– 제작 : 나

구글 드라이브의 파일을 불러오는 법

```
# 구글드라이브에 파일 업로드하면 colab 상에서 이용가능
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[7] # TDM(Term-Document Matrix): row-vocab / column-file name
    TDM = pd.read_csv('/content/drive/MyDrive/2022_2 Parrot/TDM.txt', sep = '\t', index_col = 0)
```

- 위와 같은 코드를 통해 구글 드라이브에 있는 파일을 불러올 수 있음
- 위의 경우는 MyDrive 안에 2022_2 Parrot 폴더 안에 있는 TDM.txt 파일을 불러오는 것

Jupyter notebook에서 파일을 불러오는 법

```
# jupyter notebook에서 파일 불러오기
TDM=pd.read_csv('/Users/User/Desktop/조정선/SOGANG/Parrot/6기 임원진/Python 강의자료/TDM.txt', sep = '\t', index_col = 0)

#파일의 library는 개인마다 다르다 (심지어 맥/윈도우도 다름) --> google에 물어보면 쉽게 알려줌
```

✓ 0.2s

```
TDM
```

✓ 0.3s

	01.txt	02.txt	03.txt	04.txt
14th	0	0	0	1
15	0	0	0	1
1656	0	0	0	1
1761	0	0	0	1
17th	0	0	0	1
...	...	...	...	...
years	2	0	0	0
york	0	0	1	0
you	4	0	0	0
young	0	1	0	0
your	5	0	0	0

244 rows × 4 columns

TDM.txt

원본 파일은 이렇게 생김

	01.txt	02.txt	03.txt	04.txt
14th	0	0	0	1
15	0	0	0	1
1656	0	0	0	1
1761	0	0	0	1
17th	0	0	0	1
1979	1	0	0	0
1992	1	0	0	0
25	1	0	0	0
a	1	2	2	3
about	0	0	0	2
accurate		0	0	0 1
add	0	0	1	0
admit	0	0	1	0
all	1	0	0	1
allowed	0	0	0	1
already	0	1	0	0
an	0	2	1	0
and	2	2	1	5
appearing		0	0	0 1
are	0	1	2	0
art	0	0	2	0
artist's		0	0	2 0
as	2	2	2	0
aspects	0	0	1	0
astronomer		0	0	0 1
at	0	0	2	2
back	0	0	1	0
be	1	1	2	0
become	0	1	0	0

- 왼쪽과 같이 DataFrame이 출력됨

python™

DataFrame 구조 확인하기

- shape
- head()
- tail()
- info()

```
print(TDM.shape)
```
✓ 0.2s

```
(244, 4)
```

```
TDM.info()
```
✓ 0.2s

```
<class 'pandas.core.frame.DataFrame'>
Index: 244 entries, 14th to your
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   01.txt  244 non-null    int64
 1   02.txt  244 non-null    int64
 2   03.txt  244 non-null    int64
 3   04.txt  244 non-null    int64
dtypes: int64(4)
memory usage: 9.5+ KB
```

```
TDM.head()  # default 5개
```
✓ 0.4s

	01.txt	02.txt	03.txt	04.txt
14th	0	0	0	1
15	0	0	0	1
1656	0	0	0	1
1761	0	0	0	1
17th	0	0	0	1

```
TDM.tail()
```
✓ 0.2s

	01.txt	02.txt	03.txt	04.txt
years	2	0	0	0
york	0	0	1	0
you	4	0	0	0
young	0	1	0	0
your	5	0	0	0

python™

통계 및 연산 함수

- 기본적으로 column(axis=0)을 기준으로 연산되며,
  index를 기준으로 연산하고 싶다면 (axis=1)을 추가해야 함.
- 출력 결과는 Series
- (skipna=True)를 한다면, 결측치들을 제외하고 연산함
- 통계함수: mean(), median(), var(), std(), describe(),
  idxmax(), idxmin() 등등
- 연산함수: sum(), sub(), mul(), div() 등등

각자 TDM에 하나씩 넣어서 확인해 보시길…

기술통계

```
# 기술통계: 수치 척도
# 기본적으로 column기준(axis=0)으로 연산되며, 출력결과는 Series(1차원)
# method(axis = , skipna = True): nan값이 있을 때 제외하고 연산함
TDM.sum()              # TDM.apply(sum)
# TDM.sum(axis = 1)    # TDM.apply(sum, axis = 1)
# TDM.mean()
# TDM.median()
# TDM.var()
# TDM.std(axis=1)
```

```
01.txt      85
02.txt      87
03.txt     126
04.txt     120
dtype: int64
```

describe()

- describe()를 통하여 여러가지 통계 정보들을 한 눈에 볼 수 있다.

- loc 및 iloc 원하는 값만 추출할 수 있다.

```
TDM.describe()
```

✓ 0.3s

	01.txt	02.txt	03.txt	04.txt
count	244.000000	244.000000	244.000000	244.000000
mean	0.348361	0.356557	0.516393	0.491803
std	0.778807	0.647818	1.152800	0.833910
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	1.000000	1.000000	1.000000
max	5.000000	5.000000	11.000000	5.000000

python™

describe()

- describe()를 통하여 여러가지 통계 정보들을 한 눈에 볼 수 있다.

