

기초빅데이터프로그래밍

Pandas



Pandas

- Pandas의 **Series**는 1차원 데이터를 다루는 데 효과적인 자료구조이며, **DataFrame**은 행과 열로 구성된 2차원 데이터를 다루는 데 효과적인 자료구조이다.
- Pandas의 Series는 어떤 면에서는 파이썬의 리스트와 비슷하고 어떤 면에서는 파이썬의 딕셔너리와 닮은 자료구조이다.
 - 정수를 사용해서 데이터를 선택하는 리스트 기능
 - Label을 사용해서 데이터를 선택하는 사전 기능

데이터를 Series로 저장해보기

```
from pandas import Series, DataFrame
```

```
kakao = Series([92600, 92400, 92100, 94300, 92300])  
print(kakao)
```

```
0    92600  
1    92400  
2    92100  
3    94300  
4    92300  
dtype: int64
```

Series 객체를 생성할 때 따로 인덱스를 지정하지 않으면 0부터 시작하는 정숫값으로 인덱싱된다.

```
kakao2 = Series([92600, 92400, 92100, 94300, 92300], index=['2017-03-03',  
                                                         '2017-03-06', '2017-03-07', '2017-03-08', '2017-03-09'])  
print(kakao2)
```

```
2017-03-03    92600  
2017-03-06    92400  
2017-03-07    92100  
2017-03-08    94300  
2017-03-09    92300  
dtype: int64
```

```
print(kakao2['2017-03-09'])
```

```
92300
```

```
my_ind = pd.date_range("20191203", periods=6)
my_ind
```

```
DatetimeIndex(['2019-12-03', '2019-12-04', '2019-12-05', '2019-12-06',
               '2019-12-07', '2019-12-08'],
              dtype='datetime64[ns]', freq='D')
```

```
kakao2 = pd.Series([92600, 93400, 94500, 93800, 94000, 93500], index=my_ind)
kakao2
```

```
2019-12-03    92600
2019-12-04    93400
2019-12-05    94500
2019-12-06    93800
2019-12-07    94000
2019-12-08    93500
Freq: D, dtype: int64
```

```
kakao = pd.Series([92600, 93400, 94500, 93800, 94000, 93500], index=pd.date_range("20170303", periods=6))
kakao
```

```
2017-03-03    92600
2017-03-04    93400
2017-03-05    94500
2017-03-06    93800
2017-03-07    94000
2017-03-08    93500
Freq: D, dtype: int64
```

- Pandas의 Series 객체에는 **인덱스**와 **값**이 저장되어 있는데 Series 객체의 **index**와 **values**라는 이름의 속성을 통해 접근할 수 있다.
- 예를 들어, kakao2 객체의 인덱스 값과 저장된 종가를 각각 출력하는 코드는 다음과 같이 구현할 수 있다.

```
for date in kakao2.index:  
    print(date)  
  
for ending_price in kakao2.values:  
    print(ending_price)
```

```
2017-03-03  
2017-03-06  
2017-03-07  
2017-03-08  
2017-03-09  
92600  
92400  
92100  
94300  
92300
```

Series 객체의 덧셈 연산

- 인덱싱이 서로 다른 경우에도 알아서 인덱싱이 같은 값들끼리 덧셈 연산을 수행한다.

```
mine    = Series([10, 20, 30], index=['naver', 'sk', 'kt'])
friend  = Series([10, 30, 20], index=['kt', 'naver', 'sk'])
merge   = mine + friend
print(merge)
```

```
kt      40
naver   40
sk       40
dtype: int64
```

pandas.Series

```
import numpy as np
import pandas as pd
```

```
obj=pd.Series([3,6,9,12])
print(obj)
print(type(obj))
print(len(obj))
```

```
0    3
1    6
2    9
3   12
dtype: int64
<class 'pandas.core.series.Series'>
4
```

```
obj=pd.Series([3,6,9,12], index=['a', 'b', 'c', 'd'])
obj
```

```
a    3
b    6
c    9
d   12
dtype: int64
```

```
emp={"김철수":5000, '김철호':7000, '한상민':4000, '문대용':4500} #dictionary 이용
obj=pd.Series(emp)
obj
```

```
김철수    5000
김철호    7000
문대용    4500
한상민    4000
dtype: int64
```

pandas.Series

```
import pandas as pd  
my_se=pd.Series([10,20,30])  
my_se
```

```
0    10  
1    20  
2    30  
dtype: int64
```

```
my_se[2]
```

```
30
```

```
my_se1 = pd.Series([10,20,30], index=['Samsung','Naver', 'LG'])  
my_se1
```

```
Samsung    10  
Naver      20  
LG         30  
dtype: int64
```

```
my_se1["LG"]
```

```
30
```

```
len(my_se1)
```

```
3
```

```
my_se1.min()
```

```
10
```

```
my_se1.max()
```

```
30
```


pandas.Series

```
my_se1.sort_values()
```

```
Samsung    10  
Naver      20  
LG         30  
dtype: int64
```

```
my_se1.sort_index()
```

```
LG         30  
Naver      20  
Samsung    10  
dtype: int64
```

```
my_se.sort_values(ascending=False)
```

```
2    30  
1    20  
0    10  
dtype: int64
```

```
my_se.sort_index()
```

```
0    10  
1    20  
2    30  
dtype: int64
```

where

```
my_sel.where(my_sel > 15)
```

```
Samsung    NaN
Naver      20.0
LG          30.0
dtype: float64
```

```
my_sel.where(my_sel > 15).dropna() #NaN 제거하기
```

```
Naver      20.0
LG          30.0
dtype: float64
```

```
my_sel.where(my_sel == my_sel.max()).dropna()
```

```
LG          30.0
dtype: float64
```

```
my_sel.sort_values(ascending = False).head(1)
```

```
LG          30
dtype: int64
```

pandas.Series

```
buy_stock = pd.Series([20,20,30], index=['Samsung', 'Naver', 'Kakao'])  
buy_stock
```

```
Samsung    20  
Naver      20  
Kakao      30  
dtype: int64
```

```
my_se1 + buy_stock
```

```
Kakao      NaN  
LG          NaN  
Naver      40.0  
Samsung    30.0  
dtype: float64
```

```
my_se1.add(buy_stock)
```

```
Kakao      NaN  
LG          NaN  
Naver      40.0  
Samsung    30.0  
dtype: float64
```

<https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html>

add는 이항연산이므로, operand가 NaN일 경우 대체될 value를 **fill_value**로 지정

```
new_my_se1= my_se1.add(buy_stock, fill_value=0)  
new_my_se1
```

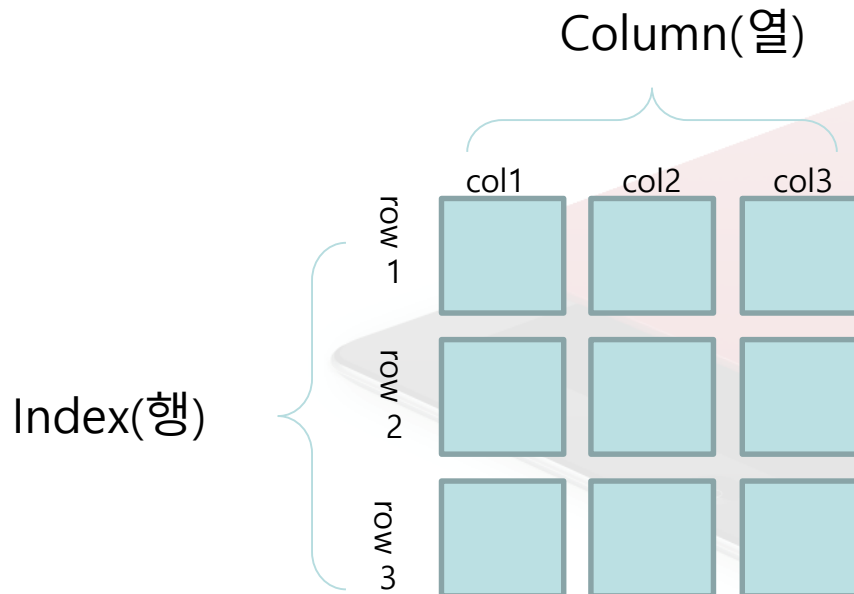
```
Kakao      30.0  
LG          30.0  
Naver      40.0  
Samsung    30.0  
dtype: float64
```

```
new_my_se2= new_my_se1.astype('int64')  
new_my_se2
```

```
Kakao      30  
LG          30  
Naver      40  
Samsung    30  
dtype: int64
```

DataFrame 구조

$n \times m$ 행렬구조를 가지는 데이터 구조이고 index와 column이 별도의 이름을 가지고, column별로 다른 데이터 타입을 가질 수 있음



```
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

```
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

#사전을 이용해서 원하는 데이터 추출하기

```
mdf = pd.DataFrame({
    'weight': [80.0, 70.4, 56.9, 66.8, 51.2],
    'height': [170, 167, 162, 164, 159],
    'gender': ['m', 'f', 'f', 'm', 'f']
})

print('몸무게 목록')
print(mdf['weight']) #몸무게 목록 출력하기
print('\n몸무게와 키 목록')
print(mdf[['weight', 'height']])
print('\n---키가 165이상')
print(mdf[mdf.height >= 165])
print('\n---남자들만... ')
print(mdf[mdf.gender == 'm'])
print('\n---키로 정렬하면...')
print(mdf.sort_values(by='height'))
print('\n---몸무게로 정렬하면...')
print(mdf.sort_values(by='weight', ascending=False))
```

mdf

	gender	height	weight
0	m	170	80.0
1	f	167	70.4
2	f	162	56.9
3	m	164	66.8
4	f	159	51.2

몸무게 목록

```
0    80.0
1    70.4
2    56.9
3    66.8
4    51.2
```

Name: weight, dtype: float64

몸무게와 키 목록

```
weight height
0    80.0    170
1    70.4    167
2    56.9    162
3    66.8    164
4    51.2    159
```

---키가 165이상

```
gender height weight
0      m    170    80.0
1      f    167    70.4
```

---남자들만...

```
gender height weight
0      m    170    80.0
3      m    164    66.8
```

---키로 정렬하면...

```
gender height weight
4      f    159    51.2
2      f    162    56.9
3      m    164    66.8
1      f    167    70.4
0      m    170    80.0
```

---몸무게로 정렬하면...

```
gender height weight
0      m    170    80.0
1      f    167    70.4
3      m    164    66.8
2      f    162    56.9
4      f    159    51.2
```

DataFrame 생성: 1 column

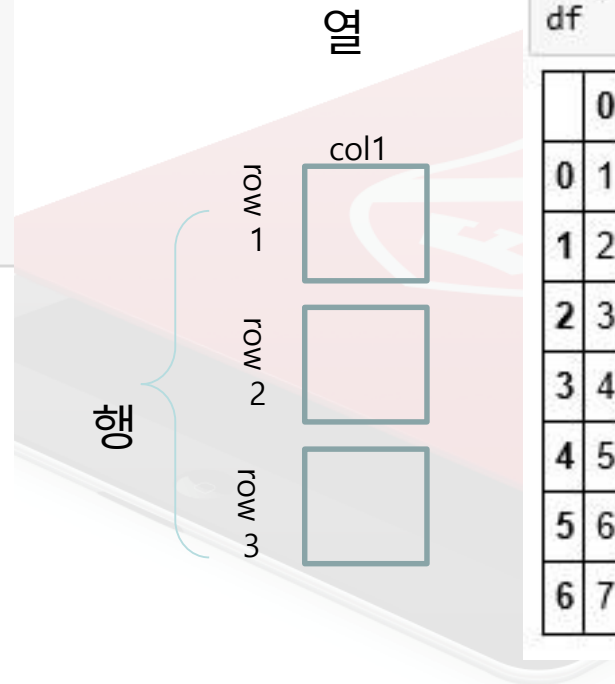
DataFrame은 기본적으로 column 단위로 데이터를 관리함

```
import pandas as pd

# Our small data set
d = [0,1,2,3,4,5,6,7,8,9]

# Create dataframe
df = pd.DataFrame(d)
print(df)
```

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



```
data = [1,2,3,4,5,6,7]
df = pd.DataFrame(data)
df
```

	0
0	1
1	2
2	3
3	4
4	5
5	6
6	7

DataFrame 생성: list/tuple

column단위로 리스트를 만들어서 **zip을 이용해서 순서쌍을 만들고** 데이터를 생성

```
names=['Bob','Jessica','Mary','John','kate']  
births=[988, 155, 77, 578,973]  
BabySet=list(zip(names,births))  
df = pd.DataFrame(data=BabySet, columns=['Names', 'Births'])  
df
```

	Names	Births
0	Bob	988
1	Jessica	155
2	Mary	77
3	John	578
4	kate	973

DataFrame 생성: dict

column단위로 리스트를 만들어서 dict에 대입해서 데이터를 생성

```
names=['Bob', 'Jessica', 'Mary', 'John','kate']  
births =[988, 155, 77, 578, 973]  
  
DicBabySet={'Names':names, 'Births':births}  
df = pd.DataFrame(data=DicBabySet)  
df
```

	Births	Names
0	988	Bob
1	155	Jessica
2	77	Mary
3	578	John
4	973	kate

DataFrame 생성: Series

두 개의 series타입에서 키값을 추출해서 자동으로 인덱스화 해서 처리

```
area_dic = {'california':423967, 'Texas': 695662, 'New York':141297, \
            'Florida': 170312, 'Illinois':149995}
pop_dic = {'california':1423967, 'Texas': 1695662, 'New York':1141297, \
           'Florida': 1170312, 'Illinois':1149995}
area = pd.Series(area_dic)
population = pd.Series(pop_dic)

states = pd.DataFrame({'Population':population, 'Area':area})
states
```

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

```
states.index
```

```
Index(['Florida', 'Illinois', 'New York', 'Texas', 'california'], dtype='object')
```

```
states.values
```

```
array([[ 170312, 1170312],
       [ 149995, 1149995],
       [ 141297, 1141297],
       [ 695662, 1695662],
       [ 423967, 1423967]], dtype=int64)
```

DataFrame 생성

- 딕셔너리를 통해 각 칼럼에 대한 데이터를 저장한 후 딕셔너리를 DataFrame 클래스의 생성자 인자로 넘겨주면 DataFrame 객체가 생성된다.

```
raw_data = {'col0': [1, 2, 3, 4],
            'col1': [10, 20, 30, 40],
            'col2': [100, 200, 300, 400]}

data = DataFrame(raw_data)
print(data)
```

	col0	col1	col2
0	1	10	100
1	2	20	200
2	3	30	300
3	4	40	400

```
data['col0']
```

0	1
1	2
2	3
3	4

Name: col0, dtype: int64

```
type(data['col0'])
```

pandas.core.series.Series

raw_data 딕셔너리는 'col0', 'col1', 'col2'라는 key 값을 가지며 각 key는 리스트 타입의 value를 가진다.

