데이터 전처리를 위한

Pandas (I)

박성호 (neowizard2018@gmail.com)

Contents

데이터프레임 개요

- 판다스(Pandas)는 데이터프레임(DataFrame)과 시리즈(Series) 라는 데이터타입(DataType)과 데이터 분석을 위한 다양한 기능을 제공 해주는 파이썬 라이브러리

데이터프레임(DataFrame)

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

시리즈(Series)

데이터프레임 생성 (from dictionary data)

➢ 데이터프레임은 dictionary 데이터 또는 list 데이터를 이용해서 생성할 수 있음

```
import pandas as pd

data_dict = { 'Name' : ['John', 'Sabre', 'Kim', 'Sato', 'Lee', 'Smith', 'David'],
              'Country' : ['USA', 'France', 'Korea', 'Japan', 'Korea', 'USA', 'USA'],
              'Age' : [ 31, 33, 28, 40, 36, 55, 48],
              'Job' : ['Student', 'Lawyer', 'Developer', 'Chef', 'Professor', 'CEO', 'Banker']
            }


df = pd.DataFrame(data_dict)

df
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

데이터프레임 생성 (from list data)

```
import pandas as pd

data_list = [ ['John', 'USA', 31, 'Student'],
              ['Sabre', 'France', 33, 'Lawyer'],
              ['Kim', 'Korea', 28, 'Developer'],
              ['Sato', 'Japan', 40, 'Chef'],
              ['Lee', 'Korea', 36, 'Professor'],
              ['Smith', 'USA', 55, 'CEO'],
              ['David', 'USA', 48, 'Banker'],
            ]

column_name = ['Name', 'Country', 'Age', 'Job']

df = pd.DataFrame(data_list, columns = column_name)

df
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

데이터프레임 기본 정보 확인

df.head(), df.tail(), df.info(), df.describe(), df.index, df.columns

df.head()

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor

df.tail()

	Name	Country	Age	Job
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

df.describe()

	Age
count	7.000000
mean	38.714286
std	9.724784
min	28.000000
25%	32.000000
50%	36.000000
75%	44.000000
max	55.000000

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 4 columns):
Name       7 non-null object
Country    7 non-null object
Age        7 non-null int64
Job        7 non-null object
dtypes: int64(1), object(3)
memory usage: 304.0+ bytes
```

데이터프레임 csv 파일 저장 df.to_csv(...)

```
# index => Yes

df.to_csv('./test_dataframe_with_index.csv', index=True)
```

```
# index => No

df.to_csv('./test_dataframe_without_index.csv', index=False)
```

```
# header => Yes

df.to_csv('./test_dataframe_with_header.csv', header=True)
```

```
# header => No

df.to_csv('./test_dataframe_without_header.csv', header=False)
```

```
# header => YES, index => NO

df.to_csv('./test_dataframe_with_header_without_index.csv', header=True, index=False)
```

```
# header => No,  index => No

df.to_csv('./test_dataframe_without_header_without_index.csv', header=False, index=False)
```

csv 파일로부터 데이터프레임 생성 pd.read_csv(...)

```
import pandas as pd
df = pd.read_csv('./test_dataframe_with_header_without_index.csv')
df
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

```
import pandas as pd
df = pd.read_csv('./test_dataframe_without_header_without_index.csv')
df
```

	John	USA	31	Student
0	Sabre	France	33	Lawyer
1	Kim	Korea	28	Developer
2	Sato	Japan	40	Chef
3	Lee	Korea	36	Professor
4	Smith	USA	55	CEO
5	David	USA	48	Banker

← 첫번째 데이터를 header 인식함

csv 파일로부터 데이터프레임 생성 pd.read_csv(...)

```
import pandas as pd

df = pd.read_csv('./test_dataframe_without_header_without_index.csv', header=None)

df
```

	0	1	2	3
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

```
cols = [ 'Name', 'Country', 'Age', 'Job' ]

df.columns = cols

df
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

데이터 전처리를 위한

Pandas (II)

박성호 (neowizard2018@gmail.com)

Contents

csv 파일로부터 데이터프레임 생성 pd.read_csv(...)

```
import pandas as pd

df = pd.read_csv('./test_dataframe_with_header_without_index.csv')

df
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

1. 열(column) 데이터 추출하기

➢ 데이터프레임(DataFrame)에서 열(column) 단위 데이터를 추출하기 위해서는 대괄호 안에 열 이름을 사용함

```
df_job = df[ 'Job' ]

df_job
```

← Series

```
0       Student
1        Lawyer
2     Developer
3          Chef
4     Professor
5           CEO
6        Banker
Name: Job, dtype: object
```

```
df_job = df[ ['Job'] ]

df_job
```

← DataFrame

	Job
0	Student
1	Lawyer
2	Developer
3	Chef
4	Professor
5	CEO
6	Banker

```
df_country_job = df[ ['Country', 'Job'] ]

df_country_job
```

	Country	Job
0	USA	Student
1	France	Lawyer
2	Korea	Developer
3	Japan	Chef
4	Korea	Professor
5	USA	CEO
6	USA	Banker

```
cols = [ 'Country', 'Job' ]

df_country_job = df[ cols ]

df_country_job
```

	Country	Job
0	USA	Student
1	France	Lawyer
2	Korea	Developer
3	Japan	Chef
4	Korea	Professor
5	USA	CEO
6	USA	Banker