- loc 및 iloc 원하는 값만 추출할 수 있다.

```
TDM.describe().loc['mean']
```

```
01.txt    0.348361
02.txt    0.356557
03.txt    0.516393
04.txt    0.491803
Name: mean, dtype: float64
```

```
TDM.describe().iloc[2]
#TDM.describe().locp['std']
```

```
01.txt    0.778807
02.txt    0.647818
03.txt    1.152800
04.txt    0.833910
Name: std, dtype: float64
```

결측치 확인

- isnull() 함수를 통해 결측치 확인가능
- isnull().sum()을 통해 해당 column에 결측치가 있는지 없는지를 확인할 수 있음

```
TDM.isnull().sum()
```
✓ 0.2s

```
01.txt    0
02.txt    0
03.txt    0
04.txt    0
dtype: int64
```

False의 경우 0으로 계산됨 ➔ 결측치가 없음

```
TDM.isnull()
```
✓ 0.4s

	01.txt	02.txt	03.txt	04.txt
14th	False	False	False	False
15	False	False	False	False
1656	False	False	False	False
1761	False	False	False	False
17th	False	False	False	False
...	...	...	...	...
years	False	False	False	False
york	False	False	False	False
you	False	False	False	False
young	False	False	False	False
your	False	False	False	False

244 rows × 4 columns

DataFrame의 연산

- 연산기호를 사용하는 방법
  (결측치를 처리하지 못함)

- 함수를 사용하는 방법
  (결측치를 처리할 수 있음)

```
b
```

	01.txt	02.txt	03.txt	04.txt
a	1	2	2	3
an	0	2	1	0
the	4	3	7	4
of	2	1	4	1
to	4	5	11	5

```
c
```

	04.txt	02.txt
to	5	5
the	4	3
a	3	2

```
b+c
```

	01.txt	02.txt	03.txt	04.txt
a	NaN	4.0	NaN	6.0
an	NaN	NaN	NaN	NaN
of	NaN	NaN	NaN	NaN
the	NaN	6.0	NaN	8.0
to	NaN	10.0	NaN	10.0

c에는 결측 되어있는 값들은
덧셈이 되지 않고 결측치로 처리된다

DataFrame의 연산

- 연산기호를 사용하는 방법
  (결측치를 처리하지 못함)

- 함수를 사용하는 방법
  (결측치를 처리할 수 있음)

```
b
```

	01.txt	02.txt	03.txt	04.txt
a	1	2	2	3
an	0	2	1	0
the	4	3	7	4
of	2	1	4	1
to	4	5	11	5

```
c
```

	04.txt	02.txt
to	5	5
the	4	3
a	3	2

```
b.add(c)
```

	01.txt	02.txt	03.txt	04.txt
a	NaN	4.0	NaN	6.0
an	NaN	NaN	NaN	NaN
of	NaN	NaN	NaN	NaN
the	NaN	6.0	NaN	8.0
to	NaN	10.0	NaN	10.0

c에는 결측 되어있는 값들은
덧셈이 되지 않고 결측치로 처리된다

DataFrame의 연산

- 연산기호를 사용하는 방법
  (결측치를 처리하지 못함)

- 함수를 사용하는 방법
  (결측치를 처리할 수 있음)

```
b
```

	01.txt	02.txt	03.txt	04.txt
a	1	2	2	3
an	0	2	1	0
the	4	3	7	4
of	2	1	4	1
to	4	5	11	5

```
c
```

	04.txt	02.txt
to	5	5
the	4	3
a	3	2

```
b.add(c, fill_value=0)
```

	01.txt	02.txt	03.txt	04.txt
a	1.0	4.0	2.0	6.0
an	0.0	2.0	1.0	0.0
of	2.0	1.0	4.0	1.0
the	4.0	6.0	7.0	8.0
to	4.0	10.0	11.0	10.0

c에는 결측 되어있는 값들은
덧셈이 되지 않고 결측치로 처리되지만
fill_value를 통하여 결측치들을
지정한 숫자로 채울 수 있다. (여기선 0)

apply

- apply(function, [axis=0])
  axis=0일 경우, column으로 계산
  axis=1일 경우, row로 계산

- applymap(function)
  함수를 DataFrame 전체에 적용

```
#b
#b.apply(sum)              # 각 열의 value들의 합
#b.apply(sum, axis=1)      # 각 행의 value들의 합
b.apply(lambda x: max(x) - min(x))     # 각 열의 최댓값-최솟값
```

```
01.txt     4
02.txt     4
03.txt    10
04.txt     5
dtype: int64
```

```
b.applymap(float)
```

	01.txt	02.txt	03.txt	04.txt
a	1.0	2.0	2.0	3.0
an	0.0	2.0	1.0	0.0
the	4.0	3.0	7.0	4.0
of	2.0	1.0	4.0	1.0
to	4.0	5.0	11.0	5.0

groupby

```
#groupby
fruit = pd.DataFrame({
    'city': ['부산', '부산', '부산', '부산', '서울', '서울', '서울'],
    'fruits': ['apple', 'orange', 'banana', 'banana', 'apple', 'apple', 'banana'],
    'price': [100, 200, 250, 300, 150, 200, 400],
    'quantity': [1, 2, 3, 4, 5, 6, 7]
})
fruit
```

	city	fruits	price	quantity
0	부산	apple	100	1
1	부산	orange	200	2
2	부산	banana	250	3
3	부산	banana	300	4
4	서울	apple	150	5
5	서울	apple	200	6
6	서울	banana	400	7

```
fruit.groupby(['city', 'fruits']).mean()
```

		price	quantity
city	fruits		
부산	apple	100.0	1.0
	banana	275.0	3.5
	orange	200.0	2.0
서울	apple	175.0	5.5
	banana	400.0	7.0

```
fruit.groupby(['fruits','city']).mean()
```

		price	quantity
fruits	city		
apple	부산	100.0	1.0
	서울	175.0	5.5
banana	부산	275.0	3.5
	서울	400.0	7.0
orange	부산	200.0	2.0

stack & unstack

```
df = pd.DataFrame({'col_1':[1,2],
                   'col_2':[3,4]},
                  index={'one','two'})
df
```

	col_1	col_2
two	1	3
one	2	4

```
stacked = df.stack()
stacked
```

```
two  col_1    1
     col_2    3
one  col_1    2
     col_2    4
dtype: int64
```

```
stacked=pd.DataFrame(stacked)
stacked
```

