[📝필사 과제 안내]

필사를 진행하며, 주요 코드나 작업에 대해 손과 눈에 최대한 익으실 수 있도록 함입니다 :)

수업을 진행하시며 이해가 잘 되지 않았던 부분을 천천히 써보시거나,

단순히 다시 한 번 따라서 진행해보시는 것만으로도 좋습니다.

새롭게 노트북파일을 만드신 뒤, 써주세요 😀

---

01. Analysis Seoul CCTV

1. 데이터 읽기

```
1 import pandas as pd
```

```
1 CCTV_Seoul = pd.read_csv("../data/01. Seoul_CCTV.csv")
2 CCTV_Seoul.head()
```

	기관명	소계	2013년도 이전	2014년	2015년	2016년
0	강남구	3238	1292	430	584	932
1	강동구	1010	379	99	155	377
2	강북구	831	369	120	138	204
3	강서구	911	388	258	184	81
4	관악구	2109	846	260	390	613

```
1 CCTV_Seoul.columns
```

```
Index(['기관명', '소계', '2013년도 이전', '2014년', '2015년', '2016년'],
dtype='object')
```

```
1 CCTV_Seoul.columns[0]
```

```
'기관명'
```

```
1 CCTV_Seoul.rename(columns={CCTV_Seoul.columns[0]: "구별"}, inplace=True)
2 CCTV_Seoul.head()
```

	구별	소계	2013년도 이전	2014년	2015년	2016년
0	강남구	3238	1292	430	584	932
1	강동구	1010	379	99	155	377
2	강북구	831	369	120	138	204
3	강서구	911	388	258	184	81
4	관악구	2109	846	260	390	613

```
pop_Seoul = pd.read_excel("../data/01. Seoul_Population.xls")
pop_Seoul.head()
```

	기간	자치구	세대	인구	인구.1	인구.2	인구.3	인구.4	인구.5	인구.6
0	기간	자치구	세대	합계	합계	합계	한국인	한국인	한국인	등록외국인
1	기간	자치구	세대	계	남자	여자	계	남자	여자	계

```
pop_Seoul = pd.read_excel(
    "../data/01. Seoul_Population.xls", header=2, usecols="B, D, G, J, N")
pop_Seoul.head()
```

	자치구	계	계.1	계.2	65세이상고령자
0	합계	10124579	9857426	267153	1365126
1	종로구	164257	154770	9487	26182
2	중구	134593	125709	8884	21384
3	용산구	244444	229161	15283	36882
4	성동구	312711	304808	7903	41273

```
pop_Seoul.rename(
    columns={
        pop_Seoul.columns[0]: "구별",
        pop_Seoul.columns[1]: "인구수",
        pop_Seoul.columns[2]: "한국인",
        pop_Seoul.columns[3]: "외국인",
        pop_Seoul.columns[4]: "고령자",
    },
    inplace=True
)
pop_Seoul.head()
```

	구별	인구수	한국인	외국인	고령자
0	합계	10124579	9857426	267153	1365126
1	종로구	164257	154770	9487	26182
2	중구	134593	125709	8884	21384
3	용산구	244444	229161	15283	36882

Pandas 기초

- Python에서 R 만큼의 강력한 데이터 핸들링 성능을 제공하는 모듈
- 단일 프로세스에서는 최대 효율
- 코딩 가능하고 응용 가능한 엑셀로 받아들여도 됨
- 누군가 스테로이드를 맞은 엑셀로 표현함

Series

- index와 value로 이루어져 있습니다
- 한 가지 데이터 타입만 가질 수 있습니다

```
1 import pandas as pd
2 import numpy as np
```

- pandas는 통상 pd
- numpy는 통상 np

```
1 pd.Series()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWar
  """Entry point for launching an IPython kernel.
Series([], dtype: float64)
```

```
1 pd.Series([1, 2])
```

```
0    1
1    2
dtype: int64
```

```
1 # pd.Series([1, 2, 3, 4], dtype=float64)
```

```
1 pd.Series([1, 2, 3, 4], dtype=np.float64)
```

```
0    1.0
1    2.0
2    3.0
3    4.0
dtype: float64
```

```
1 pd.Series([1, 2, 3, 4], dtype=str)
```

```
0    1
1    2
2    3
3    4
dtype: object
```

```
1 pd.Series(np.array([1, 2, 3]))
```

```
0    1
1    2
2    3
dtype: int64
```

```
1 pd.Series({"Key": "Value"})
```

```
Key    Value
dtype: object
```

```
1 data = pd.Series([1, 2, 3, 4, 5])
2 data
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

```
1 data = pd.Series([1, 2, 3, 4, "5"])
2 data
```

```
0    1
1    2
2    3
3    4
4    5
dtype: object
```

```
1 # 짝수를 찾고 싶다
2 # data % 2
```

```
1 data = pd.Series([1, 2, 3, 4])
2 data
```

```
0    1
1    2
2    3
3    4
dtype: int64
```

```
data % 2
```

```
0    1
1    0
2    1
3    0
dtype: int64
```

날짜 데이터

```
dates = pd.date_range("20210101", periods=6)
dates
```

```
DatetimeIndex(['2021-01-01', '2021-01-02', '2021-01-03', '2021-01-04',
               '2021-01-05', '2021-01-06'],
              dtype='datetime64[ns]', freq='D')
```

DataFrame

- pd.Series()
  - index, value
- pd.DataFrame()
  - index, value, column

```
1
```

```
# 표준정규분포에서 샘플링한 난수 생성
data = [1, 2, 3, 4]
data
```

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

```
df = pd.DataFrame(data, columns=["A"], index=["A", "B", "C", "D"])
df
```

	A

데이터 프레임 정보 탐색

- df.head()

```
1 df.head()
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096

- df.tail()

```
1 df.tail()
```

	A	B	C	D
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

- df.index

```
1 df.index
```

```
DatetimeIndex(['2021-01-01', '2021-01-02', '2021-01-03', '2021-01-04',
               '2021-01-05', '2021-01-06'],
              dtype='datetime64[ns]', freq='D')
```

```
1 df.columns
```

```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

```
1 df.values
```

```
array([[-0.73781701,  0.46598837,  2.2852592 , -0.537783  ],
       [-0.21842499,  0.0318088 ,  2.49341958,  0.79512753],
       [-1.44181122, -0.7379117 ,  1.10107334,  0.33198805],
       [-0.37915635,  0.72701723, -1.80799187, -0.26771797],
       [-0.38694081, -0.59565941, -0.37206786,  1.55509569],
       [ 0.38870316,  0.48087685,  0.06739712,  0.23001877]])
```

- df.info() : 데이터 프레임의 기본 정보 확인

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 6 entries, 2021-01-01 to 2021-01-06
Freq: D
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A       6 non-null      float64
 1   B       6 non-null      float64
 2   C       6 non-null      float64
 3   D       6 non-null      float64
dtypes: float64(4)
memory usage: 240.0 bytes
```

- df.describe() : 데이터 프레임의 기술통계 정보 확인

```
1 df.describe()
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.462575	0.062020	0.627848	0.351122
std	0.605248	0.608966	1.655208	0.752973
min	-1.441811	-0.737912	-1.807992	-0.537783
25%	-0.650098	-0.438792	-0.262202	-0.143284
50%	-0.383049	0.248899	0.584235	0.281003
75%	-0.258608	0.477155	1.989213	0.679343
max	0.388703	0.727017	2.493420	1.555096

데이터 정렬

- sort_values()
- 특정 컬럼(열)을 기준으로 데이터를 정렬합니다

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df.sort_values(by="B", ascending=False, inplace=True)
```

```
1 df
```

	A	B	C	D
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-06	0.388703	0.480877	0.067397	0.230019
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-03	-1.441811	-0.737912	1.101073	0.331988

데이터 선택

```
1 df
```

	A	B	C	D
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-06	0.388703	0.480877	0.067397	0.230019
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-03	-1.441811	-0.737912	1.101073	0.331988

```
1 # 한 개 컬럼 선택
2 df["A"]
```

```
2021-01-04   -0.379156
2021-01-06    0.388703
```

```
2021-01-01   -0.737817
2021-01-02   -0.218425
2021-01-05   -0.386941
2021-01-03   -1.441811
Name: A, dtype: float64
```

```
1 type(df["A"])
```

```
pandas.core.series.Series
```

```
1 df.A
```

```
2021-01-04   -0.379156
2021-01-06    0.388703
2021-01-01   -0.737817
2021-01-02   -0.218425
2021-01-05   -0.386941
2021-01-03   -1.441811
Name: A, dtype: float64
```

```
1 df = pd.DataFrame(data, index=dates, columns=["A", "B", "C", "D"])
2 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df.D
```

```
2021-01-01   -0.537783
2021-01-02    0.795128
2021-01-03    0.331988
2021-01-04   -0.267718
2021-01-05    1.555096
2021-01-06    0.230019
Freq: D, Name: D, dtype: float64
```

```
1 # 두 개 이상 컬럼 선택
2 df[["A", "B"]]
```

	A	B
2021-01-01	-0.737817	0.465988
2021-01-02	-0.218425	0.031809
2021-01-03	-1.441811	-0.737912

offset index

- [n:m] : n부터 m-1 까지
- 인덱스나 컬럼의 이름으로 slice 하는 경우는 끝을 포함합니다

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df[0:3]
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988

```
1 df["20210101":"20210104"]
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718

- loc : location
- index 이름으로 특정 행, 열을 선택합니다

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df.loc[:, ["A", "B"]]
```

	A	B
2021-01-01	-0.737817	0.465988
2021-01-02	-0.218425	0.031809
2021-01-03	-1.441811	-0.737912
2021-01-04	-0.379156	0.727017
2021-01-05	-0.386941	-0.595659
2021-01-06	0.388703	0.480877

```
1 df.loc["20210102":"20210104", ["A", "D"]]
```