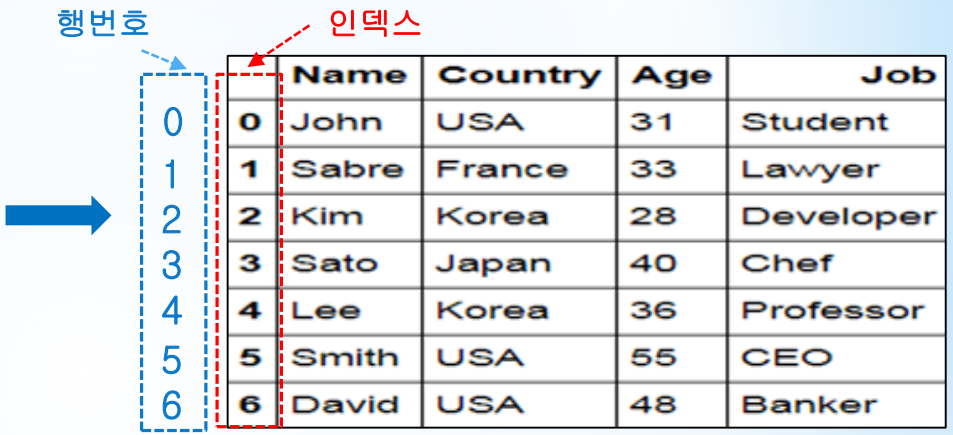
2. 인덱스, 행번호 개념

- 판다스에서는 df.loc[인덱스], df.iloc[행번호] 사용하여 행 단위로 데이터를 가져옴. 초보자라면 조건 지정이 용이한 df.loc[인덱스] 사용법부터 학습하는 것이 좋을것으로 판단됨

loc	인덱스 기준으로 행 데이터 읽기
iloc	행 번호를 기준으로 행 데이터 읽기

현재는 **인덱스**가 **행번호** 처럼 보이지만, 사실 인덱스는 문자열이나 임의의 숫자를 지정해도 무방함

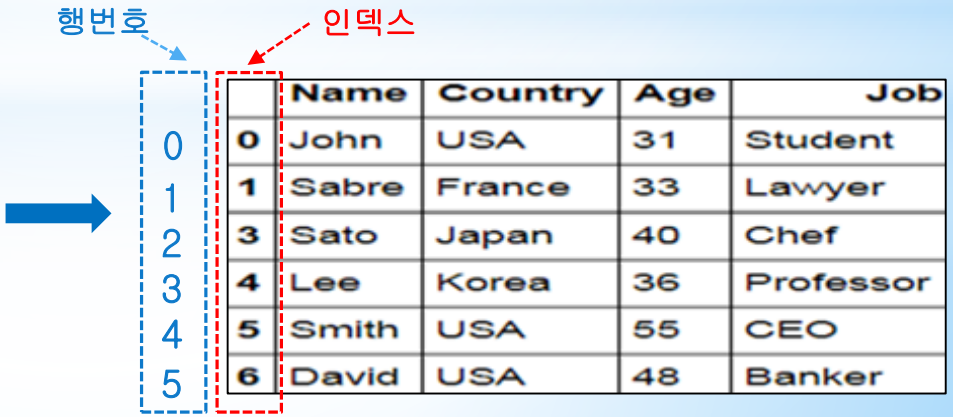
인덱스는 보통 0 부터 시작하지만 행 데이터를 추가, 삭제하면 언제든 변할 수 있음.



	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

df.drop([2]) 명령을 통해 보기와 같이 2번 인덱스를 삭제하면,

행번호는 원래와 같이 0부터 시작해서 순서대로 이어지지만, **인덱스**는 연속적인 순서가 아닌 것을 확인 할 수 있음



	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

2.1 df.loc[] 이용하여 행(row) 데이터 추출

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

[Self Study]

df.loc[0] 데이터 타입
df.loc[[0]] 데이터 타입
df.loc[0, :] 결과
df.loc[[0], :] 결과
df.loc[[0, 3], :] 결과
df.loc[0:3, :] 결과

```
df_1st_row = df.loc[ [0] ]

df_1st_row
```

	Name	Country	Age	Job
0	John	USA	31	Student

```
df_1st_4th_row = df.loc[ [0, 3] ]

df_1st_4th_row
```

	Name	Country	Age	Job
0	John	USA	31	Student
3	Sato	Japan	40	Chef

```
df_slice = df.loc[ 0:3 ]

df_slice
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef

loc 속성의 슬라이싱은 일반적인 슬라이싱과는 다르므로 주의 필요.
즉 [0:3] 인덱스 0~2 행 추출이 아닌 0~3 까지의 행을 추출함

2.1 df.loc[] 이용하여 조건에 맞는 행(row) 데이터 추출

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

[Self Study]

df.loc[df['Age']>30, :]

df.loc[df['Age']>30, ['Job']]

df.loc[(df['Age']>30) & (df['Job']=='Chef')]

df.loc[(df['Age']>30) | (df['Job']=='Chef')]

```
df.loc[df['Country']=='USA']
```

	Name	Country	Age	Job
0	John	USA	31	Student
5	Smith	USA	55	CEO
6	David	USA	48	Banker

