



- 제작 : 조정선



데이터 처리 및 분석을 위한 파이썬 모듈

- Numpy : 다차원 배열 생성 및 조작
- Pandas : Dataframe 생성 및 조작
(계산을 매우 편하게 함 = 시간 단축)
- Scipy : 과학 및 공학 연산 → 통계 분석
- Matplotlib : 시각화

Pandas

- 데이터 분석을 하는 데에 반드시 필요한 모듈 (계산을 편하게 한다)
계산을 편하게 한다는 것은 곧 시간을 줄인다는 것
- Pandas는 크게 Series와 DataFrame 두 가지로 나뉜다
물론 둘 다 알아야 한다
- `import pandas` 를 통하여 pandas 모듈을 import
`import pandas as pd`를 통하여 주로 pd로 줄여서 사용함
- 결측치 (NaN, na) : 해당 값이 존재하지 않음

Series 생성

- pd.Series() 함수를 통해 생성
- 데이터 타입은 Series
- 출력 : **index value**
index를 따로 지정하지 않는다면,
자동으로 0부터 지정되어 들어간다

```
S = pd.Series([11,28,82,3,6,8])  
print(S, '\n')  
print(type(S), '\n')  
print(len(S))
```

0	11
1	28
2	82
3	3
4	6
5	8

dtype: int64

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

6

Series 생성

- `pd.Series([value 1, value2, ...],[index1, index2, ...])`
- `pd.Series({index1:value 1, index2:value2, ...:...})`
- `pd.Series([value 1, value2, ...], index = [index1, index2 ...])` 등등...
→ 위 함수는 동일한 함수임으로 본인 취향에 따라 사용하면 됨
- value에 들어갈 수 있는 dtype은?
list, tuple, ndarray(단, 오직 1차원만)
- 당연히 indexing, slicing, step
모두 활용 가능함
→ 이걸 직접 해보기

```
S=pd.Series([11,28,82,3,6,8],  
            ['1st','2nd','3rd','4th','5th','6th'])  
print(S[0::2])
```

```
1st      11  
3rd      82  
5th       6  
dtype: int64
```

Tesla 종가 Series

```
# Tesla 종가 Series
Close = [300, 304, 292, 302, 303]
idx = pd.date_range('20220909', periods=5)
Tesla = pd.Series(Close, index=idx)
Tesla
```

```
2022-09-09    300
2022-09-10    304
2022-09-11    292
2022-09-12    302
2022-09-13    303
Freq: D, dtype: int64
```

- date_range를 통하여 index를 날짜로 나타낼 수 있다



오름차순 내림차순

- `a.sort_values()` : values를 오름차순으로 정렬
 `a.sort_values(ascending=False)` : values를 내림차순으로 정렬
- `a.sort_index()` : index를 오름차순으로 정렬
 `a.sort_index(ascending=False)` : index를 내림차순으로 정렬

```
stock = pd.Series(  
    {'LG': 73800,  
     'Samsung': 71200,  
     'Kakao': 106500})
```

```
stock  
  
LG          73800  
Samsung     71200  
Kakao       106500  
dtype: int64
```

```
stock.sort_values()  
#stock.sort_index()  
#stock.sort_index(ascending=True)  
#stock.sort_values(ascending=False)
```

```
Samsung     71200  
LG          73800  
Kakao       106500  
dtype: int64
```

```
#stock.sort_values()  
stock.sort_index()  
#stock.sort_index(ascending=True)  
#stock.sort_values(ascending=False)
```

```
Kakao       106500  
LG          73800  
Samsung     71200  
dtype: int64
```

Series 조건 검색

- `stock.where(stock<100000)`
100000 이상의 value는 결측치 처리
- `stock.where(stock==stock.min())`
최솟값을 제외한 value를 결측치 처리
max → 최댓값

```
print(stock.where(stock<100000))
```

```
LG          73800.0  
Samsung     71200.0  
Kakao       NaN  
dtype: float64
```

```
print(stock.where(stock==stock.min()))
```

```
LG          NaN  
Samsung     71200.0  
Kakao       NaN  
dtype: float64
```