	A	D
2021-01-02	-0.218425	0.795128
2021-01-03	-1.441811	0.331988
2021-01-04	-0.379156	-0.267718

```
1 df.loc["20210102":"20210104", "A":"D"]
```

	A	B	C	D
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df.loc["20210102", ["A", "B"]]
```

```
A   -0.218425
B    0.031809
Name: 2021-01-02 00:00:00, dtype: float64
```

- iloc : inter location
  - 컴퓨터가 인식하는 인덱스 값으로 선택

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df.iloc[3]
```

```
A   -0.379156
B    0.727017
C   -1.807992
D   -0.267718
Name: 2021-01-04 00:00:00, dtype: float64
```

```
1 df.iloc[3, 2]
```

```
-1.8079918710632994
```

```
1 df.iloc[3:5, 0:2]
```

	A	B
2021-01-04	-0.379156	0.727017
2021-01-05	-0.386941	-0.595659

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df.iloc[[1, 2, 4], [0, 2]]
```

	A	C
2021-01-02	-0.218425	2.493420
2021-01-03	-1.441811	1.101073
2021-01-05	-0.386941	-0.372068

```
1 df.iloc[:, 1:3]
```

	B	C
2021-01-01	0.465988	2.285259
2021-01-02	0.031809	2.493420
2021-01-03	-0.737912	1.101073
2021-01-04	0.727017	-1.807992
2021-01-05	-0.595659	-0.372068
2021-01-06	0.480877	0.067397

condition

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 # A 컬럼에서 0보다 큰 숫자(양수)만 선택
2
3 df["A"] > 0
```

```
2021-01-01    False
2021-01-02    False
2021-01-03    False
2021-01-04    False
2021-01-05    False
2021-01-06     True
Freq: D, Name: A, dtype: bool
```

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df[df["A"] > 0]
```

	A	B	C	D
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df[df > 0]
```

	A	B	C	D
2021-01-01	NaN	0.465988	2.285259	NaN
2021-01-02	NaN	0.031809	2.493420	0.795128
2021-01-03	NaN	NaN	1.101073	0.331988

- NaN : Not a Number

2021-01-05	NaN	NaN	NaN	1.555096

컬럼 추가

- 기존 컬럼이 없으면 추가
- 기존 컬럼이 있으면 수정

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df["E"] = ["one", "one", "two", "tree", "four", "seven"]
2 df
```

	A	B	C	D	E
2021-01-01	-0.737817	0.465988	2.285259	-0.537783	one
2021-01-02	-0.218425	0.031809	2.493420	0.795128	one
2021-01-03	-1.441811	-0.737912	1.101073	0.331988	two
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718	tree
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096	four
2021-01-06	0.388703	0.480877	0.067397	0.230019	seven

- isin()
- 특정 요소가 있는지 확인

```
1 df["E"].isin(["two"])
```

```
2021-01-01    False
2021-01-02    False
2021-01-03     True
2021-01-04    False
2021-01-05    False
2021-01-06    False
Freq: D, Name: E, dtype: bool
```

```
df["E"].isin(["two", "five"])
```

```
2021-01-01    False
2021-01-02    False
2021-01-03     True
2021-01-04    False
2021-01-05    False
2021-01-06    False
Freq: D, Name: E, dtype: bool
```

```
df["E"].isin(["two", "five", "three"])
```

```
2021-01-01    False
2021-01-02    False
2021-01-03     True
2021-01-04    False
2021-01-05    False
2021-01-06    False
Freq: D, Name: E, dtype: bool
```

```
df[df["E"].isin(["two", "five", "three"])]
```

	A	B	C	D	E
2021-01-03	-1.441811	-0.737912	1.101073	0.331988	two

특정 컬럼 제거

- del
- drop

```
df
```

	A	B	C	D	E

```
1 del df["E"]
2 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 df.drop(["D"], axis=1) # axis=0 가로, axis=1 세로
```

	A	B	C
2021-01-01	-0.737817	0.465988	2.285259
2021-01-02	-0.218425	0.031809	2.493420
2021-01-03	-1.441811	-0.737912	1.101073
2021-01-04	-0.379156	0.727017	-1.807992
2021-01-05	-0.386941	-0.595659	-0.372068
2021-01-06	0.388703	0.480877	0.067397

```
1 df.drop(["20210104"])
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

apply()

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096

```
1 df["A"].apply("sum")
```

```
-2.7754472042709937
```

```
1 df["A"].apply("mean")
```

```
-0.46257453404516563
```

```
1 df["A"].apply("min"), df["A"].apply("max")
```

```
(-1.4418112187047896, 0.38870315635485564)
```

```
1 df[["A", "D"]].apply("sum")
```

```
A   -2.775447
D    2.106729
dtype: float64
```

```
1 df["A"].apply(np.sum)
```

```
2021-01-01   -0.737817
2021-01-02   -0.218425
2021-01-03   -1.441811
2021-01-04   -0.379156
2021-01-05   -0.386941
2021-01-06    0.388703
Freq: D, Name: A, dtype: float64
```

```
1 df["A"].apply(np.mean)
```

```
2021-01-01   -0.737817
2021-01-02   -0.218425
2021-01-03   -1.441811
2021-01-04   -0.379156
2021-01-05   -0.386941
2021-01-06    0.388703
Freq: D, Name: A, dtype: float64
```

```
1 df["A"].apply(np.std)
```

```
2021-01-01    0.0
2021-01-02    0.0
```

```
2021-01-03    0.0
2021-01-04    0.0
2021-01-05    0.0
2021-01-06    0.0
Freq: D, Name: A, dtype: float64
```

```
1 df.apply(np.sum)
```

```
A   -2.775447
B    0.372120
C    3.767090
D    2.106729
dtype: float64
```

```
1 df
```

	A	B	C	D
2021-01-01	-0.737817	0.465988	2.285259	-0.537783
2021-01-02	-0.218425	0.031809	2.493420	0.795128
2021-01-03	-1.441811	-0.737912	1.101073	0.331988
2021-01-04	-0.379156	0.727017	-1.807992	-0.267718
2021-01-05	-0.386941	-0.595659	-0.372068	1.555096
2021-01-06	0.388703	0.480877	0.067397	0.230019

```
1 def plusminus(num):
2     return "plus" if num > 0 else "minus"
```

```
1 df["A"].apply(plusminus)
```

```
2021-01-01    minus
2021-01-02    minus
2021-01-03    minus
2021-01-04    minus
2021-01-05    minus
2021-01-06     plus
Freq: D, Name: A, dtype: object
```

```
1 df["A"].apply(lambda num: "plus" if num > 0 else "minus")
```

```
2021-01-01    minus
2021-01-02    minus
2021-01-03    minus
2021-01-04    minus
2021-01-05    minus
2021-01-06     plus
Freq: D, Name: A, dtype: object
```

---

2. CCTV 데이터 훑어보기

```
1 CCTV_Seoul.head()
```

	구별	소계	2013년도 이전	2014년	2015년	2016년
0	강남구	3238	1292	430	584	932
1	강동구	1010	379	99	155	377
2	강북구	831	369	120	138	204
3	강서구	911	388	258	184	81
4	관악구	2109	846	260	390	613

```
1 CCTV_Seoul.tail()
```

	구별	소계	2013년도 이전	2014년	2015년	2016년
20	용산구	2096	1368	218	112	398
21	은평구	2108	1138	224	278	468
22	종로구	1619	464	314	211	630
23	중구	1023	413	190	72	348
24	중랑구	916	509	121	177	109

```
1 CCTV_Seoul.sort_values(by="소계", ascending=True).head(5)
```

	구별	소계	2013년도 이전	2014년	2015년	2016년
9	도봉구	825	238	159	42	386
2	강북구	831	369	120	138	204
5	광진구	878	573	78	53	174
3	강서구	911	388	258	184	81
24	중랑구	916	509	121	177	109

```
1 CCTV_Seoul.sort_values(by="소계", ascending=False).head(5)
```

	구별	소계	2013년도 이전	2014년	2015년	2016년
0	강남구	3238	1292	430	584	932

```
# 기존 컬럼이 없으면 추가, 있으면 수정
CCTV_Seoul["최근증가율"] = (
    (CCTV_Seoul["2016년"] + CCTV_Seoul["2015년"] + CCTV_Seoul["2014년"]) / CCTV_Se
)

CCTV_Seoul.sort_values(by="최근증가율", ascending=False).head(5)
```

	구별	소계	2013년도 이전	2014년	2015년	2016년	최근증가율
22	종로구	1619	464	314	211	630	248.922414
9	도봉구	825	238	159	42	386	246.638655
12	마포구	980	314	118	169	379	212.101911
8	노원구	1566	542	57	451	516	188.929889
1	강동구	1010	379	99	155	377	166.490765

3. 인구현황 데이터 훑어보기

```
pop_Seoul.head()
```

	구별	인구수	한국인	외국인	고령자
0	합계	10124579	9857426	267153	1365126
1	종로구	164257	154770	9487	26182
2	중구	134593	125709	8884	21384
3	용산구	244444	229161	15283	36882
4	성동구	312711	304808	7903	41273

```
pop_Seoul.tail()
```

	구별	인구수	한국인	외국인	고령자
21	관악구	520929	503297	17632	70046
22	서초구	445401	441102	4299	53205
23	강남구	561052	556164	4888	65060
24	송파구	671173	664496	6677	76582
25	강동구	440359	436223	4136	56161

```
1 pop_Seoul.drop([0], axis=0, inplace=True)
2 pop_Seoul.head()
```