```
df.loc[df['Age']>30]
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

3. df.loc[] 이용한 행과 열 데이터 동시 추출

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

[Self Study]

```
for index in range(len(df)):

    print(type(df.loc[ index, ['Name' ,'Country']]))
    print(df.loc[ index, ['Name' ,'Country']])
    print('=====================================')
    print(type(df.loc[ [index], ['Name' ,'Country']]))
    print(df.loc[ [index], ['Name' ,'Country']])
    print('=====================================')
```

```
df.loc[ [1], : ]
```

	Name	Country	Age	Job
1	Sabre	France	33	Lawyer

```
df.loc[ [1, 3], ['Name', 'Job'] ]
```

	Name	Job
1	Sabre	Lawyer
3	Sato	Chef

```
df.loc[ :, : ]
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

4. 데이터프레임 행과 열 삭제 – df.drop()

4.1 행 삭제 df.drop(index, axis=0) # axis = 0 행, axis = 1 열

```
df = df.drop(1, axis=0)

df
```

	Name	Country	Age	Job
0	John	USA	31	Student
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

```
df = df.drop([3, 5], axis=0)

df
```

	Name	Country	Age	Job
0	John	USA	31	Student
2	Kim	Korea	28	Developer
4	Lee	Korea	36	Professor
6	David	USA	48	Banker

```
df = df.reset_index()

df
```

	index	Name	Country	Age	Job
0	0	John	USA	31	Student
1	2	Kim	Korea	28	Developer
2	4	Lee	Korea	36	Professor
3	6	David	USA	48	Banker

[Self Study]

데이터 프레임 df 생성 후,

df.drop(1, axis=0, inplace=True)

df.drop([3, 5], axis=0, inplace=True)

df.reset_index(inplace=True)

4. 데이터프레임 행과 열 삭제 – df.drop()

4.2 열 삭제 df.drop(column name, axis=1) # axis = 0 행, axis = 1 열

```
df = df.drop('Age', axis=1)
df
```

	Name	Country	Job
0	John	USA	Student
1	Sabre	France	Lawyer
2	Kim	Korea	Developer
3	Sato	Japan	Chef
4	Lee	Korea	Professor
5	Smith	USA	CEO
6	David	USA	Banker

```
df = df.drop(['Name', 'Job'], axis=1)
df
```

	Country
0	USA
1	France
2	Korea
3	Japan
4	Korea
5	USA
6	USA

[Self Study]

데이터 프레임 df 생성 후,

df.drop('Age', axis=1, inplace=True)

df.drop(['Name', 'Job'], axis=1, inplace=True)

5. 데이터프레임 행과 열 추가

5.1 행 추가 df.append(dict_new_data, ignore_index=True)

```
new_data = { 'Name' : 'Park',
             'Country' : 'Korea',
             'Age' : 36,
             'Job' : 'Chef'
           }

df = df.append(new_data, ignore_index=True)

df
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker
7	Park	Korea	36	Chef

```
new_data = { 'Name' : 'Koga',
             'Country' : 'Japan',
             'Age' : 26,
             'Job' : 'Player'
           }

df = df.append(new_data, ignore_index=True)

df
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker
7	Park	Korea	36	Chef
8	Koga	Japan	26	Player

5. 데이터프레임 행과 열 추가

5.2 열 추가 df['column'], df.assign()

```
df['New_Col1'] = df['Age'] / 2.0

df
```

	Name	Country	Age	Job	New_Col1
0	John	USA	31	Student	15.5
1	Sabre	France	33	Lawyer	16.5
2	Kim	Korea	28	Developer	14.0
3	Sato	Japan	40	Chef	20.0
4	Lee	Korea	36	Professor	18.0
5	Smith	USA	55	CEO	27.5
6	David	USA	48	Banker	24.0

```
add_val_1 = df['Age'].values

add_val_2 = df['New_Col1'].values

df = df.assign(ADD_1=add_val_1, ADD_2=add_val_2)

df
```

	Name	Country	Age	Job	New_Col1	ADD_1	ADD_2
0	John	USA	31	Student	15.5	31	15.5
1	Sabre	France	33	Lawyer	16.5	33	16.5
2	Kim	Korea	28	Developer	14.0	28	14.0
3	Sato	Japan	40	Chef	20.0	40	20.0
4	Lee	Korea	36	Professor	18.0	36	18.0
5	Smith	USA	55	CEO	27.5	55	27.5
6	David	USA	48	Banker	24.0	48	24.0

6. 데이터프레임 합치기 pd.concat()

6.1 위 아래 방향으로 합치기 pd.concat([df1, df2], axis=0, ignore_index=True)

```
data1 = { 'Name' : ['John', 'Sabre'],
          'Country' : ['USA', 'France'],
          'Age' : [31, 33],
          'Job' : ['Student', 'Lawyer']
        }

df1 = pd.DataFrame(data1)

df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer

```
data2 = { 'Name' : ['Lee', 'Smith'],
          'Country' : ['Korea', 'USA'],
          'Age' : [36, 55],
          'Job' : ['Professor', 'CEO']
        }

df2 = pd.DataFrame(data2)

df2
```

	Name	Country	Age	Job
0	Lee	Korea	36	Professor
1	Smith	USA	55	CEO

6. 데이터프레임 합치기 pd.concat()