DataFrame에는 3개의 Series 객체가 있고, 이는 'col0', 'col1', 'col2'라는 key에 각각 대응되는 값(value)이다. 'col0', 'col1', 'col2'라는 key를 통해 value에 해당하는 Series 객체에 접근할 수 있다.

'col0', 'col1', 'col2'라는 문자열은 DataFrame의 각 칼럼을 인덱싱하는데 사용

```
raw_data = {'Col0':[1,2,3,4],
            'Col1':[10,20,30,40],
            "Col2":[100,200,300,400]}
data = pd.DataFrame(raw_data)
data
```

	Col0	Col1	Col2
0	1	10	100
1	2	20	200
2	3	30	300
3	4	40	400

```
data = pd.DataFrame(raw_data, index=['r0','r1','r2','r3'])
data
```

	Col0	Col1	Col2
r0	1	10	100
r1	2	20	200
r2	3	30	300
r3	4	40	400

```
data['Col2']
```

```
r0    100
r1    200
r2    300
r3    400
Name: Col2, dtype: int64
```

- Column순서 변경
- Index 순서 변경

```
raw_data = {'Col0':[1,2,3,4],
            'Col1':[10,20,30,40],
            "Col2": [100,200,300,400]}
data = pd.DataFrame(raw_data, columns=['Col1','Col2','Col0'] )
data
```

	Col1	Col2	Col0
0	10	100	1
1	20	200	2
2	30	300	3
3	40	400	4

```
df=data.reindex(index=[0,3,2,1])
df
```

	Col1	Col2	Col0
0	10	100	1
3	40	400	4
2	30	300	3
1	20	200	2

- Column 추가
- 원하는 위치의 값을 변경: `loc` 옵션

```
import numpy as np
df['Col3']=np.nan
df
```

	Col1	Col2	Col0	Col3
0	10	100	1	NaN
3	40	400	4	NaN
2	30	300	3	NaN
1	20	200	2	NaN

```
df.loc[1,['Col3']]=2000
df
```

	Col1	Col2	Col0	Col3
0	10	100	1	NaN
3	40	400	4	NaN
2	30	300	3	NaN
1	20	200	2	2000.0

- Column 간 연산
- 행 추가

```
df['NewCol'] = df['Col1'] + df['Col2']
df
```

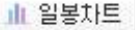
	Col1	Col2	Col0	Col3	NewCol
0	10	100	1	NaN	110
3	40	400	4	NaN	440
2	30	300	3	NaN	330
1	20	200	2	2000.0	220

```
df.loc[4] = np.nan
df
```

	Col1	Col2	Col0	Col3	NewCol
0	10.0	100.0	1.0	NaN	110.0
3	40.0	400.0	4.0	NaN	440.0
2	30.0	300.0	3.0	NaN	330.0
1	20.0	200.0	2.0	2000.0	220.0
4	NaN	NaN	NaN	NaN	NaN

실습: 주가를 DataFrame으로 저장해보기

- 16.02.23~16.02.29일 사이의 일자별 주가 중 시가, 고가, 저가, 종가를 DataFrame으로 저장해보자.
- 참고로 시가, 고가, 저가, 종가는 영어로 open, high, low, close라고 부르며 약어로 OHLC라고 한다.

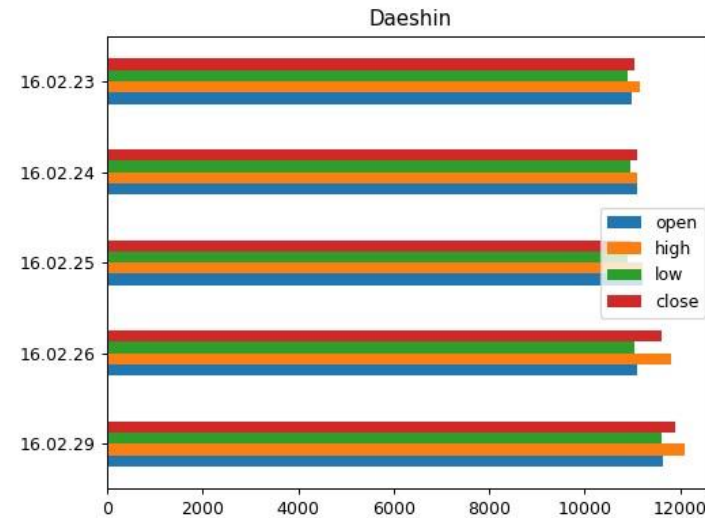
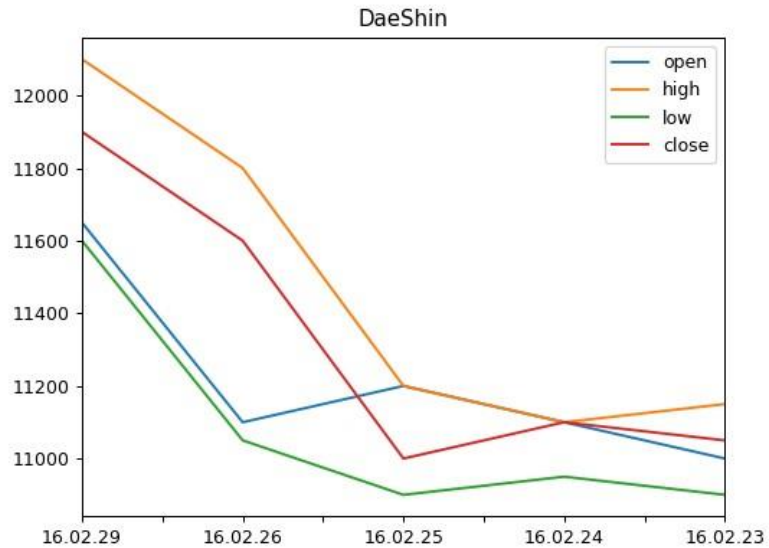
일자별 주가  자세히▶

일자	시가	고가	저가	종가	전일비	등락률	거래량
16.02.29	11,650	12,100	11,600	11,900	▲ 300	+2.59%	225,844
16.02.26	11,100	11,800	11,050	11,600	▲ 600	+5.45%	385,241
16.02.25	11,200	11,200	10,900	11,000	▼ 100	-0.90%	161,214
16.02.24	11,100	11,100	10,950	11,100	▲ 50	+0.45%	77,201
16.02.23	11,000	11,150	10,900	11,050	▲ 100	+0.91%	113,131
16.02.22	10,950	11,050	10,850	10,950	▼ 100	-0.90%	138,387
16.02.19	10,950	11,100	10,800	11,050	- 0	0.00%	76,105
16.02.18	11,050	11,200	10,950	11,050	▲ 250	+2.31%	83,611
16.02.17	11,150	11,300	10,800	10,800			
16.02.16	10,950	11,200	10,850	11,100			

	open	high	low	close
16.02.29	11650	12100	11600	11900
16.02.26	11100	11800	11050	11600
16.02.25	11200	11200	10900	11000
16.02.24	11100	11100	10950	11100
16.02.23	11000	11150	10900	11050

그림 13.9 2차원 형태의 데이터 (출처: 다음 증권)

실습: 그래프로 그리기



주가를 DataFrame으로 저장해보기

- 16.02.23~16.02.29일 사이의 일자별 주가 중 시가, 고가, 저가, 종가를 DataFrame으로 저장해보자.
- 참고로 시가, 고가, 저가, 종가는 영어로 open, high, low, close라고 부르며 약어로 OHLC라고 한다.

```
daeshin = {'open': [11650, 11100, 11200, 11100, 11000],
           'high': [12100, 11800, 11200, 11100, 11150],
           'low' : [11600, 11050, 10900, 10950, 10900],
           'close': [11900, 11600, 11000, 11100, 11050]}
```

```
daeshin_day = DataFrame(daeshin)
print(daeshin_day)
```

	close	high	low	open
0	11900	12100	11600	11650
1	11600	11800	11050	11100
2	11000	11200	10900	11200
3	11100	11100	10950	11100
4	11050	11150	10900	11000

- DataFrame 객체에서 칼럼의 순서는 DataFrame 객체를 생성할 때 **columns**라는 키워드를 지정 할 수 있다.

```
daeshin_day = DataFrame(daeshin, columns=['open', 'high', 'low', 'close'])  
print(daeshin_day)
```

	open	high	low	close
0	11650	12100	11600	11900
1	11100	11800	11050	11600
2	11200	11200	10900	11000
3	11100	11100	10950	11100
4	11000	11150	10900	11050

- DataFrame에서 인덱스 역시 DataFrame 객체를 생성하는 시점에 index를 통해 지정할 수 있다. 먼저 인덱싱에 사용할 값을 만든 후 이를 DataFrame 객체 생성 시점에 지정한다.

```
date = ['16.02.29', '16.02.26', '16.02.25', '16.02.24', '16.02.23']
daeshin_day = DataFrame(daeshin, columns=['open', 'high', 'low', 'close'], index=date)
print(daeshin_day)
```

	open	high	low	close
16.02.29	11650	12100	11600	11900
16.02.26	11100	11800	11050	11600
16.02.25	11200	11200	10900	11000
16.02.24	11100	11100	10950	11100
16.02.23	11000	11150	10900	11050

```
kakao=pd.Series([92600, 93400, 94500, 93800, 94000, 93500], index=pd.date_range("20180227", periods=6))
kakao
```

2018-02-27	92600
2018-02-28	93400
2018-03-01	94500
2018-03-02	93800
2018-03-03	94000
2018-03-04	93500

Freq: D, dtype: int64

date_range

`pd.date_range` 함수를 쓰면 모든 날짜/시간을 일일이 입력할 필요 없이 시작일과 종료일 또는 시작일과 기간(`periods=`)을 입력하면 범위 내의 인덱스를 생성해 준다.

- `freq` 인수로 특정한 날짜만 생성되도록 할 수도 있다. 많이 사용되는 `freq` 인수값은 다음과 같다.
- s: 초
- T: 분
- H: 시간
- D: 일(day)
- B: 주말이 아닌 평일
- W: 주(일요일)
- W-MON: 주(월요일)
- M: 각 달(month)의 마지막 날
- MS: 각 달의 첫날
- BM: 주말이 아닌 평일 중에서 각 달의 마지막 날
- BMS: 주말이 아닌 평일 중에서 각 달의 첫날
- WOM-2THU: 각 달의 두번째 목요일
- Q-JAN: 각 분기의 첫달의 마지막 날
- Q-DEC: 각 분기의 마지막 달의 마지막 날
- 보다 자세한 내용은 다음 웹사이트를 참조한다.
- <http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases>

DataFrame 칼럼, 로우 선택

- DataFrame 객체의 column에 접근하려면 **칼럼 이름**을, row에 접근하려면 **loc** 메소드를 통해 **인덱스값**을 이용한다.

```
close = daeshin_day['close']
print(close)
```

```
16.02.29    11900
16.02.26    11600
16.02.25    11000
16.02.24    11100
16.02.23    11050
Name: close, dtype: int64
```

```
day_data=daeshin_day.loc['16.02.24']
print(day_data)
print(type(day_data))
```

```
open      11100
high      11100
low       10950
close     11100
Name: 16.02.24, dtype: int64
<class 'pandas.core.series.Series'>
```

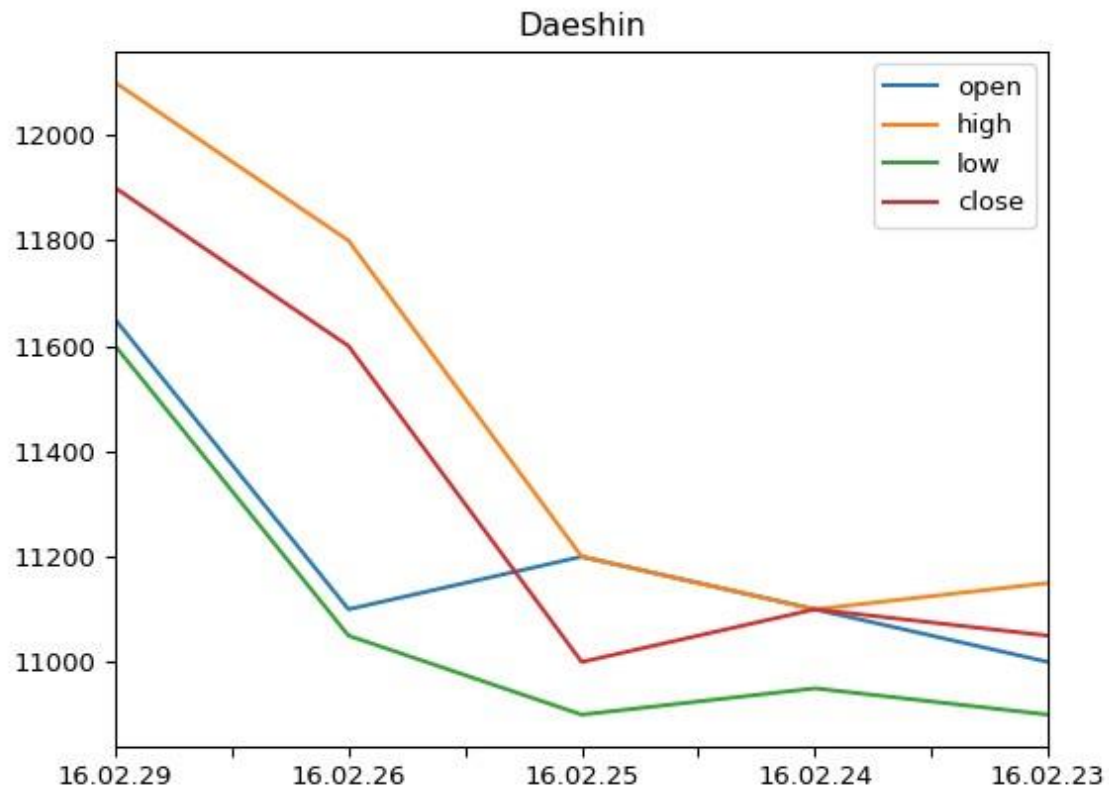
- DataFrame 객체의 칼럼 이름과 인덱스 값을 확인하려면 각각 **columns**와 **index** 속성을 사용한다.

```
print(daeshin_day.columns)
print(daeshin_day.index)
```

