

파이썬 라이브러리 중 가장 널리 사용되는 것을 말하자면 'numpy'와 'pandas'라고 할 수 있다. 'pandas'라는 패키지는 excel에서 하던 작업을 아주 쉽게 처리하기 위한 것이라고 생각하면 이해하기 쉽다. 기본적인 사항들은 chatGPT나 온라인에서 많은 자료가 있으니 참고하길 바란다. 아래 문제들을 해결해 가면서 pandas가 하는 일에 대해서 이해해 보도록 하자.

문제1. 아래 데이터를 입력시켜서 AAA칼럼의 원소가 5이하 이면, 'Yes', 아니면 'No'를 입력하는 새로운 칼럼 'TF'를 생성해 보아라.

	AAA	BBB	CCC
0	4	10	100
1	5	20	50
2	6	30	-30
3	7	40	-50

먼저 위 자료를 'pandas'의 데이터프레임(DataFrame) 유형으로 만들어 보자. 아래 자료에서 맨 왼쪽에 표시되는 0,1,2,3은 행을 지칭하는 인덱스(index)라고 되어 있으며, 이 인덱스는 다른 값으로 변경할 수도 있다.

```
import pandas as pd # pandas를 pd라는 별칭으로 불러온다.
df=pd.DataFrame(data={'AAA':[4,5,6,7],
                      'BBB':[10,20,30,40],
                      'CCC':[100,50,-30,-50]})
df
```

	AAA	BBB	CCC
0	4	10	100
1	5	20	50
2	6	30	-30
3	7	40	-50

DataFrame()안에서 사용된 data에는 사전형(dictionary)으로 정의된 데이터가 들어가 있는데, 이는 아래와 같이 자료 따로, 칼럼 이름 따로 지정할 수도 있다.

```
# 데이터를 rows로 따로 입력하여 만든다...
D=[[4,10,100],[5,20,50],[6,30,-30],[7,40,-50]]
# columns 인자로 데이터 할당명을 따로 입력해도 됨
df=pd.DataFrame(D,columns=['AAA','BBB','CCC']) #data=D라고 해도 상관없다.
df
```

	AAA	BBB	CCC
0	4	10	100
1	5	20	50
2	6	30	-30
3	7	40	-50

```
# 데이터를 rows로 따로 입력하여 만든다...
D=[[4,10,100],[5,20,50],[6,30,-30],[7,40,-50]]
df=pd.DataFrame(D)
df.columns=['AAA','BBB','CCC']
df
```

	AAA	BBB	CCC
0	4	10	100
1	5	20	50
2	6	30	-30
3	7	40	-50

먼저 $x \leq 5$ 이면 'Yes' 아니면 'No'를 출력하는 사용자 정의함수를 작성해 보자.

```
# 새로운 함수 작성
import numpy as np
def df_hst(x):
    if x<=5:
        return 'Yes'
    else:
        return 'No'
df_hst(10)

'No'
```

데이터프레임에서 apply()라는 메서드를 사용하면 특정 칼럼에 위에서 만든 함수를 이용해 판단을 할 수 있다.

```
# 'TF'라는 새로운 칼럼을 만든다...
# apply(Lambda x: 함수 정의 사용)
df['TF']=df.AAA.apply(lambda x:df_hst(x))
df
```

	AAA	BBB	CCC	TF
0	4	10	100	Yes
1	5	20	50	Yes
2	6	30	-30	No
3	7	40	-50	No

이 함수 안에서 사용된 lambda는 사용자 정의함수를 만드는 또다른 방법 중에 하나이니 기억해 두자.

```
f=lambda x: 2*x
f(2)

4
```

이 문제의 경우 'numpy.where()'라는 내장함수를 이용해도 된다.

```
import numpy as np
# TF1이라는 새로운 칼럼을 추가로 만든다...
# where(조건, 참실행, 거짓실행)
df['TF1']=np.where(df.AAA<=5, 'Yes', 'No')
df
```

	AAA	BBB	CCC	TF	TF1
0	4	10	100	Yes	Yes
1	5	20	50	Yes	Yes
2	6	30	-30	No	No
3	7	40	-50	No	No

참고로, 만일 위 데이터에서 AAA칼럼의 데이터가 5이하인 행들만 추출하려면 아래와 같이 하면 된다.

```
# loc[] 라는 메서드를 사용하여..
TF=df.AAA<=5 # 조건 생성
TF
```

	0	1	2	3
TF	True	True	False	False

Name: AAA, dtype: bool

```
# 조건에 맞는 행만 추출
df.loc[TF]
```

	AAA	BBB	CCC	TF	TF1
0	4	10	100	Yes	Yes
1	5	20	50	Yes	Yes

```
# 조건에 맞지 않는 행만 추출
df.loc[~TF]
```

	AAA	BBB	CCC	TF	TF1
2	6	30	-30	No	No
3	7	40	-50	No	No

여기서 'loc[]'라는 메서드는 행이나 열의 이름으로 위치를 찾는데 사용된다. 위 예제에서 보면 df.loc[1,'AAA']라고 하면 인덱스 이름이 1인 행, 칼럼의 이름이 'AAA'인 원소 즉, 5를 지칭하게 된다.

```
df.loc[1,'AAA']
```

5

'loc[]' 메서드와 유사하게 'iloc[]'라는 메서드도 있는데, 'iloc[]'는 숫자로 인덱싱을 하는데 차이가 있다.

```
# iloc는 인덱싱을 하여 각 자료의 위치를 찾을 때 유용하다..
df.iloc[0,0] # 첫번째 행, 첫번째 열을 의미
```

4

```
df.iloc[1,:] # 두번째 행을 추출한다..
```

	AAA	BBB	CCC	TF	TF1
1	5	20	50	Yes	Yes

Name: 1, dtype: object

'at'이라는 메서드를 사용하면, 인덱스의 이름과 열의 이름을 바탕으로 특정 원소를 호출하는 방법도 가능하다.

```
# 인덱스 값이 2, 열의 값이 'BBB'인 공유
df.at[2,'BBB']
```

30

문제2. 아래와 같은 데이터프레임에서 column 이름이 'A'로 시작하는 column들을 모두 추출하여라.