6.1 위 아래 방향으로 합치기 pd.concat([df1, df2], axis=0, ignore_index=True)

```
df3 = pd.concat([df1, df2], axis=0)

df3
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
0	Lee	Korea	36	Professor
1	Smith	USA	55	CEO

```
df4 = pd.concat([df1, df2], axis=0, ignore_index=True)

df4
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Lee	Korea	36	Professor
3	Smith	USA	55	CEO

6. 데이터프레임 합치기 pd.concat()

6.2 좌우 방향으로 합치기 pd.concat([df1, df2], axis=1)

```
data1 = { 'Name' : ['John', 'Sabre'],
          'Country' : ['USA', 'France'],
          'Age' : [31, 33],
          'Job' : ['Student', 'Lawyer']
        }

df1 = pd.DataFrame(data1)

df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer

```
data5 = { 'Salary' : 1000,
          'Hobby' : ['Run'],
        }

df5 = pd.DataFrame(data5)

df5
```

	Salary	Hobby
0	1000	Run

```
df6 = pd.concat([df1, df5], axis=1)

df6
```

	Name	Country	Age	Job	Salary	Hobby
0	John	USA	31	Student	1000.0	Run
1	Sabre	France	33	Lawyer	NaN	NaN

7. 데이터프레임 열 순서 변경 및 특정 열 제외

7.1 열 순서 변경

```
df
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

```
column_name = [ 'Age', 'Job', 'Country', 'Name' ]

df1 = df[column_name]

df1
```

	Age	Job	Country	Name
0	31	Student	USA	John
1	33	Lawyer	France	Sabre
2	28	Developer	Korea	Kim
3	40	Chef	Japan	Sato
4	36	Professor	Korea	Lee
5	55	CEO	USA	Smith
6	48	Banker	USA	David

7. 데이터프레임 열 순서 변경 및 특정 열 제외

7.2 특정 열 제외

```
df
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

```
df2 = df[df.columns.difference(['Age', 'Job'])]

df2
```

	Country	Name
0	USA	John
1	France	Sabre
2	Korea	Kim
3	Japan	Sato
4	Korea	Lee
5	USA	Smith
6	USA	David

데이터 전처리를 위한

Pandas (III)

박성호 (neowizard2018@gmail.com)

Contents

Missing Data (NaN, None 등) 처리 1

➢ 판다스 read_csv(…) 이용하여 다음과 같은 데이터 읽어 옴 (Missing Data 확인)

Name	Country	Age	Job
John	USA	31	Student
Sabre	France	33	
Kim	Korea	28	Developer
Sato		40	Chef
Lee	Korea	36	Professor
Smith	USA	55	CEO
David	USA		Banker

test_missing_data.csv

```
import pandas as pd

try:
    df = pd.read_csv('./test_missing_data.csv')
except Exception as err:
    print(str(err))
```

```
df
```

	Name	Country	Age	Job
0	John	USA	31.0	Student
1	Sabre	France	33.0	NaN
2	Kim	Korea	28.0	Developer
3	Sato	NaN	40.0	Chef
4	Lee	Korea	36.0	Professor
5	Smith	USA	55.0	CEO
6	David	USA	NaN	Banker

Missing Data (NaN, None 등) 처리 2 – isnull(), dropna()

- Missing Data 개수 확인

  df.isnull().sum()

```
df.isnull().sum()

Name       0
Country    1
Age        1
Job        1
dtype: int64
```

- 각 열(column)에 있는 각각의 Data 개수 확인 (NaN 제외한 데이터 개수)

  df['Name'].value_counts(), df['Country'].value_counts(),…

- NaN 값이 있는 행(row) 모두 제거

  df.dropna()

```
df1 = df.dropna()

df1
```

	Name	Country	Age	Job
0	John	USA	31.0	Student
2	Kim	Korea	28.0	Developer
4	Lee	Korea	36.0	Professor
5	Smith	USA	55.0	CEO

Missing Data (NaN, None 등) 처리 3 – fillna()

- Missing Data 를 특정 값으로 변경하기 (각 열의 NaN)

  df['열이름'].fillna(변경값, inplace=True)

```
df['Country'].fillna('Spain')
df['Age'].fillna(100.0)
df['Job'].fillna('Reporter')

df
```

	Name	Country	Age	Job
0	John	USA	31.0	Student
1	Sabre	France	33.0	NaN
2	Kim	Korea	28.0	Developer
3	Sato	NaN	40.0	Chef
4	Lee	Korea	36.0	Professor
5	Smith	USA	55.0	CEO
6	David	USA	NaN	Banker

```
df['Country'].fillna('Spain', inplace=True)
df['Age'].fillna(100.0, inplace=True)
df['Job'].fillna('Reporter', inplace=True)

df
```

	Name	Country	Age	Job
0	John	USA	31.0	Student
1	Sabre	France	33.0	Reporter
2	Kim	Korea	28.0	Developer
3	Sato	Spain	40.0	Chef
4	Lee	Korea	36.0	Professor
5	Smith	USA	55.0	CEO
6	David	USA	100.0	Banker

Missing Data (NaN, None 등) 처리 4 – fillna()

- Missing Data 를 특정 값으로 변경하기 (모든 NaN)

  df.fillna(변경값, inplace=True)