	구별	인구수	한국인	외국인	고령자
1	종로구	164257	154770	9487	26182
2	중구	134593	125709	8884	21384
3	용산구	244444	229161	15283	36882
4	성동구	312711	304808	7903	41273
5	광진구	372298	357703	14595	43953

```
1 pop_Seoul["구별"].unique()
```

```
array(['종로구', '중구', '용산구', '성동구', '광진구', '동대문구', '중랑구', '성북구', '강북구',
       '도봉구', '노원구', '은평구', '서대문구', '마포구', '양천구', '강서구', '구로구', '금천구',
       '영등포구', '동작구', '관악구', '서초구', '강남구', '송파구', '강동구'],
dtype=object)
```

```
1 len(pop_Seoul["구별"].unique())
```

```
25
```

```
1 # 외국인비율, 고령자비율
2
3 pop_Seoul["외국인비율"] = pop_Seoul["외국인"] / pop_Seoul["인구수"] * 100
4 pop_Seoul["고령자비율"] = pop_Seoul["고령자"] / pop_Seoul["인구수"] * 100
5 pop_Seoul.head()
```

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
1	종로구	164257	154770	9487	26182	5.775705	15.939656
2	중구	134593	125709	8884	21384	6.600640	15.887899
3	용산구	244444	229161	15283	36882	6.252148	15.088118
4	성동구	312711	304808	7903	41273	2.527254	13.198448
5	광진구	372298	357703	14595	43953	3.920247	11.805865

```
1 pop_Seoul.sort_values(["인구수"], ascending=False).head(5)
```

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
24	송파구	671173	664496	6677	76582	0.994825	11.410173

```
1 pop_Seoul.sort_values(["외국인"], ascending=False).head(5)
```

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
19	영등포구	402024	368550	33474	53981	8.326369	13.427308
17	구로구	441559	410742	30817	58794	6.979135	13.315095
18	금천구	253491	235154	18337	34170	7.233787	13.479769
21	관악구	520929	503297	17632	70046	3.384722	13.446362
6	동대문구	366011	350647	15364	55718	4.197688	15.223040

```
1 pop_Seoul.sort_values(["외국인비율"], ascending=False).head()
```

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
19	영등포구	402024	368550	33474	53981	8.326369	13.427308
18	금천구	253491	235154	18337	34170	7.233787	13.479769
17	구로구	441559	410742	30817	58794	6.979135	13.315095
2	중구	134593	125709	8884	21384	6.600640	15.887899
3	용산구	244444	229161	15283	36882	6.252148	15.088118

```
1 pop_Seoul.sort_values(by="고령자", ascending=False).head()
```

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
24	송파구	671173	664496	6677	76582	0.994825	11.410173
16	강서구	608255	601691	6564	76032	1.079153	12.500021
12	은평구	491202	486794	4408	74559	0.897390	15.178888
11	노원구	558075	554403	3672	74243	0.657976	13.303409
21	관악구	520929	503297	17632	70046	3.384722	13.446362

```
1 pop_Seoul.sort_values(by="고령자비율", ascending=False).head()
```

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
0	강북구	328002	324479	3523	56530	1.074079	17.234651

Pandas에서 데이터 프레임을 병합하는 방법

- pd.concat()
- pd.merge()
- pd.join()

pd.merge(left, right)

```
# 딕셔너리 안의 리스트 형태

left = pd.DataFrame({
    "key": ["K0", "K4", "K2", "K3"],
    "A": ["A0", "A1", "A2", "A3"],
    "B": ["B0", "B1","B2", "B3"]
})
left
```

	key	A	B
0	K0	A0	B0
1	K4	A1	B1
2	K2	A2	B2
3	K3	A3	B3

```
# 리스트 안의 딕셔너리 형태

right = pd.DataFrame([
    {"key":"K0", "C":"C0", "D":"D0"},
    {"key":"K1", "C":"C1", "D":"D1"},
    {"key":"K2", "C":"C2", "D":"D2"},
    {"key":"K3", "C":"C3", "D":"D3"},
])
right
```

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

pd.merge()

- 두 데이터 프레임에서 컬럼이나 인덱스를 기준으로 잡고 병합하는 방법
- 기준이 되는 컬럼이나 인덱스를 키값이라고 합니다
- 기준이 되는 키값은 두 데이터 프레임에 모두 포함되어 있어야 합니다

```
1 pd.merge(left, right, how="inner", on="key")
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K2	A2	B2	C2	D2
2	K3	A3	B3	C3	D3

```
1 pd.merge(left, right, how="left", on="key")
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K4	A1	B1	NaN	NaN
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

```
1 left
```

	key	A	B
0	K0	A0	B0
1	K4	A1	B1
2	K2	A2	B2
3	K3	A3	B3

```
1 right
```

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

```
1 pd.merge(left, right, how="right", on="key")
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	NaN	NaN	C1	D1
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

```
1 pd.merge(left, right, how="outer", on="key")
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K4	A1	B1	NaN	NaN
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3
4	K1	NaN	NaN	C1	D1

---

4. 두 데이터 합치기

```
1 CCTV_Seoul.head(1)
```

	구별	소계	2013년도 이전	2014년	2015년	2016년	최근증가율
0	강남구	3238	1292	430	584	932	150.619195

```
1 pop_Seoul.head(1)
```

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
1	종로구	164257	154770	9487	26182	5.775705	15.939656

```
1 data_result = pd.merge(CCTV_Seoul, pop_Seoul, on="구별")
2 data_result.head()
```

	구별	소계	2013년도 이전	2014년	2015년	2016년	최근증가율	인구수	한국인	외국인	고령자	
0	강남	3238	1292	430	584	932	150.619195	561052	556164	4888	65060	0.

년도별 데이터 컬럼 삭제

- del
- drop()

```
1 del data_result["2013년도 이전"]
```

```
1 del data_result["2014년"]
```

```
1 data_result.head(3)
```

	구별	소계	2015년	2016년	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고
0	강남구	3238	584	932	150.619195	561052	556164	4888	65060	0.871220	11.
1	강동구	1010	155	377	166.490765	440359	436223	4136	56161	0.939234	12.
2	강북구	831	138	204	125.203252	328002	324479	3523	56530	1.074079	17.

```
1 data_result.drop(["2015년", "2016년"], axis=1, inplace=True)
```

```
1 data_result.head()
```

	구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율
0	강남구	3238	150.619195	561052	556164	4888	65060	0.871220	11.596073
1	강동구	1010	166.490765	440359	436223	4136	56161	0.939234	12.753458
2	강북구	831	125.203252	328002	324479	3523	56530	1.074079	17.234651
3	강서구	911	134.793814	608255	601691	6564	76032	1.079153	12.500021
4	관악구	2109	149.290780	520929	503297	17632	70046	3.384722	13.446362

인덱스 변경

- set_index()
- 선택한 컬럼을 데이터 프레임의 인덱스로 지정

```
1 data_result.set_index("구별", inplace=True)
2 data_result.head()
```

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율
강남구	3238	150.619195	561052	556164	4888	65060	0.871220	11.596073
강동구	1010	166.490765	440359	436223	4136	56161	0.939234	12.753458
강북구	831	125.203252	328002	324479	3523	56530	1.074079	17.234651
강서구	911	134.793814	608255	601691	6564	76032	1.079153	12.500021

상관계수

- corr()
- correlation 의 약자입니다
- 상관계수가 0.2 이상인 데이터를 비교

```
1 data_result.corr()
```

	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자
소계	1.000000	-0.264378	0.232555	0.227852	0.030421	0.163905	-0.045956	-0.26
최근증가율	-0.264378	1.000000	-0.097165	-0.086341	-0.156421	-0.072251	-0.047102	0.19
인구수	0.232555	-0.097165	1.000000	0.998151	-0.167243	0.936737	-0.601076	-0.63
한국인	0.227852	-0.086341	0.998151	1.000000	-0.226853	0.936155	-0.645463	-0.62
외국인	0.030421	-0.156421	-0.167243	-0.226853	1.000000	-0.175318	0.838612	-0.02
고령자	0.163905	-0.072251	0.936737	0.936155	-0.175318	1.000000	-0.620300	-0.34
외국인비율	-0.045956	-0.047102	-0.601076	-0.645463	0.838612	-0.620300	1.000000	0.24
고령자비율	-0.267841	0.190396	-0.637414	-0.628360	-0.021147	-0.348840	0.242816	1.00

```
1 data_result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 25 entries, 강남구 to 중랑구
Data columns (total 8 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   소계      25 non-null     int64  
 1   최근증가율   25 non-null     float64
 2   인구수     25 non-null     int64  
 3   한국인     25 non-null     int64  
 4   외국인     25 non-null     int64  
 5   고령자     25 non-null     int64  
 6   외국인비율   25 non-null     float64
 7   고령자비율   25 non-null     float64
dtypes: float64(3), int64(5)
memory usage: 1.8+ KB
```

```
1 data_result["CCTV비율"] = data_result["소계"] / data_result["인구수"]
2 data_result["CCTV비율"] = data_result["CCTV비율"] * 100
```

```
1 data_result.head()
```

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율
강남구	3238	150.619195	561052	556164	4888	65060	0.871220	11.596073	0.577130
강동구	1010	166.490765	440359	436223	4136	56161	0.939234	12.753458	0.229358
강북구	831	125.203252	328002	324479	3523	56530	1.074079	17.234651	0.253352
강서구	911	134.793814	608255	601691	6564	76032	1.079153	12.500021	0.149773
관악구	2109	149.290780	520929	503297	17632	70046	3.384722	13.446362	0.404854

```
1 data_result.sort_values(by="CCTV비율", ascending=False).head()
```

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율
종로구	1619	248.922414	164257	154770	9487	26182	5.775705	15.939656	0.985651
용산구	2096	53.216374	244444	229161	15283	36882	6.252148	15.088118	0.857456
중구	1023	147.699758	134593	125709	8884	21384	6.600640	15.887899	0.760069
강남구	3238	150.619195	561052	556164	4888	65060	0.871220	11.596073	0.577130
금천구	1348	100.000000	253491	235154	18337	34170	7.233787	13.479769	0.531774

```
1 data_result.sort_values(by="CCTV비율", ascending=True).head()
```