		0
two	col_1	1
	col_2	3
one	col_1	2
	col_2	4

```
stacked.unstack()
```

	0	
	col_1	col_2
two	1	3
one	2	4

```
stacked.unstack(level=0)
```

	0	
	two	one
col_1	1	2
col_2	3	4

stack의 경우 Series로 출력
unstack의 경우 DataFrame

level을 통해서 multi index의 어떤 데이터를 보낼지 선택 가능

stack 을 하고 unstack을 해야만하는건 아니고
그냥 unstack도 할 수 있는 것 같다

python™

heatmap

```
grd_sur = grd_sur[['survived']]
grd_sur
```

		survived
pclass	sex	
1	female	0.968085
	male	0.368852
2	female	0.921053
	male	0.157407
3	female	0.500000
	male	0.135447

```
# heatmap
# stack을 하지 않았을 때
sns.heatmap(grd_sur)
```

```
<AxesSubplot:ylabel='pclass-sex'>
```

python™

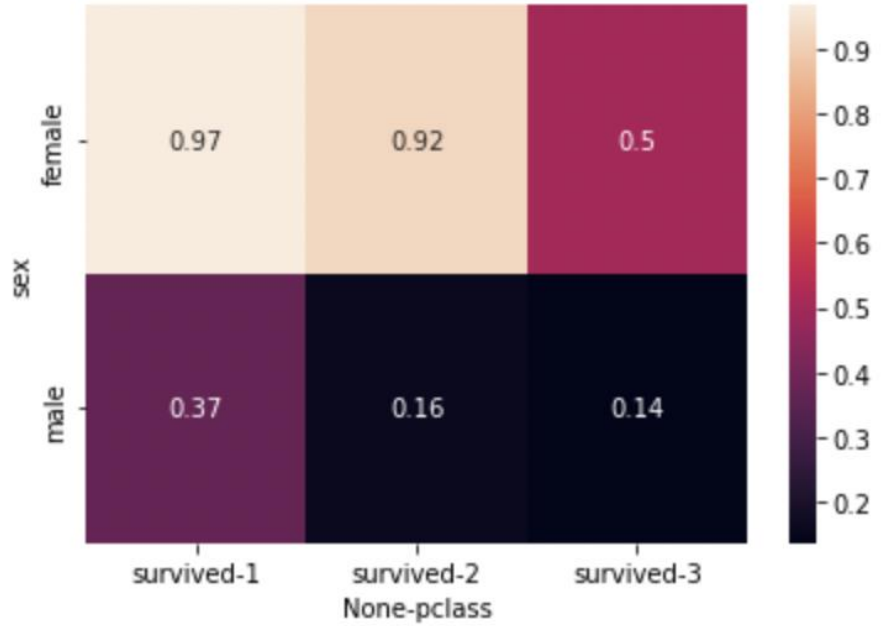
heatmap

```
grd_sur.unstack(0)
```

	survived		
pclass	1	2	3
sex			
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

```
# pclass별 생존율 히트맵 시각화
sns.heatmap(grd_sur.unstack(0), annot=True)
```

```
<AxesSubplot:xlabel='None-pclass', ylabel='sex'>
```



annot = True의 경우 각 칸의 비율을 알려줌

python™

pivot

- 데이터 재구조화
- 데이터 axis를 원하는대로 설정하는 함수

```
df
```

	Date	Typecode	Volumne
0	20220901	A	10
1	20220901	B	100
2	20220901	C	1000
3	20220902	A	20
4	20220902	B	200
5	20220902	C	2000
6	20220903	A	30
7	20220903	B	300
8	20220903	C	3000

```
df_pivoted1 = df.pivot(index='Date', columns='Typecode', values='Volumne')
df_pivoted1
```

Typecode	A	B	C
Date			
20220901	10	100	1000
20220902	20	200	2000
20220903	30	300	3000

index를 'Date'로 설정
columns를 'Typecode'로 설정
values를 'Volumne'으로 설정

```
df_pivoted2=pd.pivot_table(df, index='Date', columns='Typecode', values='Volumne')
df_pivoted2
```

Typecode	A	B	C
Date			
20220901	10	100	1000
20220902	20	200	2000
20220903	30	300	3000

pd.pivot_table과
df.pivot은 같은 함수이다

python™

Example

```
import numpy as np
#np.random.seed(1)
df = pd.DataFrame(np.random.choice([' ', None, 'apple', 'banana', 'orange'],
                                   30, replace = True,
                                   p = [0.05,0.15,0.25,0.25,0.30]).reshape(10,3),
                  index = list('abcdefghij'),
                  columns = list('ABC'))
df
```

	A	B	C
a	orange	banana	None
b	orange	None	banana
c	apple	banana	banana
d	None	apple	apple
e	None	banana	orange
f	banana	banana	apple
g	banana	apple	None
h	banana	banana	banana
i		apple	None
j	apple	orange	apple

```
df.C.value_counts()
# df.C.value_counts().idxmax()
```

```
banana    3
apple     3
orange    1
Name: C, dtype: int64
```

value_counts(): DataFrame의 column에 사용,
DataFrame의 value값이 index, value의 빈도가 value로
출력 빈도는 내림차순