```
df_test = pd.read_csv('./test_missing_data.csv')

df_test
```

	Name	Country	Age	Job
0	John	USA	31.0	Student
1	Sabre	France	33.0	NaN
2	Kim	Korea	28.0	Developer
3	Sato	NaN	40.0	Chef
4	Lee	Korea	36.0	Professor
5	Smith	USA	55.0	CEO
6	David	USA	NaN	Banker

```
df_test.fillna('AAA', inplace=True)

df_test
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	AAA
2	Kim	Korea	28	Developer
3	Sato	AAA	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	AAA	Banker

[appendix] mean(), median(), replace()

- fillna() 에서 NaN 을 특정 값으로 변경할때 mean() 또는 median() 등으로 바꾸는 경우가 많음 (통계의 오류는 감안 해야함)

```
df_stat = pd.read_csv('./test_missing_data.csv')

df_stat
```

	Name	Country	Age	Job
0	John	USA	31.0	Student
1	Sabre	France	33.0	NaN
2	Kim	Korea	28.0	Developer
3	Sato	NaN	40.0	Chef
4	Lee	Korea	36.0	Professor
5	Smith	USA	55.0	CEO
6	David	USA	NaN	Banker

```
print('Age mean = ', df_stat['Age'].mean())
print('Age median = ', df_stat['Age'].median())
```

```
Age mean =  37.166666666666664
Age median =  34.5
```

[appendix] mean(), median(), replace()

- replace() 함수 이용하여 NaN ⇒ 특정값 또는 특정값 ⇒ NaN 으로 변경하는 경우도 있음 (특정값은 일반적으로 outlier 경우가 일반적임)

```
import numpy as np

df_stat['Age'].replace(np.nan, 50, inplace=True)

df_stat
```

	Name	Country	Age	Job
0	John	USA	31.0	Student
1	Sabre	France	33.0	NaN
2	Kim	Korea	28.0	Developer
3	Sato	NaN	40.0	Chef
4	Lee	Korea	36.0	Professor
5	Smith	USA	55.0	CEO
6	David	USA	50.0	Banker

```
import numpy as np

df_stat['Job'].replace('CEO', np.nan, inplace=True)

df_stat
```

	Name	Country	Age	Job
0	John	USA	31.0	Student
1	Sabre	France	33.0	NaN
2	Kim	Korea	28.0	Developer
3	Sato	NaN	40.0	Chef
4	Lee	Korea	36.0	Professor
5	Smith	USA	55.0	NaN
6	David	USA	50.0	Banker

다중분류 (Multi-Classification) 예제

diabetes.csv 파일을 이용한 기존 예제는 이항 분류 문제였음. 즉 기존 예제 코드는 정답을 0 또는 1 로 나타내는 이항분류(binary classification) 문제 였으나, 이러한 정답을 아래의 각각의 조건에 맞게 다중 분류(0 또는 1)로 나타내어 코드를 구현하시오

[1] 정답(t_data) 를 one-hot encoding 으로 나타낸 후에, 다중 분류를 할수 있는 모델을 구현하시오

[2] 정답(t_data) 를 one-hot encoding 으로 나타내지 말고, 다중 분류를 할수 있는 모델을 구현하시오

※ 예제와 같이 Sequential API방식으로 먼저 구현하고, 시간 여유가 된다면 각각의 모델을 Sequential / Functional API 방식으로 구현해 보시오)

다중분류 (Multi-Classification) 예제

diabetes.csv 파일을 이용한 기존 예제는 이항 분류 문제였음. 즉 기존 예제 코드는 정답을 0 또는 1 로 나타내는 이항분류(binary classification) 문제 였으나, 이러한 정답을 아래의 각각의 조건에 맞게 다중 분류(0 또는 1)로 나타내어 코드를 구현하시오

[1] 정답(t_data) 를 one-hot encoding 으로 나타낸 후에, 다중 분류를 할수 있는 모델을 구현하시오

[2] 정답(t_data) 를 one-hot encoding 으로 나타내지 말고, 다중 분류를 할수 있는 모델을 구현하시오

※ 예제와 같이 Sequential API방식으로 먼저 구현하고, 시간 여유가 된다면 각각의 모델을 Sequential / Functional API 방식으로 구현해 보시오)

- Neural Network (MNIST example) -

박성호 (neowizard2018@gmail.com)