Series 연산

- add 함수를 통하여 두 개의 Series를 합하여 볼 수 있다.
- fill_value의 함수는 교집합이 아닌 index에 대하여 value값을 0으로 취급해줌으로써 좀 더 쉬운 연산을 가능하게 해준다.

```
print(stock, '\n'); print(changes)
```

```
LG          73800
Samsung     71200
Kakao       106500
dtype: int64
```

```
Kakao       1500
LG          1200
SK          -1500
dtype: int64
```

```
print(stock.add(changes, fill_value=0).astype('int'))
```

```
Kakao       108000
LG          75000
SK          -1500
Samsung     71200
dtype: int64
```

index, value의 name

- ✓ **name**을 통해 value의 제목을 붙일 수 있음
단, value의 경우 제목을 붙인 것은 DataFrame에만 나타남
- ✓ **index.name**을 통해 index의 제목을 붙일 수 있음

```
#name
stock.name = 'price'    # dataframe 내 column 제목
stock.index.name = 'market'
stock
```

```
market      여기엔 없다!
LG           73800
Samsung     71200
Kakao       106500
Name: price, dtype: int64
```

```
pd.DataFrame(stock)
```

price	
market	
LG	73800
Samsung	71200
Kakao	106500

여기엔 있다!

index 추가/제거

• index 추가

✓ `stock['SK']=122000`
`stock`

```
market
LG          73800
Samsung     71200
Kakao       106500
SK          122000
Name: price, dtype: int64
```

✓ `stock.append(pd.Series({'Naver':345000}))`
`#stock`

```
LG          73800
Samsung     71200
Kakao       106500
SK          122000
Naver       345000
dtype: int64
```

(append는 영구적으로 추가하지 않음!
#을 제거해보고 stock를 실행해보기

• index 제거

✓ `del stock['SK']`
`stock`

```
market
LG          73800
Samsung     71200
Kakao       106500
Name: price, dtype: int64
```

✓ `stock.drop(['LG'])`
`#stock`

```
market
Samsung     71200
Kakao       106500
Name: price, dtype: int64
```

(drop은 영구적으로 제거하지 않음!
#을 제거해보고 stock를 실행해보기

apply

- apply 함수를 활용하여 모든 index의 value에 함수를 적용시킬 수 있다.
- applymap 함수를 활용하면 DataFrame 전체에 함수를 적용시킬 수 있다.

```
# apply: 전체 value에 적용
stock['SK'] = 122000 'SK' 라는 index를 추가
stock
✓ stock.apply(float) 모든 value들을 float화
stock.apply(lambda x: x/stock.sum())
```

```
market
LG          0.197590
Samsung     0.190629
Kakao       0.285141
SK          0.326640
Name: price, dtype: float64
```

DataFrame 생성

- **column** 단위로 데이터를 관리
- ✓ `pd.DataFrame()`을 통해 생성
- `dtype`은 DataFrame
- `print`로 출력하면 표가 사라짐
이유는 저도 몰라요...

```
a = pd.DataFrame([[10,20,30], [40,50,60],[70,80,90]])  
print(a)  
print(type(a))  
print(len(a))  
a
```

```
      0   1   2  
0  10  20  30  
1  40  50  60  
2  70  80  90  
<class 'pandas.core.frame.DataFrame'>  
3
```

	0	1	2
0	10	20	30
1	40	50	60
2	70	80	90

DataFrame의 index, value의 name

- DataFrame(data, [columns = ???, index = ???])
- data에 들어갈 수 있는 dtype은?
→ list/tuple, ndarray, dict, Series

```
# DataFrame columns, index
data = [[10,20,30],[40,50,60],[70,80,90]]
a = pd.DataFrame(data, columns = ['1차', '2차', '3차'], index = ['One', 'Two', 'Three'])
a
```

	1차	2차	3차
One	10	20	30
Two	40	50	60
Three	70	80	90

```
print(a.index)      # row 명
print(a.columns)    # columns 명
print(a.values)     # values 값
```

```
Index(['One', 'Two', 'Three'], dtype='object')
Index(['1차', '2차', '3차'], dtype='object')
[[10 20 30]
 [40 50 60]
 [70 80 90]]
```

yfinance

- ✓ import yfinance as yf
- 금융데이터의 경우 야후의 금융데이터를 불러올 수 있다
- 테슬라.. 애플.. 삼성.. 등등 각자 고유에 맞는 약자를 입력하면 된다
- ✓ yf.download('회사이름')

```
# yfinance
# yahoo finance에서 주식 데이터를 받아올 수 있는 모듈
tsla = yf.download('TSLA')
aapl = yf.download('AAPL')
```

```
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
```