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율
강서구	911	134.793814	608255	601691	6564	76032	1.079153	12.500021	0.149773
송파구	1081	104.347826	671173	664496	6677	76582	0.994825	11.410173	0.161061
중랑구	916	79.960707	412780	408226	4554	59262	1.103251	14.356800	0.221910
강동구	1010	166.490765	440359	436223	4136	56161	0.939234	12.753458	0.229358
광진구	878	53.228621	372298	357703	14595	43953	3.920247	11.805865	0.235833

---

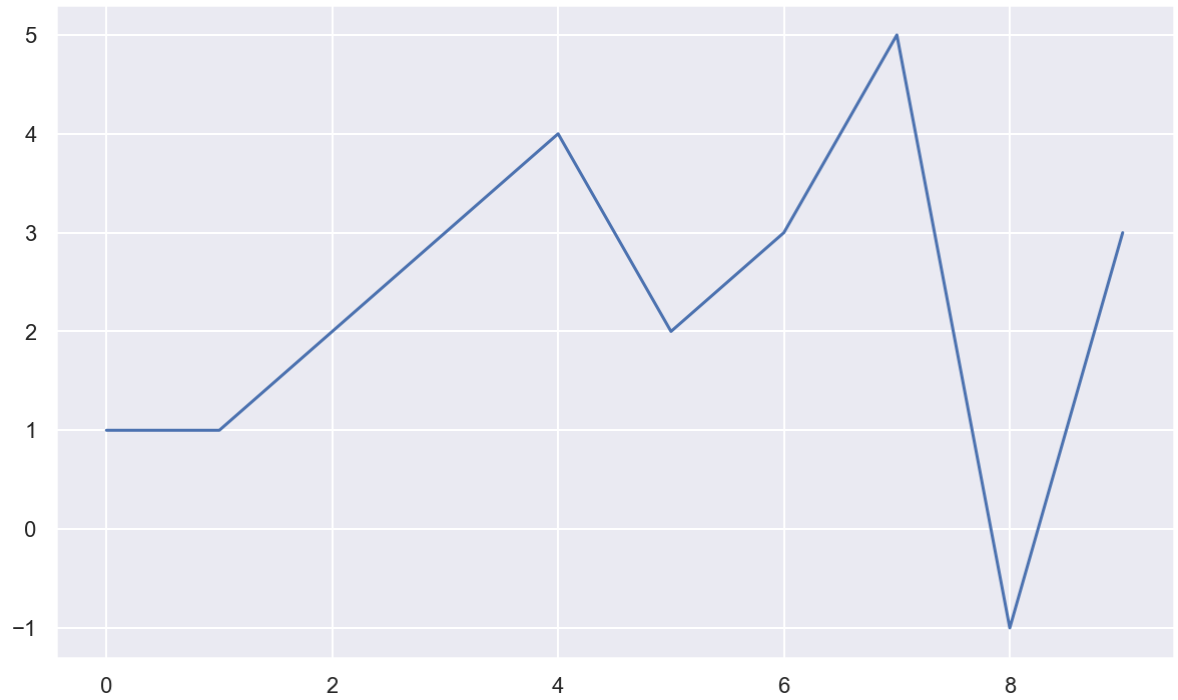
matplotlib 기초

```
1 import matplotlib.pyplot as plt
2 from matplotlib import rc
3
4 rc("font", family="Arial Unicode MS") # Windows : Malgu Gothic
5 # %matplotlib inline
6 get_ipython().run_line_magic("matplotlib", "inline")
```

matplotlib 그래프 기본 형태

```
plt.figure(figsize=(10, 6))
plt.plot(x, y)
plt.show
```

```
1 plt.figure(figsize=(10, 6))
2 plt.plot([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [1, 1, 2, 3, 4, 2, 3, 5, -1, 3])
3 plt.show()
```



예제1: 그래프 기초

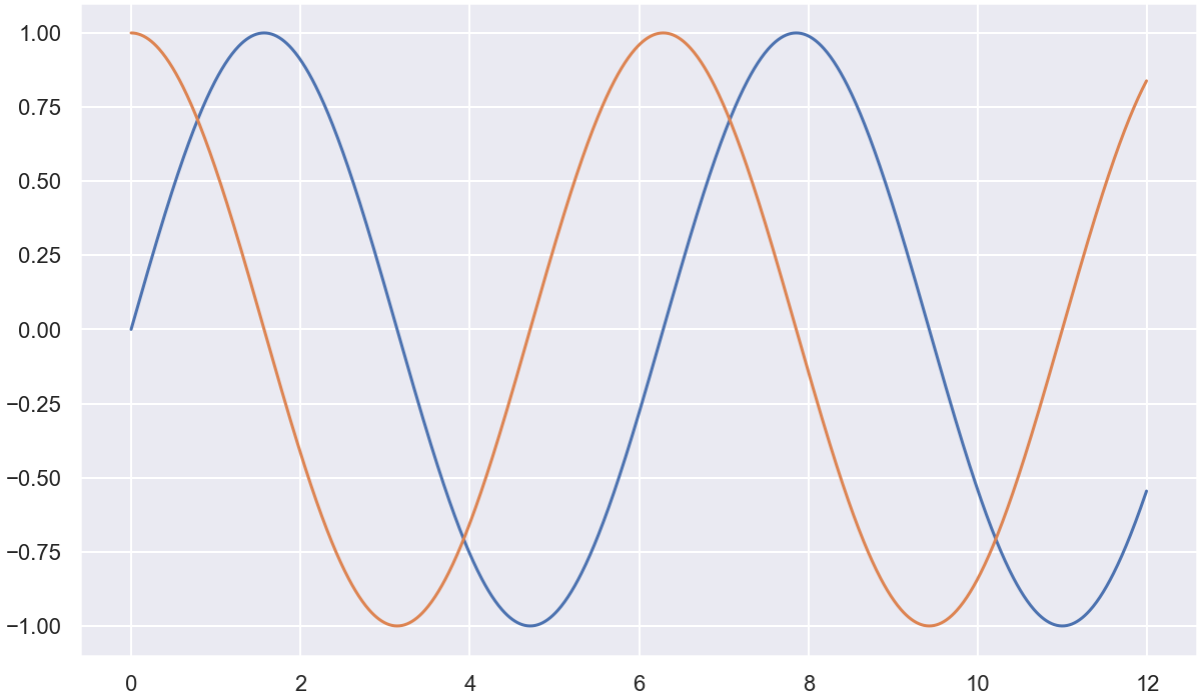
삼각함수 그리기

- np.arange(a, b, s): a부터 b까지 s의 간격
- np.sin(value)

```
1 import numpy as np
2
```

```
t = np.arange(0, 12, 0.01)
y = np.sin(t)
```

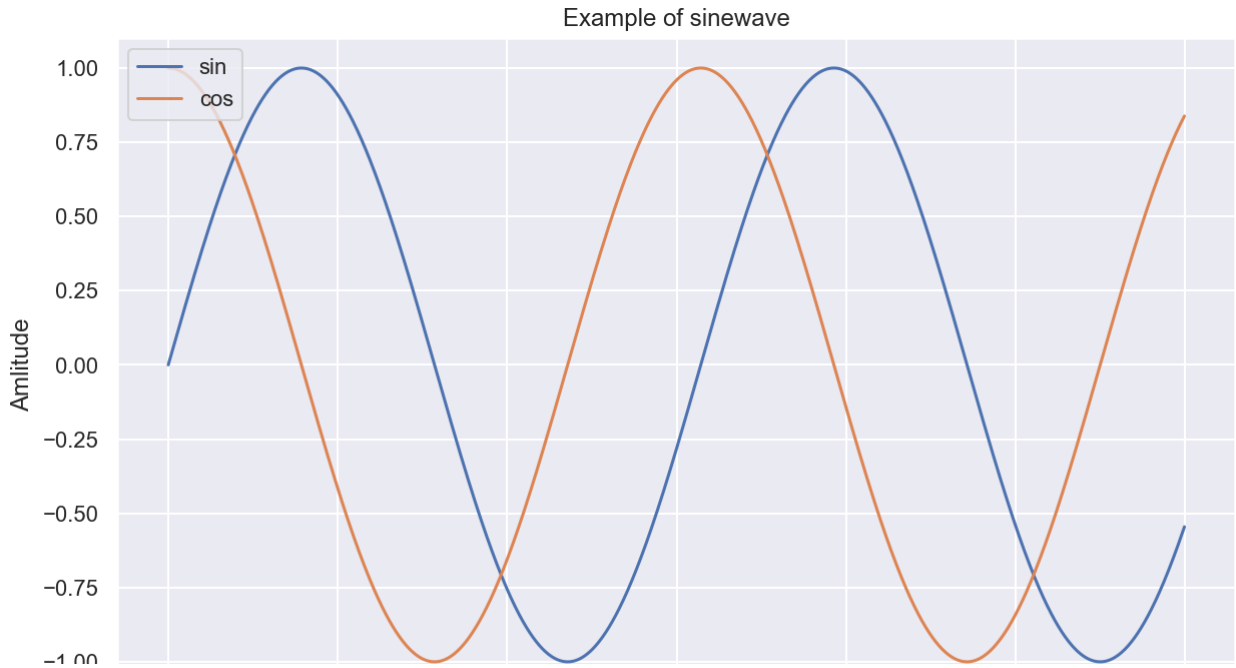
```
plt.figure(figsize=(10, 6))
plt.plot(t, np.sin(t))
plt.plot(t, np.cos(t))
plt.show()
```