	A	A1	C	DA
0	1	4	7	10
1	2	5	8	11
2	3	6	9	12

데이터프레임의 'filter' 메서드를 사용하면 열의 이름을 바탕으로 다양한 작업을 할 수 있다. 먼저 열 이름에 'A'가 들어가는 열들을 모두 출력하려면 아래와 같이 하면 된다.

```
import pandas as pd

data = {'A': [1, 2, 3],
        'A1': [4, 5, 6],
        'C': [7, 8, 9],
        'DA': [10, 11, 12]}
df = pd.DataFrame(data)
df
```

	A	A1	DA
0	1	4	10
1	2	5	11
2	3	6	12

```
# 열 이름에 A를 포함하는 열 추출
df.filter(like='A')

# 인덱스 값이 2인 행에서
# 열 이름에 A를 포함하는 열 추출
df.loc[2].filter(like='A')
```

	A
2	3

Name: 2, dtype: int64

'A'로 시작하는 열들을 선택하기 위해서는 정규식 표현¹을 사용해야 하는데, 매개변수에서 'regex'로 지정할 수 있도록 되어 있다.

```
# 열 이름이 A로 시작하는 열 추출
df.filter(regex='^A')
```

	A	A1
0	1	4
1	2	5
2	3	6

```
# 인덱스 값이 1인 행에서
# 열 이름이 A로 시작하는 열 추출
df.loc[1].filter(regex='^A')
```

	A
1	2

Name: 1, dtype: int64

문제3. '성별' 칼럼에는 '남자','여자'가 랜덤하게 1000개, '생사여부' 칼럼에는 0(dead),1(alive) 숫자가 랜덤하게 1000개가 만들어지도록 하여 이를 데이터프레임으로 만들어라. 성별에 따른 생사여부의 숫자를 세어서 표시해 보아라.

먼저 'random'이라는 패키지와 'numpy' 패키지를 사용해서 필요한 개수만큼 랜덤하게 생성시킨다.

```
# 중복해서 랜덤추출할 때 사용하는 패키지 사용
import random as rd
import numpy as np
Sex = ['남자', '여자']
# choice(N,k=1000)은 N에서 중복을 허락하여 1000개를 추출함
D1 = rd.choices(Sex, k=1000)
# 0,1 중 1000개의 난수 생성
D2 = np.random.randint(2, size=1000)
# 데이터프레임으로 만들기
df = pd.DataFrame(data={'성별': D1, '생사여부': D2})
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
#   Column    Non-Null Count  Dtype
---  -
0   성별      1000 non-null   object
1   생사여부  1000 non-null   int32
dtypes: int32(1), object(1)
memory usage: 11.8+ KB
```

이 문제는 데이터프레임에서 제공하는 'pivot_table()'이라는 메서드를 이용하면 편리하다.

¹ Regular expression으로 're' 라는 표준 라이브러리에 자세한 사용법이 들어 있다.

```
# pivot_table() 사용하기
# index에 따른 values의 값들을 모아서 aggfunc에 설정된 함수대로 처리해준다.
# aggfunc를 따로 지정하지 않으면 디폴트 값으로 평균값으로 출력시킨다.
df.pivot_table(values='생사여부', index='성별', aggfunc=np.sum)
```

생사여부	
성별	
남자	262
여자	224

데이터프레임에는 'pivot_table()'과 유사한 'groupby()'라는 메서드도 있으니 참고하자.

```
# groupby 사용하기
# groupby([A,B,C])[D].f: A,B,C에 대해 D를 f에 의해 정렬하라.
df.groupby(['성별', '생사여부'])['생사여부'].count()
```

성별	생사여부	
남자	0	246
	1	262
여자	0	268
	1	224

Name: 생사여부, dtype: int64

문제4. '그룹'이라는 칼럼을 만들어 A,B,C 세가지 중 하나를 무작위로 1000개 만들고, '실험1' 칼럼에는 10부터 50까지의 자연수의 난수 1000개, '실험2' 칼럼에는 100부터 300까지의 자연수 난수 1000개를 생성하자. A,B,C에 따른 실험1, 실험2의 최대, 최소값 등 기본적 통계량을 추출해 보자.

```
import random as rd # rd라는 별명으로 불러올
import numpy as np
Catgry=['A','B','C']
# choice(N,k=1000)은 N에서 중복을 허락하여 1000개를 추출할
D1=rd.choices(Catgry,k=1000)

# 10~50까지 1000개의 난수 생성
D2=np.random.randint(10,51,size=1000)

# 100~300까지 1000개의 난수 생성
D3=np.random.randint(100,301,size=1000)

# 데이터프레임으로 만들기
df=pd.DataFrame(data={'그룹':D1, '실험1':D2, '실험2':D3})
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 3 columns):
#   Column    Non-Null Count  Dtype
---  ---
0   그룹      1000 non-null   object
1   실험1     1000 non-null   int32
2   실험2     1000 non-null   int32
dtypes: int32(2), object(1)
memory usage: 15.8+ KB
```

먼저 그룹에 따라 실험1과 실험2의 최대값 최소값을 추출해 보자.

```
# '그룹'에 따른 최대값, 최소값 구하기
grouped_df.groupby('그룹').# '그룹' 항목을 중심으로
grouped.min().# 최소값 구하기
```

실험1 실험2		
그룹		
A	10	100
B	10	100
C	10	101

```
grouped.max().# 최대값 구하기
```

실험1 실험2		
그룹		
A	50	299
B	50	299
C	50	300

A,B,C에 따라 실험1의 평균(mean), 표준편차(std), 최소값(min), 최대값(max) 등을 살펴보려면 describe() 라는 메서드를 사용한다.

<pre># 실험1에 대해서 A,B,C에 대해 기본통계량 추출하기 grouped.describe()['실험1']</pre>								
	count	mean	std	min	25%	50%	75%	max
그룹								
A	322.0	29.813665	11.452845	10.0	20.0	30.0	40.0	50.0
B	338.0	29.286982	11.986041	10.0	19.0	29.0	40.0	50.0
C	340.0	29.673529	12.445369	10.0	18.0	28.5	40.0	50.0

'pivot_table()'의 내부 매개변수 'aggfunc'를 이용해서도 기본적인 값들을 추출할 수 있다.

