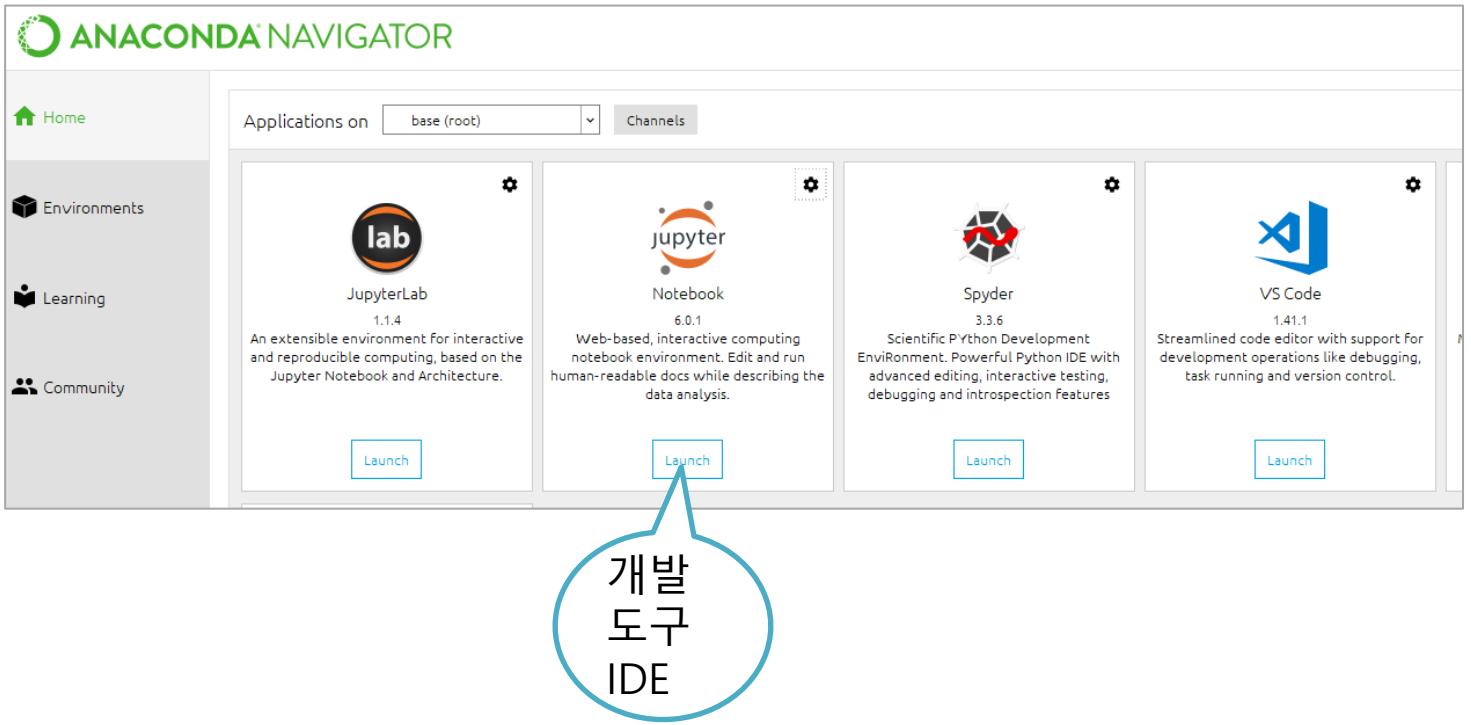
  - **구글 드라이브 – Colab 실행**

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

데이터 분석

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

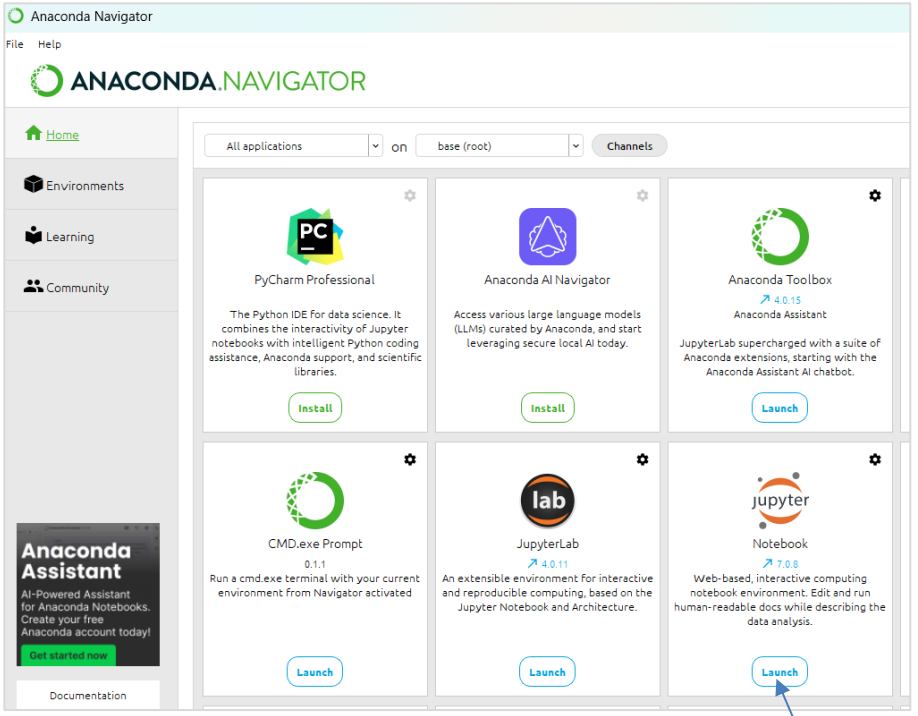
www.anaconda.com

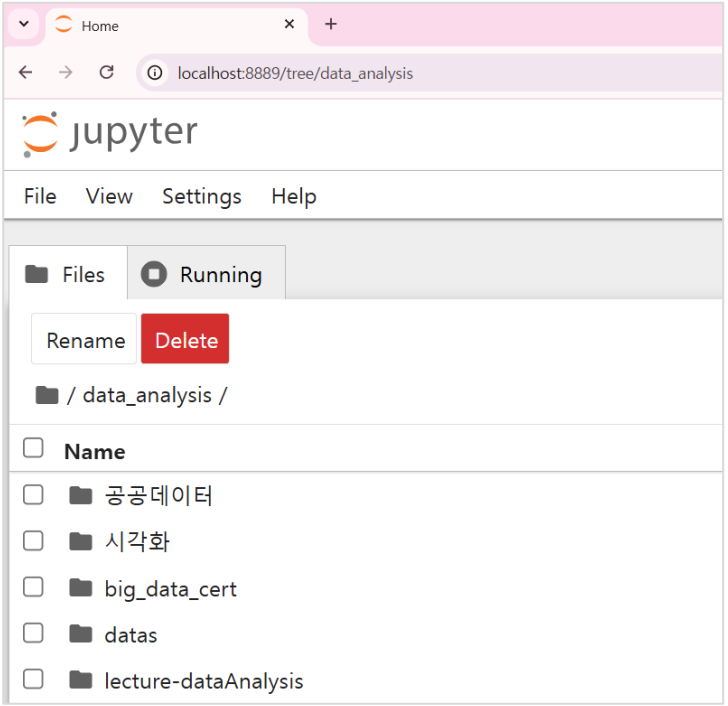


데이터 분석

- **파이썬 프로그래밍을 위한 IDE(통합개발환경)**

1. 주피터 노트북(Jupyter Notebook)