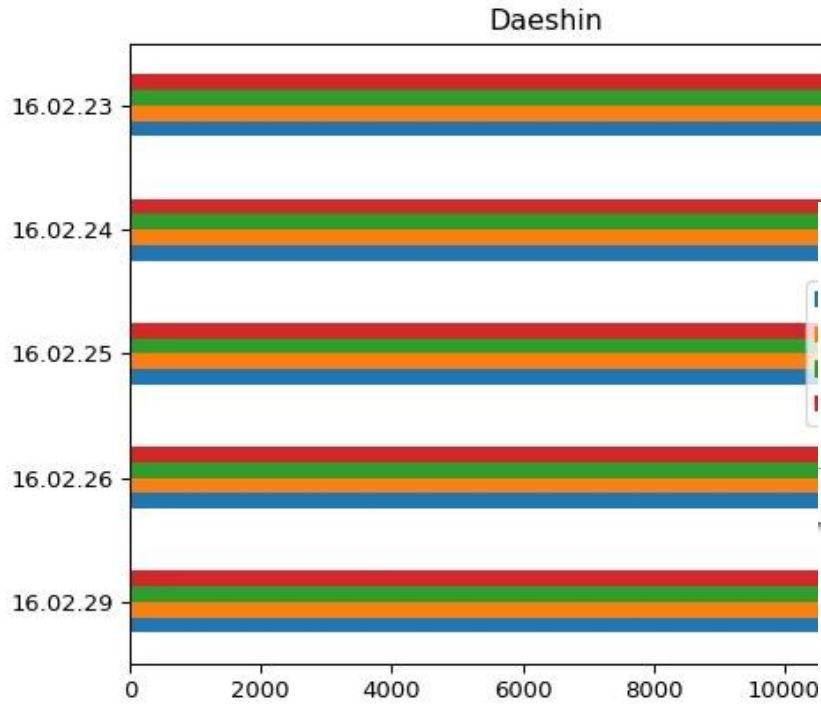
```
Index(['open', 'high', 'low', 'close'], dtype='object')
Index(['16.02.29', '16.02.26', '16.02.25', '16.02.24', '16.02.23'], dtype='object')
```



```
%matplotlib nbagg
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
dae_day.plot(title='Daeshin')
```



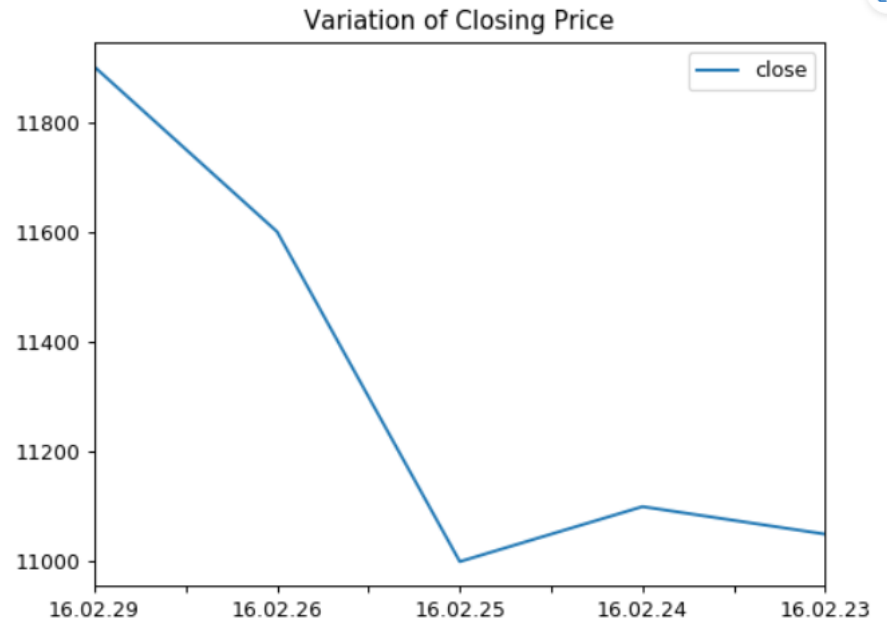

```
dae_day.plot(title='Daeshin', kind="barh")
```



```
dae_close = dae_day['close']  
dae_close
```

```
16.02.29    11900  
16.02.26    11600  
16.02.25    11000  
16.02.24    11100  
16.02.23    11050  
Name: close, dtype: int64
```

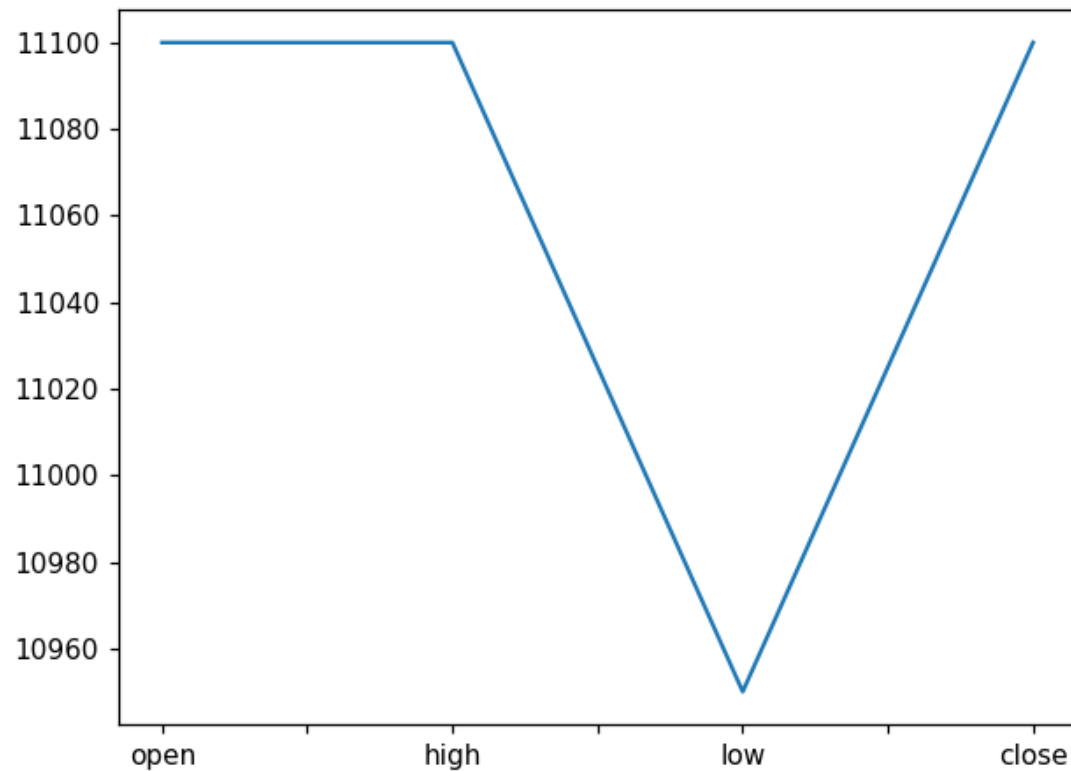
```
dae_close.plot(title='Variation of Closing Price', legend='close')
```



```
day_data=daeshin_day.loc['16.02.24']
print(day_data)
print(type(day_data))
```

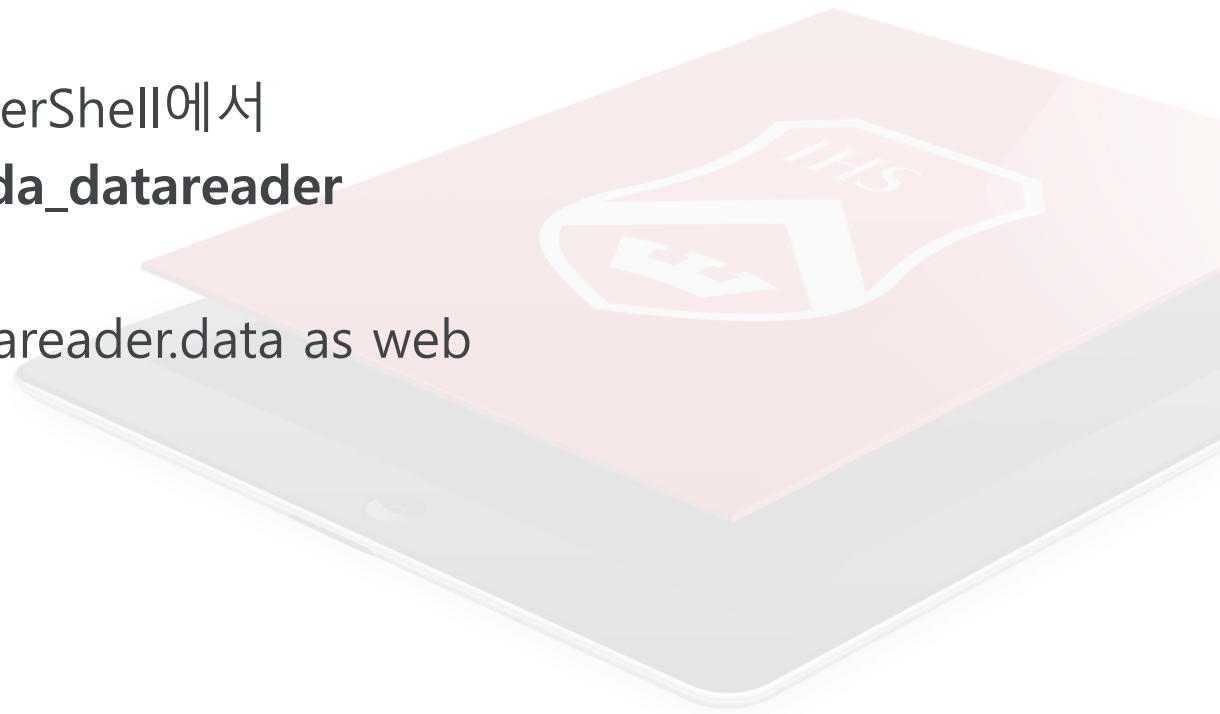
```
open      11100
high      11100
low       10950
close     11100
Name: 16.02.24, dtype: int64
<class 'pandas.core.series.Series'>
```

```
day_data.plot()
```



DataReader 이용하기

- **pandas_datareader** 패키지를 추가로 설치
- Anaconda Prompt에서
 - **conda** install panda-datareader
- 또는 Windows PowerShell에서
 - **pip** install panda_datareader
- `import pandas_datareader.data as web`

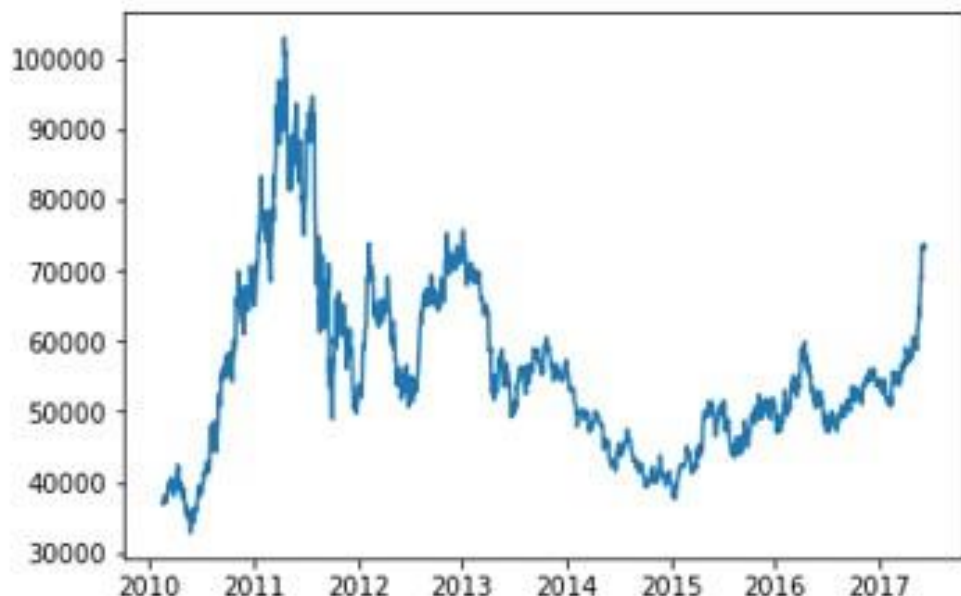


```
import pandas_datareader.data as web
import datetime
start = datetime.datetime(2010, 2, 19)
end = datetime.datetime.today()
gs = web.DataReader("078930", "google", start, end)
```

```
gs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1804 entries, 2010-02-19 to 2017-06-05
Data columns (total 5 columns):
Open      1804 non-null float64
High      1804 non-null float64
Low       1804 non-null float64
Close     1804 non-null float64
Volume    1804 non-null int64
dtypes: float64(4), int64(1)
memory usage: 84.6 KB
```

```
import matplotlib.pyplot as plt
plt.plot(gs['Close'])
plt.show()
```



```
import pandas_datareader.data as web
import datetime
start = datetime.datetime(2017, 2, 19)
end = datetime.datetime.today()
gs = web.DataReader("078930.KS", 'yahoo', start, end)
```

```
gs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1415 entries, 2017-02-20 to 2022-11-28
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  -
0   High        1415 non-null   float64
1   Low         1415 non-null   float64
2   Open        1415 non-null   float64
3   Close       1415 non-null   float64
4   Volume      1415 non-null   float64
5   Adj Close   1415 non-null   float64
dtypes: float64(6)
memory usage: 77.4 KB
```

	High	Low	Open	Close	Volume	Adj Close
Date						
2017-02-20	54800.0	53800.0	54300.0	54500.0	134852.0	44448.437500
2017-02-21	55200.0	54300.0	54300.0	55000.0	129964.0	44856.218750
2017-02-22	54900.0	53800.0	54900.0	54200.0	162382.0	44203.765625
2017-02-23	54400.0	53600.0	54400.0	53900.0		
2017-02-24	54400.0	53500.0	54000.0	54000.0		

```
import matplotlib.pyplot as plt
plt.plot(gs['Close'])
plt.show()
```