<pre># pivot_table 이용하기 df.pivot_table(values=['실험1','실험2'],index='그룹',aggfunc=['min','max','count','mean'])</pre>							
	min		max		count		mean
	실험1	실험2	실험1	실험2	실험1	실험2	실험1
그룹							
A	10	100	50	299	322	322	29.813665
B	10	100	50	299	338	338	29.286982
C	10	101	50	300	340	340	29.673529

문제5. 아래와 같은 자료를 작성하여 DataFrame을 만들고, 짝수칼럼(c2,c4,c6,c8,c10)만의 부분합을 구하여라. 유사하게 홀수행들(r1,r3,r5,r7,r9)의 부분합을 구하여라.

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
r1	0	1	2	3	4	5	6	7	8	9
r2	10	11	12	13	14	15	16	17	18	19
r3	20	21	22	23	24	25	26	27	28	29
r4	30	31	32	33	34	35	36	37	38	39
r5	40	41	42	43	44	45	46	47	48	49
r6	50	51	52	53	54	55	56	57	58	59
r7	60	61	62	63	64	65	66	67	68	69
r8	70	71	72	73	74	75	76	77	78	79
r9	80	81	82	83	84	85	86	87	88	89
r10	90	91	92	93	94	95	96	97	98	99

특정한 포맷을 가진 데이터를 생성할 때에도 'print()' 문에서 썼던 f' ' 형식을 이용하면 손쉽게 만들어

낼 수 있고, 'numpy'의 'reshape()' 메서드로 array형 데이터의 모양을 원하는 형태로 변형하는데 많이 쓰이고 있다.

```
C=[f'c{i}' for i in range(1,11)] # c1, ..., c10 생성
R=[f'r{i}' for i in range(1,11)] # r1, ..., r10 생성
D=np.array([i for i in range(100)]) # 0,1,2,...,99 생성
D=D.reshape(10,10) # 10 by 10으로 재배치 시킴
df=pd.DataFrame(data=D,columns=C,index=R)
df
```

일부 칼럼의 부분합을 구하기 위해서 먼저 부분합을 하는 칼럼들을 하나의 칼럼으로 인식하도록 사전형 자료를 만들어 주고, 이를 바탕으로 연산하면 된다.

```
# 칼럼 합 구하기
# f문을 이용해 dictionary 형태 자료 생성
# {'c2': 'C_Grp', 'c4': 'C_Grp', 'c6': 'C_Grp', 'c8': 'C_Grp', 'c10': 'C_Grp'}
dic={f'c{2*i}': 'C_Grp' for i in range(1,6)}
df.groupby(dic,axis=1).sum() # axis=0은 행 방향, axis=1은 칼럼방향이 됨
```

	C_Grp
r1	25
r2	75
r3	125
r4	175
r5	225
r6	275
r7	325
r8	375
r9	425
r10	475

행들의 부분합도 같은 방식으로 하면 된다.

```
# row들에 대한 부분합 구하기
# f문을 이용해 dictionary 형태 자료 생성
# {'r1': 'R_Grp', 'r3': 'R_Grp', 'r5': 'R_Grp', 'r7': 'R_Grp', 'r9': 'R_Grp'}
dic={f'r{2*i-1}': 'R_Grp' for i in range(1,6)}
df.groupby(dic,axis=0).sum() # axis=0은 행 방향도 된다
```

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
R_Grp	200	205	210	215	220	225	230	235	240	245

또다른 방식으로 아래와 같이 사용자 정의 함수를 만들어서 인덱스들을 그룹핑하여 이들의 합을 구하는 것도 가능하다.


```
# 사용자 정의 함수를 만들어서..
R_sum=[f'r{2*i-1}' for i in range(1,6)] # r1,r3,r5,r7,r9
def row_sum(x):
    if x in R_sum: # x가 R_sum에 속한다면
        return 'R_Grp'

df.groupby(row_sum,axis=0).sum()
```

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
R_Grp	200	205	210	215	220	225	230	235	240	245

문제6. 웹페이지 kaggle.com에 접속하여 [Titanic 데이터](#)를 다운받아서, 압축을 풀면 Titanic.csv 라는 파일이 존재한다. 이 파일을 열어서 데이터프레임으로 읽어오자.

chatGPT를 사용해 질문하면, 아래와 같이 친절하게 코드까지 알려준다. 하고 싶은 작업이 있으면 chatGPT를 적극 활용하면 많은 도움이 된다².

pandas로 csv파일을 읽어오려면?

Pandas를 사용하여 CSV 파일을 읽어오려면 `pandas` 라이브러리의 `read_csv()` 함수를 사용합니다. 이 함수는 주어진 CSV 파일을 DataFrame 객체로 읽어옵니다. 다음은 Pandas를 사용하여 CSV 파일을 읽는 기본적인 방법입니다:

```
python
import pandas as pd

# CSV 파일 읽기
df = pd.read_csv('파일경로/파일이름.csv')

# DataFrame 출력
print(df)
```

'pandas'의 내장함수 'read_csv()' 함수는 csv 파일³을 읽어서 데이터프레임으로 저장해 주는 역할을 한다.

² 아래는 ChatGPT3.5를 이용해 질문과 답을 캡처한 예이다.

³ csv(comma-separated values)파일은 각 항목을 특정한 구분자(,)로 처리하여 텍스트 기반으로 많은 자료를 저장할 때, 널리 사용된다. MS-Excel 데이터는 저장할 수 있는 레코드 수에 한계가 존재하지만, csv는 상대적으로 자유롭다고 볼 수 있다.


```

import pandas as pd # pandas 패키지를 pd라는 별명으로 불러옴
file='C:/Temp/Titanic.csv' # 파일 위치 지정
df=pd.read_csv(file) # csv 파일을 읽어서 DataFrame으로 저장 해줄
df.info() # dataframe의 정보 표시

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column      Non-Null Count  Dtype
---  -
0   PassengerId  891 non-null    int64
1   Survived     891 non-null    int64
2   Pclass       891 non-null    int64
3   Name         891 non-null    object
4   Sex          891 non-null    object
5   Age          714 non-null    float64
6   SibSp        891 non-null    int64
7   Parch        891 non-null    int64
8   Ticket       891 non-null    object
9   Fare         891 non-null    float64
10  Cabin        204 non-null    object
11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB

```

위 데이터는 12개 칼럼에 891개의 행으로 되어 있음을 알 수 있다. 칼럼들의 데이터 속성을 살펴보면 int64, object, float64 등으로 분류되어 있다.