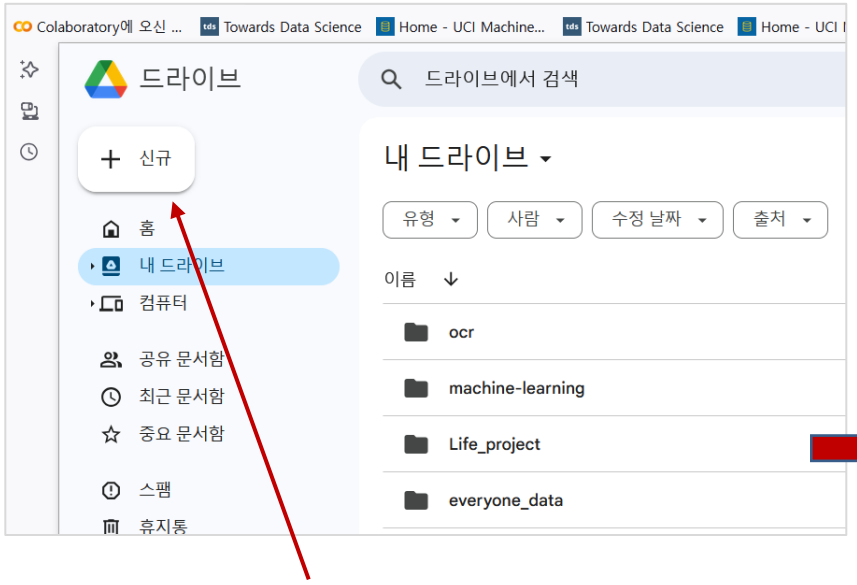


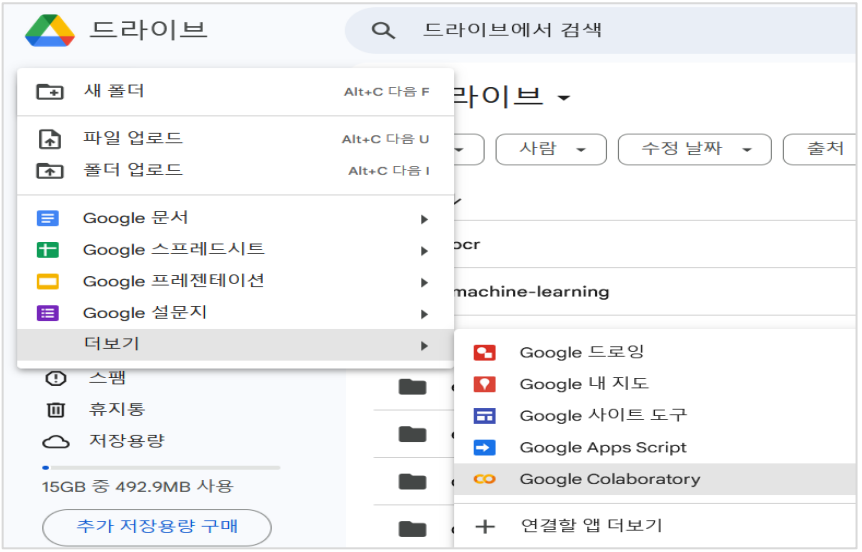
launch 클릭

데이터 분석

- **파이썬 프로그래밍을 위한 IDE(통합개발환경)**

2. 구글 코랩(Colaboratory)



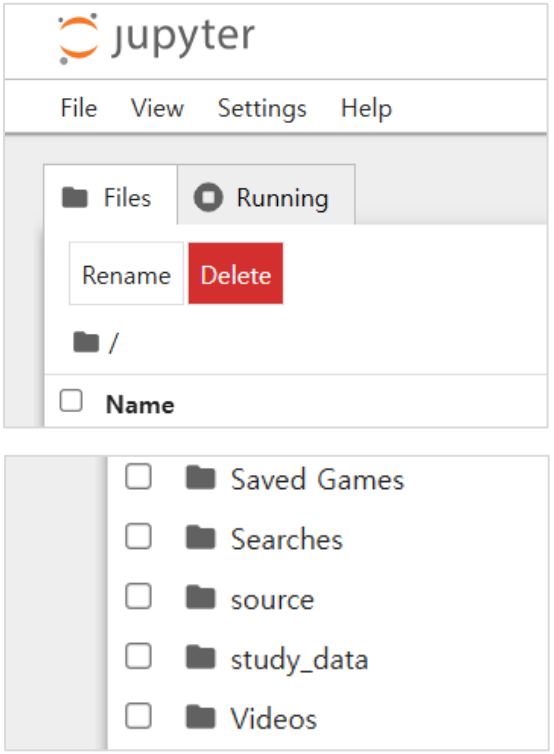


신규 > 더보기 > Google Colaboratory

jupyter notebook 사용

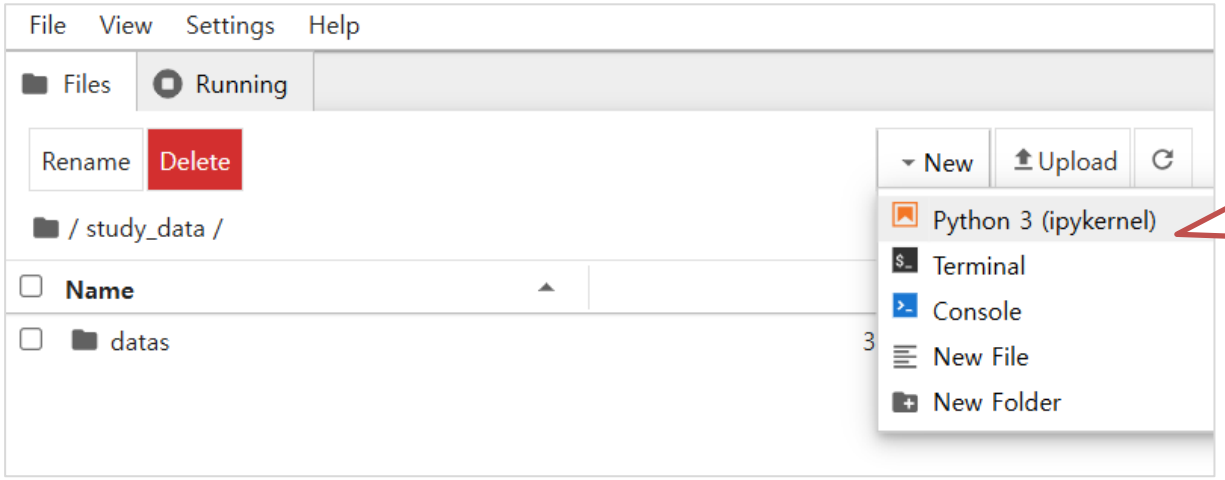
❖ 작업 디렉터리 생성

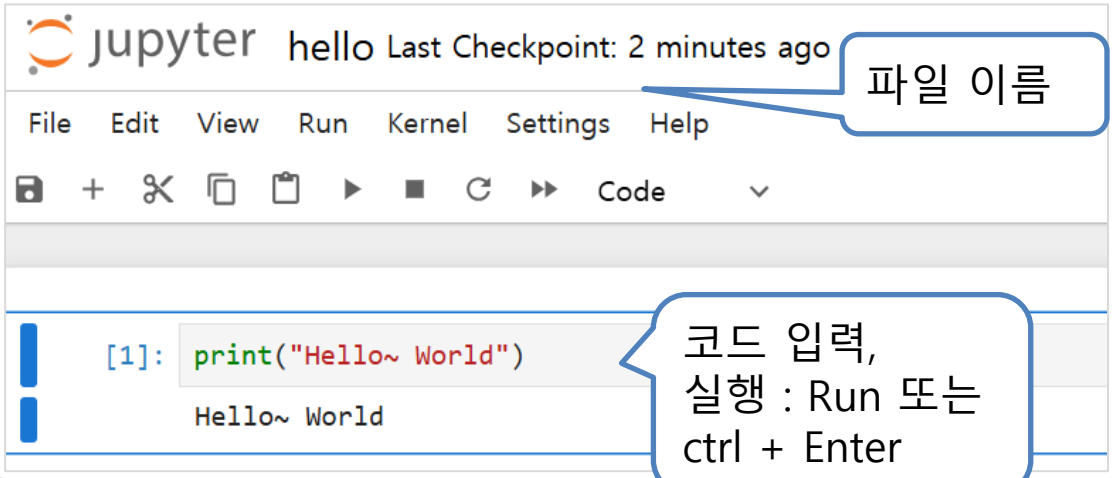
- User > Administrator 하위에
  study_data 폴더 생성 > 하위에 data 폴더 생성