...
2022-11-22	47050.0	46250.0	46500.0	47050.0
2022-11-23	47450.0	46550.0	47050.0	47150.0
2022-11-24	47100.0	46000.0	47100.0	46150.0
2022-11-25	46650.0	46100.0	46200.0	46600.0
2022-11-28	46550.0	45200.0	46550.0	46200.0

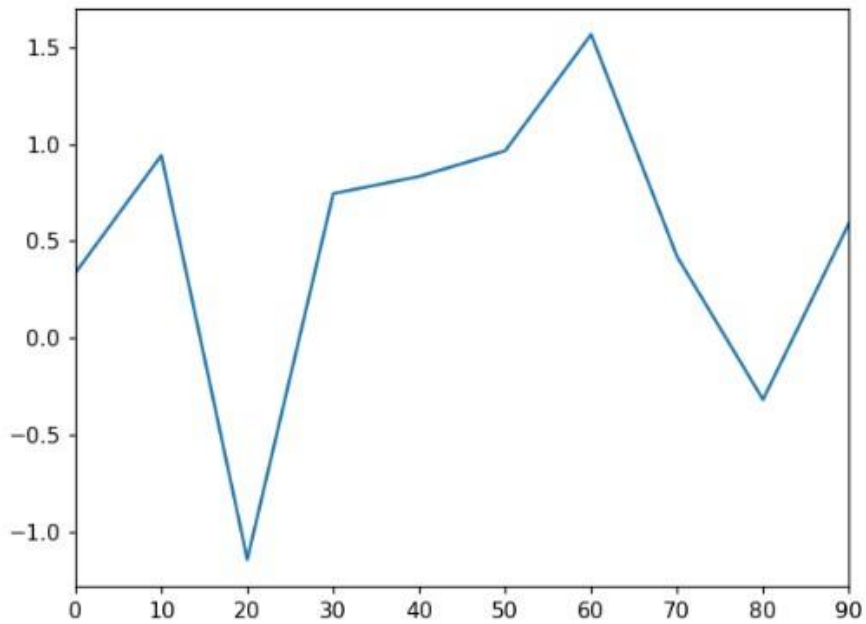


```
%matplotlib nbagg
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
a = pd.Series(np.random.randn(10), index = np.arange(0, 100, 10))
a
```

```
0      0.336954
10     0.942193
20    -1.139543
30     0.745335
40     0.833991
50     0.965838
60     1.565770
70     0.421018
80    -0.315718
90     0.592613
dtype: float64
```

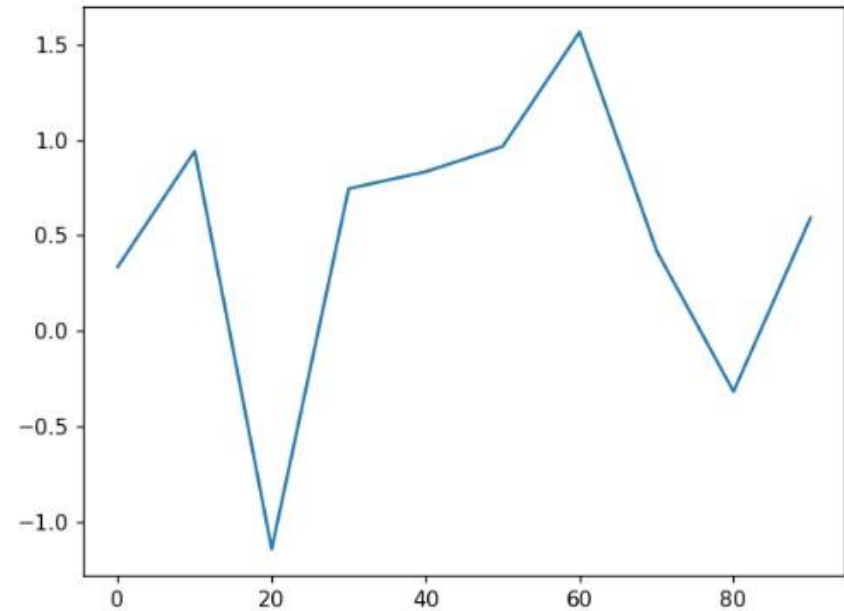
a.plot()

```
a.plot()
```



plt.plot(a)

```
plt.plot(a)
```

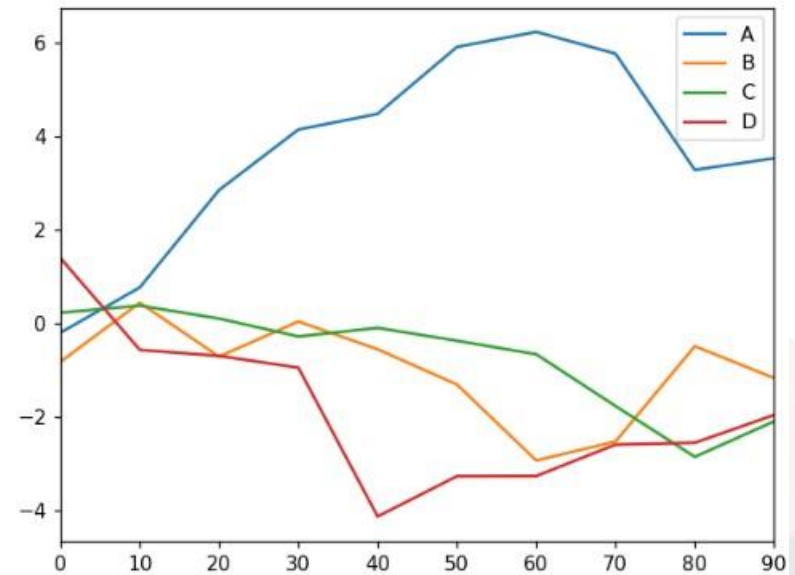



```
df = pd.DataFrame(np.random.randn(10,4).cumsum(axis=0), \
                  columns = ["A", "B", "C", "D"], \
                  index=np.arange(0,100,10))
```

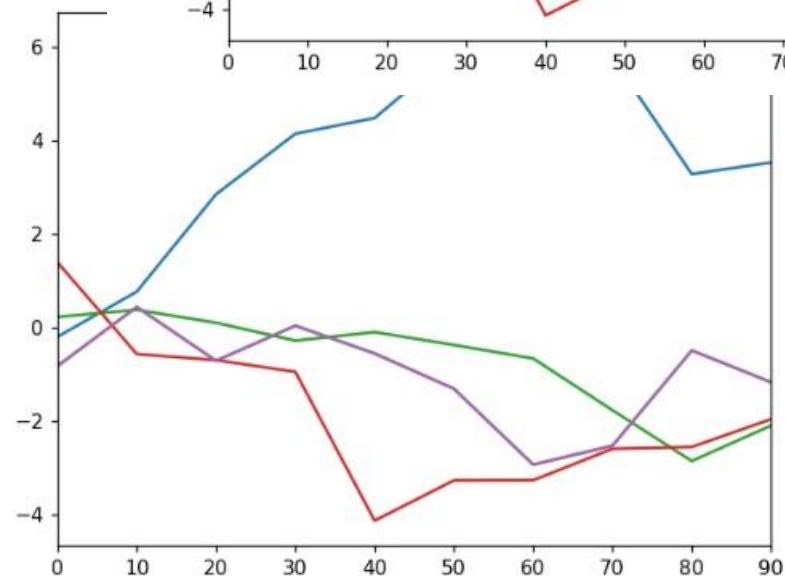
df

	A	B	C	D
0	-0.195147	-0.818633	0.231205	1.397892
10	0.771331	0.444291	0.382609	-0.563171
20	2.854510	-0.707699	0.105789	-0.692718
30	4.147936	0.043988	-0.276413	-0.944034
40	4.483857	-0.550648	-0.093270	-4.125725
50	5.914736	-1.304638	-0.368625	-3.265200
60	6.237437	-2.928266	-0.657638	-3.261847
70	5.771313	-2.523559	-1.764859	-2.589371
80	3.286360	-0.485514	-2.850688	-2.549705
90	3.534146	-1.165568	-2.093120	-1.955858

df.plot()



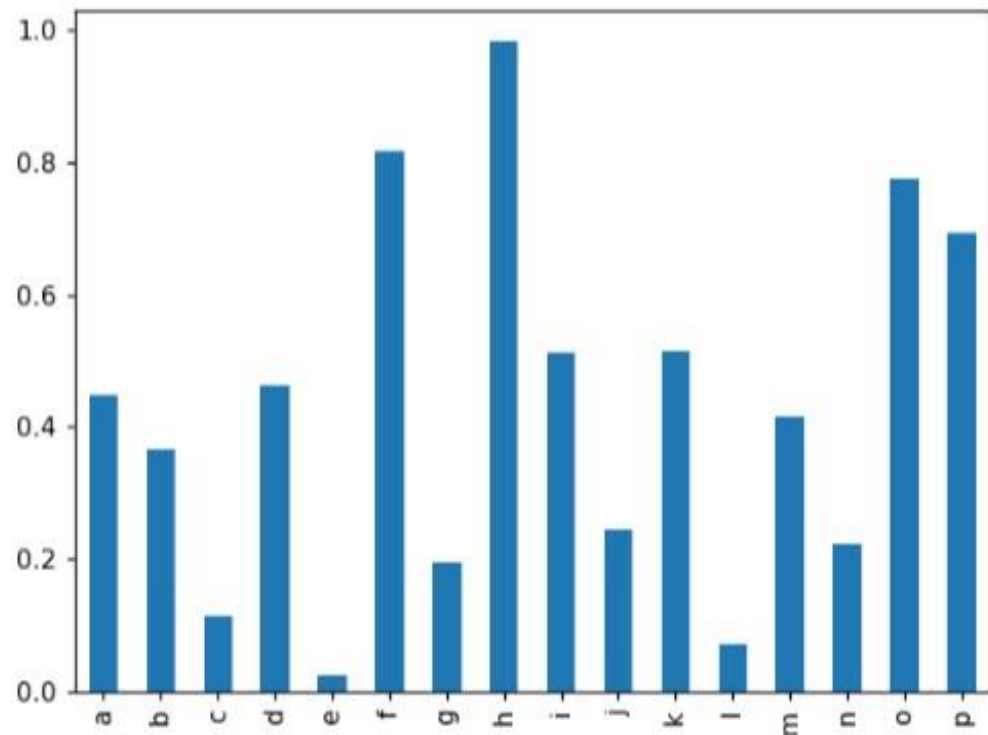
plt.plot(df)



```
s2 = pd.Series(np.random.rand(16), index=list("abcdefghijklmnop"))
s2
```

```
a    0.447705
b    0.366086
c    0.112897
d    0.461951
e    0.024518
f    0.816570
g    0.194222
h    0.982399
i    0.512912
j    0.243593
k    0.513772
l    0.070270
m    0.416423
n    0.221580
o    0.774268
p    0.692939
dtype: float64
```

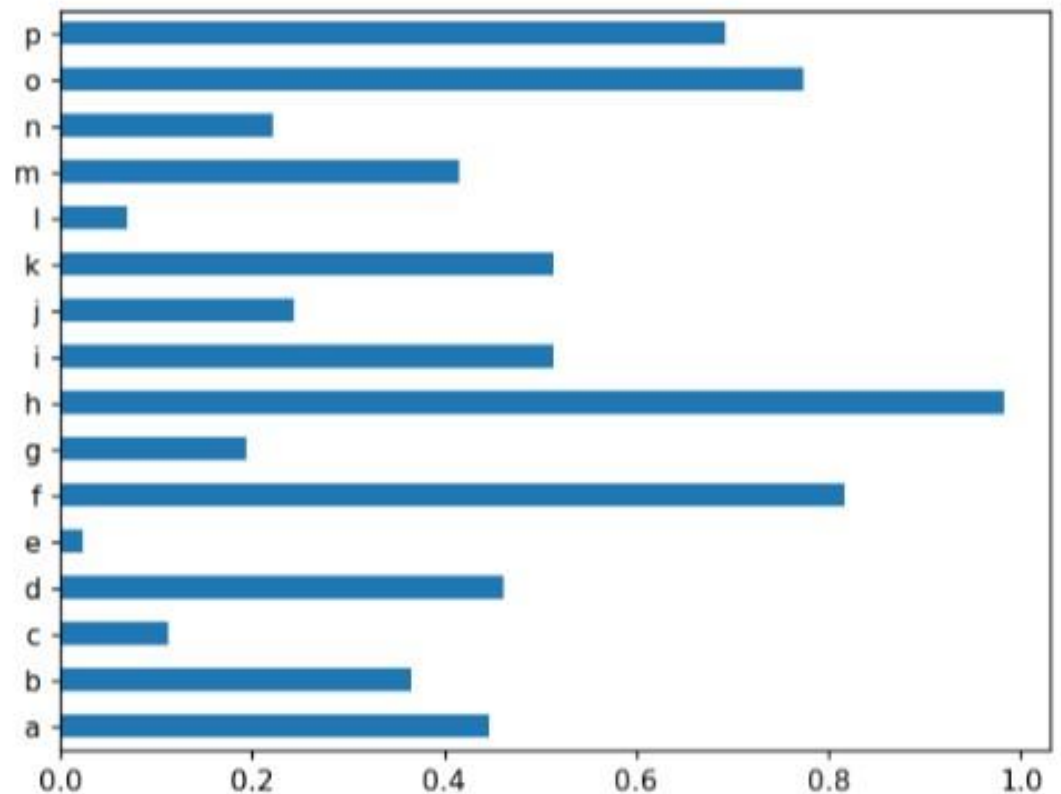
```
s2.plot(kind="bar")
```



```
s2 = pd.Series(np.random.rand(16), index=list("abcdefghijklmnop"))
s2
```

```
a    0.447705
b    0.366086
c    0.112897
d    0.461951
e    0.024518
f    0.816570
g    0.194222
h    0.982399
i    0.512912
j    0.243593
k    0.513772
l    0.070270
m    0.416423
n    0.221580
o    0.774268
p    0.692939
dtype: float64
```