'Survived' 칼럼의 값의 분포를 보기 위해 value_counts()라는 메서드를 사용할 수 있는데, 이 값의 분포를 보면 0이 549개, 1이 342개로 되어 있다⁴.

```

# Survived 칼럼의 값의 분포 보기
df.Survived.value_counts()

0    549
1    342
Name: Survived, dtype: int64

```

문제7. 타이타닉 데이터프레임 데이터에서 ['PassengerId','Name','Ticket']의 칼럼을 삭제해 보자.

불필요한 칼럼을 삭제할 때는 'drop()'이라는 메서드를 사용하면 된다. 인자 'inplace=True'라고 설정하지 않으면 바뀐 값으로 df 값을 수정하지 않는다.

⁴ 데이터 설명을 찾아보면 0은 사망, 1은 생존을 의미한다고 한다.

```
# 불필요한 컬럼 제거 하기
# inplace=True로 설정하지 않으면, df가 수정되지 않을
df.drop(columns=['PassengerId','Name','Ticket'],inplace=True)
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column      Non-Null Count  Dtype
---  -
0   PassengerId  891 non-null    int64
1   Survived     891 non-null    int64
2   Pclass       891 non-null    int64
3   Name         891 non-null    object
4   Sex          891 non-null    object
5   Age          714 non-null    float64
6   SibSp        891 non-null    int64
7   Parch        891 non-null    int64
8   Ticket       891 non-null    object
9   Fare         891 non-null    float64
10  Cabin        204 non-null    object
11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
# 불필요한 컬럼 제거 하기
df.drop(columns=['PassengerId','Name','Ticket'],inplace=True)
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Survived     891 non-null    int64
1   Pclass       891 non-null    int64
2   Sex          891 non-null    object
3   Age          714 non-null    float64
4   SibSp        891 non-null    int64
5   Parch        891 non-null    int64
6   Fare         891 non-null    float64
7   Cabin        204 non-null    object
8   Embarked     889 non-null    object
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

만일, 위 데이터에서 Age 칼럼의 데이터가 없는 행들을 모두 삭제하고 싶다면 'dropna()' 메서드를 사용할 수 있다. 'Age'와 'Cabin'에서 동시에 데이터가 없는 행들만 삭제하려면 df.dropna(subset=['Age','Cabin'],how='all')과 같이 how라는 매개변수를 추가하여 설정해 주면 된다⁵.

```
# Age의 데이터가 없는 행 삭제
df.dropna(subset=['Age'],inplace=True)
df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 714 entries, 0 to 890
Data columns (total 12 columns):
#   Column      Non-Null Count  Dtype
---  -
0   PassengerId  714 non-null    int64
1   Survived     714 non-null    int64
2   Pclass       714 non-null    int64
3   Name         714 non-null    object
4   Sex          714 non-null    object
5   Age          714 non-null    float64
6   SibSp        714 non-null    int64
7   Parch        714 non-null    int64
8   Ticket       714 non-null    object
9   Fare         714 non-null    float64
10  Cabin        185 non-null    object
11  Embarked     712 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 72.5+ KB
```

문제8. 타이타닉 데이터프레임에서 'Age', 'Cabin', 'Embarked'의 칼럼은 891개에서 몇 개가 모자란 칼럼들이다. 빈 데이터가 있는 경우 이를 채워 넣어야 할 경우가 있다. 'Age' 칼럼의 빈 데이터는 평균값으로 채워넣고, 'Cabin'과 'Embarked'는 문자 'N'으로 채워 넣어라.

⁵ how='any'로 설정하면 하나라도 값이 없으면 해당 행을 삭제하게 된다.

'fillna()' 메서드를 활용하면 NaN 데이터를 특정 값으로 채워넣기 쉽다. df.Age.mean()은 'Age' 칼럼의 평균값을 계산하게 되고, 이 값으로 NaN 데이터를 대체하게 된다.

```
# missing data 만들어 넣기
df['Age'].fillna(df['Age'].mean(),inplace=True)
df['Cabin'].fillna('N',inplace=True)
df['Embarked'].fillna('N',inplace=True)
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Survived    891 non-null    int64
1   Pclass      891 non-null    int64
2   Sex         891 non-null    object
3   Age         891 non-null    float64
4   SibSp       891 non-null    int64
5   Parch       891 non-null    int64
6   Fare        891 non-null    float64
7   Cabin       891 non-null    object
8   Embarked    891 non-null    object
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

작업이 완료된 후 데이터 정보를 살펴보면 모두 891개로 정렬되어 있음을 확인할 수 있다.

문제9. 타이타닉 데이터프레임에서 'Sex'에 따른 'Survived'의 숫자를 비교해 보자.

먼저 'pivot_table()'을 사용하여 생존자의 수를 세어 보면 아래와 같다.

```
# Sex에 따른 Survived의 수와 평균이 어떻게 되는지 프로 계산해 보아라..
df.pivot_table(values='Survived',index='Sex',aggfunc='sum')
```