yfinance

- Tesla 데이터 불러오기

```
tsla.tail()
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2023-03-08	185.039993	186.500000	180.000000	182.000000	182.000000	151897800
2023-03-09	180.250000	185.179993	172.509995	172.919998	172.919998	170023800
2023-03-10	175.130005	178.289993	168.440002	173.440002	173.440002	191007900
2023-03-13	167.460007	177.350006	163.910004	174.479996	174.479996	167790300
2023-03-14	177.309998	183.800003	177.139999	183.259995	183.259995	143430400

yfinance

• 애플 데이터 불러오기

```
aapl.tail()
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2023-03-08	152.809998	153.470001	151.830002	152.869995	152.869995	47204800
2023-03-09	153.559998	154.539993	150.229996	150.589996	150.589996	53833600
2023-03-10	150.210007	150.940002	147.610001	148.500000	148.500000	68524400
2023-03-13	147.809998	153.139999	147.699997	150.470001	150.470001	84457100
2023-03-14	151.279999	153.399994	150.100006	152.589996	152.589996	73628400

DataFrame의 indexing

```
tsla=tsla.tail()  
tsla['Close'] # column 이름으로만 indexing 가능  
#tsla['2023-03-08'] # error --> index 이름으로는 indexing 불가능 loc 사용필요
```

```
Date  
2023-03-08    182.000000  
2023-03-09    172.919998  
2023-03-10    173.440002  
2023-03-13    174.479996  
2023-03-14    183.259995  
Name: Close, dtype: float64
```

tsla.Close

```
Date  
2023-03-08    182.000000  
2023-03-09    172.919998  
2023-03-10    173.440002  
2023-03-13    174.479996  
2023-03-14    183.259995  
Name: Close, dtype: float64
```

✓ DataFrame['column명']을 통해서 indexing 가능

• DataFrame.'column명'을 통해서 indexing 가능 → type: Series

```
print(type(tsla['Close']))
```

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

DataFrame의 indexing

```
tsla[['Open', 'Close']]
```

	Open	Close
Date		
2023-03-08	185.039993	182.000000
2023-03-09	180.250000	172.919998
2023-03-10	175.130005	173.440002
2023-03-13	167.460007	174.479996
2023-03-14	177.309998	183.259995

✓ DataFrame[['column1', 'column2']]
을 통해서 여러 개를 indexing 가능

- DataFrame 형태로 출력
dtype : DataFrame

DataFrame의 indexing

```
tsla.loc['2023-03-14']
```

Open	1.773100e+02
High	1.838000e+02
Low	1.771400e+02
Close	1.832600e+02
Adj Close	1.832600e+02
Volume	1.434304e+08

Name: 2023-03-14 00:00:00, dtype: float64

```
tsla.iloc[1] # 2023-03-09
```

Open	1.802500e+02
High	1.851800e+02
Low	1.725100e+02
Close	1.729200e+02
Adj Close	1.729200e+02
Volume	1.700238e+08

Name: 2023-03-09 00:00:00, dtype: float64

- ✓ DataFrame.loc['index명']을 통해서 indexing 가능
- ✓ DataFrame.iloc[숫자]를 통해서 indexing 가능

DataFrame의 slicing

```
# slicing
tsla.loc['2023-03-08':'2023-03-14':2, 'Open':'Close']
#tsla.iloc[0:3,0:3]    # 2023-03-08 ~ 2023-03-10
```

	Open	High	Low	Close
Date				
2023-03-08	185.039993	186.500000	180.000000	182.000000
2023-03-10	175.130005	178.289993	168.440002	173.440002
2023-03-14	177.309998	183.800003	177.139999	183.259995

- ✓ DataFrame.loc['index1':'index2', 'column1:column2']을 통해서 slicing 가능

DataFrame의 수정

- `reindex`를 통하여 `column`과 `index`의 순서를 바꿀 수 있다

✓ `tsla.reindex(columns=['Open', 'Close', 'Adj Close', 'High', 'Low', 'Volume'])`