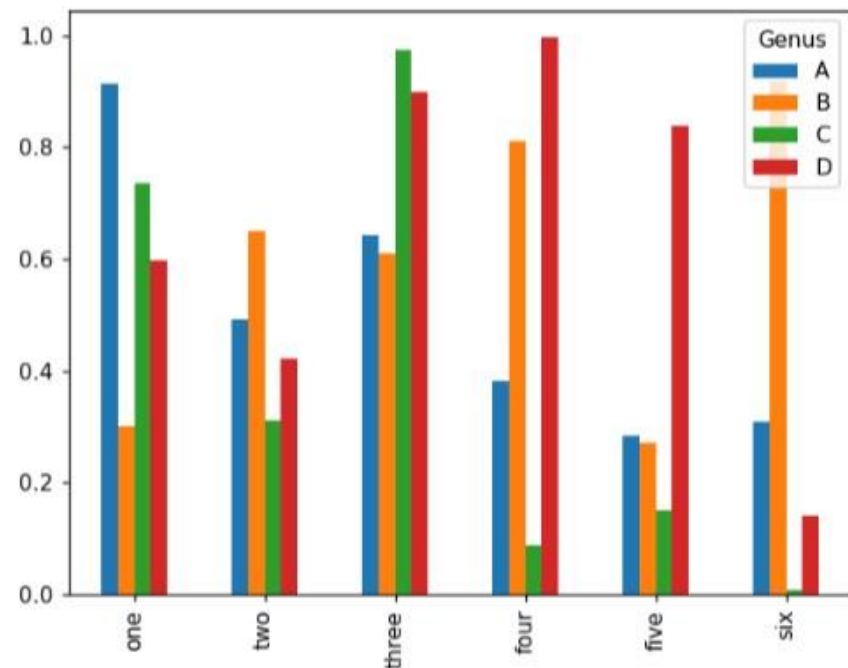
```
s2.plot(kind="barh")
```



```
df2 = pd.DataFrame(np.random.rand(6, 4),
                    index=['one', "two", "three", "four", "five", "six"],
                    columns=pd.Index(["A", "B", "C", "D"], name="Genus"))
df2
```

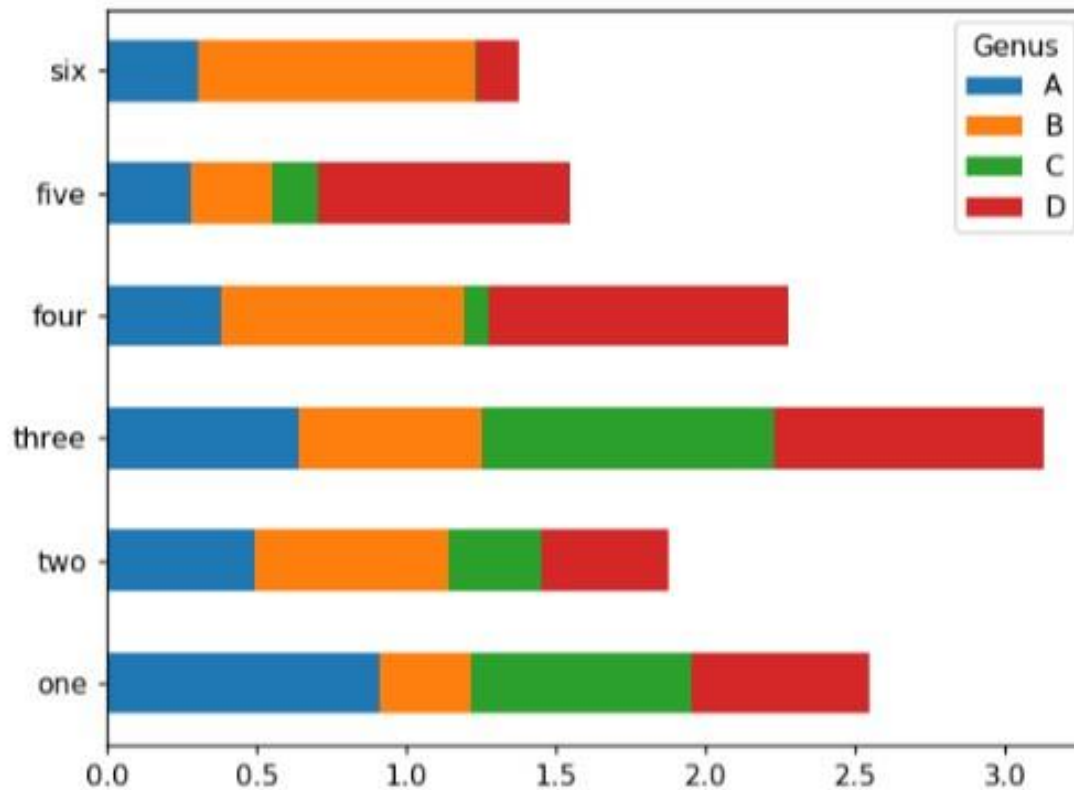
Genus	A	B	C	D
one	0.418906	0.236360	0.356705	0.734447
two	0.607312	0.303439	0.386386	0.028987
three	0.682558	0.194159	0.673478	0.713587
four	0.435695	0.141532	0.976216	0.189504
five	0.560502	0.049765	0.696089	0.068235
six	0.290436	0.671938	0.569046	0.369884

```
df2.plot(kind="bar")
```



Stacked = True

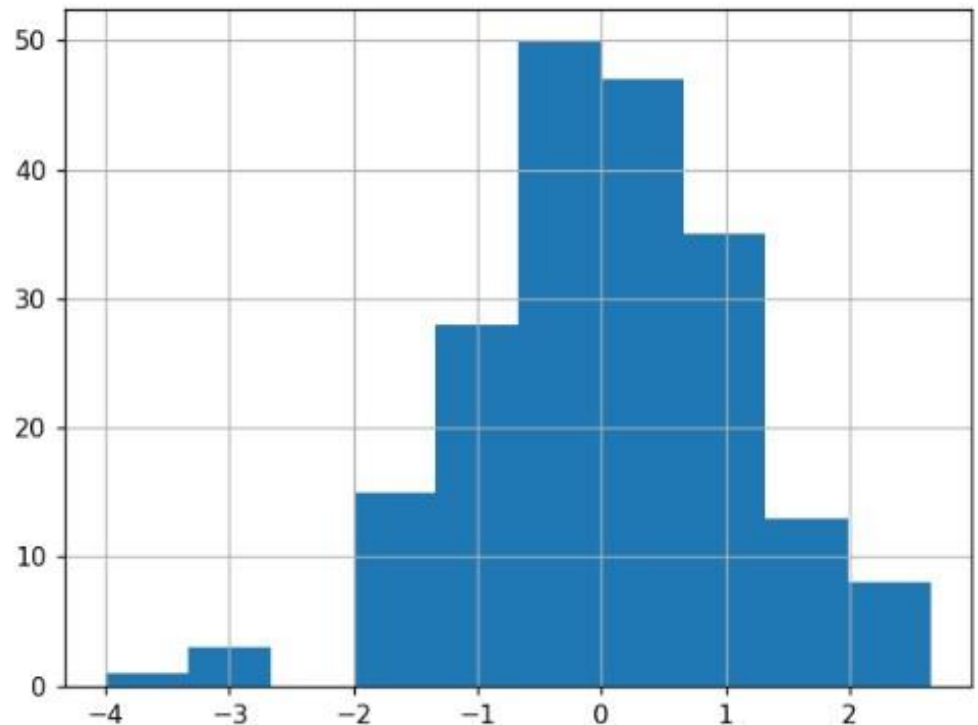
```
df2.plot(kind="barh", stacked=True)
```



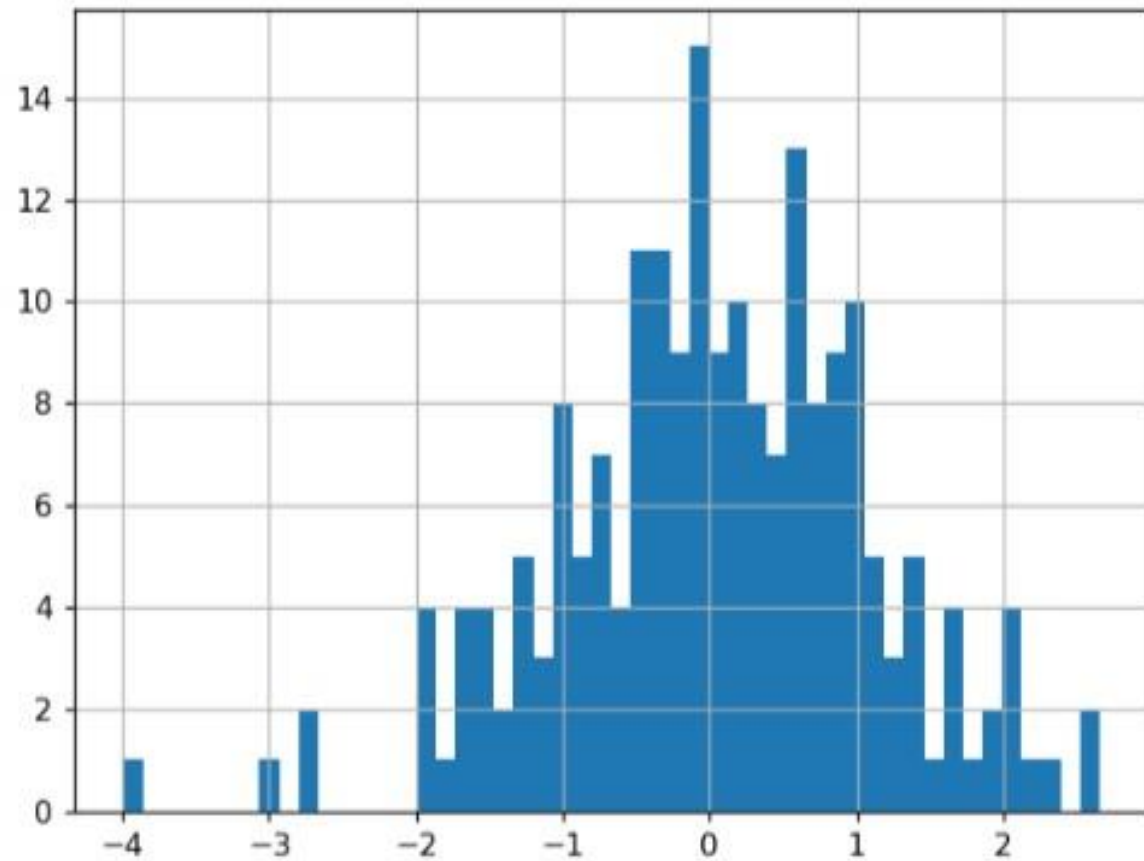
```
s3 = pd.Series(np.random.normal(0, 1, size=200))
s3
```

```
0    -0.894760  --  .....
1     0.828095  ...
2    -0.483131  170  1.035331
3    -0.598732  171  -0.986877
4    -0.216426  172  0.033864
5    -0.035249  173  1.473756
6    -0.014753  174  0.302862
7     0.077966  175  -1.431808
8     0.558808  176  0.763044
9    -1.527285  177  -0.411425
10     0.134735  178  -0.187996
11    -0.119358  179  -1.612416
12    -0.743607  180  -0.170691
13     1.354945  181  -0.688637
14     1.159680  182  -0.096160
15     0.385966  183  -0.448952
16    -0.696737  184  -0.888805
17    -3.001067  185  1.089468
18     0.779795  186  -0.287810
19    -1.550678  187  1.069154
20     0.977719  188  0.633246
21     0.207527  189  -0.080853
22    -0.309065  190  -0.575455
23    -1.255309  191  0.825637
24     0.293596  192  -0.488726
25     0.518539  193  0.219815
26    -0.008871  194  1.648556
27     1.280560  195  0.598652
28    -1.909763  196  -2.763028
29     0.153304  197  -1.289101
                198  1.431606
                199  -0.528180
                ...
170    1.035331  dtype: float64
```

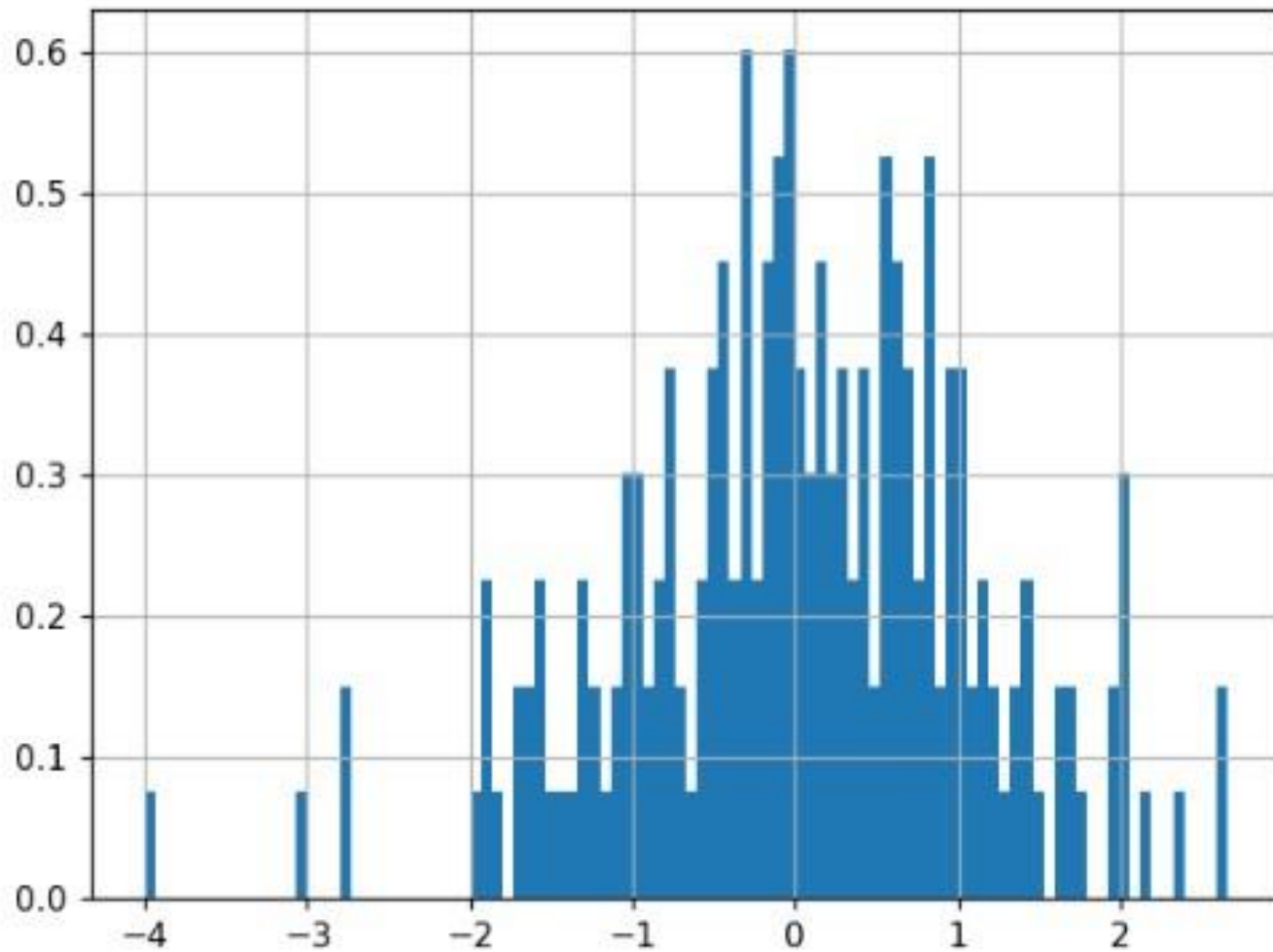
```
s3.hist()
```



```
s3.hist(bins=50)
```