- ramdom함수를 통해 랜덤으로 DataFrame 생성
- None ← 결측치

Q. None은 무엇이고 NaN과는 무엇이 다른가요??

결측치 (None, NaN)

- None :
  존재하지 않음, 비어있음, Null을 뜻함. dtype은 NoneType

- NaN (Not a Number) :
  특정 숫자로 표현할 수 없음 을 뜻함. dtype은 float

다만 pandas가 친절한 것은..
None이 float으로 쓰일 때 자동으로
NaN 값을 넣어서 NaN으로 처리를 한다

```
type(None)
```

```
NoneType
```

```
type(np.nan)
```

```
float
```

```
np.nan+np.nan
```

```
nan
```

```
None+None
```

```
TypeError
```

```
np.isnan(np.nan)
```

```
True
```

```
np.isnan(None)
```

```
TypeError
```

python™

결측치 (None, NaN)

- NaN은 데이터가 비어 있는 것이 아니다. 다만 수치화를 못할 뿐.
  다만 None 같은 경우는 데이터가 아예 비어 있어서 계산을 할 수 없다.
  따라서 이 둘은 서로 완전히 다른 개념

- Pandas는 때에 따라서 None을 자동으로 NaN 변환을 해준다.
  (계산을 편하게 하기 위함)

```
a=pd.DataFrame([np.nan, None, 1, 2])
a
```

	0
0	NaN
1	NaN
2	1.0
3	2.0

```
b=pd.DataFrame([np.nan, None, 'Sangwook', 'Jeongseon'])
b
```

	0
0	NaN
1	None
2	Sangwook
3	Jeongseon

Random Sampling

- 이항분포: np.random.binomial(n,p,size)
- 포아송분포: np.random.poisson(lam, size)
- 정규분포: np.random.normal(mu, sigma, size)
- t분포: np.random.standard_t(df, size)
- 균등분포: np.random,uniform(low, high, size)
- F-분포: np.random.f(dfnum, dfden, size)
- 카이제곱분포: np.random.chisquare(df,size)

시각화

- plot()
  꺾은선 그래프

- hist()
  히스토그램

- matplotlib.pyplot
  시각화 모듈

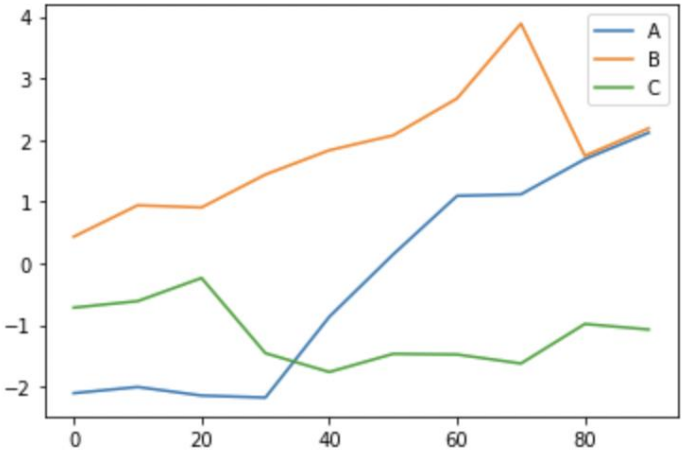
- + plotly (이쁨)

1. 예쁨
2. 드래그해서 크게 볼 수 있음
3. 원하는 데이터만 쉽게 볼 수 있음
4. 아주 유용한 시각화 툴
5. 발표할때도 많이씀

```
# Numpy로 난수를 발생시키고 시각화하기
a = pd.DataFrame(np.random.randn(10,3).cumsum(axis=0),
                 columns = ['A','B','C'], index = np.arange(0,100,10))
print(a)
a.plot()
```

```
           A         B         C
0  -2.104410  0.432340 -0.717898
10 -2.004536  0.941451 -0.612521
20 -2.143078  0.906159 -0.237341
30 -2.175790  1.440604 -1.456530
40 -0.866357  1.834043 -1.761129
50  0.145235  2.075012 -1.466862
60  1.094515  2.673826 -1.477579
70  1.119771  3.887708 -1.623380
80  1.689659  1.744163 -0.982494
90  2.118683  2.189206 -1.071973

<AxesSubplot:>
```



python™

시각화

- 분석한 데이터를 이해하기 쉽도록 그림이나 그래프로 나타낸 것

- Pandas만으로도 시각화가 가능하다
  그러나 일반적으로 Matplotlib를 이용하여 시각화를 많이 함

- 한글화가 안되어 있기 때문에 따로 한글패치를 해줘야 함
  (실습파일에 있으니 확인해보세요!)