Survived	
Sex	
female	197
male	93

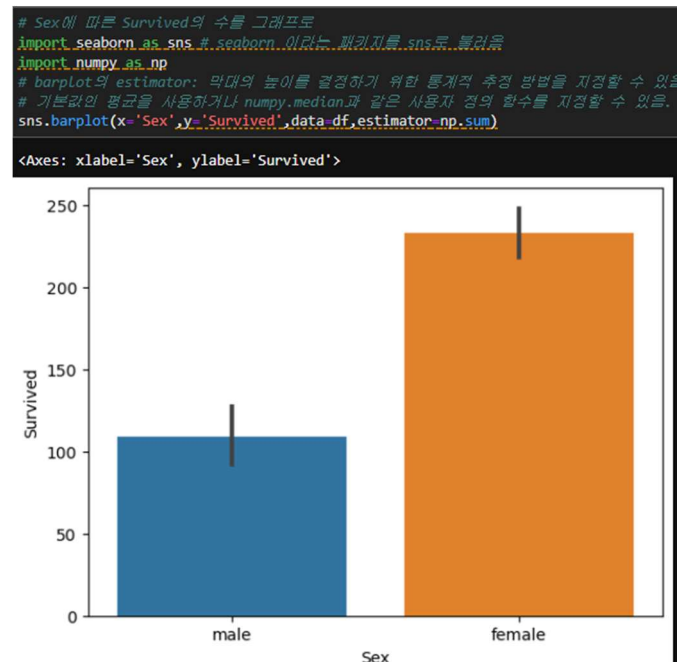
다른 방식으로 'groupby()'를 활용하여 보자.

```
# groupby()로 살펴보기
df.groupby(['Sex','Survived'])['Survived'].count()
```

Sex	Survived	
female	0	64
	1	197
male	0	360
	1	93

Name: Survived, dtype: int64

아래는 그래픽 패키지의 하나인 'seaborn'을 설치⁶하고 'barplot()' 함수를 사용해 'Sex'에 따른 'Survived'의 분포를 그려보았다.



문제10. 타이타닉 데이터프레임에서 'Cabin' 칼럼을 살펴보면 복잡한 문자열이 있다. 각 문자열의 첫 알파벳으로 데이터를 모두 수정해 보아라.

먼저 'Cabin' 칼럼의 데이터를 살펴보면 아래와 같다.

```
# Cabin 칼럼의 값의 분포 출력해보면 복잡한 문자열이 있다. 첫 알파벳으로 모두 수정하자
df['Cabin'].value_counts()

N          687
C23 C25 C27    4
G6             4
B96 B98        4
C22 C26        3
...
E34           1
C7            1
C54           1
E36           1
C148          1
Name: Cabin, Length: 148, dtype: int64
```

'Cabin' 칼럼의 각 문자열의 첫 알파벳만 선택하려면 'str[]' 메서드를 활용하여 df.Cabin.str[0]이나 df.Cabin.str[:1]과 같이 사용하면 된다.

⁶ conda install -c conda-forge seaborn

```
df['Cabin']=df['Cabin'].str[:1] # 첫번째 문자열 선택
df['Cabin'].value_counts()

N    687
C     59
B     47
D     33
E     32
A     15
F     13
G      4
T      1
Name: Cabin, dtype: int64
```

타이타닉 데이터를 읽어와서 데이터를 원하는 대로 변형한 후, 이 데이터를 Excel형태로 저장할 수 있다. 파이썬에서 Excel을 제어하는 패키지 중에 하나인 'openpyxl'을 설치⁷하면, 데이터프레임을 Excel 형태로 저장할 수 있다.

```
# 다른 데이터로 저장하기
df.to_excel('C:/Temp/TT.xlsx') # 엑셀 데이터 저장
df.to_csv('C:/Temp/TT.csv') # csv 형태로 저장

# 저장된 Excel 파일을 읽어와서 데이터 프레임으로 보기
df1=pd.read_excel('C:/Temp/TT.xlsx')
df1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 714 entries, 0 to 713
Data columns (total 10 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Unnamed: 0    714 non-null    int64
1   Survived      714 non-null    int64
2   Pclass        714 non-null    int64
3   Sex           714 non-null    object
4   Age           714 non-null    float64
5   SibSp         714 non-null    int64
6   Parch         714 non-null    int64
7   Fare          714 non-null    float64
8   Cabin         714 non-null    object
9   Embarked      714 non-null    object
dtypes: float64(2), int64(5), object(3)
memory usage: 55.9+ KB
```

Excel 워크북을 열고 수정하는 또다른 패키지 중의 하나인 'xlwings'를 설치⁸하면 데이터프레임을 Excel로 열어볼 수도 있다. 아래와 같이 'xlwings.view()'라는 메서드를 사용하면 자신의 PC에 있는 Excel을 구동시켜 df를 Excel에서 살펴볼 수 있게 된다.

⁷ conda install -c conda-forge openpyxl

⁸ conda install -c conda-forge xlwings

```
import xlwings as xw # xlwings 라이브러리 xw 별명으로 읽어온다.
xw.view(df1)
```

Unnamed: 0	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	0	3 male	22	1	0	7.25	N	S
1	1	1	1 female	38	1	0	71.2833	C	C
2	2	1	3 female	26	0	0	7.925	N	S
3	3	1	1 female	35	1	0	53.1	C	S
4	4	0	3 male	35	0	0	8.05	N	S
5	6	0	1 male	54	0	0	51.8625	E	S

문제11. 웹페이지 [kaggle.com](https://www.kaggle.com)에 접속하여 [Stock Exchange Data](#)를 다운받아서, 압축을 풀면 indexData.csv 라는 파일이 존재한다. 이 파일을 열어서 데이터프레임으로 읽어오자.

앞의 타이타닉 생존자와 같이 csv 자료를 읽어올 수 있다. 자료의 양을 보면 레코드 수가 112,457개이고, Index와 Date 칼럼을 제외하면 모두 float64 데이터로 되어 있음을 알 수 있다.

```
import pandas as pd # pandas 패키지를 pd라는 별명으로 불러옴
file='C:/Temp/indexData.csv' # 파일 위치 지정
df=pd.read_csv(file) # csv 파일을 읽어서 DataFrame으로 저장 해줌
df.info() # dataframe의 정보 표시

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112457 entries, 0 to 112456
Data columns (total 8 columns):
#   Column      Non-Null Count  Dtype  
---  -
0    Index      112457 non-null object  
1    Date       112457 non-null object  
2    Open       110253 non-null float64  
3    High       110253 non-null float64  
4    Low        110253 non-null float64  
5    Close      110253 non-null float64  
6    Adj Close  110253 non-null float64  
7    Volume     110253 non-null float64  
dtypes: float64(6), object(2)
memory usage: 6.9+ MB
```