	Open	Close	Adj Close	High	Low	Volume
Date						
2023-03-08	185.039993	182.000000	182.000000	186.500000	180.000000	151897800
2023-03-09	180.250000	172.919998	172.919998	185.179993	172.509995	170023800
2023-03-10	175.130005	173.440002	173.440002	178.289993	168.440002	191007900
2023-03-13	167.460007	174.479996	174.479996	177.350006	163.910004	167790300
2023-03-14	177.309998	183.259995	183.259995	183.800003	177.139999	143430400

DataFrame의 수정

- rename을 통하여 column의 이름을 수정할 수 있다.

```
✓tsla.rename(columns={'Open': 'open'})
```

	open	High	Low	Close	Adj Close	Volume
Date						
2023-03-08	185.039993	186.500000	180.000000	182.000000	182.000000	151897800
2023-03-09	180.250000	185.179993	172.509995	172.919998	172.919998	170023800
2023-03-10	175.130005	178.289993	168.440002	173.440002	173.440002	191007900
2023-03-13	167.460007	177.350006	163.910004	174.479996	174.479996	167790300
2023-03-14	177.309998	183.800003	177.139999	183.259995	183.259995	143430400

DataFrame의 수정

- rename을 통하여 index의 이름을 바꿀 수 있다...?

```
tsla.rename(index={'2023-03-14': 'Yesterday'})
```

IndexError

Traceback (most recent call last)

Input In [73], in <cell line: 1>()

```
----> 1 tsla.rename(index={'2023-03-14': 'Yesterday'})
```

- 이유는 Date에 있는 dtype이 datetime이기 때문

```
print(type(tsla.index))
```

```
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
```


DataFrame의 수정

- lambda를 이용하여 dtype을 바꾼 후 바꾸면 된다

```
Tsla=tsla.rename(index=lambda x: x.strftime('%Y-%m-%d'))
```

```
# Tsla.rename(index={'2023-03-14': 'Yesterday'}, inplace=True)
Tsla = Tsla.rename(index={'2023-03-14': 'Yesterday'}) # inplace = True를 사용한 것과 같다
Tsla
```

- 즉 index를 넣으면 column과 같이 바뀐다

	Open	High	Low	Close	Adj Close	Volume
Date						
2023-03-08	185.039993	186.500000	180.000000	182.000000	182.000000	151897800
2023-03-09	180.250000	185.179993	172.509995	172.919998	172.919998	170023800
2023-03-10	175.130005	178.289993	168.440002	173.440002	173.440002	191007900
2023-03-13	167.460007	177.350006	163.910004	174.479996	174.479996	167790300
Yesterday	177.309998	183.800003	177.139999	183.259995	183.259995	143430400

DataFrame의 수정

- DataFrame.loc['index명']을 통하여 index를 추가할 수 있음

```
# index 추가  
Tsla.loc['Tomorrow']=[304.829987, 310.119995, 310.720001, 310.750000, 310.750000, 54613900]  
Tsla
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2023-03-08	185.039993	186.500000	180.000000	182.000000	182.000000	151897800.0
2023-03-09	180.250000	185.179993	172.509995	172.919998	172.919998	170023800.0
2023-03-10	175.130005	178.289993	168.440002	173.440002	173.440002	191007900.0
2023-03-13	167.460007	177.350006	163.910004	174.479996	174.479996	167790300.0
Yesterday	177.309998	183.800003	177.139999	183.259995	183.259995	143430400.0
Tomorrow	304.829987	310.119995	310.720001	310.750000	310.750000	54613900.0

DataFrame의 수정

- ✓ DF.drop()을 통하여 삭제할 수 있음
- del 을 통하여 삭제할수 있기도 함

inplace = 제자리에, 가동할 준비가 되어 있는.

```
# index 및 column 삭제
# del Tsla['Adj Close']
Tsla.drop(['Yesterday'], axis=0, inplace = True)
Tsla.drop(['Adj Close'], axis=1, inplace = True)
Tsla
```

inplace = True
변경사항이 즉시 데이터프레임에 적용된다

	Open	High	Low	Close	Volume
Date					
2023-03-08	185.039993	186.500000	180.000000	182.000000	151897800.0
2023-03-09	180.250000	185.179993	172.509995	172.919998	170023800.0
2023-03-10	175.130005	178.289993	168.440002	173.440002	191007900.0
2023-03-13	167.460007	177.350006	163.910004	174.479996	167790300.0
Tomorrow	304.829987	310.119995	310.720001	310.750000	54613900.0

DataFrame의 수정

- ✓ `set_index()`를 통하여 column을 index로 설정할 수 있다
- ✓ `DF.sort_values(by='column')`을 통하여 value 값으로 정렬할 수 있다.
- ✓ `DF.sort_index`를 통하여 index로 정렬할 수 있다.

lecture

	Names	Score
0	Amy	13
1	Bob	45
2	Cindy	68
3	James	91

`lecture.set_index(['Names'])`

Score	
Names	
Amy	13
Bob	45
Cindy	68
James	91