- 1. 격자무늬 추가
- 2. 그래프 제목 추가
- 3. x축, y축 제목 추가
- 4. 주황색, 파란색 선 데이터 의미 구분

```
def drawGraph():

    plt.figure(figsize=(10, 6))
    plt.plot(t, np.sin(t), label="sin")
    plt.plot(t, np.cos(t), label="cos")
    plt.grid(True)
    plt.legend(loc=2) # 범례
    plt.title("Example of sinewave")
    plt.xlabel("time")
    plt.ylabel("Amplitude") # 진폭
    plt.show()
```

```
drawGraph()
```

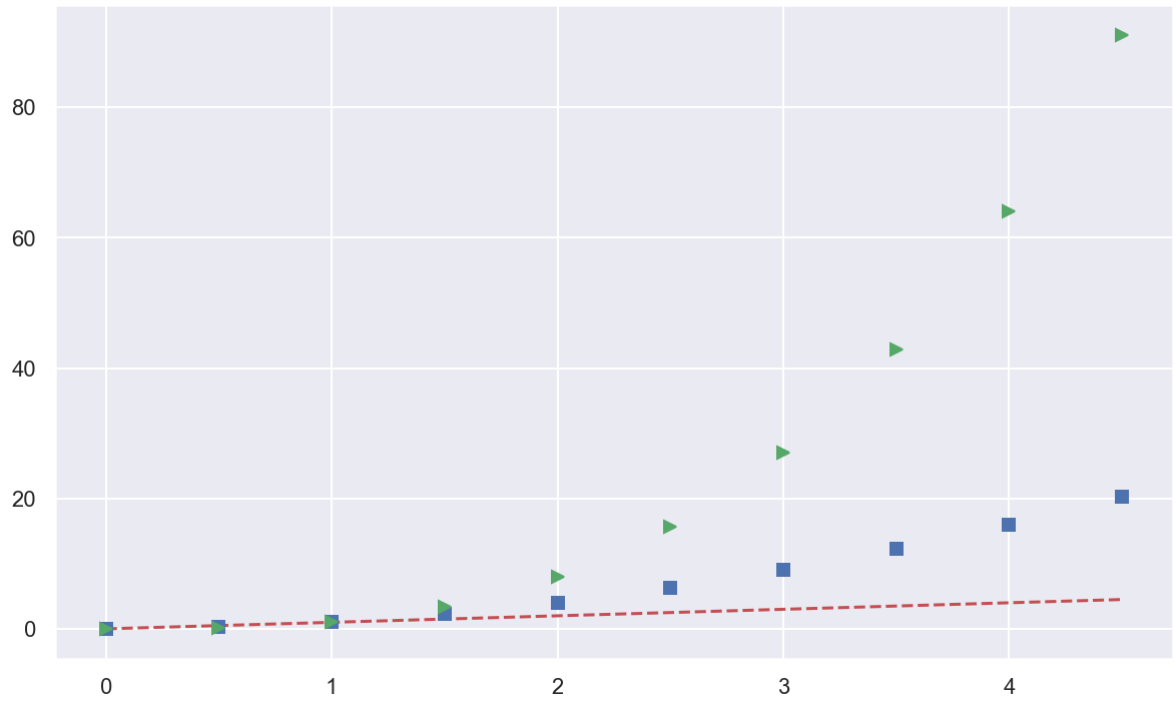


예제2: 그래프 커스텀

```
1 t = np.arange(0, 5, 0.5)
2 t
```

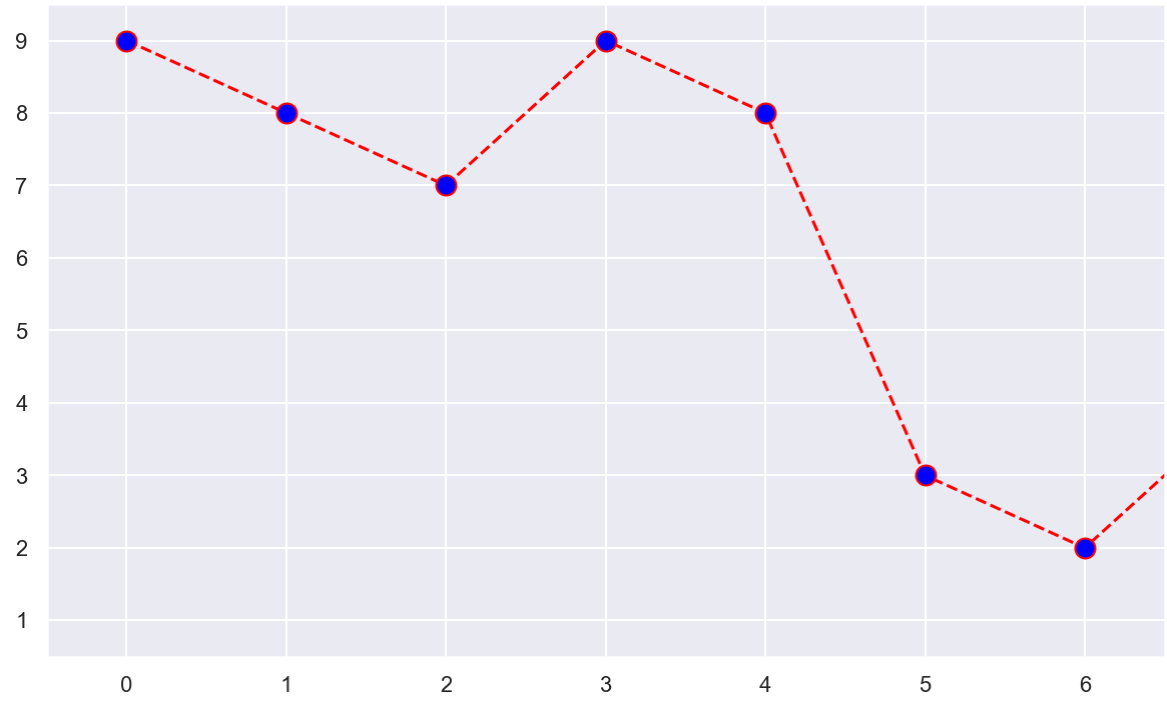
```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

```
1 plt.figure(figsize=(10, 6))
2 plt.plot(t, t, "r--") # red ----
3 plt.plot(t, t ** 2, "bs")
4 plt.plot(t, t ** 3, "g>")
5 plt.show()
```



```
# t = [0, 1, 2, 3, 4, 5, 6]
t = list(range(0, 7))
y = [1, 4, 5, 8, 9, 5, 3]
```

```
def drawGraph():

    plt.figure(figsize=(10, 6))
    plt.plot(
        t,
        y,
        color="red",
        linestyle="--",
        marker="o",
        markerfacecolor="blue",
        markersize=10,
    )

    plt.xlim([-0.5, 6.5])
    plt.ylim([0.5, 9.5])
    plt.show()

drawGraph()
```

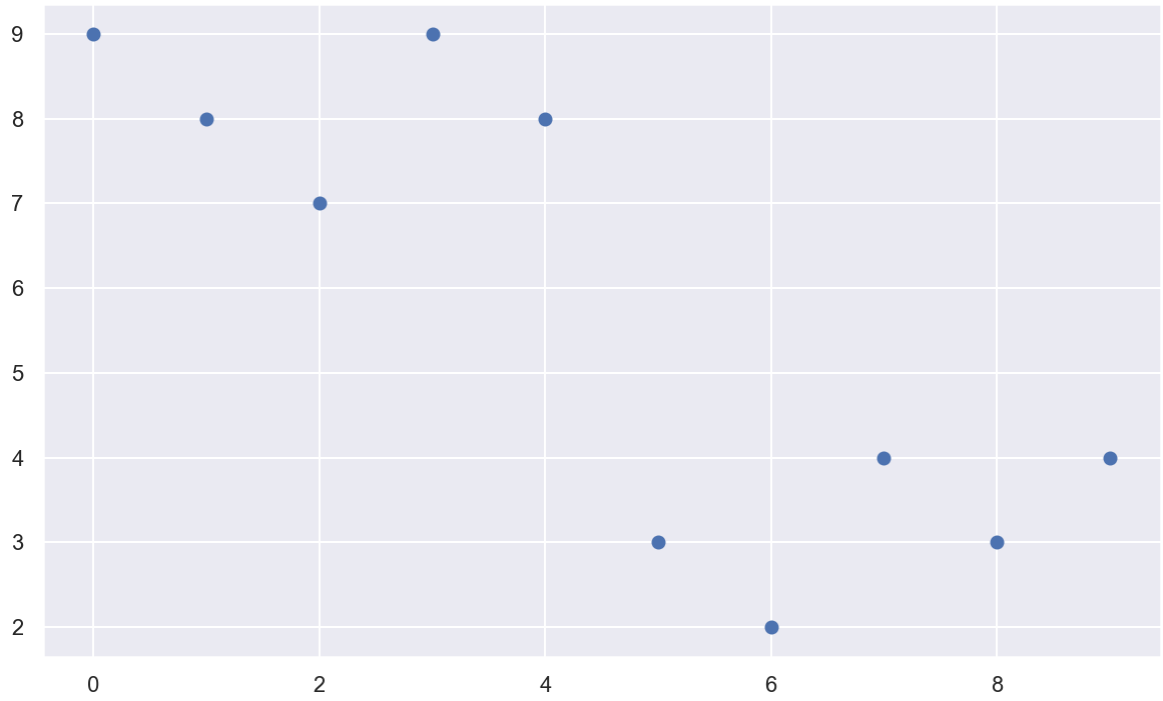


예제3: scatter plot

```
t = np.array(range(0, 10))
y = np.array([9, 8, 7, 9, 8, 3, 2, 4, 3, 4])
```

```
def drawGraph():

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

```
    plt.scatter(t, y)
    plt.show()

drawGraph()
```




```
colormap = t

def drawGraph():

    plt.figure(figsize=(10, 6))
    plt.scatter(t, y, s=150, c=colormap, marker="<")
    plt.colorbar()
    plt.show()

drawGraph()
```