MNIST Example https://youtu.be/AyvicBsP8tE

[1] MNIST Data 생성 및 확인

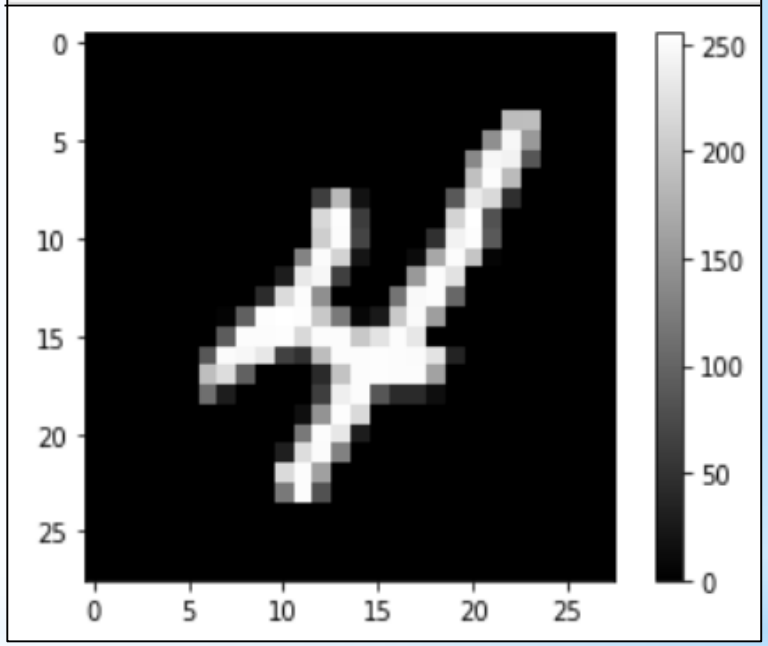
```
import tensorflow as tf
import numpy as np

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to_categorical
```

```
(x_train, t_train), (x_test, t_test) = mnist.load_data()

print('')
print('x_train.shape = ', x_train.shape, ', t_train.shape = ', t_train.shape)
print('x_test.shape = ', x_test.shape, ', t_test.shape = ', t_test.shape)
```

```
x_train.shape =  (60000, 28, 28) , t_train.shape =  (60000,)
x_test.shape =  (10000, 28, 28) , t_test.shape =  (10000,)
```

```python
import matplotlib.pyplot as plt

# 25개의 이미지 출력
plt.figure(figsize=(6, 6))

for index in range(25):    # 25 개 이미지 출력

    plt.subplot(5, 5, index + 1)  # 5행 5열
    plt.imshow(x_train[index], cmap='gray')
    plt.axis('off')


plt.show()
```

```python
plt.imshow(x_train[9], cmap='gray')
plt.colorbar()
plt.show()
```



[2] 데이터 전처리

```
# x_train, x_test 값 범위를 0 ~ 1 사이로 정규화

x_train = x_train / 255.0
x_test = x_test / 255.0

# 정규화 결과 확인
print('train max = ', x_train[0].max(),', train min = ', x_train[0].min())
print('test max = ', x_train[0].max(),', test min = ', x_train[0].min())
```

```
train max =  1.0 , train min =  0.0
test max =  1.0 , test min =  0.0
```

```
# 정답 데이터 one-hot encoding
t_train = to_categorical(t_train, 10)
t_test = to_categorical(t_test, 10)

# one-hot encoding 확인
print('train label = ', t_train[0], ', decimal value = ', np.argmax(t_train[0]))
print('test label = ', t_test[0], ', decimal value = ', np.argmax(t_test[0]))
```

```
train label =  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.] , decimal value =  5
test label =  [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.] , decimal value =  7
```

[3] 모델 구축 및 컴파일

```
model = Sequential()      # model 생성

model.add(Flatten(input_shape=(28, 28, 1)))

model.add(Dense(100, activation='relu'))

model.add(Dense(10, activation='softmax'))
```

```
from tensorflow.keras.optimizers import SGD

model.compile(optimizer=SGD(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 100)               78500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

[4] 모델 학습

```
hist = model.fit(x_train, t_train, epochs=30, validation_split=0.2)
```

```
Epoch 2/30
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3682 - accuracy: 0.8980 - val_loss: 0.3169 - val_accuracy: 0.9116
Epoch 3/30
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3163 - accuracy: 0.9109 - val_loss: 0.2799 - val_accuracy: 0.9219
Epoch 4/30
1500/1500 [==============================] - 4s 3ms/step - loss: 0.2858 - accuracy: 0.9196 - val_loss: 0.2592 - val_accuracy: 0.9267
Epoch 5/30
```

.....

```
Epoch 28/30
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0985 - accuracy: 0.9731 - val_loss: 0.1217 - val_accuracy: 0.9658
Epoch 29/30
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0959 - accuracy: 0.9739 - val_loss: 0.1174 - val_accuracy: 0.9668
Epoch 30/30
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0936 - accuracy: 0.9746 - val_loss: 0.1151 - val_accuracy: 0.9678
```

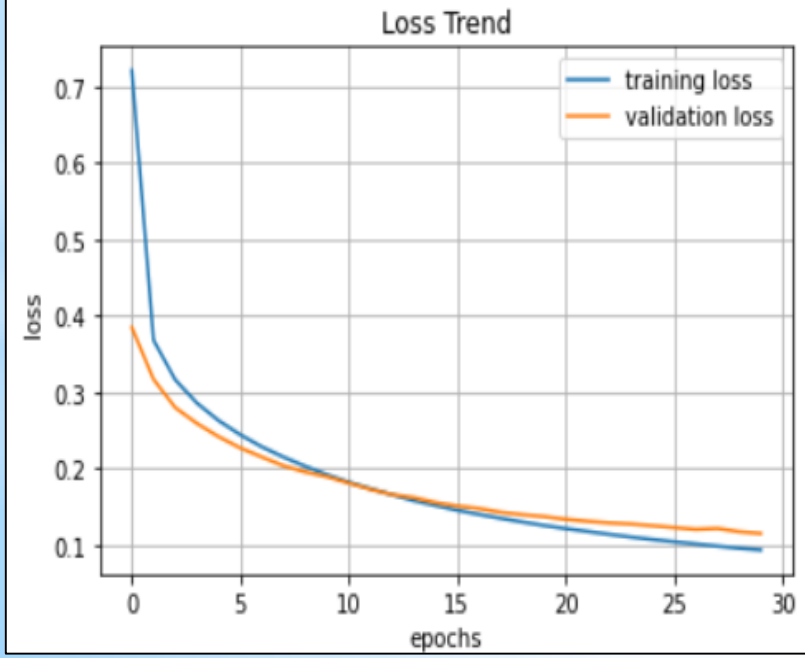
[5] 모델 (정확도) 평가