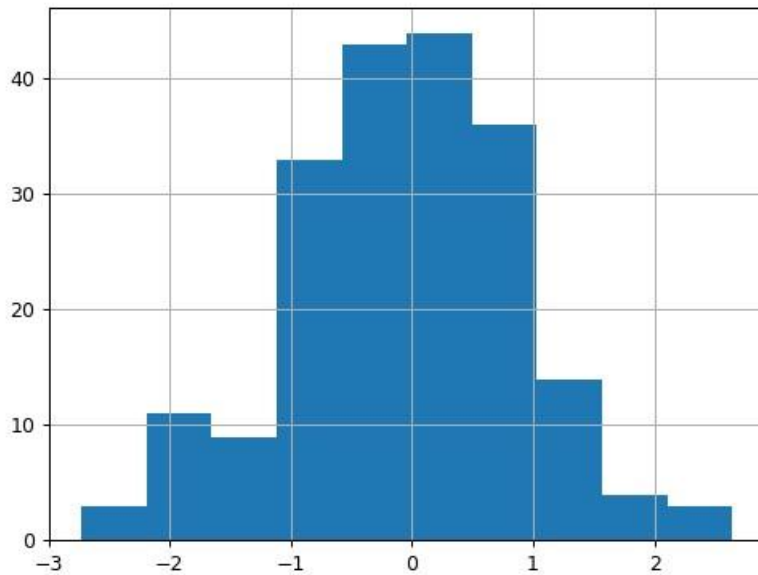



```
s3.hist(bins=100, normed=True)
```

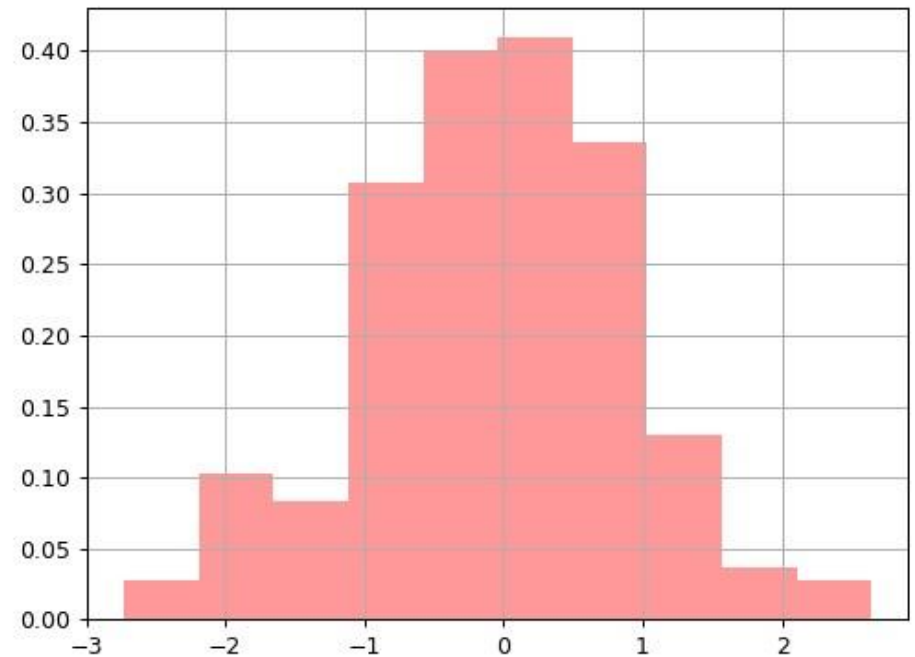


hist함수 :파라미터

facecolor는 색깔, alpha는 투명도 표시



```
: s3.hist(normed=True, facecolor='r', alpha=0.4)
```



scatter

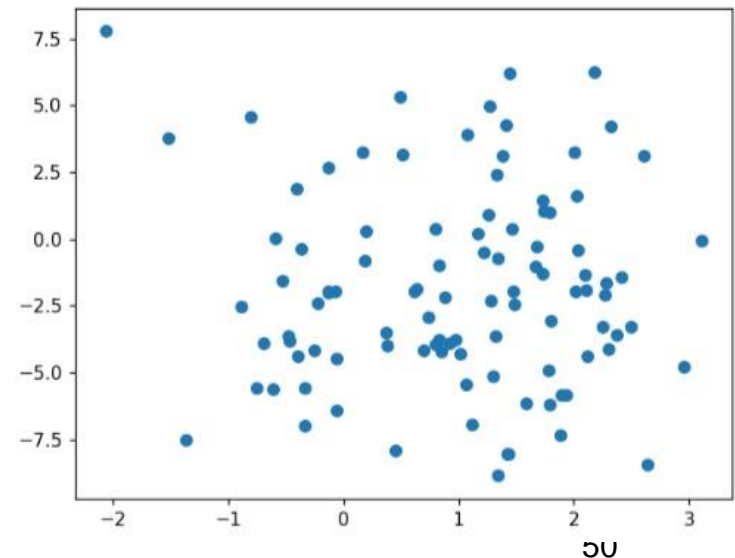
```
x1 = np.random.normal(1,1, size=(100, 1))
x2 = np.random.normal(-2, 4, size=(100, 1))
x = np.concatenate((x1, x2), axis=1)
x
```

```
array([[ 0.19313506,  0.31312163],
       [-0.47424759, -3.6463171 ],
       [-0.39566273, -4.39315792],
       [ 1.58981876, -6.13784689],
       [ 0.82694043, -0.97478759],
       [-1.36406218, -7.52887931],
       [ 1.32639017,  2.4169649 ],
       [ 2.30069265, -4.10133103],
       [-0.40316729,  1.87375378],
       [-0.21853915, -2.40974112],
       [ 1.47694488, -1.96760802],
       [ 1.42585904, -8.0605759 ],
       [-2.06136554,  7.81256667],
       [ 1.06650113, -5.43389854],
       [ 1.46705916,  0.38011986],
       [ 2.2663812 , -2.0689719 ],
       [ 1.93725699, -5.84513859],
       [ 2.25169038, -3.27098125],
       [-0.80174507,  4.59140458],
       [-0.68720265, -3.91389168],
       [ 2.01903759, -1.94005522],
       [ 0.18252244, -0.78753369],
       [ 2.41669538, -1.40658701],
       [-0.06693979, -1.94870717],
       [-0.7513464 , -5.56498656],
       [-0.13268378, -1.94952599],
       [ 1.21816097, -0.47712306],
       [ 1.25555707,  0.90667213],
       [ 2.32595465,  4.24346478],
       [-1.52017399,  3.77346622],
       [ 2.3737178 , -3.56550352],
       [ 0.85144142, -4.22281907],
       [-0.13133085,  2.68650806],
       [ 1.07774921,  3.92385452],
       [-0.36599726, -0.36424637],
       [ 2.11846786, -4.38793622],
       [ 1.31668586, -3.62571651],
       [ 1.27920239, -2.31842658],
       [ 0.92087715, -3.89298929],
       [ 2.95758481, -4.75543334],
```

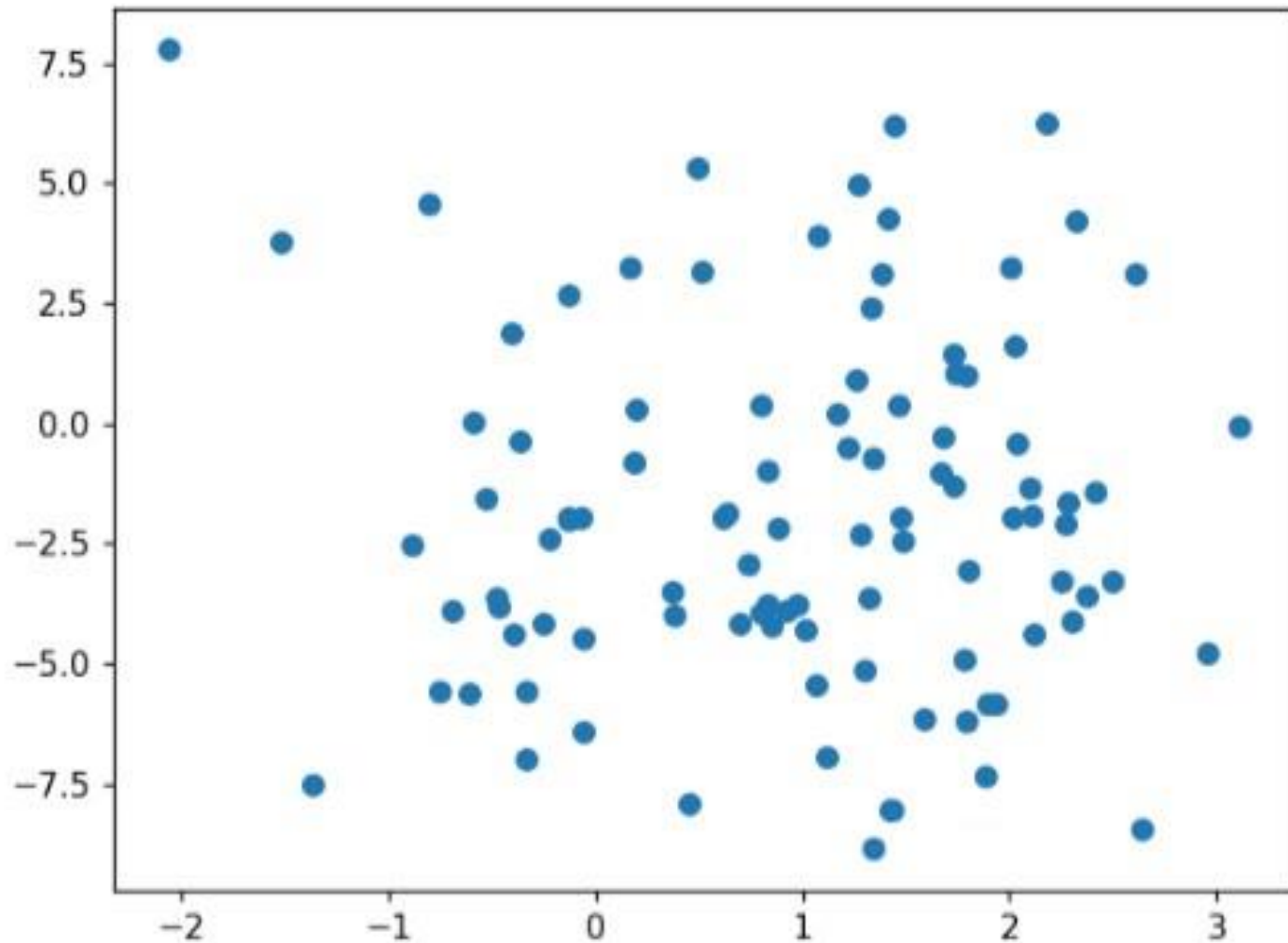
```
df3 = pd.DataFrame(x, columns=["x1", "x2"])
df3
```

	x1	x2
0	0.193135	0.313122
1	-0.474248	-3.646317
2	-0.395663	-4.393158
3	1.589819	-6.137847
4	0.826940	-0.974788
5	-1.364062	-7.528879
6	1.326390	2.416965
7	2.300693	-4.101331
8	-0.403167	1.873754
9	-0.218539	-2.409741

```
plt.scatter(df3["x1"], df3["x2"])
```



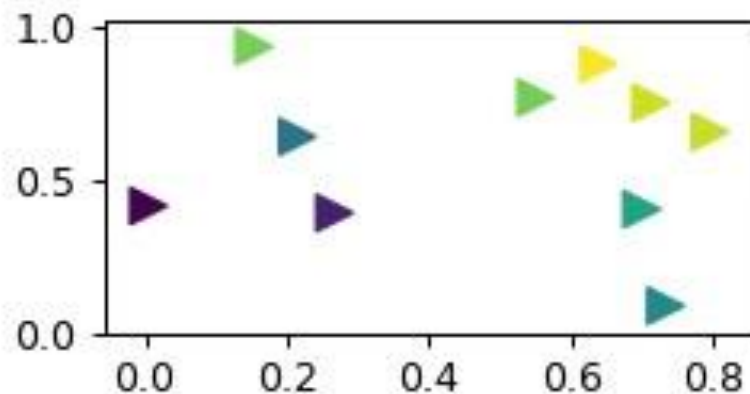
```
plt.scatter(df3["x1"], df3["x2"])
```



scatter 함수 : 모양과 색 입히기

s는 크기, c는 색상, marker는 삼각형

```
x = np.random.rand(10)
y = np.random.rand(10)
z = np.sqrt(x**2+y**2)
plt.subplot(321)
plt.scatter(x,y, s=80, c=z, marker=">")
plt.show()
```



pie 함수

색상 기본 순서 colors=('b', 'g', 'r', 'c', 'm', 'y', 'k', 'w')

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

data = [5,10,30,20,7,8,10,10]

plt.pie(data)
plt.show()
```

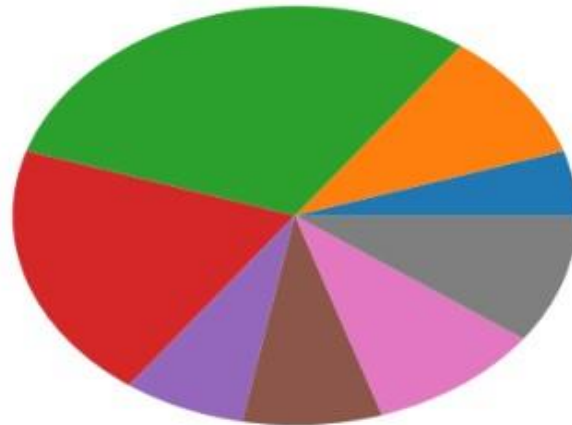


character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