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)
```

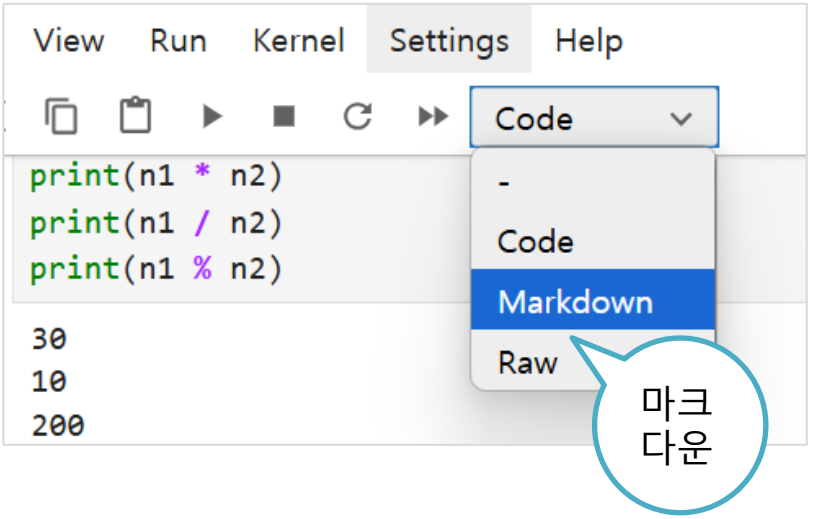
```
['계란', '라면', '콩나물']
계란
라면
콩나물
```

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)

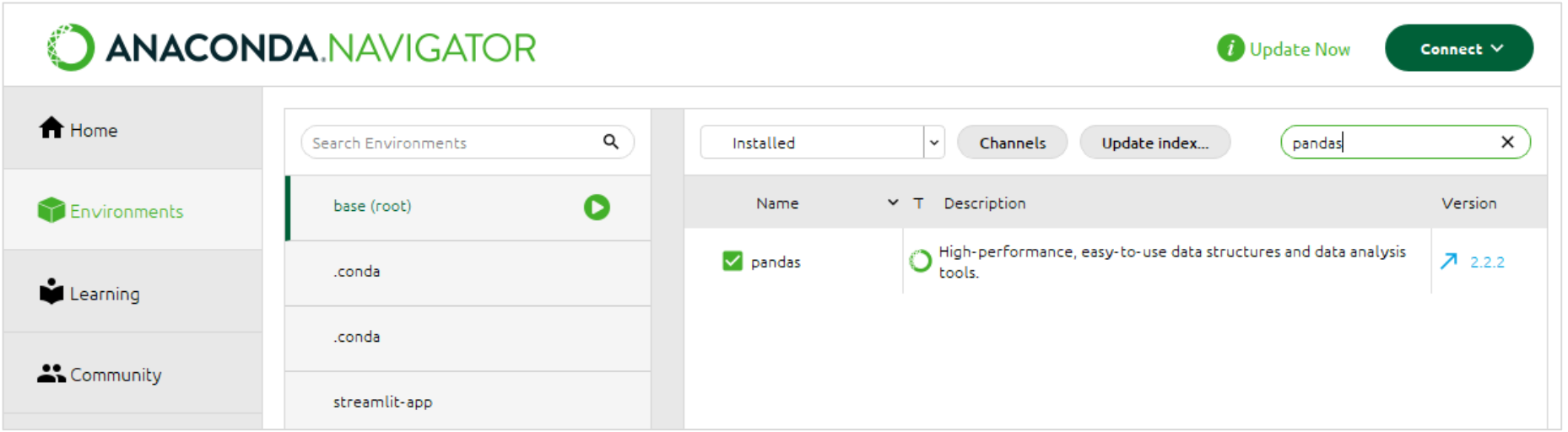
- **판다스**
  - 판다스(pandas) 라이브러리는 데이터를 수집하고 정리하는 데 최적화된 도구이다.
  - 가장 많이 사용되는 프로그래밍 언어인 **파이썬(python)**을 기반으로 한다.
  - **시리즈(Series)**와 **데이터프레임(DataFrame)**이라는 구조화된 데이터 형식을 제공한다.
  - 판다스 설치 - 아나콘다 플랫폼에 이미 설치됨

```
- 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)

```python
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

CSV 파일 만들기

- **데이터 프레임을 CSV 파일로 변환**

  CSV(Comma Seperated Value) 파일: 쉼표(,)로 데이터를 구분하여 저장하는 텍스트 파일 형식이다. 주로 표형태의 데이터를 저장하는 데 사용하며 엑셀에서 쉽고 열고 편집할 수 있음

☞ CSV 파일로 변환

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

☞ CSV 파일 읽기

pd.read_csv("computer.csv")

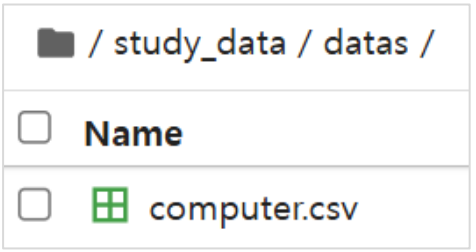
CSV 파일 만들기

- **데이터 프레임을 CSV 파일로 변환**

CSV 파일이란?

쉼표(,)로 데이터를 구분하여 저장하는 텍스트 파일 형식이다.

```
import pandas as pd

# csv 파일 만들기 - index 제외
df.to_csv("datas/computer.csv", index=False)

# csv 파일 읽기
computer = pd.read_csv("datas/computer.csv")
computer
```



	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)

데이터프레임

- **데이터 프레임 - 행/열 선택 : loc[ ] 사용**

```
# 행 선택 - loc[행(행이름)]
print(df)

print(df.loc[0])
print(df.loc[0:1])

# 행과 열 선택 - loc[행이름, 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"]])
```

```
품명           키보드
가격          20000
제품상세          유선
할인가       16000.0
Name: 0, dtype: object
    품명     가격 제품상세      할인가
0  키보드  20000   유선  16000.0
1  마우스  32000   무선  25600.0
========================================
키보드
32000
0    키보드
1    마우스
Name: 품명, dtype: object
    품명     가격
0  키보드  20000
1  마우스  32000
```

데이터프레임

- **데이터 프레임 - 행/열 선택 : iloc[ ] 사용**

```
# 행 선택 - iloc[행(인덱스번호)]
# print(df)

print(df.iloc[0])
print(df.iloc[0:2]) # 끝인덱스-1
print("=========================================")

# 행과 열 선택 - loc[행(인덱스번호), 열(인덱스번호)]
print(df.iloc[0, 0])
print(df.iloc[1, 1])
print(df.iloc[0:2, 0])
print(df.iloc[0:2, 0:2])
```