Date 칼럼의 앞과 뒤의 5개씩의 정보를 미리 살펴보면 아래와 같이 1965년 12월31일부터 2021년 6월 2일까지의 영업일 기준의 데이터라고 추측할 수 있다.

df.Date.head(5)		df.Date.tail(5)	
0	1965-12-31	112452	2021-05-27
1	1966-01-03	112453	2021-05-28
2	1966-01-04	112454	2021-05-31
3	1966-01-05	112455	2021-06-01
4	1966-01-06	112456	2021-06-02
Name: Date, dtype: object		Name: Date, dtype: object	

또한 Index 칼럼의 값들의 분포를 살펴보면 NYA, HSI 등 여러가지 주가지수 이름으로 구분되어 있음을 알 수 있다.

```
# df.Index의 값의 분포 보기
df.Index.value_counts()
```

N225	14500
NYA	13948
IXIC	12690
GSPTSE	10776
HSI	8750
GDAXI	8606
SSMI	7830
KS11	6181
TWII	6010
000001.SS	5963
399001.SZ	5928
N100	5507
NSEI	3381
J203.JO	2387
Name: Index, dtype: int64	

문제12. 위에서 읽어온 데이터프레임의 칼럼 중 'Date'라는 칼럼의 데이터형은 object로 되어 있다. 이를 날짜형(datetime)으로 변환하고, 2019년 6월의 데이터만 추출하여라.

df의 컬럼(column)중 하나인 'Date'라는 것을 실제 날짜 데이터로 인식시키기 위해서는 먼저 아래와 같은 변환작업을 거쳐야 한다. 아래 chatGPT에 질문을 하면서 원하는 라이브러리나 함수 및 매개변수에 대한 정보를 찾아볼 수 있다.

```
# 특정 칼럼을 선택할 때는 df['Date'] 혹은 df.Date 으로 선택해도 된다.
df['Date'] = pd.to_datetime(df.Date) # to_datetime() 을 이용해 날짜 데이터로 변환
df.Date.info() # 'Date' 칼럼에 대한 정보 출력
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 112457 entries, 0 to 112456
Series name: Date
Non-Null Count  Dtype
-----
112457 non-null  datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 878.7 KB
```

Date를 데이터프레임의 datetime 자료형으로 만들어 두고, datetime을 index로 설정해 두면 자료 추

출 등의 다양한 작업을 좀더 쉽게 할 수 있다는 장점이 있다.

```
# 2019년 6월의 데이터만 추출하자
# Date를 Index로 설정하면 날짜와 관련된 다양한 작업을 쉽게 할 수 있다...
df.set_index('Date', inplace=True)
df.loc['2019-06'].# Index가 2019-06으로 시작되는 모든 행을 추출한다...
```

	Index	Open	High	Low	Close	Adj Close	Volume
Date							
2019-06-03	NYA	12288.509770	12372.820310	12273.089840	12341.820310	12341.820310	3.966700e+09
2019-06-04	NYA	12341.820310	12564.370120	12341.820310	12560.580080	12560.580080	3.810430e+09
2019-06-05	NYA	12607.349610	12618.219730	12530.150390	12614.700200	12614.700200	3.548830e+09
2019-06-06	NYA	12631.089840	12710.669920	12606.120120	12675.660160	12675.660160	3.396410e+09
2019-06-07	NYA	12725.379880	12814.809570	12725.379880	12765.860350	12765.860350	3.220250e+09
...
2019-06-24	N100	1071.739990	1075.569946	1067.920044	1069.329956	1069.329956	1.659836e+08
2019-06-25	N100	1064.829956	1068.750000	1064.829956	1066.729980	1066.729980	1.628686e+08
2019-06-26	N100	1064.439941	1069.709961	1062.959961	1063.819946	1063.819946	1.747766e+08
2019-06-27	N100	1064.910034	1065.780029	1057.010010	1062.380005	1062.380005	1.698720e+08
2019-06-28	N100	1062.119995	1072.229980	1061.969971	1071.439941	1071.439941	1.823296e+08

271 rows × 7 columns

문제13. 앞의 주가지수 데이터프레임에서 Index 칼럼의 값이 NYA인 행들만 추출하여 NYA_df라는 데이터 프레임으로 추출하고, 2019년 6월부터 2020년 5월까지의 Adj Close의 그래프를 그려보아라.

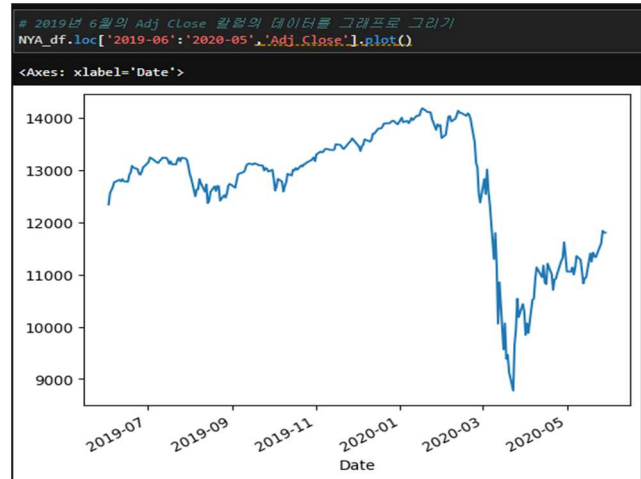
Date를 데이터프레임의 인덱스로 설정한후, 'Index' 칼럼의 값이 'NYA' 행만을 추출하여 NYA_df로 저장하면 아래와 같이 13,948개의 행이 추출된다.

```
# NYA 인덱스 값 추출
NYA_df=df.loc[df.Index=='NYA']
NYA_df.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 13948 entries, 1965-12-31 to 2021-05-28
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Index       13948 non-null  object
1   Open        13947 non-null  float64
2   High        13947 non-null  float64
3   Low         13947 non-null  float64
4   Close       13947 non-null  float64
5   Adj Close   13947 non-null  float64
6   Volume      13947 non-null  float64
dtypes: float64(6), object(1)
memory usage: 871.8+ KB
```

Date에 따른 특정 칼럼의 plot은 아래와 같이 'plot()'메서드⁹를 사용하면 된다.