```

%matplotlib nbagg
import numpy as np
import matplotlib.pyplot as plt
data=[5, 10, 30, 20, 7, 8, 10, 10]
plt.pie(data)

```



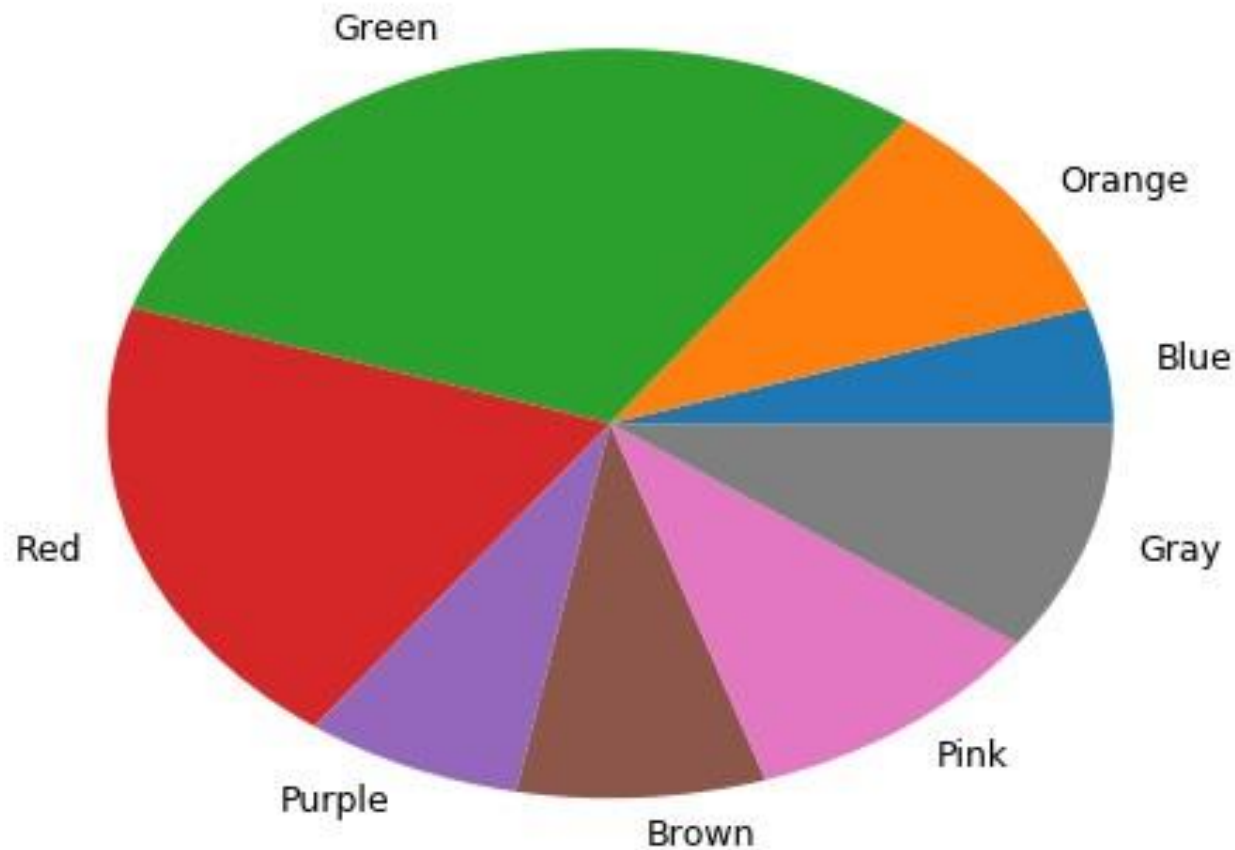
```

([<matplotlib.patches.Wedge at 0x1a99ed1ae10>,
 <matplotlib.patches.Wedge at 0x1a99ed21a90>,
 <matplotlib.patches.Wedge at 0x1a99ed27710>,
 <matplotlib.patches.Wedge at 0x1a99ed71390>,
 <matplotlib.patches.Wedge at 0x1a99ed71fd0>,
 <matplotlib.patches.Wedge at 0x1a99ed78c50>,
 <matplotlib.patches.Wedge at 0x1a99ed818d0>,
 <matplotlib.patches.Wedge at 0x1a99ed89550>],
 [<matplotlib.text.Text at 0x1a99ed215c0>,
 <matplotlib.text.Text at 0x1a99ed27240>,
 <matplotlib.text.Text at 0x1a99ed27e80>,
 <matplotlib.text.Text at 0x1a99ed71b00>,
 <matplotlib.text.Text at 0x1a99ed78780>,
 <matplotlib.text.Text at 0x1a99ed81400>,
 <matplotlib.text.Text at 0x1a99ed81f98>,
 <matplotlib.text.Text at 0x1a99ed89cc0>])

```

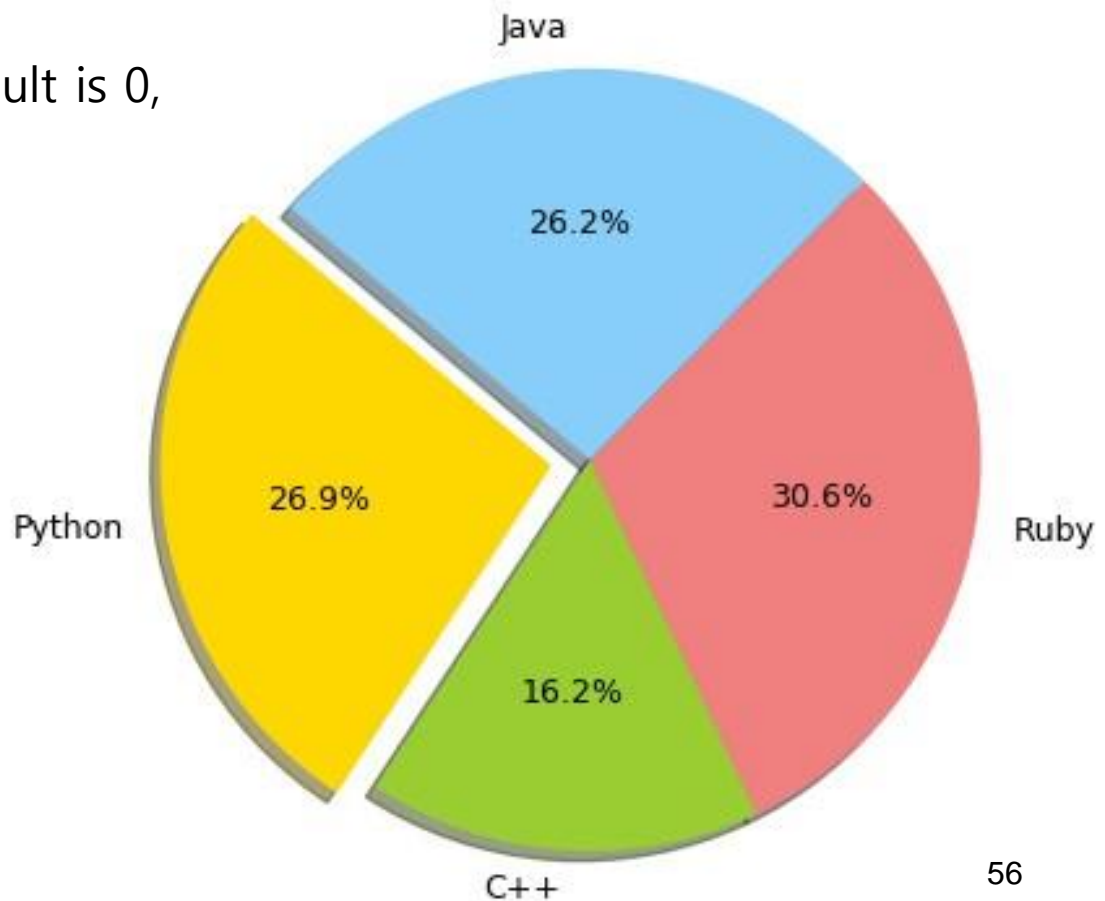


```
data = [5,10,30,20,7,8,10,10]  
label = ['Blue', 'Orange', 'Green', 'Red', 'Purple', 'Brown', 'Pink', 'Gray']  
plt.pie(data, labels=label)  
plt.show()
```



실습

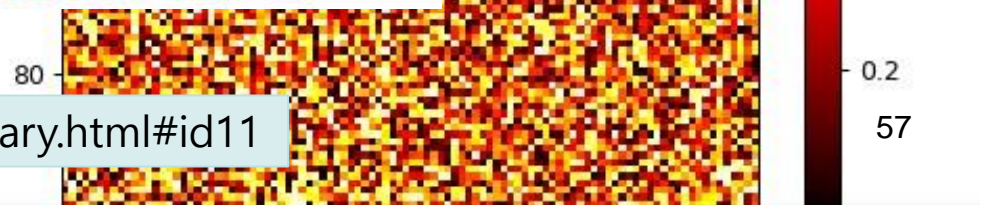
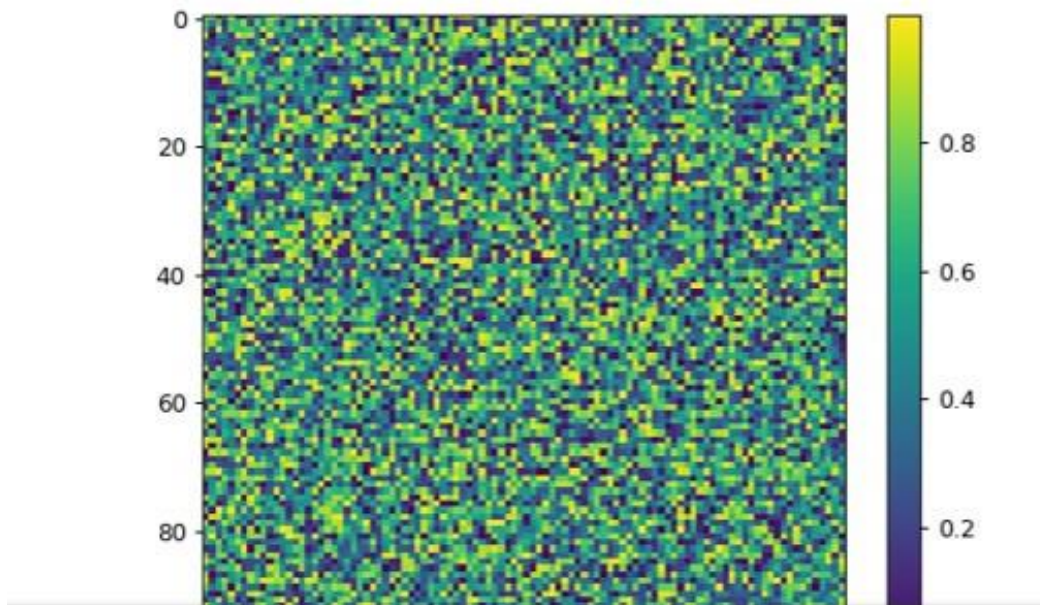
- * slice labels
- * auto-labeling the percentage (autopct)
- * offsetting a slice with "explode"
- * drop-shadow
- * custom start angle : default is 0, counter-clockwise



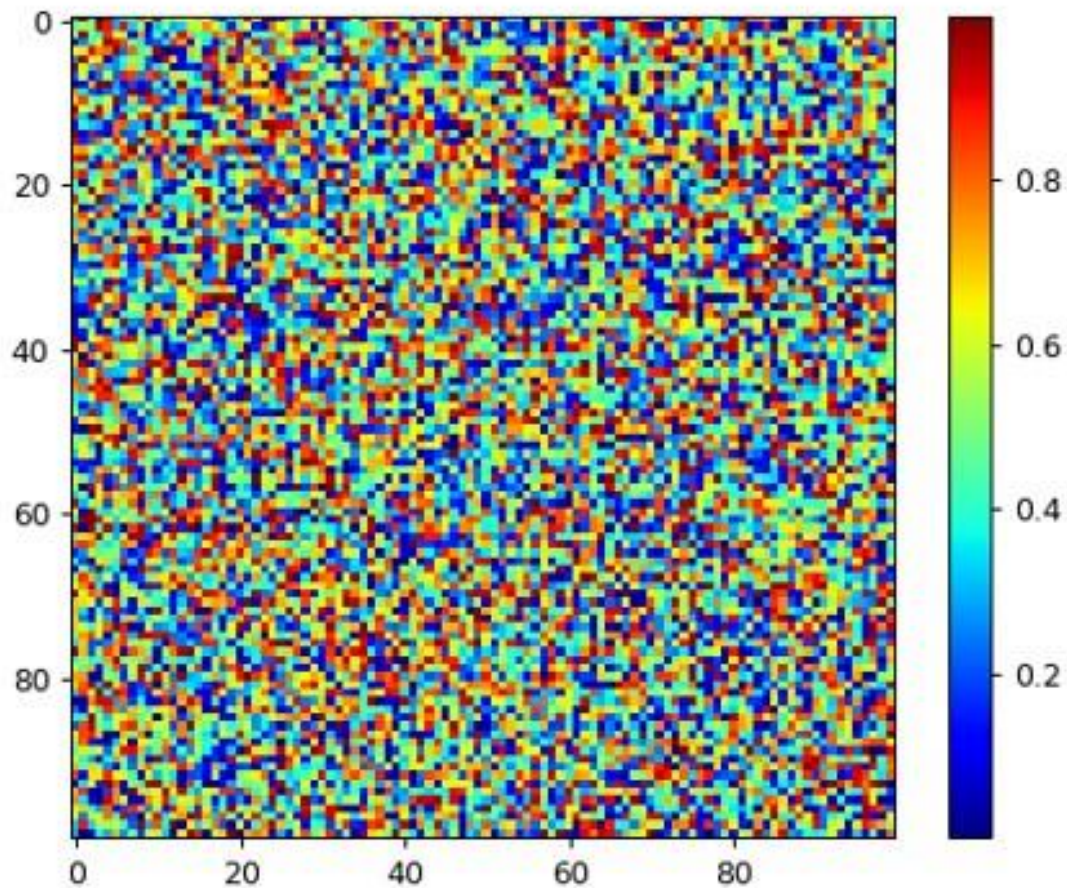
imshow() 함수

- imshow() 함수를 이용해서 이미지 출력
- colorbar 함수를 이용 옆에 colorbar를 출력