```
품명          키보드
가격         20000
제품상세         유선
할인가      16000.0
Name: 0, dtype: object
    품명     가격 제품상세      할인가
0  키보드  20000   유선  16000.0
1  마우스  32000   무선  25600.0
=========================================
키보드
32000
0    키보드
1    마우스
Name: 품명, dtype: object
    품명     가격
0  키보드  20000
1  마우스  32000
```

데이터프레임

- 데이터 프레임 - 요소값 변경(업데이트)

df.loc[행이름, 열이름] = “변경할 값 "

df.iloc[행번호, 열번호] = “변경할 값 "

- 데이터 프레임 - 요소값 추가

df[열] = “추가할 칼럼“ => 열 추가

df.loc[행] = “추가할 값“ => 행 추가

데이터프레임

- **데이터 프레임 - 행/열 요소 변경 및 추가**

```
# 요소값 변경 및 추가
df2 = df.copy() #원본 유지 복사
df2

# 품명 '마우스'를 '무선마우스'로 변경
df2.loc[1, "품명"] = "무선마우스"
# 무선 마우스의 가격을 35000으로 변경
df2.loc[1, "가격"] = 35000
# 무선 마우스의 할인가 수정
df2.loc[1, "할인가"] = df2.loc[1, "가격"] * 0.8
df2

# 행 추가
df2.loc[3] = ["키보드", 40000, "무선", 0.0]
df2.loc[3, "할인가"] = df2.loc[3, "가격"] * 0.8
df2
```

	품명	가격	제품상세	할인가
0	키보드	20000	유선	16000.0
1	무선마우스	35000	무선	28000.0
2	USB 메모리	11000	128GB	8800.0
3	키보드	40000	무선	32000.0

데이터프레임

- 데이터 프레임 - 행/열 삭제

df.drop(행, axis=0) => **행 삭제**

df.drop(열, axis=1) => **열 삭제**

```
# 행 삭제 - drop(행번호, axis=0)
# 열 삭제 - drop(칼럼명, axis=1)
df3 = df2.copy()
df3

# 3행 삭제
df3 = df3.drop(3, axis=0)
df3

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

	품명	가격
0	키보드	20000
1	무선마우스	35000
2	USB 메모리	11000

데이터프레임

- 데이터 프레임 – 인덱스 설정(변경)

df.copy() – 데이터프레임 복사

df.set_index("칼럼명") – 인덱스 설정

```
가격          20000
제품상세          유선
할인가       16000.0
Name: 키보드, dtype: object
        가격 제품상세      할인가
품명
키보드  20000   유선  16000.0
마우스  32000   무선  25600.0
             가격    제품상세       할인가
품명
키보드        20000      유선   16000.0
마우스        32000      무선   25600.0
USB 메모리   11000  128GB    8800.0
```

데이터프레임

- **데이터 프레임 – 인덱스 설정(변경)**

```
# 인덱스 설정 - set_index(칼럼명)
df4 = df.copy()
df4

df4 = df4.set_index("품명")
df4

# 행, 열 선택 - loc[ ]
print(df4.loc["키보드"])
print(df4.loc[["키보드", "마우스"]])
print(df4.loc["키보드":"USB 메모리"])
print("=========================================")

# 행, 열 선택 - iloc[ ]
print(df4.iloc[0])
print(df4.iloc[0:2])
print(df4.iloc[0:3])
```

KBO 팀순위 스크랩핑

- KBO 프로야구 팀 순위

팀 순위

팀 순위

일자별 연도별

2025년

2025

순위	팀명	경기	승	패	무	승률	게임차
1	한화	87	52	33	2	0.612	0
2	LG	88	48	38	2	0.558	4.5
3	롯데	89	47	39	3	0.547	5.5
4	KIA	88	45	40	3	0.529	7
5	KT	89	45	41	3	0.523	7.5

KBO 팀순위 스크랩핑

- KBO 프로야구 팀 순위

```
import requests
from bs4 import BeautifulSoup

# KBO 팀순위 사이트
url = 'https://www.koreabaseball.com/Record/TeamRank/TeamRank.aspx'
response = requests.get(url)
html = BeautifulSoup(response.text, 'html.parser')

# 첫번째 테이블 검색
first_table = html.select_one('table > tbody')
# print(first_table)

# nth-of-type(특정 열 선택)
순위 = first_table.select('tr > td:nth-of-type(1)')
# for i in 순위:
#   print(i.text)
[int(i.text) for i in 순위]
```

KBO 팀순위 스크랩핑

- KBO 프로야구 팀 순위

```
팀 = first_table.select('tr > td:nth-of-type(2)')
[i.text for i in 팀]

승 = first_table.select('tr > td:nth-of-type(4)')
[int(i.text) for i in 승]

패 = first_table.select('tr > td:nth-of-type(5)')
[int(i.text) for i in 패]

무 = first_table.select('tr > td:nth-of-type(6)')
[int(i.text) for i in 무]

승률 = first_table.select('tr > td:nth-of-type(7)')
[float(i.text) for i in 승률]
```

```
[0.612, 0.558, 0.547, 0.529, 0.523, 0.512, 0.5, 0.494, 0.424, 0.307]
```

KBO 팀순위 스크랩핑

- 텍스트 파일에 쓰고 읽기

```
# 텍스트 파일 쓰기

with open("datas/2025_kbo.txt", 'w') as f:
    f.write("========== 2025 한국 프로야구 성적표 ==========\n")
    head = "순위\t팀\t승\t패\t무\t승률"
    f.write(head)
    f.write('\n')
    for i in range(len(팀)): # 10번 반복
        txt = f'{순위[i].text}\t{팀[i].text}\t{승[i].text}\t{패[i].text}\t \
{무[i].text}\t{승률[i].text}\n'
        f.write(txt)
```

KBO 팀순위 스크랩핑

- 텍스트 파일에 쓰고 읽기

```
# 텍스트 파일 읽기

with open("datas/2025_kbo.txt", 'r') as f:
    data = f.read()
    print(data)
```

실습 문제

- 데이터 프레임 만들고 csv 파일로 변환하기

kbo_2025.csv

```
순위,팀,승,패,무,승률
1,한화,52,33,2,0.612
2,LG,48,38,2,0.558
3,롯데,47,39,3,0.547
4,KIA,45,40,3,0.529
5,KT,45,41,3,0.523
6,SSG,43,41,3,0.512
7,NC,40,40,5,0.5
8,삼성,43,44,1,0.494
9,두산,36,49,3,0.424
10,키움,27,61,3,0.307
```

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 프레임을 CSV 파일로 변환

```
# 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

```
# 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())
```

	메뉴	가격	분류	할인가
0	김밥	3000	한식	2700
1	비빔밥	7000	한식	6300
2	자장면	5500	중식	4950
3	우동	5500	일식	4950
4	김밥	3500	한식	3150
5	자장면	6000	중식	5400
6	김밥	4000	한식	3600

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
round(수, 자리수)	반올림
groupby(칼럼명)	그룹화

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])
```

데이터 전처리

- 그룹핑 - groupby()

```
import pandas as pd

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

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

# csv 파일 읽기
food = pd.read_csv("./datas/food2.csv")
food
```

```
메뉴,가격,분류,수량
김밥,3000,한식,10
비빔밥,7000,한식,10
자장면,5500,중식,15
우동,5500,일식,5
김밥,3500,한식,7
자장면,6000,중식,7
김밥,4000,한식,8
```

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

- **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

- **결측 데이터 프레임 생성**

```python
import pandas as pd
import numpy as np #수치 계산 라이브러리

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

# 칼럼 추가
df['C'] = np.nan
df

# csv 파일로 변환
df.to_csv("./datas/data_nan.csv", index=False)
```

data_nan.csv

	A	B	C
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**

```
# csv 파일 읽기
df_nan = pd.read_csv("./datas/data_nan.csv")
df_nan

# 데이터 탐색
# df_nan.info()
# df_nan.isna().sum()

# 결측치가 있는 1행 삭제
# inplace=True (실행중 즉시 삭제)
df_nan.drop(1, axis=0, inplace=True)
df_nan

# 결측치가 있는 열 삭제
df_nan = df_nan.drop('C', axis=1)
df_nan
```