그릴 수 있는 그래프의 종류

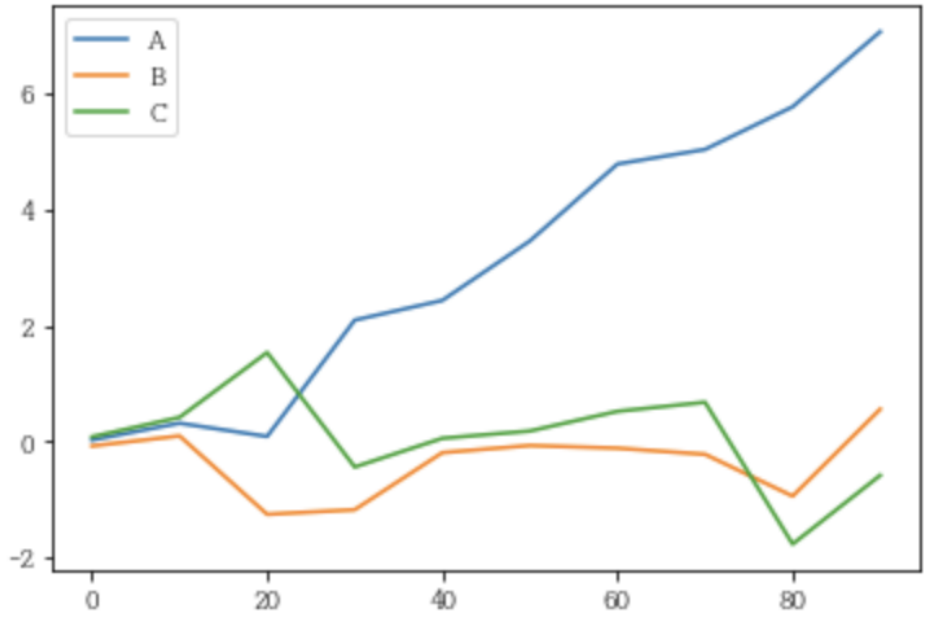
```
import cufflinks as cf
cf.help()
```

```
Use 'cufflinks.help(figure)' to see the list of available parameters for the given figure.
Use 'DataFrame.iplot(kind=figure)' to plot the respective figure
Figures:
        bar
        box
        bubble
        bubble3d
        candle
        choroplet
        distplot
        heatmap
        histogram
        ohlc
        pie
        ratio
        scatter
        scatter3d
        scattergeo
        spread
        surface
        violin
```

Plot

```
# Numpy로 난수를 발생시키고 시각화하기
a = pd.DataFrame(np.random.randn(10,3).cumsum(axis=0),
                 columns = ['A','B','C'], index = np.arange(0,100,10))
a.plot()
```

```
<AxesSubplot:>
```

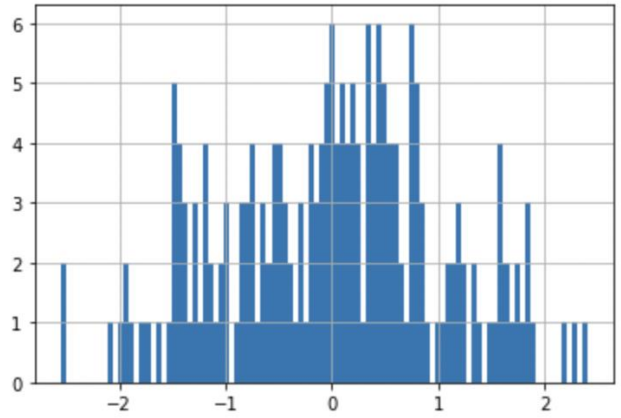


python™

Histogram

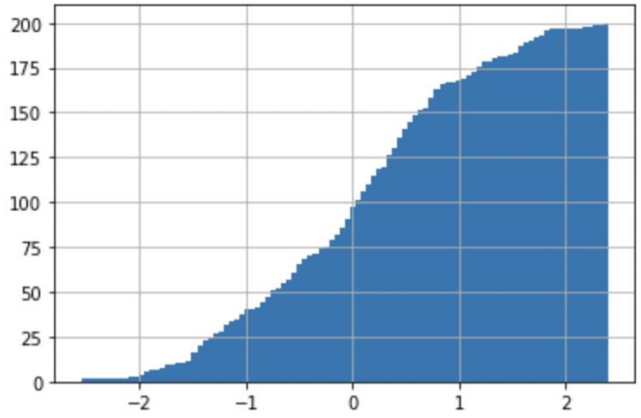
```
b.hist(bins=100)     # bins = 막대의 갯수
```

```
<AxesSubplot:>
```



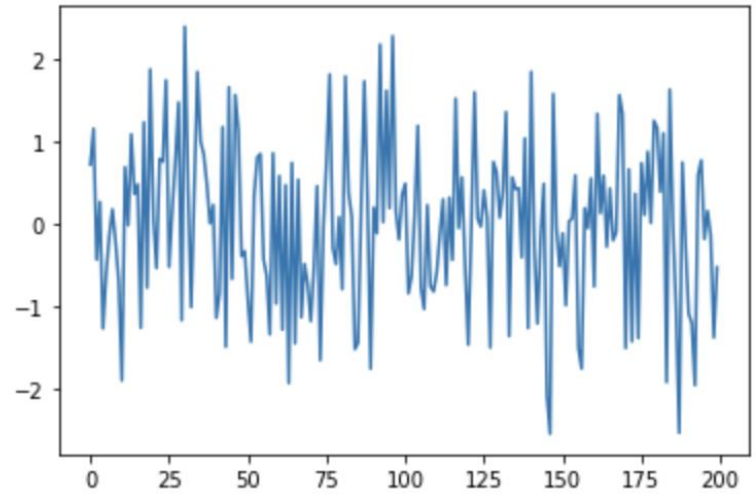
```
b.hist(bins=100, cumulative = True)     # cumulative는 누적치
```

```
<AxesSubplot:>
```



```
# plot()
b = pd.Series(np.random.normal(0,1, size = 200))     # 정규분포
b.plot()
```

```
<AxesSubplot:>
```