```
model.evaluate(x_test, t_test)
```

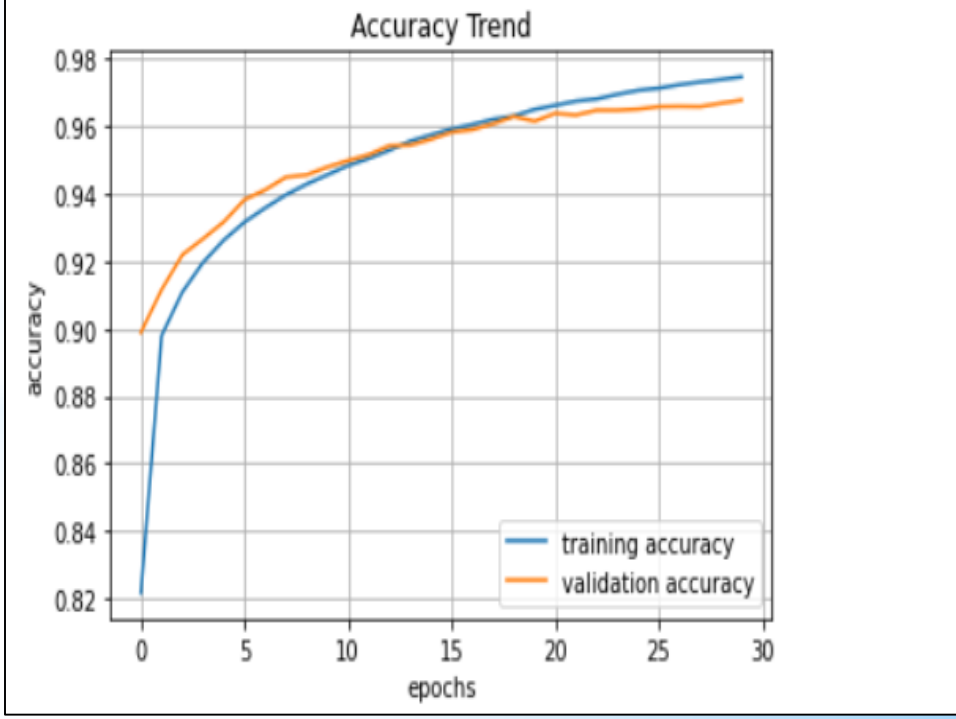
```
313/313 [==============================] - 1s 3ms/step - loss: 0.1105 - accuracy: 0.9678
[0.11048293113708496, 0.9678000211715698]
```

[6] 손실 및 정확도 추세

```
import matplotlib.pyplot as plt

plt.title('Loss Trend')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.grid()

plt.plot(hist.history['loss'], label='training loss')
plt.plot(hist.history['val_loss'], label='validation loss')
plt.legend(loc='best')

plt.show()
```




```
plt.title('Accuracy Trend')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.grid()

plt.plot(hist.history['accuracy'], label='training accuracy')
plt.plot(hist.history['val_accuracy'], label='validation accuracy')
plt.legend(loc='best')

plt.show()
```



[7] 예측

```
pred = model.predict(x_test)

print(pred.shape)

print(pred[:5])   # 모델이 예측한 pred[:5] 필기체 손글씨 숫자와 정답을 비교하시오
```

numpy.random.choice() 함수를 이용해서 x_test 에서 임의로 서로 다른 5개의 데이터를 추출해서 model.predict() 실행하시오

project overview

➢ 데이터

✓ UCI Machine Learning Repository의 wine quality data set 사이트에서, winequality-red.csv, winequality-white.csv 파일 다운로드 (https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/)

➢ 프로젝트

와인 타입(wine type)을 red, white 두가지 타입으로 예측

✓ 아키텍처, node 개수, epoch, optimizer 등을 바꾸어 가면서 구현함 최소 2개이상 구현

```
red_df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

```
white_df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

표준화, 정규화

표준화(Standardization): 데이터의 피처 각각이 평균이 0, 분산이 1인 가우시안 정규분포를 가진 값으로 변환하는 작업을 표준화라고 함.

$$x_{i\_new} = \frac{x_i - \mathrm{mean}(x)}{stdev(x)}$$

실제 구현시에는 사이킷런의 StandardScaler를 사용해 표준화를 진행하는것이 일반적임

정규화(Normalization): 서로 다른 피처들의 크기를 통일하기 위해 크기를 변화해주는 것.

$$x_{i\_new} = \frac{x_i - \min(x)}{max(x) - \min(x)}$$

실제 구현시에는 사이킷런에서 제공하는 MinMaxScaler는 음수 값이 없으면 0 ~ 1의 값으로, 음수 값이 있으면 -1 ~ 1의 값으로 변환해준다.

표준화, 정규화