```
# 정렬
lecture.sort_values(by='Score')
#lecture.sort_values(by='Score', ascending=False)
#lecture.sort_index() # index
#lecture.sort_index(axis=1, ascending=False) # column명 내림차순
```

	Names	Score
0	Amy	13
1	Bob	45
3	James	71
2	Cindy	98

axis=0 → column 정렬
axis=1 → index 정렬
ascending = True → 오름차순
ascending = False → 내림차순

DataFrame의 수정

- DF.index/columns.name을 통하여 index와 column의 이름을 붙여줄 수 있다

```
✓ lecture.index.name='INDEX'  
✓ lecture.columns.name='COLUMNS'  
lecture
```

COLUMNS	Names	Score
INDEX		
0	Amy	13
1	Bob	45
2	Cindy	98
3	James	71

- DF.T를 통하여 DataFrame을 transpose 할 수 있음

```
✓ lecture.T
```

INDEX	0	1	2	3
COLUMNS				
Names	Amy	Bob	Cindy	James
Score	13	45	98	71

DataFrame 조건 검색

- Series에서의 조건 검색과 같은 방법을 통해 조건 검색을 실시한다

Series 조건 검색

- `stock.where(stock<100000)`
100000 이상의 value는 결측치 처리
- `stock.where(stock==stock.min())`
최솟값을 제외한 value를 결측치 처리
max → 최댓값

```
print(stock.where(stock<100000))
```

```
LG          73800.0  
Samsung     71200.0  
Kakao       NaN  
dtype: float64
```

```
print(stock.where(stock==stock.min()))
```

```
LG          NaN  
Samsung     71200.0  
Kakao       NaN  
dtype: float64
```

- stock 1을 stock의 DataFrame이라고 한다면?

DataFrame 조건 검색

- ✓ `stock1.where(stock1<100000)`
100000 이상의 value는 결측치 처리
- ✓ `stock1.where(stock1==stock1.min())`
최솟값을 제외한 value를 결측치 처리
max → 최댓값

```
stock1.where(stock1<100000)
```

Final	
LG	73800.0
Samsung	71200.0
Kakao	NaN

```
stock1.where(stock1==stock1.min())
```

Final	
LG	NaN
Samsung	71200.0
Kakao	NaN

DataFrame 합치기

- ✓ concat 함수를 활용하면 두 개의 DataFrame을 합칠 수 있다!
→ axis=0이면 index, 1이면 column (default는 0)

states

	Population	Area	Year
California	1423967	423967	2022
Texas	1695662	695662	2022
New York	1141297	141297	2022
Florida	1170312	170312	2022
Illinois	1149995	149995	2022

states1

	Population	Area
Seoul	423967	23967
Daegu	195662	15662
Busan	141297	11297

```
states2=pd.concat([states, states1])  
states2
```

	Population	Area	Year
California	1423967	423967	2022.0
Texas	1695662	695662	2022.0
New York	1141297	141297	2022.0
Florida	1170312	170312	2022.0
Illinois	1149995	149995	2022.0
Seoul	423967	23967	NaN
Daegu	195662	15662	NaN
Busan	141297	11297	NaN