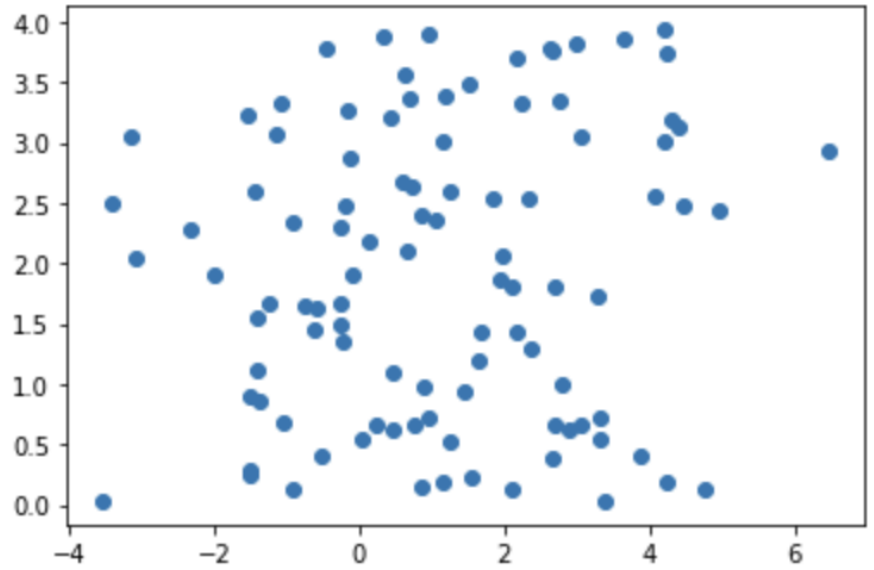
plot과의 비교.. 딱봐도 보이죠?

python™

Scatter

```
# scatter
x = np.concatenate([np.random.normal(1,2, size = (100,1)), np.random.uniform(0,4, size = (100,1))], axis =1)
scat = pd.DataFrame(x, columns=['x1','x2'])
# scat.head()
plt.scatter(scat['x1'], scat['x2'])
```

```
<matplotlib.collections.PathCollection at 0x7fe540f23fd0>
```



python™

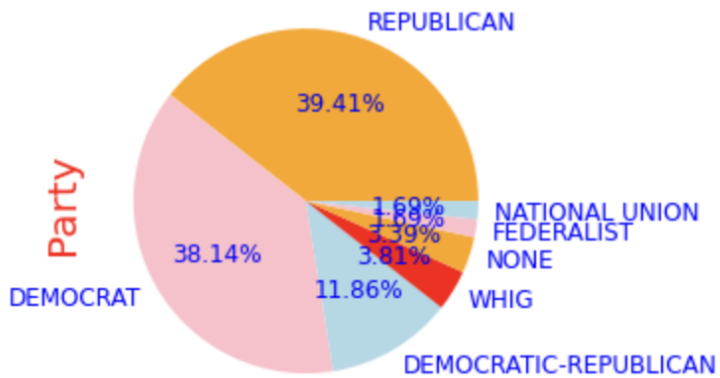
역대 미국 대통령 데이터/ Pie

```
Presidents
```

Years	President	Party
1789	George Washington	None
1790	George Washington	None
1791	George Washington	None
1792	George Washington	None
1793	George Washington	None
...	...	...
2018	Donald Trump	Republican
2019	Donald Trump	Republican
2020	Donald Trump	Republican
2021	Joe Biden	Democrat
2022	Joe Biden	Democrat

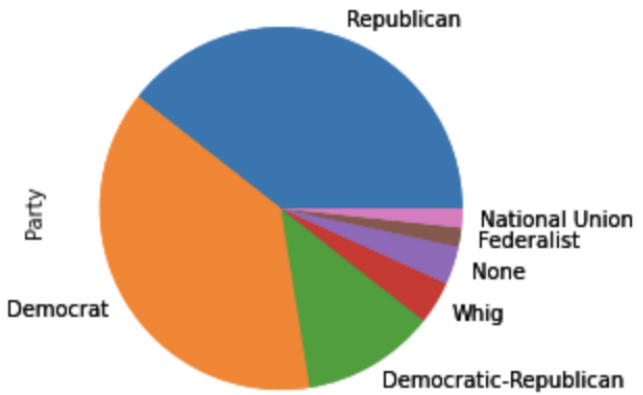
```
party.plot.pie(textprops={'fontsize':12, 'color':'blue'},
               colors= ['orange','pink','lightblue','red'],
               labels = [i.upper() for i in party.index], autopct = '%0.2f%%')
plt.ylabel(party.name, fontsize = 20, color = 'red')
```

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




```
party.plot(kind='pie')
party.plot.pie()
```

```
<AxesSubplot:ylabel='Party'>
```