	A	B
0	1.0	4.0
2	3.0	NaN

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)
```

```
메뉴    1
가격    1
분류    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())
```

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([1, 2, 3])	합계(리스트, 튜플)
np.mean([1, 2, 3])	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)**

```
# 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])
```

```
[1 2 3]
(3,)
1
3
3
[1 2 3]
[1 2 3]
[1 2]
```

Numpy(넘파이)

- **Numpy(Numeric 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:])
```

```
[[1 2 3]
 [4 5 6]]
(2, 3)
1
5
[[1 2 3]
 [4 5 6]]
[[5 6]]
```

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)
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
[0 2 4 6 8]
[[0 1 2 3 4]
 [5 6 7 8 9]]
[0 1 2 3 4 5 6 7 8 9]
```

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)
```

```
nan
<class 'float'>
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]]
[[1 1 1]
 [1 1 1]]
```

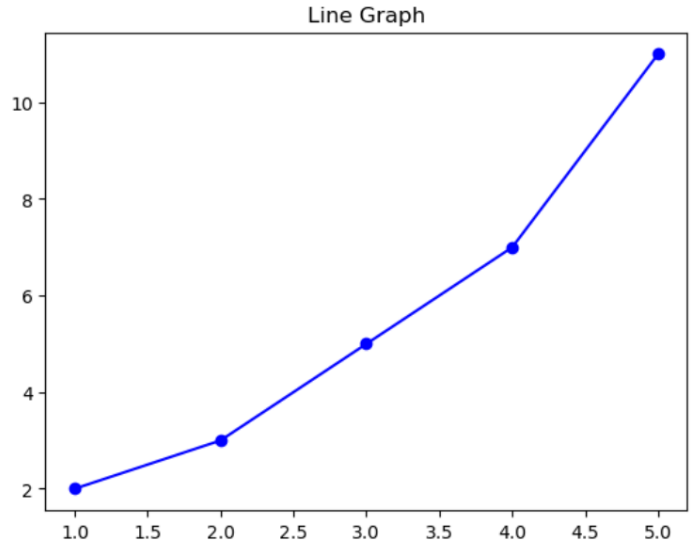
시각화 도구 – matplotlib

- **matplotlib 라이브러리**
  - Python에서 가장 널리 사용되는 시각화 라이브러리이다.
  - 데이터 분석, 과학 시각화, 보고서 작성 등 다양한 분야에서 그래프를 그릴 때 사용됨
  - 주요 그래프 종류

종류	함수명	예시
선 그래프	plot()	plt.plot(x, y)
막대 그래프(수직)	bar()	plt.bar(x, y)
막대 그래프(수평)	barh()	plt.barh(x, y)
히스토그램	hist()	plt.hist(data)
파이차트	pie()	plt.pie(data)

시각화 도구 - matplotlib

- 선 그래프

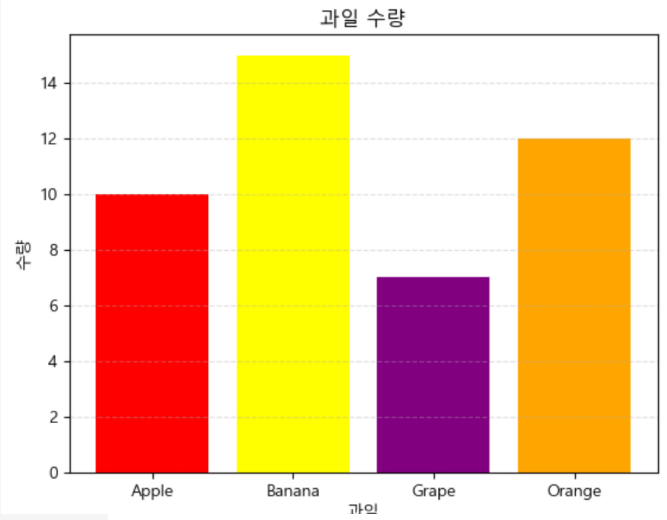



```
import matplotlib.pyplot as plt

# 데이터 준비
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

# 선 그래프 그리기
plt.plot(x, y, label='Prime numbers', color='blue', marker='o')
plt.title('Line Graph') #제목
plt.legend() #범례

# 그래프 출력
plt.show()
```

시각화 도구 - matplotlib

- 막대 그래프

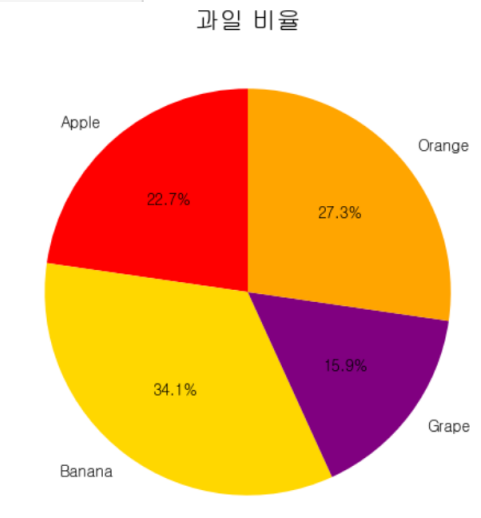
```
import matplotlib.pyplot as plt

# 데이터 준비
fruits = ['Apple', 'Banana', 'Grape', 'Orange']
counts = [10, 15, 7, 12]
color = ['red', 'yellow', 'purple', 'Orange']

# 한글 깨짐 방지 - Batang, Gulim 사용
plt.rc('font', family="Malgun Gothic")

# 막대 그래프
plt.bar(fruits, counts, color=color)
plt.title('과일 수량')
plt.xlabel('과일') # x축 제목
plt.ylabel('수량') # y축 제목

# y축 grid 추가
# plt.grid()
plt.grid(axis='y', linestyle='--', alpha=0.3)
plt.show()
```



시각화 도구 - matplotlib

- 원형 차트(파이 차트)

```
import matplotlib.pyplot as plt

# 데이터
fruits = ['Apple', 'Banana', 'Grape', 'Orange']
counts = [10, 15, 7, 12]
colors = ['red', 'gold', 'purple', 'orange']

plt.rc('font', family="Gulim")  # 한글 폰트

plt.figure(figsize=(6,6)) #그래프 크기(가로-6인치, 세로-6

# 파이 차트
plt.pie(counts, labels=fruits, autopct='%1.1f%%',
        startangle=90, colors=colors)

plt.title('과일 비율', fontsize=16, fontweight='bold')
plt.show()
```