```
import pandas as pd

df = pd.read_csv('./kaggle_diabetes.csv', sep=',')

df.describe()
```

```
# 표준화 (Standardization)

from sklearn.preprocessing import StandardScaler

std_cols = ['Pregnancies', 'Glucose', 'BloodPressure','SkinThickness',
            'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age' ]

scaler = StandardScaler()

df_std = scaler.fit_transform(df[std_cols])

print(type(df_std))

df_std = pd.DataFrame(df_std, columns=std_cols)

df_std['Outcome'] = df['Outcome'].values

df_std.describe()
```

표준화, 정규화

```
# 정규화 (Normalization)

from sklearn.preprocessing import MinMaxScaler

norm_cols = ['Pregnancies', 'Glucose', 'BloodPressure','SkinThickness',
             'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age' ]

scaler = MinMaxScaler()

df_norm = scaler.fit_transform(df[norm_cols])

print(type(df_norm))

df_norm = pd.DataFrame(df_norm, columns=std_cols)

df_norm['Outcome'] = df['Outcome'].values

df_norm.describe()
```

[예제] 당뇨병 발병 예측

1. https://www.kaggle.com/uciml/pima-Indians-diabetes-database 접속 후,

2. Download 메뉴를 통해서 데이터 다운 후 kaggle_diabetes.csv 파일로 이름 변경

3. 머신러닝/딥러닝 기본 프로세스를 바탕으로, 당뇨병 발병 확률을 70% 이상으로 예측

1. 데이터 로드 및 기본 정보 확인

```
import matplotlib
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv('./kaggle_diabetes.csv')

df.head()
```

```
# 전체 히스토그램을 살펴본다
df.hist()

plt.tight_layout()
plt.show()
```

```
# 개별 히스토그램을 살펴본다

df['BloodPressure'].hist()

plt.tight_layout()
plt.show()
```

```
df.info()
```

```
df.describe()
```

2. 데이터 전처리

```
# missing value 확인

df.isnull().sum()
```

```
for col in df.columns:
    missing_rows = df.loc[df[col]==0].shape[0]
    print(col + ": " + str(missing_rows))
```

```
import numpy as np

# outlier 처리

df['Glucose'] = df['Glucose'].replace(0, np.nan)
df['BloodPressure'] = df['BloodPressure'].replace(0, np.nan)
df['SkinThickness'] = df['SkinThickness'].replace(0, np.nan)
df['Insulin'] = df['Insulin'].replace(0, np.nan)
df['BMI'] = df['BMI'].replace(0, np.nan)

# missing value 처리

df['Glucose'] = df['Glucose'].fillna(df['Glucose'].mean())
df['BloodPressure'] = df['BloodPressure'].fillna(df['BloodPressure'].mean())
df['SkinThickness'] = df['SkinThickness'].fillna(df['SkinThickness'].mean())
df['Insulin'] = df['Insulin'].fillna(df['Insulin'].mean())
df['BMI'] = df['BMI'].fillna(df['BMI'].mean())
```

2. 데이터 전처리

```
for col in df.columns:
    missing_rows = df.loc[df[col]==0].shape[0]
    print(col + ": " + str(missing_rows))
```

```
# 데이터 표준화

from sklearn.preprocessing import StandardScaler

scale_cols = ['Pregnancies', 'Glucose', 'BloodPressure','SkinThickness',
              'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age' ]

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[scale_cols])

print(type(df_scaled))

df_scaled = pd.DataFrame(df_scaled, columns=scale_cols)

df_scaled['Outcome'] = df['Outcome'].values  # 원본 DataFrame 보존
```

2. 데이터 전처리

```
# feature column, label column 추출 후 DataFrame 생성

feature_df = df_scaled[df_scaled.columns.difference(['Outcome'])]

label_df = df_scaled['Outcome']

print(feature_df.shape, label_df.shape)
```

```
# pandas <=> numpy

feature_np = feature_df.to_numpy().astype('float32')
label_np = label_df.to_numpy().astype('float32')

print(feature_np.shape, label_np.shape)
```

3. 머신러닝 / 딥러닝

```
s = np.arange(len(feature_np))

np.random.shuffle(s)

feature_np = feature_np[s]
label_np = label_np[s]
```

```
# train / test data 분리

split = 0.15

test_num = int(split*len(label_np))

x_test = feature_np[0:test_num]
y_test = label_np[0:test_num]

x_train = feature_np[test_num:]
y_train = label_np[test_num:]

print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()

model.add(Dense(1, activation='sigmoid', input_shape=(8,) ))
```

```
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=1e-3),
              loss='binary_crossentropy', metrics=['accuracy'])
```

```
from datetime import datetime

start_time = datetime.now()

hist = model.fit(x_train, y_train, epochs=400, validation_data=(x_test, y_test))

end_time = datetime.now()

print('elapsed time => ',end_time-start_time)
```

```
model.evaluate(x_test, y_test)                # 또는 pred = model.predict(...)
```

```
import matplotlib.pyplot as plt

plt.title('loss trend')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.grid()

plt.plot(hist.history['loss'], label='train loss')
plt.plot(hist.history['val_loss'], label='validation loss')
plt.legend(loc='best')

plt.show()
```

```
import matplotlib.pyplot as plt

plt.title('accuracy trend')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.grid()

plt.plot(hist.history['accuracy'], label='train accuracy')
plt.plot(hist.history['val_accuracy'], label='validation accuracy')
plt.legend(loc='best')

plt.show()
```