예제4: Pandas에서 plot 그리기

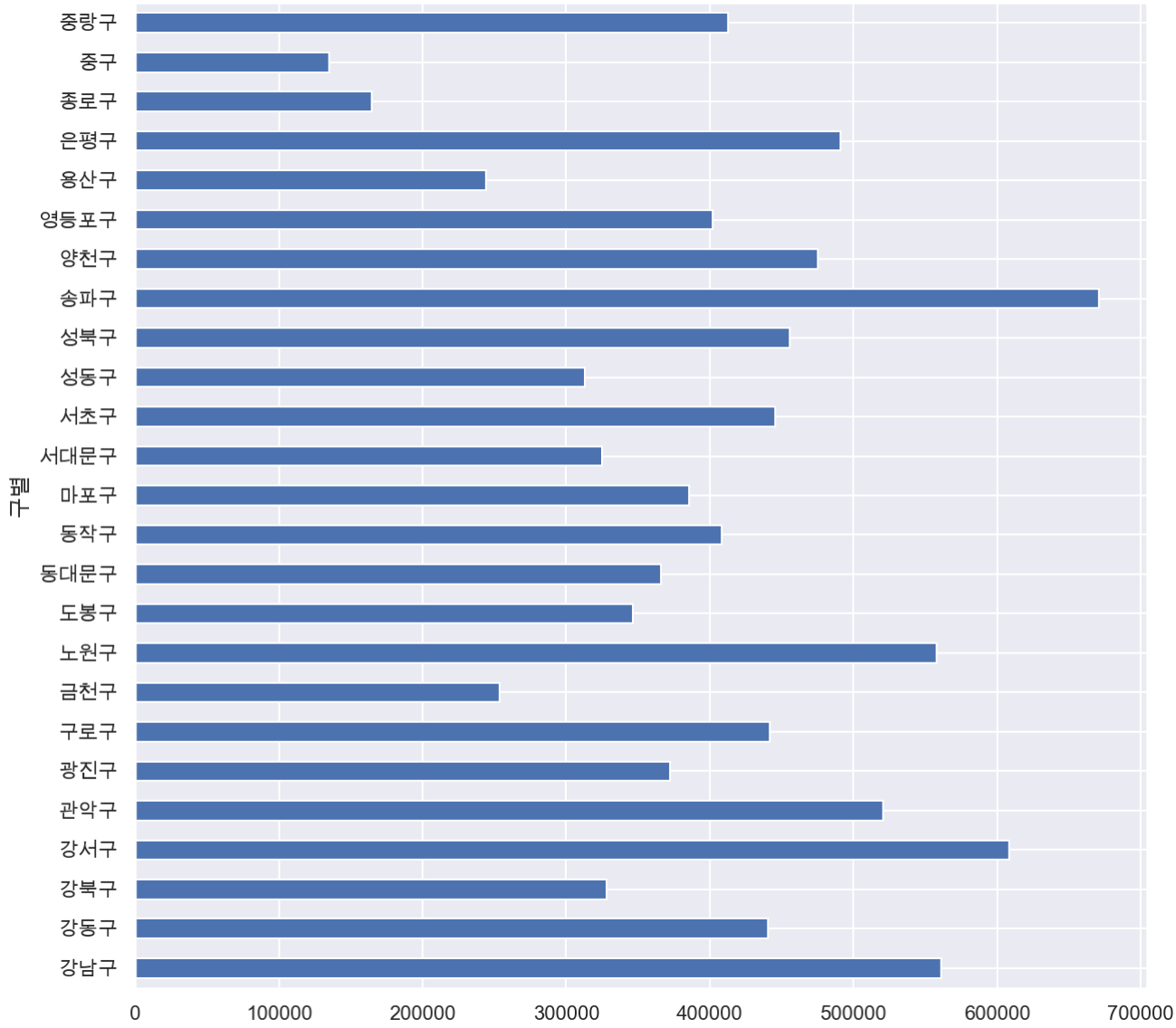
- matplotlib 을 가져와서 사용합니다

```
1 data_result.head()
```

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율
강남구	3238	150.619195	561052	556164	4888	65060	0.871220	11.596073	0.577130
강동구	1010	166.490765	440359	436223	4136	56161	0.939234	12.753458	0.229358
강북구	831	125.203252	328002	324479	3523	56530	1.074079	17.234651	0.253352
강서구	911	134.793814	608255	601691	6564	76032	1.079153	12.500021	0.149773
관악구	2109	149.290780	520929	503297	17632	70046	3.384722	13.446362	0.404854

```
1 data_result["인구수"].plot(kind="bar", figsize=(10, 10));
```


```
1 data_result["인구수"].plot(kind="barh", figsize=(10, 10));
```



---

5. 데이터 시각화

```
1 import matplotlib.pyplot as plt
2 # import matplotlib as mpl
3 from matplotlib import rc
4
5 plt.rcParams["axes.unicode_minus"] = False # 마이너스 부호 때문에 한글이 깨질 수가 있어 주
6 rc("font", family="Arial Unicode MS") # Windows: Malgun Gothic
7 # %matplotlib inline
8 get_ipython().run_line_magic("matplotlib", "inline")
```

```
1 data_result.head()
```

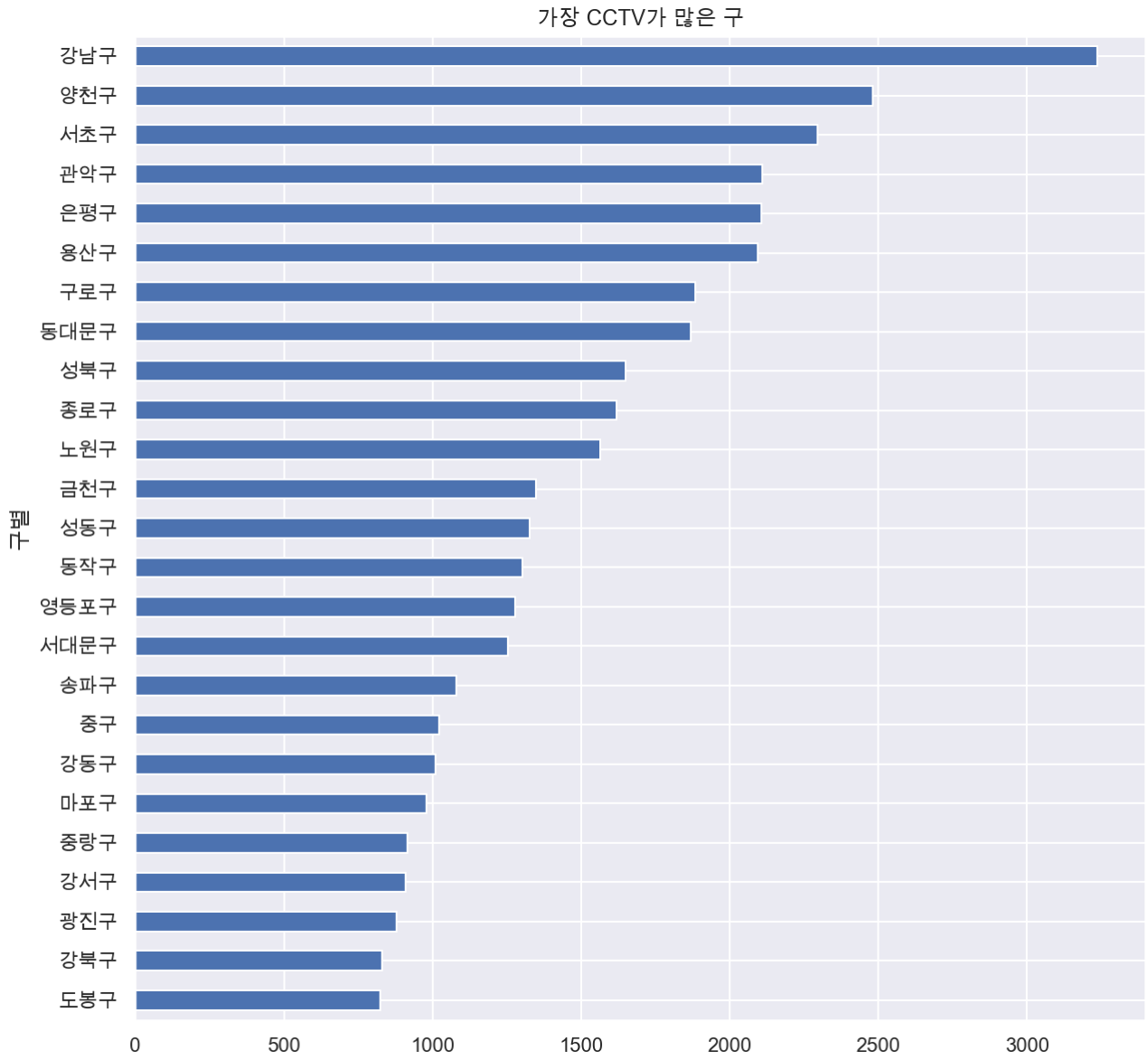
구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율
강남구	3238	150.619195	561052	556164	4888	65060	0.871220	11.596073	0.577130
강동구	1010	166.490765	440359	436223	4136	56161	0.939234	12.753458	0.229358
강북구	831	125.203252	328002	324479	3523	56530	1.074079	17.234651	0.253352
강서구	911	134.793814	608255	601691	6564	76032	1.079153	12.500021	0.149773
관악구	2109	149.290780	520929	503297	17632	70046	3.384722	13.446362	0.404854

소계 컬럼 시각화

```
1 data_result["소계"].plot(kind="barh", grid=True, figsize=(10, 10));
```


```
1 def drawGraph():
2     data_result["소계"].sort_values().plot(
```

```
3         kind="barh", grid=True, title="가장 CCTV가 많은 구", figsize=(10, 10));
4 drawGraph()
```