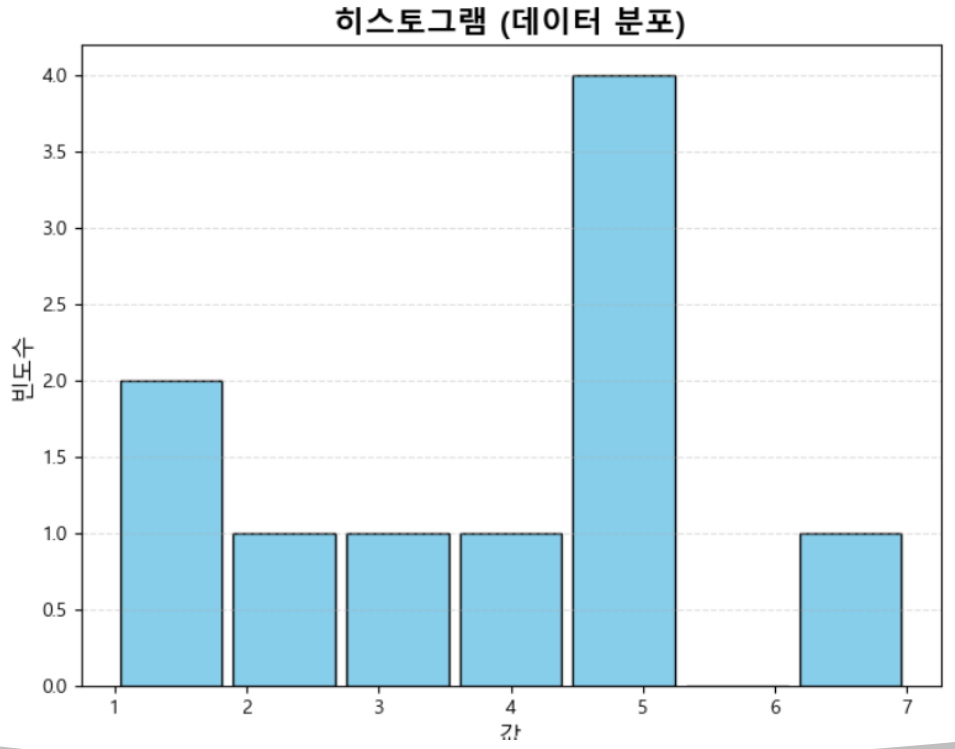


시각화 도구 - matplotlib

- 히스토그램

  히스토그램은 데이터를 여러 구간으로 나누고, 각 구간에 속하는 데이터 개수를 세서 막대로 표시합니다.

  bins=7 이라면 데이터 범위를 7개의 구간으로 나눠서 막대를 그립니다.



시각화 도구 - matplotlib

- 히스토그램

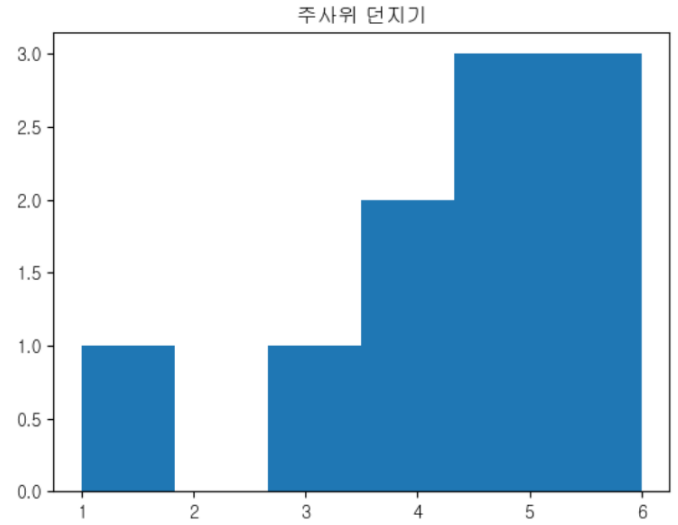
```
import matplotlib.pyplot as plt

# 데이터
data = [1, 1, 2, 3, 4, 5, 5, 5, 5, 7]

plt.rc('font', family='Batang')  # 한글 폰트 설정
plt.figure(figsize=(8,6))        # 그래프 크기

# 히스토그램 그리기
plt.hist(data, bins=7, color='skyblue', edgecolor='black', rwidth=0.9)

# 제목과 축 이름
plt.title('히스토그램 (데이터 분포)', fontsize=16, fontweight='bold')
plt.xlabel('값', fontsize=12)
plt.ylabel('빈도수', fontsize=12)

# y축 그리드 추가
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()
```

시각화 도구 - matplotlib

- 히스토그램

```
import matplotlib.pyplot as plt
import random

#
numbers = []
for i in range(10):
    dice = random.randint(1, 6)
    numbers.append(dice)

print(numbers)

plt.rc('font', family='Malgun Gothic')  # 한글 폰트 설정
plt.figure(figsize=(8,6))       # 그래프 크기

# 히스토그램 그리기
plt.hist(numbers, bins=6)

# 제목과 축 이름
plt.title('히스토그램 (주사위 던지기)', fontsize=16, fontweight='bold')
plt.xlabel('값', fontsize=12)
plt.ylabel('빈도수', fontsize=12)

# y축 그리드 추가
plt.grid(axis='y', linestyle='--', alpha=0.4)

plt.show()
```

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



seaborn 라이브러리

- **seaborn 라이브러리**
  - Python의 데이터 시각화 라이브러리로, 통계적 그래프를 쉽게 그릴 수 있도록 matplotlib을 기반으로 만들어졌다
  - pandas의 DataFrame과 잘 통합되어 있어, 데이터를 시각적으로 분석하고자 할 때 매우 유용하다

- **설치 방법**

  pip install seaborn

타이타닉호 데이터 분석

- 타이타닉호 데이터 준비

```
import seaborn as sns

df = sns.load_dataset('titanic')
df.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no

타이타닉호 데이터 분석

- 타이타닉호 데이터 정보(결측치 확인)

#	Column	Non-Null Count	Dtype
0	survived	891 non-null	int64
1	pclass	891 non-null	int64
2	sex	891 non-null	object
3	age	714 non-null	float64
4	sibsp	891 non-null	int64
5	parch	891 non-null	int64
6	fare	891 non-null	float64
7	embarked	889 non-null	object
8	class	891 non-null	category
9	who	891 non-null	object
10	adult_male	891 non-null	bool
11	deck	203 non-null	category
12	embark_town	891 non-null	object
13	alive	891 non-null	object
14	alone	891 non-null	bool

#	Column	Non-Null Count	Dtype
0	survived	891 non-null	int64
1	pclass	891 non-null	int64
2	sex	891 non-null	object
3	age	714 non-null	float64
4	sibsp	891 non-null	int64
5	parch	891 non-null	int64
6	fare	891 non-null	float64
7	embarked	889 non-null	object
8	class	891 non-null	category
9	who	891 non-null	object
10	adult_male	891 non-null	bool
11	embark_town	891 non-null	object
12	alive	891 non-null	object
13	alone	891 non-null	bool

타이타닉호 데이터 분석 및 처리

- 타이타닉호 데이터 분석 및 처리

```
# df.info()
# df.isnull().sum()
print(df.isna().sum())
df['deck'].value_counts(dropna=False)

# 데이터 전처리
df2 = df.copy() #원본 복사

# deck열 삭제
df2 = df2.drop('deck', axis=1)
# df2.info()
df2.columns # 칼럼 확인
```

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'embark_town', 'alive',
       'alone'],
      dtype='object')
```

타이타닉호 데이터 분석 및 처리

- 타이타닉호 데이터 분석 및 처리

```
# # age열 결측 여부 확인
print(df['age'].isnull())
print(df['age'].head(10))

# 데이터 대체 - 평균값 이용
mean_age = df2['age'].mean()
print(mean_age) #29.699

# 평균값으로 채우기
df2['age'] = df2['age'].fillna(mean_age)
print(df2['age'].head(10))
```

```
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
```

```
0    22.000000
1    38.000000
2    26.000000
3    35.000000
4    35.000000
5    29.699118
6    54.000000
```

타이타닉호 데이터 분석 및 처리

- 타이타닉호 데이터 분석 및 처리

```
# embark_town 열 데이터 대체 - 최빈값 이용
print(df2['embark_town'][825:830])

most_freq = df['embark_town'].mode()[0]
print(most_freq) #Southampton

# 최빈값으로 채우기
df['embark_town'] = df['embark_town'].fillna(most_freq)
print(df['embark_town'][825:830])
```

```
825    Queenstown
826    Southampton
827      Cherbourg
828    Queenstown
829            NaN
```

```
825    Queenstown
826    Southampton
827      Cherbourg
828    Queenstown
829    Southampton
```

seaborn 라이브러리

- **타이타닉호 - 성별에 따른 생존자 수 그래프**

```python
import seaborn as sns
import matplotlib.pyplot as plt

# 예제 데이터셋 로드
df = sns.load_dataset('titanic')

# 데이터 확인
print(df.head())

# 성별에 따른 생존자 수 시각화 (막대그래프)
sns.countplot(data=df, x='sex', hue='survived')
plt.title('성별에 따른 생존자 수')
plt.show()
```