```
states3=pd.concat([states, states1], axis=1)  
states3
```

	Population	Area	Year	Population	Area
California	1423967.0	423967.0	2022.0	NaN	NaN
Texas	1695662.0	695662.0	2022.0	NaN	NaN
New York	1141297.0	141297.0	2022.0	NaN	NaN
Florida	1170312.0	170312.0	2022.0	NaN	NaN
Illinois	1149995.0	149995.0	2022.0	NaN	NaN
Seoul	NaN	NaN	NaN	423967.0	23967.0
Daegu	NaN	NaN	NaN	195662.0	15662.0
Busan	NaN	NaN	NaN	141297.0	11297.0

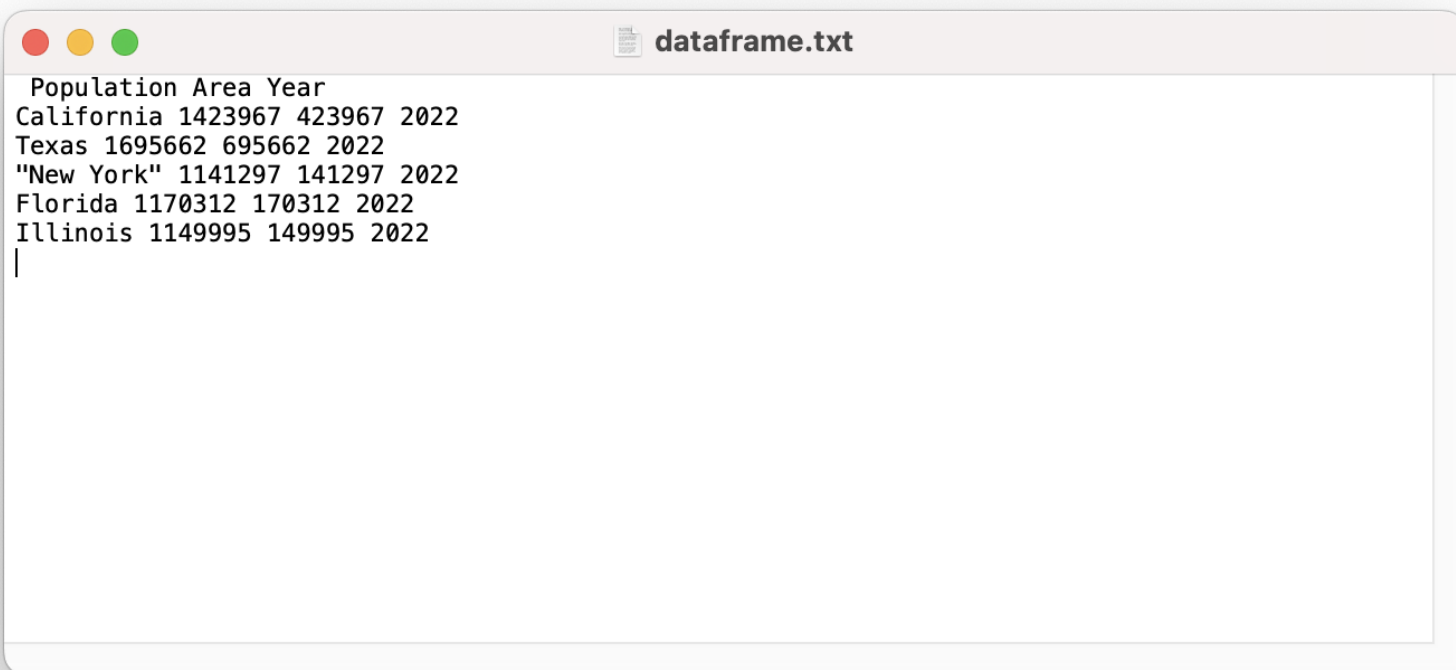
결측치에 관련된 함수

- ✓ a.isnull() : NaN이라면 True로 반환 (아니면 False)
- ✓ a.fillna(x) : 모든 NaN의 값을 x로 채움
- ✓ a.dropna : NaN인 index를 아예 제외

→ Series, DataFrame 모두 같은 적용

DataFrame 파일 저장

```
# dataframe 파일 저장
states.to_csv('dataframe.txt', sep = ' ')
# 파일 dataframe으로 불러오기
file = pd.read_csv('dataframe.txt', encoding = '', index_col = 0)
```



dataframe.txt

```
Population Area Year
California 1423967 423967 2022
Texas 1695662 695662 2022
"New York" 1141297 141297 2022
Florida 1170312 170312 2022
Illinois 1149995 149995 2022
|
```

- to_csv()를 통해 다음과 같이 파일 생성
- file을 통해 출력할 수 있음

file

Population Area Year		
California	1423967	423967 2022
Texas	1695662	695662 2022
New York	1141297	141297 2022
Florida	1170312	170312 2022
Illinois	1149995	149995 2022



- 제작 : 나