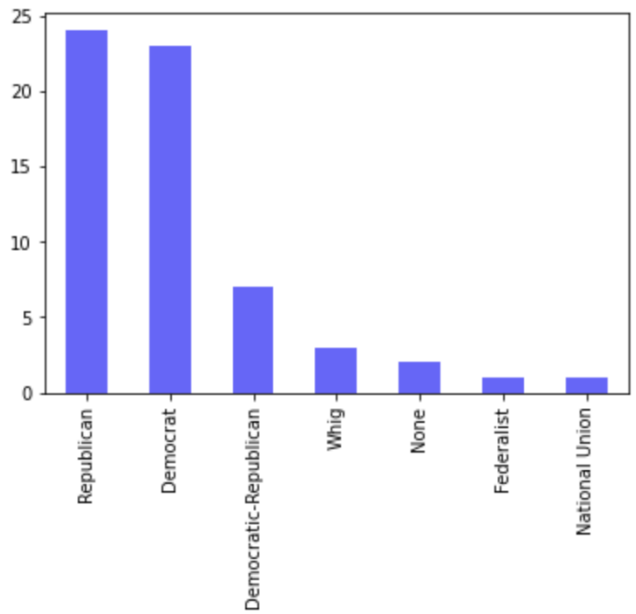


python™

역대 미국 대통령 데이터/ Bar

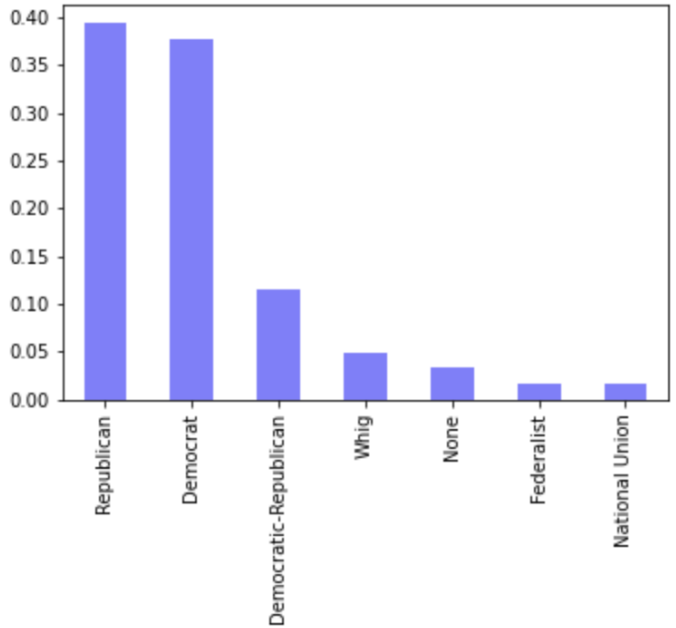
```
party.plot(kind='bar', color = 'blue', alpha = 0.6)     # alpha: 투명도
#party.plot.bar(color = 'blue', alpha = 0.6)
```

```
<AxesSubplot:>
```




```
# Series party의 값을 상대빈도로 변환
party.apply(lambda x: x/party.sum())
party.apply(lambda x: x/party.sum()).plot.bar(color='blue', alpha = 0.5)
```

```
<AxesSubplot:>
```



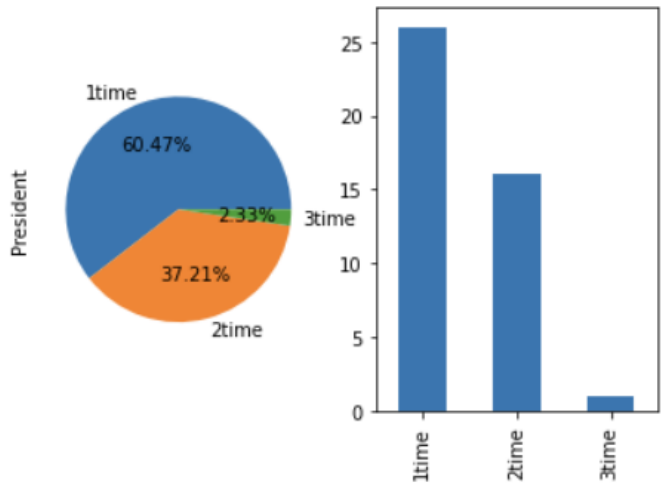
python™

역대 미국 대통령 데이터

- 동일 인명 대통령의 취임횟수(1회/2회/3회)를 산출하여 시각화

```
a = df['President'].value_counts().value_counts()
plt.subplot(1,2,1)
a.plot.pie(labels = [str(i) + 'time' for i in a.index],
           autopct = '%0.2f%%')
plt.subplot(1,2,2)
a.plot.bar()
plt.xticks(range(3), [str(i) + 'time' for i in a.index])
```

```
([<matplotlib.axis.XTick at 0x16f69030d90>,
  <matplotlib.axis.XTick at 0x16f69030d60>,
  <matplotlib.axis.XTick at 0x16f690530a0>],
 [Text(0, 0, '1time'), Text(1, 0, '2time'), Text(2, 0, '3time')])
```