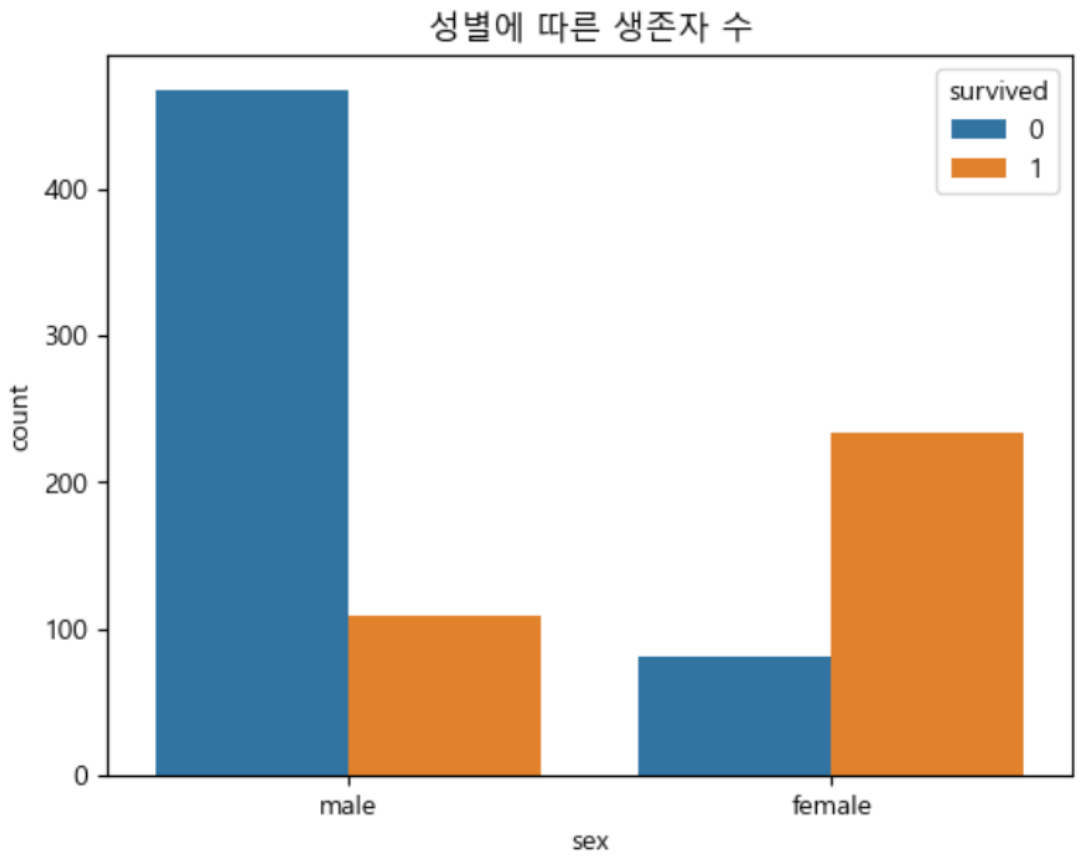
seaborn 라이브러리

- 타이타닉호 - 성별에 따른 생존자 수 그래프

```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

seaborn 라이브러리

- **타이타닉호 - 성별에 따른 생존자 수 그래프**

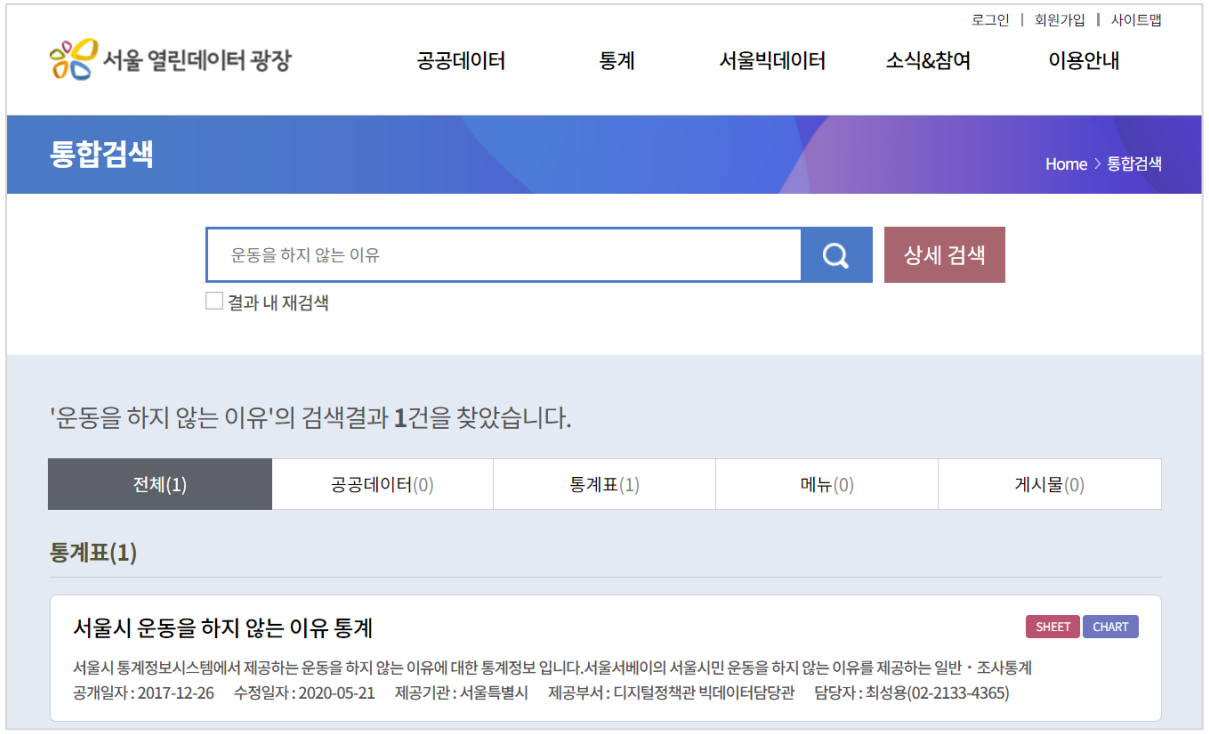


공공데이터 분석 및 시각화

- 자료 수집
  - 열린 데이터 광장(data.seoul.go.kr)

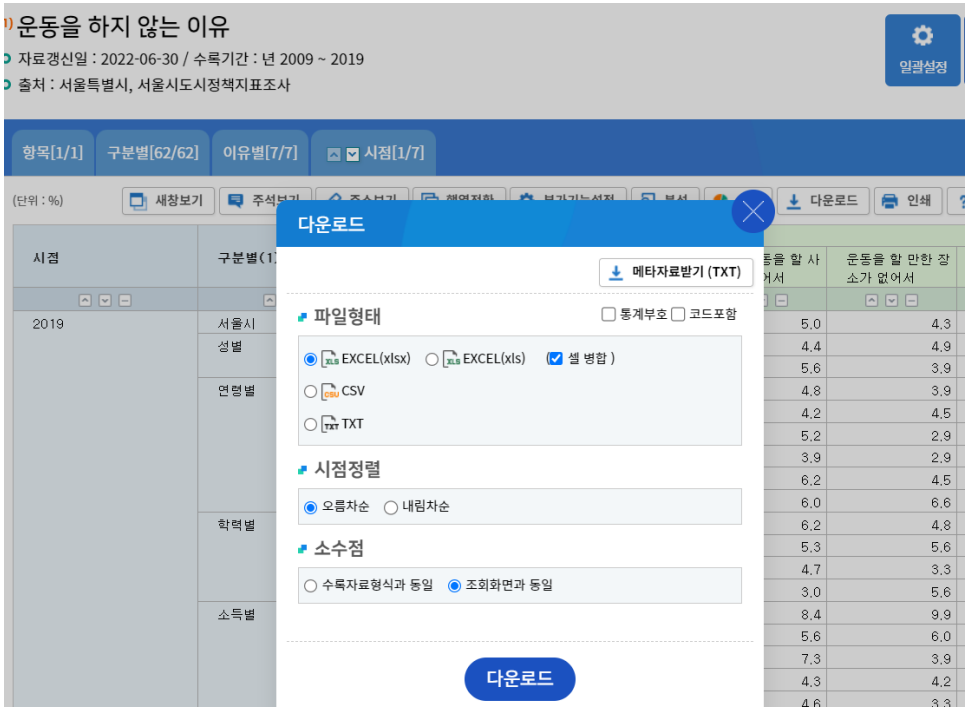
서울시 운동을 하지 않는 이유 통계

- 자료 수집
  - 운동을 하지 않는 이유 검색



서울시 운동을 하지 않는 이유 통계

- 자료 수집
  - excel 파일 다운로드



서울시 운동을 하지 않는 이유 통계

- 판다스로 엑셀 파일 읽어 오기
  - **pd.read_excel**(파일명)

