lab-2-numpy-pandas-typesofdata-2

January 23, 2024

Ratchanon Tarawan 65070503464

1 Lab 2: Numpy, Pandas, and Types of Data

Objectives: - To be more familiar with Numpy and Pandas libraries - To gain more hands-on experience working with different types of data

1.1 [1] Numpy

1.1.1 1.0) import numpy library

[1]:
```
import numpy as np
```

1.1.2 1.1) ndarray initialization

Construct using python list

[2]:
```
# 1d ndarray from 1d python list
list_a1=[1,2,3.5]
arr_a1=np.array(list_a1)
arr_a1
```

[2]:
```
array([1. , 2. , 3.5])
```

[3]:
```
# 2d ndarray from 2d python list (list of list)
list_a2=[[1,2],[3,4],[5,6]]
arr_a2=np.array(list_a2)
arr_a2
```

[3]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

[4]:
```
list_a3=[[[1,2],[2,3]],[[3,4],[4,5]]]
arr_a3=np.array(list_a3)
arr_a3
```

[4]:
```
array([[[1, 2],
        [2, 3]],

       [[3, 4],
        [4, 5]]])
```

or construct using some numpy classes and functions

[5]:
```
np.zeros(5)
```

[5]:
```
array([0., 0., 0., 0., 0.])
```

[6]:
```
np.ones((3,4),dtype=float)
```

[6]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[7]:
```
np.full((4,),999)
```

[7]:
```
array([999, 999, 999, 999])
```

[8]:
```
np.arange(3,10,2)
```

[8]:
```
array([3, 5, 7, 9])
```

[9]:
```
np.linspace(10,15,11)
```

[9]:
```
array([10. , 10.5, 11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. ])
```

[10]:
```
np.random.choice(['a','b'],9)
```

[10]:
```
array(['a', 'b', 'b', 'a', 'a', 'a', 'a', 'a', 'a'], dtype='<U1')
```

[11]:
```
np.random.randn(10)
```

[11]:
```
array([ 0.44104474,  0.13829339, -1.00189576,  0.12488257,  0.64871406,
        0.95528791,  0.34665574, -0.59161612, -1.215097  ,  1.31445034])
```

1.1.3 1.2) ndarray properties

[12]:
```
list_a=[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
arr_a=np.array(list_a)
arr_a
```

[12]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
```

```
       [ 9, 10, 11, 12]])
```

[13]:
```
arr_a.ndim
```

[13]:
```
2
```

[14]:
```
arr_a.shape
```

[14]:
```
(3, 4)
```

[15]:
```
arr_a.dtype
```

[15]:
```
dtype('int64')
```

[16]:
```
arr_a.size
```

[16]:
```
12
```

1.1.4 1.3) Reshaping & Modification

from this original ndarray

[17]:
```
arr_a
```

[17]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to convert into 3D array

[18]:
```
arr_a.reshape((2,2,3))
```

[18]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

sometimes you may resize for same dimension where only known some dimension, insert -1 for unknown len

[19]:
```
arr_a.reshape((-1,6))
```

[19]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
```

Would you like to try this?

[23]:
```
arr_a.reshape((-1,5))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-286d5aa6424c> in <cell line: 1>()
----> 1 arr_a.reshape((-1,5))

ValueError: cannot reshape array of size 12 into shape (5)
```

[Q1] From the above cell, explain in your own words why it worked or did not work.

Ans: Because 5 can't divisible with all element in array

Next, try to append any value(s) into exist 2darray

[21]:
```
np.append(arr_a,13)
```

[21]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13])
```

[24]:
```
np.append(arr_a,arr_a[0])
```

[24]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,  1,  2,  3,  4])
```

[25]:
```
np.append(arr_a,arr_a[0].reshape((1,-1)),axis=0)
```

[25]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [ 1,  2,  3,  4]])
```

[26]:
```
np.append(arr_a,arr_a[:,0].reshape((-1,1)),axis=1)
```

[26]:
```
array([[ 1,  2,  3,  4,  1],
       [ 5,  6,  7,  8,  5],
       [ 9, 10, 11, 12,  9]])
```

[ ]:
```
np.concatenate([arr_a,arr_a])
```

[28]:
```
np.concatenate([arr_a,arr_a],axis=1)
```

[28]:
```
array([[ 1,  2,  3,  4,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  5,  6,  7,  8],
       [ 9, 10, 11, 12,  9, 10, 11, 12]])
```

1.1.5 1.4) indexing & slicing

from this original array again

[29]:
```
arr_a
```

[29]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to access all element at the first row

[30]:
```
arr_a[1]
```

[30]:
```
array([5, 6, 7, 8])
```

then you would like to access the second element from the first row

[32]:
```
arr_a[1][2]
```

[32]:
```
7
```

[31]:
```
arr_a[1,2]
```

[31]:
```
7
```

Next, try to access all element start from 1th in the first row

[34]:
```
arr_a[1,1:]
```

[34]:
```
array([6, 7, 8])
```

[33]:
```
arr_a[:2,1:]
```

[33]:
```
array([[2, 3, 4],
       [6, 7, 8]])
```

sometimes you may specify some row number using list within indicing

[35]:
```
arr_a[[1,2,1],1:]
```

[35]:
```
array([[ 6,  7,  8],
       [10, 11, 12],
       [ 6,  7,  8]])
```

1.1.6 1.5) Boolean slicing

based on this original array

[36]:
```
arr_a
```

[36]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
```

```
       [ 9, 10, 11, 12]])
```

try to filter all elements which more than 5

[37]:
```
arr_a>5
```

[37]:
```
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True]])
```

Next, try to filter all elements which more than 5 and less than 10

[38]:
```
(arr_a>5)&(arr_a<10)
```

[38]:
```
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True, False, False, False]])
```

Run the cell below and answer a question.