```
ao=np.random.random((100,100))
plt.imshow(ao)
plt.colorbar()
```



```
aou=np.random.random((100,100))
plt.imshow(aou)
plt.jet() # a spectral map with dark endpoints, blue-cyan-yellow-red
plt.colorbar()
```




```
aou=np.random.random((100,100))
plt.imshow(aou)
plt.hot() #sequential black-red-yellow-white
plt.colorbar()
```

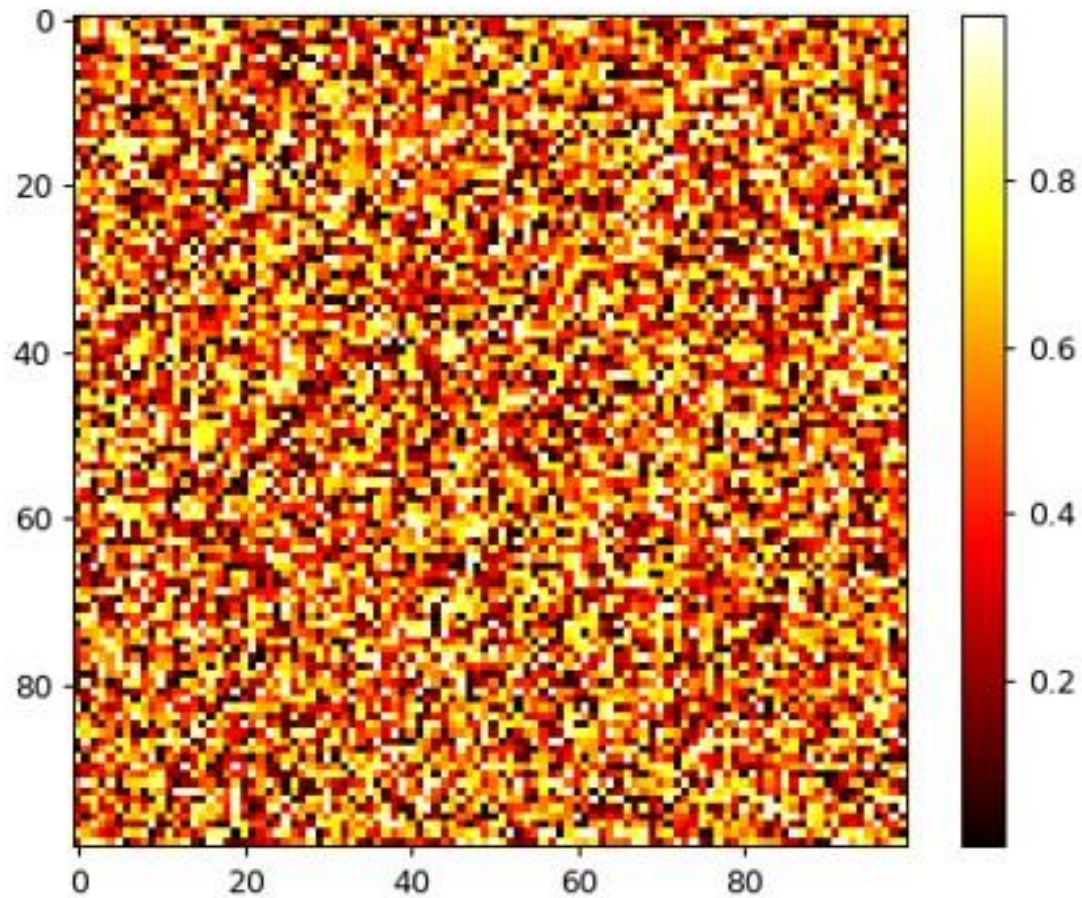


image.read 함수

이미지 파일을 읽고 이를 ndarray로 전환해서 imshow함수로 그래프 출력

```
: import matplotlib.image as mpimg
img = mpimg.imread("민들레꽃.jpg")
print (type(img))
plt.imshow(img)
plt.hot()
plt.colorbar()
plt.show()
```

```
<class 'numpy.ndarray'>
```



```
import matplotlib.image as mg
im=mg.imread("민들레꽃.jpg")
print(type(im))
plt.imshow(im)
```

```
<class 'numpy.ndarray'>
```



실습: 이미지만 출력하기



이미지 처리시 좌표축 제거하기

axis('off')를 이용해서 이미지만 출력

```
import matplotlib.image as mpimg  
img = mpimg.imread("민들레꽃.jpg")  
print (type(img))  
plt.imshow(img)  
plt.axis('off')  
plt.show()
```

```
<class 'numpy.ndarray'>
```



예제: 출생률 데이터

- <https://raw.githubusercontent.com/jakevdp/data-CDCbirths/master/births.csv>
- HTTP로 다운로드 : curl
- 파일을 콘솔로 출력
 - !curl https://raw.githubusercontent.com/jakevdp/data-CDCbirths/master/births.csv
- 지정한 이름의 파일로 출력
 - !curl -o birth.csv https://raw.githubusercontent.com/jakevdp/data-CDCbirths/master/births.csv
- 서버의 filename으로 출력
 - !curl -O birth.csv <https://raw.githubusercontent.com/jakevdp/data-CDCbirths/master/births.csv>
- birth = pd.read_csv('birth.csv')
- birth.head()


```
births = pd.read_csv('https://raw.githubusercontent.com/jakevdp/data-CDCbirths/master/births.csv')
```

```
births.head()
```

	year	month	day	gender	births
0	1969	1	1	F	4046
1	1969	1	1	M	4440
2	1969	1	2	F	4454
3	1969	1	2	M	4548
4	1969	1	3	F	4548

```
births["decade"] = 10 * (births['year'] // 10)
births.pivot_table('births', index='decade', columns='gender', aggfunc='sum')
```

gender	F	M
decade		
1960	1753634	1846572
1970	16263075	17121550
1980	18310351	19243452
1990	19479454	20420553
2000	18229309	19106428

```
%matplotlib nbagg
import seaborn as sns
sns.set()
births.pivot_table('births', index='year', columns='gender', aggfunc='sum').plot()
plt.ylabel('total births per year')
```



```
quartiles=np.percentile(births['births'], [25, 50, 75])  
mu=quartiles[1]  
sig = 0.75*(quartiles[2] - quartiles[0])
```

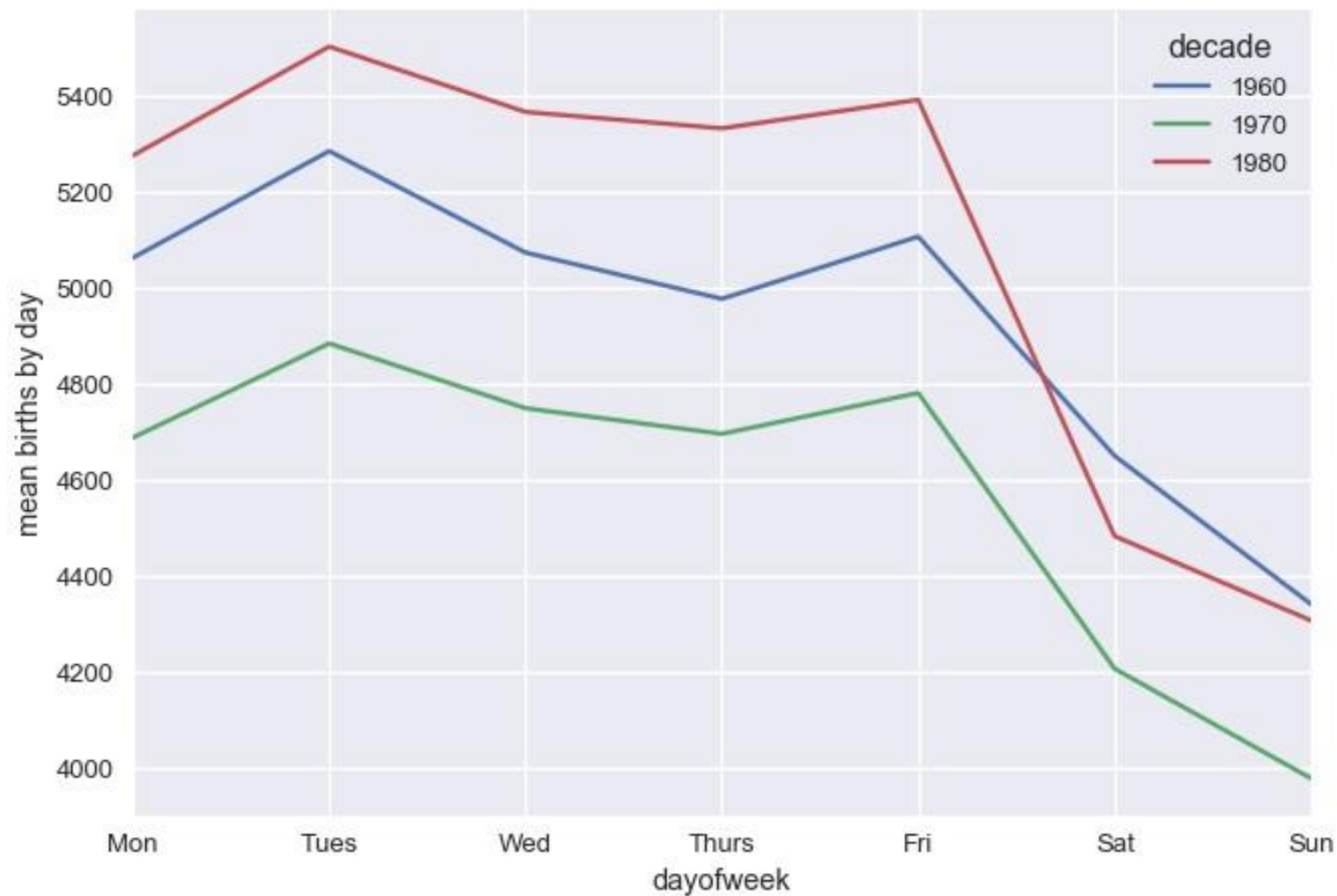
```
births =births.query('(births > @mu - 5 * @sig) & (births < @mu + 5*@sig)')
```

```
births['day']= births['day'].astype(int)
```

```
births.index = pd.to_datetime(10000*births.year + 100*births.month + births.day, format='%Y%m%d')  
births['dayofweek'] = births.index.dayofweek
```

```
births.pivot_table('births', index='dayofweek', columns='decade', aggfunc='mean').plot()  
plt.gca().set_xticklabels(['Mon', 'Tues', "Wed", 'Thurs', 'Fri', 'Sat', 'Sun'])  
plt.ylabel('mean births by day')
```

```
births.pivot_table('births', index='dayofweek', columns='decade', aggfunc='mean').plot()  
plt.gca().set_xticklabels(['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun'])  
plt.ylabel('mean births by day')
```



```
births_by_date = births.pivot_table('births', [births.index.month, births.index.day])
births_by_date.head()
```

```
1  1    4009.225
   2    4247.400
   3    4500.900
   4    4571.350
   5    4603.625
```

Name: births, dtype: float64

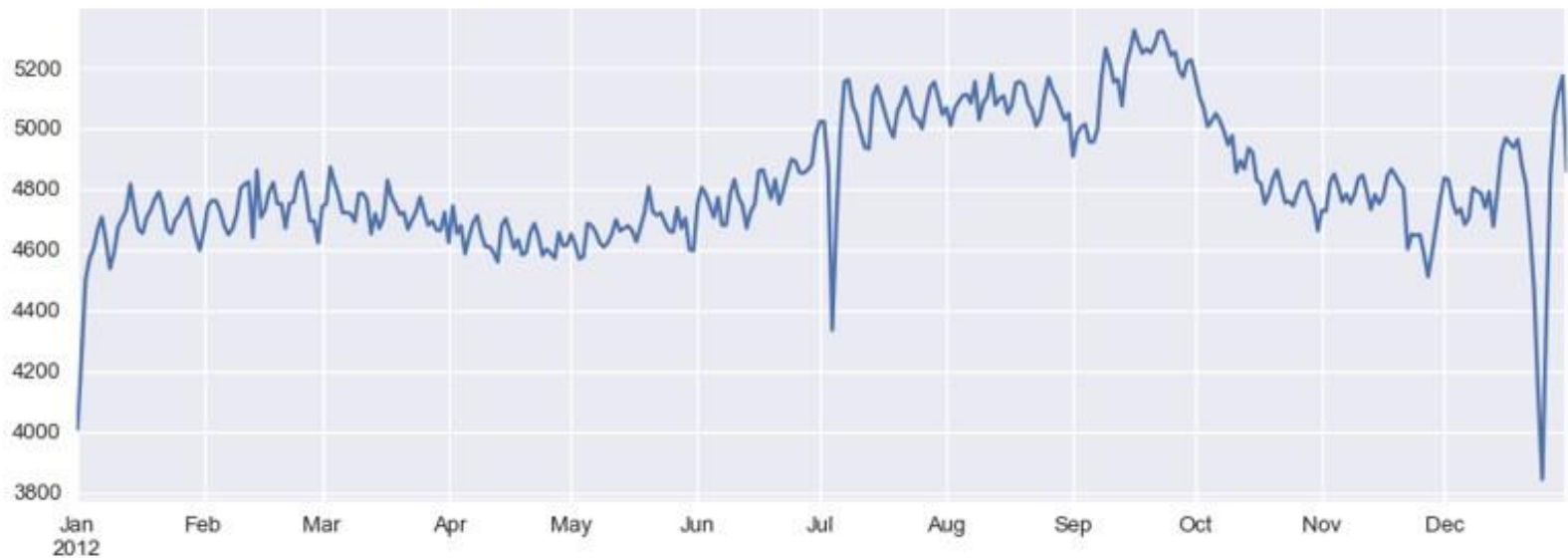
```
births_by_date.index = [pd.datetime(2012, month, day) for (month, day) in births_by_date.index]
births_by_date.head()
```

```
2012-01-01    4009.225
2012-01-02    4247.400
2012-01-03    4500.900
2012-01-04    4571.350
2012-01-05    4603.625
```

Name: births, dtype: float64

```
fig, ax = plt.subplots(figsize=(12,4))
births_by_date.plot()
```

```
fig, ax = plt.subplots(figsize=(12,4))  
births_by_date.plot()
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x277c8efd278>
```