```
import pandas as pd
import warnings
warnings.simplefilter('ignore')

# 제목행의 1번 인덱스만 출력(0번은 숨김)
not_exercise = pd.read_excel("./datas/exercise.xlsx", header=1)
not_exercise.head()
```

	시점	구분별 (1)	구분별 (2)	운동을 할 충분한 시간이 없어서	함께 운동을 할 사람이 없어서	운동을 할 만한 장소가 없어서
0	2019.0	서울시	소계	46.8	5.0	4.3
1	NaN	성별	남자	52.4	4.4	4.9
2	NaN	NaN	여자	42.5	5.6	3.9
3	NaN	연령별	10대	55.3	4.8	3.9
4	NaN	NaN	20대	46.0	4.2	4.5

서울시 운동을 하지 않는 이유 통계

- 데이터 전처리
  - **'시점' 칼럼 삭제**

```
# '시점' 칼럼 확인
print(not_exercise.columns[0])

# 칼럼 삭제 - 객체.drop()
not_exercise = not_exercise.drop('시점', axis=1)
not_exercise
```

시점

	구분별(1)	구분별(2)	운동을 할 충분한 시간이 없어서	함께 운동을 할 사람이 없어서
0	서울시	소계	46.8	5.0
1	성별	남자	52.4	4.4

서울시 운동을 하지 않는 이유 통계

- 데이터 전처리
  - **칼럼명 변경**

```
# 칼럼명 변경 - 객체.rename(), inplace=True(즉시 저장 및 실행)
not_exercise.rename(columns={'구분별(1)':'대분류', '구분별(2)':'분류'}, inplace=True)
not_exercise
```

	대분류	분류	운동을 할 충분한 시간이 없어서	함께 운동을 할 사람이 없어서	운동을
0	서울시	소계	46.8	5.0	
1	성별	남자	52.4	4.4	
2	NaN	여자	42.5	5.6	
3	연령별	10대	55.3	4.8	
4	NaN	20대	46.0	4.2	

서울시 운동을 하지 않는 이유 통계

- 데이터 전처리
  - **행 삭제**

45	NaN	영등포구	50.2
46	NaN	동작구	39.1
47	NaN	관악구	44.4
48	NaN	서초구	49.9
49	NaN	강남구	40.7
50	NaN	송파구	47.6
51	NaN	강동구	47.0

```
]: # 행 삭제 - 객체.drop(index=range(시작행, 끝행-1))
not_exercise.drop(index=range(22, 52), inplace=True)
not_exercise
```

서울시 운동을 하지 않는 이유 통계

- 데이터 전처리
  - **행 수정**

```
# 2행의 NaN을 '성별'로 변경 - '대분류' 칼럼의 2행을 수정
not_exercise.loc[2, '대분류'] = '성별'
not_exercise
```

	대분류	분류	운동을 할 충분한 시간이 없어서	함께 운동을 할 사람이 없어서
0	서울시	소계	46.8	5.0
1	성별	남자	52.4	4.4
2	성별	여자	42.5	5.6

서울시 운동을 하지 않는 이유 통계

- 데이터 전처리
  - **조건 검색**

```
# 대분류가 성별인 항목만 검색

not_exercise['대분류'] == '성별'
not_exercise[not_exercise['대분류'] == '성별']
```

	대분류	분류	운동을 할 충분한 시간이 없어서	함께 운동을 할 사람이 없어서
1	성별	남자	52.4	4.4
2	성별	여자	42.5	5.6

서울시 운동을 하지 않는 이유 통계

- 데이터 전처리
  - **깊은 복사 & 얕은 복사**

```
# 원본 유지 - 깊은 복사

not_ex_gender = not_exercise[not_exercise['대분류'] == '성별'].copy()  #깊은 복사
not_ex_gender
```

얕은 복사, 깊은 복사

```
# 얕은 복사
a = [1, 2, 3, 4]
a
```

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

```
b = a
b
```

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

```
b[1] = 10
b
```

```
[1, 10, 3, 4]
```

```
a   # b리스트를 변경하면 a(원본)도 똑같이 변경됨
```

```
[1, 10, 3, 4]
```

```
# 깊은 복사
a = [1, 2, 3, 4]
a
```

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

```
b = a.copy()
b
```

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

```
b[1] = 10
b
```

```
[1, 10, 3, 4]
```

```
a   # b리스트를 변경해도 a는 원본을 유지함
```

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

서울시 운동을 하지 않는 이유 통계

- 데이터 전처리
  - **칼럼 삭제**

```
# 대분류 칼럼 삭제
# not_ex_gender.drop("대분류", axis=1, inplace=True)
not_ex_gender.drop(columns='대분류', inplace=True)
not_ex_gender
```

	분류	운동을 할 충분한 시간이 없어서	함께 운동을 할 사람이 없어서
1	남자	52.4	4.4
2	여자	42.5	5.6

서울시 운동을 하지 않는 이유 통계

- 데이터 전처리
  - **인덱스 설정**

```
# index를 '분류'로 세팅
not_ex_gender.set_index('분류', inplace=True)
not_ex_gender
```

분류	운동을 할 충분한 시간이 없어서	함께 운동을 할 사람이 없어서
남자	52.4	4.4
여자	42.5	5.6

서울시 운동을 하지 않는 이유 통계

- 시각화
  - **파이 차트 그리기**

fig(figure) : 그래프를 그릴 공간

ax(axis) : 그 공간중 사용할 부분

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2)  # subplots() - 작은 그래프(1행 2열)
plt.rc('font', family='Malgun Gothic')

not_ex_gender['운동을 할 충분한 시간이 없어서'].plot.pie(ax=ax[0],
                autopct='%.1f%%', startangle=90)
ax[0].set_title('운동을 할 충분한 시간이 없어서')
ax[0].set_ylabel('')

not_ex_gender['운동을 싫어해서'].plot.pie(ax=ax[1],
                autopct='%.1f%%', startangle=90)
ax[1].set_title('운동을 싫어해서')
ax[1].set_ylabel('')
```

서울시 운동을 하지 않는 이유 통계

- 시각화
  - **파이 차트 그리기**

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