⁹ plot() 함수는 pandas뿐만 아니라, matplotlib.pyplot, sympy 등에도 존재하며, 각 패키지별로 조금씩 사용방법이 다를 수 있으니 참고하자. 데이터프레임에 사용하는 plot()함수는 사실상 matplotlib의



일반적인 파이썬 슬라이싱 [a:b]라고 하면 b는 포함되지 않지만, 아래에서 보면 2020-05의 데이터가 빠지지 않고 포함되어 있음에 주의하길 바란다.

```
NYA_df.loc['2019-06':'2020-05', 'Adj.Close'].tail(10)
```

Date	Adj Close
2020-05-15	10947.32031
2020-05-18	11402.23047
2020-05-19	11248.96973
2020-05-20	11420.04004
2020-05-21	11351.59961
2020-05-22	11331.96973
2020-05-26	11603.00000
2020-05-27	11837.53027
2020-05-28	11804.91016
2020-05-29	11802.95020

Name: Adj Close, dtype: float64

문제14. 첫날의 종가가 P1, 다음날의 종가가 P2라고 할 때, 연속복리수익율은 자연로그¹⁰ $\log(P2) - \log(P1)$ 과 같이 정의된다¹¹. 앞 문제의 NYA_df의 'Adj Close' 칼럼의 자료를 바탕으로 연속복리수익율 칼럼 'Return'을 작성해 보아라.

plot을 차용하고 있다고 한다.

¹⁰ 밑이 자연상수(e)인 log를 말한다. 수학교과서에는 주로 ln을 쓰기도 하는데 numpy에서는 log()를 사용한다. 참고로 밑이 10인 상용로그는 log10()인 함수가 따로 있다. 자연상수 값에 대한 설명은 뒷 장을 참고하기 바란다.

¹¹ 부록 참조

```
import numpy as np
NYA_df['Return']=np.log(NYA_df['Adj Close'])-np.log(NYA_df['Adj Close'].shift(1))
NYA_df.head(5)
```

C:\Users\hst00\AppData\Local\Temp\ipykernel_6300\193346024.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/1d_introductions.html#1d_introductions
us-a-copy

```
NYA_df['Return']=np.log(NYA_df['Adj Close'])-np.log(NYA_df['Adj Close'].shift(1))
```

	Index	Open	High	Low	Close	Adj Close	Volume	Return
Date								
1965-12-31	NYA	528.690002	528.690002	528.690002	528.690002	528.690002	0.0	NaN
1966-01-03	NYA	527.210022	527.210022	527.210022	527.210022	527.210022	0.0	-0.002803
1966-01-04	NYA	527.840027	527.840027	527.840027	527.840027	527.840027	0.0	0.001194
1966-01-05	NYA	531.119995	531.119995	531.119995	531.119995	531.119995	0.0	0.006195
1966-01-06	NYA	532.070007	532.070007	532.070007	532.070007	532.070007	0.0	0.001787

먼저 자연로그 함수를 사용하기 위해 아래와 같이 'numpy'를 불러와 자연로그인지 확인해 보자.

```
# 자연로그 사용하기
import numpy as np
# 자연로그 함수 log() 사용하기
np.log(np.exp(1)) # exp(1)=e 이다

1.0
```

두번째는 데이터프레임의 'shift()' 메서드의 기능을 살펴보기 위해 아래와 같은 예제를 만들어 보았다. 긴 데이터를 밀려쓰거나 당겨쓰기에 매우 유용한 기능이니 알아두도록 하자¹².

```
# shift() 살펴보기
D=NYA_df["Adj Close"].head(5) # 특정 칼럼의 자료의 앞 5개 추출
Test_df=pd.DataFrame(data={'Price':D}) # 데이터프레임으로 구축
Test_df['lag_1']=Test_df.Price.shift(1) # 1개씩 밀려쓰게됨
Test_df['lag_2']=Test_df.Price.shift(2) # 2개씩 밀려쓰게됨
Test_df['lag_-1']=Test_df.Price.shift(-1) # 1개씩 당겨 쓰게됨
Test_df
```

	Price	lag_1	lag_2	lag_-1
Date				
1965-12-31	528.690002	NaN	NaN	527.210022
1966-01-03	527.210022	528.690002	NaN	527.840027
1966-01-04	527.840027	527.210022	528.690002	531.119995
1966-01-05	531.119995	527.840027	527.210022	532.070007
1966-01-06	532.070007	531.119995	527.840027	NaN

¹² NYA_df['Adj Close']라고 선택하면 pandas의 'series'라는 자료형으로 추출된다. 만일 데이터프레임으로 추출하고 싶다면, NYA_df[['Adj Close']]와 같이 []를 하나 더 씌워주면 되니 참고하자.

이 문제의 해에서 사용한 아래 코드를 자세히 살펴보면 한 칼럼에 들어있는 숫자 전체를 대상으로 마치 하나의 숫자를 연산하는 것처럼 보인다. 이 연산을 MS-Excel에서 구현했다면 cell마다 일일이 연산을 해야 하는 번거로움이 있었을 것이다.

```
NYA_df['Return']=np.log(NYA_df['Adj Close'])-np.log(NYA_df['Adj Close'].shift(1))
```

이와 유사하게 주가분석시에 많이 쓰이는 이동평균(Moving Average)을 구하려면 rolling()과 mean()을 조합하여 사용하면 된다. 새로 만들어진 'MA' 칼럼의 처음 2개는 값이 없으니 NaN으로 처리됨을 확인하고 넘어가자.

```
# 이동평균(Moving Average) 구하기
Test_df['MA']=Test_df.Price.rolling(3).mean() # 3개씩 이동평균을 만든다.
Test_df.head(10) # 앞 10개 데이터 추출
```

	Price	lag_1	lag_2	lag_-1	MA
Date					
1965-12-31	528.690002	NaN	NaN	527.210022	NaN
1966-01-03	527.210022	528.690002	NaN	527.840027	NaN
1966-01-04	527.840027	527.210022	528.690002	531.119995	527.913350
1966-01-05	531.119995	527.840027	527.210022	532.070007	528.723348
1966-01-06	532.070007	531.119995	527.840027	NaN	530.343343

문제15. YA_df에서 매달 말일의 데이터 항만을 추출하여라.