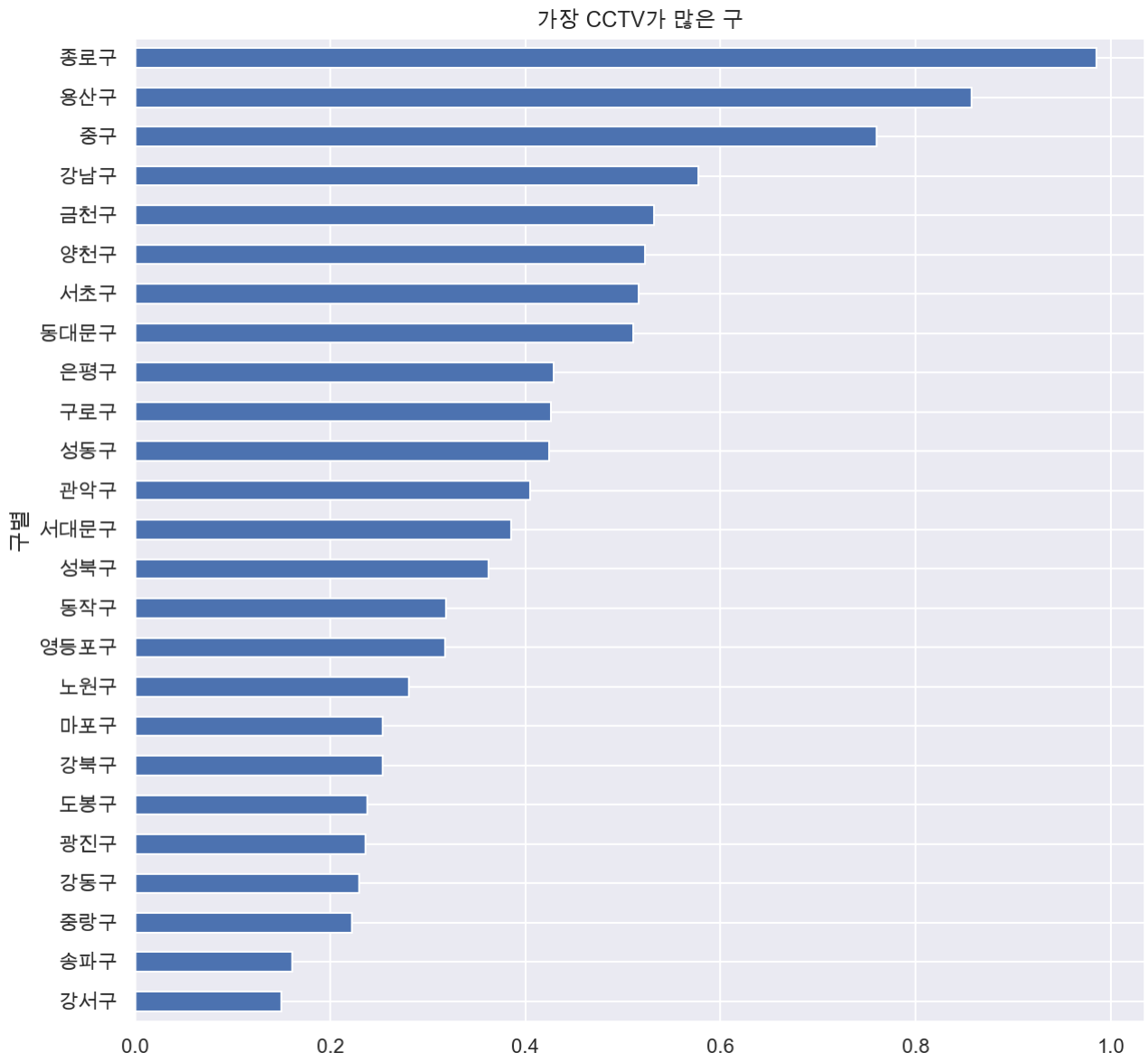



```
1 data_result.head()
```

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율
강남구	3238	150.619195	561052	556164	4888	65060	0.871220	11.596073	0.577130
강동구	1010	166.490765	440359	436223	4136	56161	0.939234	12.753458	0.229358
강북구	831	125.203252	328002	324479	3523	56530	1.074079	17.234651	0.253352
강서구	911	134.793814	608255	601691	6564	76032	1.079153	12.500021	0.149773
관악구	2109	149.290780	520929	503297	17632	70046	3.384722	13.446362	0.404854

CCTV비율 컬럼 시각화

```
def drawGraph():
    data_result["CCTV비율"].sort_values().plot(
        kind="barh", grid=True, title="가장 CCTV가 많은 구", figsize=(10, 10));
drawGraph()
```



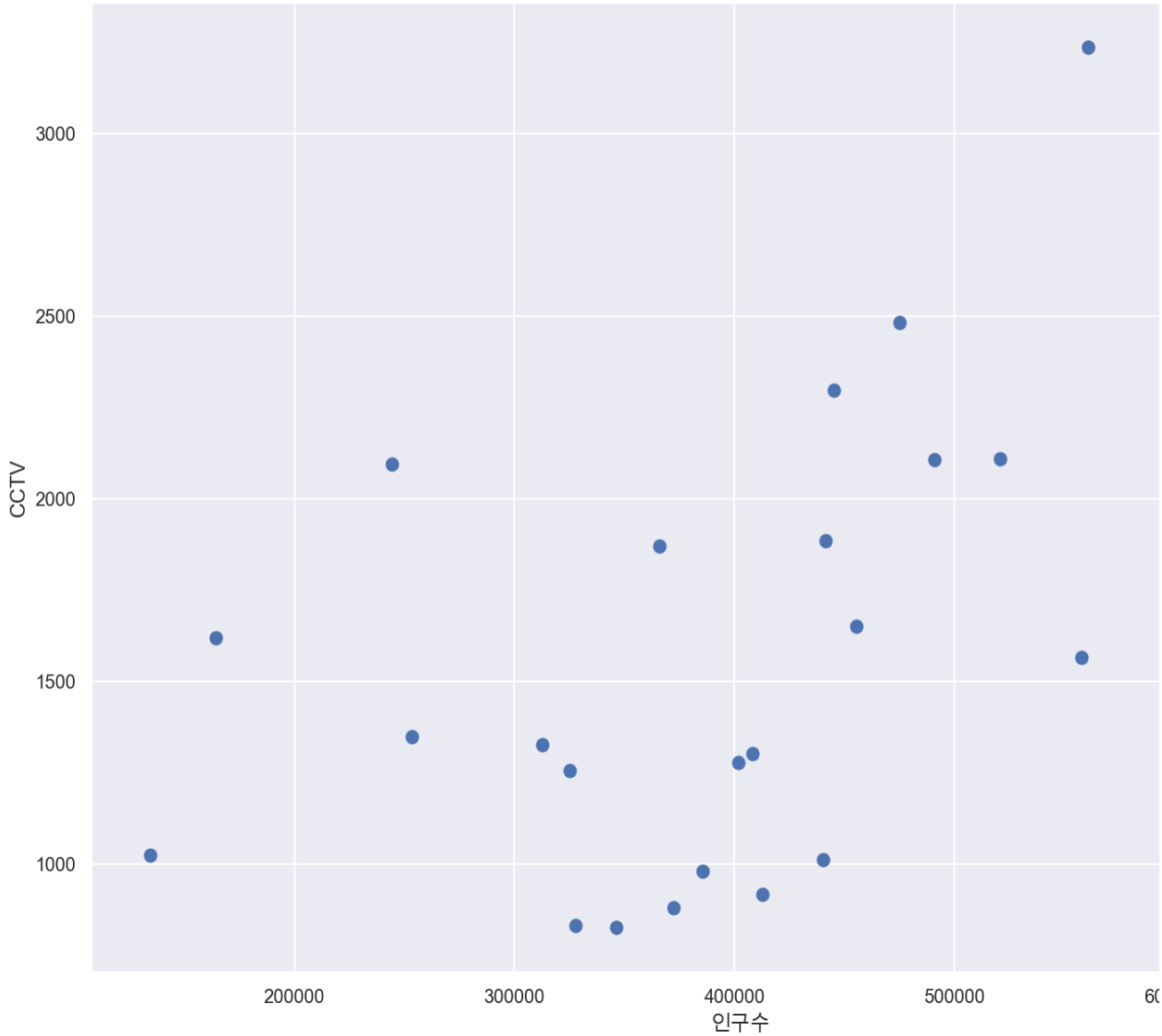
6. 데이터의 경향 표시

```
data_result.head()
```

	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율
강남구	3238	150.619195	561052	556164	4888	65060	0.871220	11.596073	0.577130

인구수와 소계 컬럼으로 scatter plot 그리기

```
def drawGraph():

    plt.figure(figsize=(14, 10))
    plt.scatter(data_result["인구수"], data_result["소계"], s=50)
    plt.xlabel("인구수")
    plt.ylabel("CCTV")
    plt.grid(True)
    plt.show()
drawGraph()
```



Numpy를 이용한 1차 직선 만들기

- np.polyfit(): 직선을 구성하기 위한 계수를 계산
- np.poly1d(): polyfit 으로 찾은 계수로 파이썬에서 사용할 수 있는 함수로 만들어주는 기능

```
import numpy as np
```

```
fp1 = np.polyfit(data_result["인구수"], data_result["소계"], 1)
fp1
```

```
array([1.11155868e-03, 1.06515745e+03])
```

```
f1 = np.poly1d(fp1)
f1
```

```
poly1d([1.11155868e-03, 1.06515745e+03])
```

```
f1(400000)
```