a = 대통령 취임 횟수

plt.subplot : 그래프가 그려지는 위치
때에 따라 축을 공유하도록 그래프 위치를 배치할 수 있다

autopct : 차지하는 비중

xticks : x축 눈금

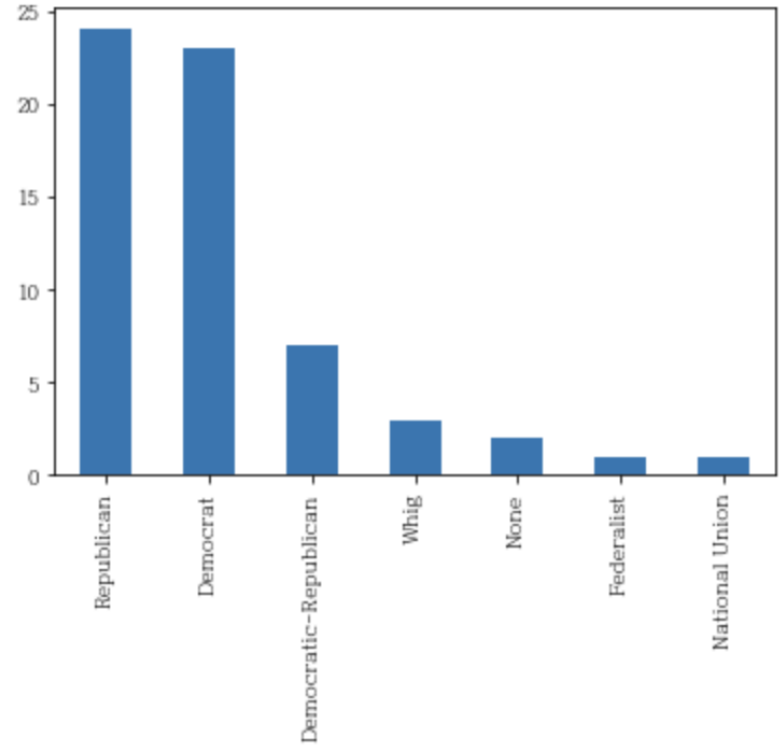
python™

역대 미국 대통령 데이터

- 정당 별 plot 만들기

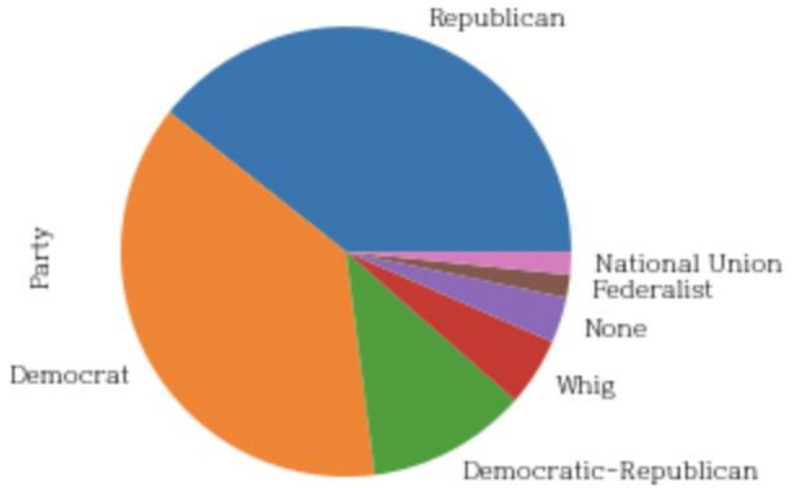
```
# 2-1. 정당별 bar plot 만들기
df.Party.value_counts().plot.bar()
```

```
<AxesSubplot:>
```




```
# 2-2. 정당별 pie chart 만들기
df.Party.value_counts().plot.pie()
```

```
<AxesSubplot:ylabel='Party'>
```



python™

역대 미국 대통령 데이터

- 정당별 역대 미국 대통령 동일 인물 취임 횟수

```
# 4. 상위 계층: Party, 하위 계층: 취임횟수 DataFrame
b = df.groupby(['Party', 'President']).size().groupby('Party').value_counts()
pd.DataFrame(b)
```

		0
Party		
Democrat	**1**	8
	2	6
	3	1
ETC	**1**	6
	2	4
Republican	**1**	12
	2	6

```
# unstack: 이원분할 빈도 교차표
c = b.unstack().fillna(0).astype('int')
c
```

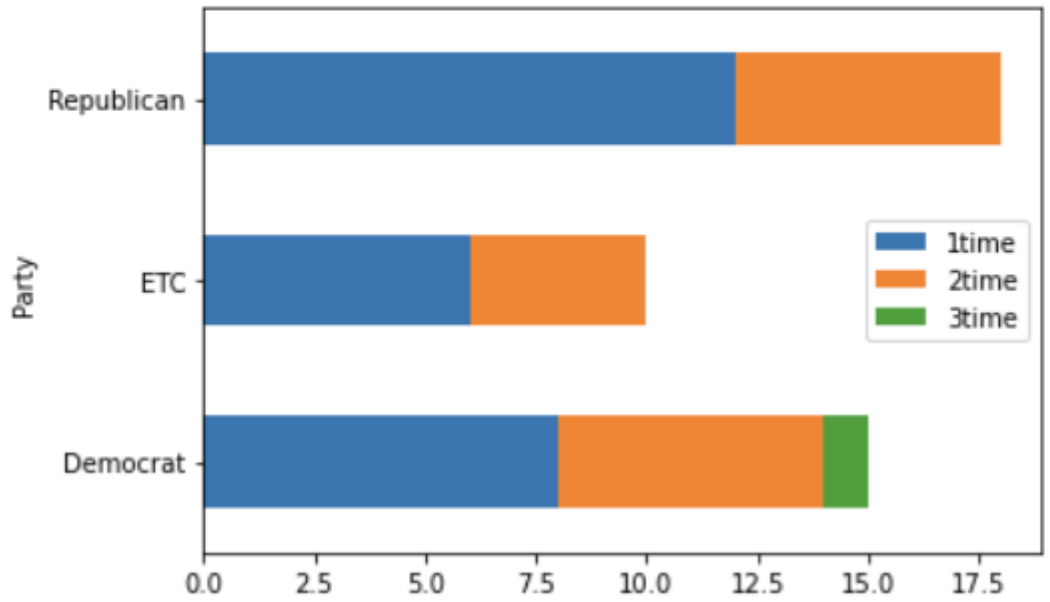
	1	**2**	**3**
Party			
Democrat	8	6	1
ETC	6	4	0
Republican	12	6	0

역대 미국 대통령 데이터

- 정당별 역대 미국 대통령 동일 인물 취임 횟수

```
c.columns = [str(i) + 'time' for i in c.columns]
c.plot.barh(stacked = True)
```

```
<AxesSubplot:ylabel='Party'>
```



python™

– 제작 : 나