데이터프레임의 인덱스가 datetime으로 설정되어 있을 때 resample() 메서드를 활용해 아래와 같이 쉽게 만들어 낼 수가 있다.

```
df_new=NYA_df.resample('M').last()
df_new.head(5)
```

	Index	Open	High	Low	Close	Adj Close	Volume	Return
Date								
1965-12-31	NYA	528.690002	528.690002	528.690002	528.690002	528.690002	0.0	NaN
1966-01-31	NYA	533.340027	533.340027	533.340027	533.340027	533.340027	0.0	-0.004359
1966-02-28	NYA	524.559998	524.559998	524.559998	524.559998	524.559998	0.0	0.001202
1966-03-31	NYA	511.450012	511.450012	511.450012	511.450012	511.450012	0.0	0.005588
1966-04-30	NYA	521.179993	521.179993	521.179993	521.179993	521.179993	0.0	0.000000

위에서 사용된 'M'대신 'W'를 사용하면 주간 샘플링으로 바꿀 수도 있으며, last() 함수 대신, mean(), sum(), min(), max() 등의 다양한 함수를 사용할 수 있다. 더 자세한 사항은 구글링이나 chatGPT를 활

용하자¹³.

문제16. NYA_df의 'Adj Close'를 대상으로 분기별 시작값(open), 최고값(high), 최저값(low), 마지막값(close) 칼럼을 만들어 보아라.

앞의 문제에서 사용된 resample()과 ohlc()라는 메서드를 사용하면 아래와 같이 간단히 만들어 낼 수 있다.

```
NYA_ohlc=NYA_df['Adj Close'].resample('Q').ohlc()
NYA_ohlc.head(5)
```

Date	open	high	low	close
1965-12-31	528.690002	528.690002	528.690002	528.690002
1966-03-31	527.210022	539.890015	501.510010	511.450012
1966-06-30	515.580017	529.109985	476.450012	485.440002
1966-09-30	490.200012	500.559998	426.329987	436.700012
1966-12-31	427.390015	475.079987	416.290009	462.279999

시계열 데이터프레임을 분석하기 위해서는 반드시 인덱스를 datetime의 자료형으로 변환한 후 사용 하여야 하는 것을 잊지 말자.

문제17. 2024년 1월1일부터 3000일간의 일간 데이터를 생성하고, 1000부터 3000까지의 자연수 난수를 3000개 생성하여 '종가'라는 이름의 칼럼으로 데이터프레임을 작성해 보아라.

'pandas.date_range()' 함수는 주어진 일련의 날짜나 시간을 생성하는 데 사용되는 강력한 기능 중 하나다. 이 함수는 다양한 방식으로 날짜 범위를 생성하고 조작할 수 있어 데이터 분석 및 시계열 데이터 처리에 매우 유용하다.

¹³ 예를 들어 resample('W-FRI').ffill()과 같이 사용하면 매주 금요일 데이터를 샘플링하게 되고, ffill()은 해당 금요일 자료가 없을 시, 직전 금요일 데이터로 채워 넣는다는 의미를 갖게 된다.

```
import pandas as pd
import numpy as np
# date_range() 함수를 이용하여 날짜 데이터 생성
# 매개변수 periods는 데이터 개수, freq='D'는 일간 데이터 생성
# freq가 'W'이면 주간, 'W-MON'이면 월요일마다, 'M'이면 월간, 'Q'이면 분기당
D_date=pd.date_range('2024-1-1',periods=3000,freq='D')
Close=np.random.randint(1000,3001,size=3000) #1000부터 3000까지의 자연수 난수
df=pd.DataFrame(data={'종가':Close},index=D_date)
df.head(10) # 처음 10개의 데이터 보기
```

	증가
2024-01-01	2971
2024-01-02	1309
2024-01-03	1878
2024-01-04	2368
2024-01-05	1643
2024-01-06	2516
2024-01-07	1358
2024-01-08	2741
2024-01-09	1778
2024-01-10	1447

주말을 제외한 월요일부터 금요일까지의 날짜를 생성하기 위해서는 freq='B'로 설정하면 된다.

```
# 주말 제외한 10개의 날짜 생성
import pandas as pd

D0=pd.date_range('2023-09-25',periods=10,freq='B')
D0

DatetimeIndex(['2023-09-25', '2023-09-26', '2023-09-27', '2023-09-28',
               '2023-09-29', '2023-10-02', '2023-10-03', '2023-10-04',
               '2023-10-05', '2023-10-06'],
              dtype='datetime64[ns]', freq='B')
```

주말이외에 대한민국의 휴일 정보까지 반영하고 싶다면, 조금 더 복잡해진다. 먼저 대한민국을 포함하여 여러 나라의 달력 정보를 가지고 있는 라이브러리인 holidays를 설치¹⁴해야 한다.

¹⁴ pip install holidays


```

# 대한민국의 달력기준으로 데이터 만들기
import holidays # 별도로 설치하여야 함

# 주말을 제외한 대한민국의 휴일 정보를 가져온다...
# 아래는 2023년만의 휴일 정보를 가져오도록 설정했다...
kor_holidays=list(holidays.KOR(years=range(2023,2024)).keys()) #list() 데이터로 변환
kor_holidays.append('2023-10-02') # 법정공휴일 추가
kor_holidays=pd.to_datetime(kor_holidays) # 데이터프레임 날짜로 변환

# Business days로 10개의 날짜 생성
D_samples=pd.date_range('2023-09-25',periods=10,freq='B')

# 생성된 날짜에서 대한민국 휴일 제외
# 아래에서 사용된 isin()은 존재하는지를 확인하는 내장함수
D0=D_samples[~D_samples.isin(kor_holidays)] # 휴일정보에 포함되지 않는 날짜만 추출
D0

DatetimeIndex(['2023-09-25', '2023-09-26', '2023-09-27', '2023-10-04',
               '2023-10-05', '2023-10-06'],
              dtype='datetime64[ns]', freq=None)

```