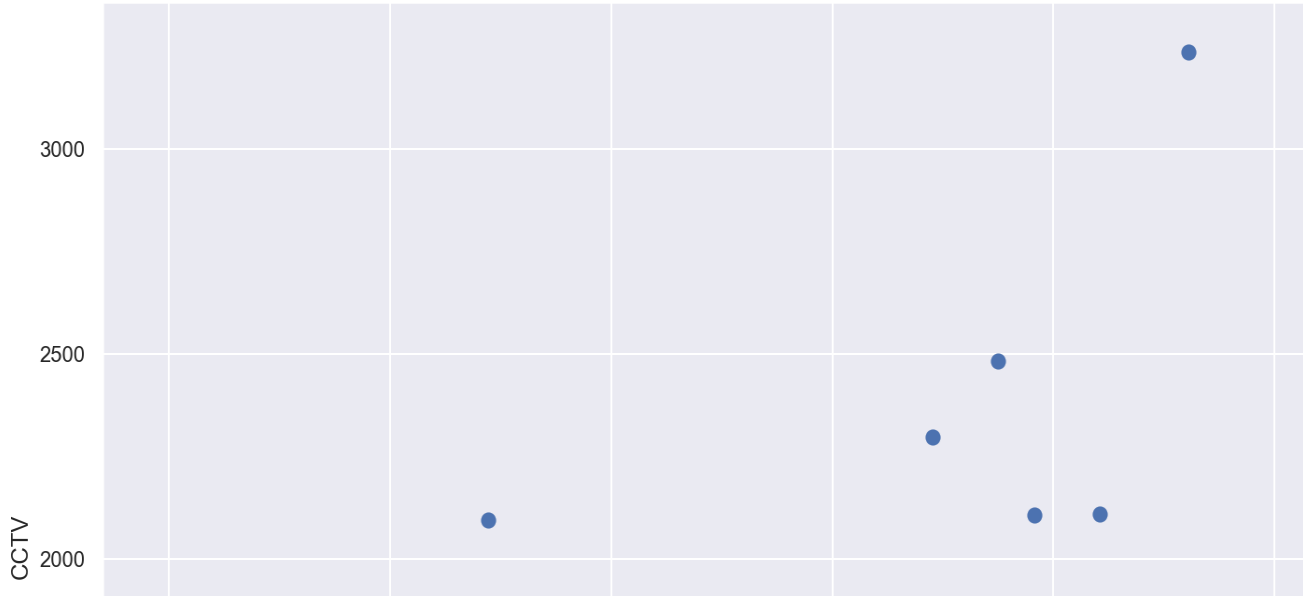
```
1509.7809252413335
```

- 인구가 40만인 구에서 서울시의 전체 경향에 맞는 적당한 CCTV 수는?

```
fx = np.linspace(100000, 700000, 100)
```

- 경향선을 그리기 위한 X 데이터 생성
- np.linspace(a, b, n): a부터 b까지 n개의 등간격 데이터 생성

```
def drawGraph():

    plt.figure(figsize=(14, 10))
    plt.scatter(data_result["인구수"], data_result["소계"], s=50)
    plt.plot(fx, f1(fx), ls="dashed", lw=3, color="g")
    plt.xlabel("인구수")
    plt.ylabel("CCTV")
    plt.grid(True)
    plt.show()
drawGraph()
```



7. 강조하고 싶은 데이터를 시각화해보자

그래프 다듬기

경향과의 오차 만들기

- 경향(trend)과의 오차를 만들자
- 경향은 f1 함수에 해당 인구를 입력
- f1(data_result["인구수"])


```
1 fp1 = np.polyfit(data_result["인구수"], data_result["소계"], 1)
2 f1 = np.poly1d(fp1)
3 fx = np.linspace(100000, 700000, 100)
```

```
1 data_result.head(3)
```

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율
강남구	3238	150.619195	561052	556164	4888	65060	0.871220	11.596073	0.577130
강동구	1010	166.490765	440359	436223	4136	56161	0.939234	12.753458	0.229358
강북구	831	125.203252	328002	324479	3523	56530	1.074079	17.234651	0.253352

```
1 data_result["오차"] = data_result["소계"] - f1(data_result["인구수"])
```

```
1 data_result.head(1)
```

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율	

```
# 경향과 비교해서 데이터의 오차가 너무 나는 데이터를 계산

df_sort_f = data_result.sort_values(by="오차", ascending=False) # 내림차순
df_sort_t = data_result.sort_values(by="오차", ascending=True) # 오름차순
```

```
# 경향 대비 CCTV를 많이 가진 구
df_sort_f.head()
```

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율	
강남구	3238	150.619195	561052	556164	4888	65060	0.871220	11.596073	0.577130	1
양천구	2482	34.671731	475018	471154	3864	55234	0.813443	11.627770	0.522507	
용산구	2096	53.216374	244444	229161	15283	36882	6.252148	15.088118	0.857456	
서초구	2297	63.371266	445401	441102	4299	53205	0.965198	11.945415	0.515715	
은평구	2108	85.237258	491202	486794	4408	74559	0.897390	15.178888	0.429151	

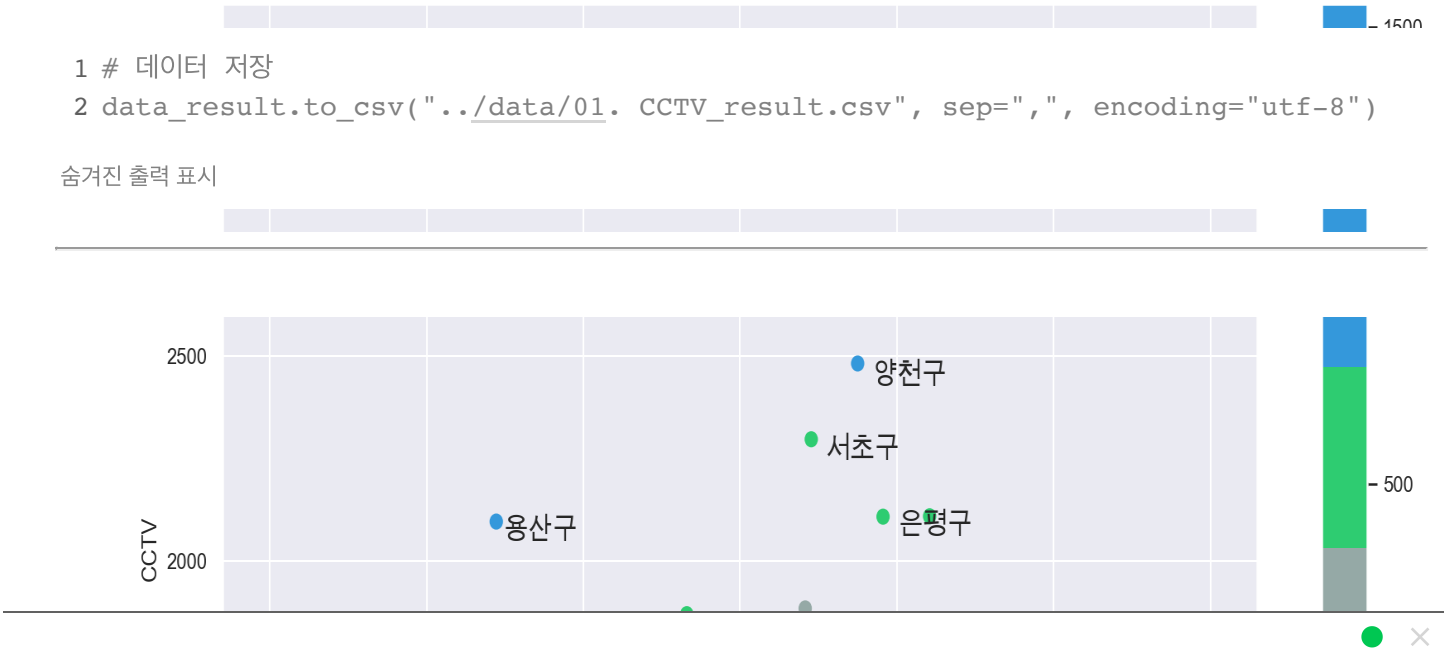
```
# 경향 대비 CCTV를 적게 가진 구
df_sort_t.head()
```

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율	CCTV비율	
강서구	911	134.793814	608255	601691	6564	76032	1.079153	12.500021	0.149773	-8
송파구	1081	104.347826	671173	664496	6677	76582	0.994825	11.410173	0.161061	-7
도봉구	825	246.638655	346234	344166	2068	53488	0.597284	15.448512	0.238278	-6
중랑구	916	79.960707	412780	408226	4554	59262	1.103251	14.356800	0.221910	-6
광진구	878	53.228621	372298	357703	14595	43953	3.920247	11.805865	0.235833	-6

```
from matplotlib.colors import ListedColormap

# colormap 을 사용자 정의(user define)로 세팅
color_step = ["#e74c3c", "#2ecc71", "#95a9a6", "#2ecc71", "#3498db", "#3498db"]
my_cmap = ListedColormap(color_step)
```

```
def drawGraph():

    plt.figure(figsize=(14, 10))
    plt.scatter(data_result["인구수"], data_result["소계"], s=50, c=data_result["오
    plt.plot(fx, f1(fx), ls="dashed", lw=3, color="g")

```

```
    for n in range(5):
        # 상위 5개
        plt.text(
            df_sort_f["인구수"][n] * 1.02, # x 좌표
            df_sort_f["소계"][n] * 0.98,  # y 좌표
            df_sort_f.index[n], # title
            fontsize=15,
        )

        # 하위 5개
        plt.text(
            df_sort_t["인구수"][n] * 1.02,
            df_sort_t["소계"][n] * 0.98,
            df_sort_t.index[n],
            fontsize=15
        )


    plt.xlabel("인구수")
    plt.ylabel("CCTV")
    plt.colorbar()
    plt.grid(True)
    plt.show()
drawGraph()
```




```
1 # 데이터 저장
2 data_result.to_csv("../data/01. CCTV_result.csv", sep=",", encoding="utf-8")
```

숨겨진 출력 표시