[39]:
```
arr_a[(arr_a>5)&(arr_a<10)]
```

[39]:
```
array([6, 7, 8, 9])
```

[Q2] From the above cell, explain in your own words how the output came about?

Ans: Because the array was filter value > 5 and < 10 to boolean array then create new array with condition element

Try running the cell below.

[40]:
```
arr_a[(arr_a>5) and (arr_a<10)]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-40-78eb1746bbfd> in <cell line: 1>()
----> 1 arr_a[(arr_a>5) and (arr_a<10)]

ValueError: The truth value of an array with more than one element is ambiguous
 ↪Use a.any() or a.all()
```

[Q3] Explain in your own words why the above cell gives an error.

Ans: This is not valid for numpy array, it need to use bitwise operation.

[Q4] And what should be written instead so that the code is error-free?

Ans: arr_a[(arr_a>5) & (arr_a<10)]

1.1.7 1.6) Basic operations

[41]:
```
list_b=[[1,2,3,4],[1,2,3,4],[1,2,3,4]]
arr_b=np.array(list_b)
arr_b
```

[41]:
```
array([[1, 2, 3, 4],
       [1, 2, 3, 4],
       [1, 2, 3, 4]])
```

This is some operations for only 1 array

[42]:
```
np.sqrt(arr_b)
```

[42]:
```
array([[1.        , 1.41421356, 1.73205081, 2.        ],
       [1.        , 1.41421356, 1.73205081, 2.        ],
       [1.        , 1.41421356, 1.73205081, 2.        ]])
```

This is some operations for 2 arrays with the same shape

[43]:
```
arr_a-arr_b
```

[43]:
```
array([[0, 0, 0, 0],
       [4, 4, 4, 4],
       [8, 8, 8, 8]])
```

[44]:
```
np.add(arr_a,arr_b)
```

[44]:
```
array([[ 2,  4,  6,  8],
       [ 6,  8, 10, 12],
       [10, 12, 14, 16]])
```

Next, try to operate with 1 array and one numeric variable

[45]:
```
arr_a*3
```

[45]:
```
array([[ 3,  6,  9, 12],
       [15, 18, 21, 24],
       [27, 30, 33, 36]])
```

[46]:
```
1+arr_a**2
```

[46]:
```
array([[  2,   5,  10,  17],
       [ 26,  37,  50,  65],
       [ 82, 101, 122, 145]])
```

Try to play with 2 arrays with different shape

[47]:
```
arr_c=np.array([1,2,3])
arr_d=np.array([[3],[5],[8]])
```

[48]:
```
arr_c-arr_d
```

[48]:
```
array([[-2, -1,  0],
       [-4, -3, -2],
       [-7, -6, -5]])
```

1.1.8 1.7) Basic aggregations

[49]:
```
arr_a
```

[49]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[50]:
```
arr_a.sum()
```

[50]:
```
78
```

[51]:
```
arr_a.mean()
```

[51]:
```
6.5
```

[52]:
```
arr_a.min()
```

[52]:
```
1
```

[53]:
```
arr_a.max()
```

[53]:
```
12
```

[54]:
```
arr_a.std()
```

[54]:
```
3.452052529534663
```

1.1.9 1.8) ndarray axis

[55]:
```
arr_a
```

[55]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[56]:
```
arr_a.sum(axis=0)
```

[56]:
```
array([15, 18, 21, 24])
```

[57]:
```
arr_a.sum(axis=1)
```

[57]:
```
array([10, 26, 42])
```

[Q5] Summarize the value of the argument *axis*, what is the value for row-wise summation and column-wise summation, respectively?

Ans: axis = 0 is column, and axis = 1 is row

2 [2] Pandas

2.0.1 2.0) Series

[58]:
```
import pandas as pd
import numpy as np
```

[59]:
```
pd.Series(np.random.randn(6))
```

[59]:
```
0    0.456768
1    0.488107
2   -0.823382
3    0.888762
4    1.274571
5    1.626542
dtype: float64
```

[60]:
```
pd.Series(np.random.randn(6), index=['a','b','c','d','e','f'])
```

[60]:
```
a   -0.769893
b   -0.343562
c   -0.020193
d   -1.961106
e   -0.653805
f    0.577878
dtype: float64
```

2.0.2 2.1) Constructing Dataframe

Constructing DataFrame from a dictionary

[61]:
```
d = {'col1':[1,2], 'col2': [3,4]}
```

[ ]:
```
df = pd.DataFrame(data=d)
df
```

[64]:
```
d2 = {'Name':['Joe','Nat','Harry','Sam','Monica'],
      'Age': [20,21,19,20,22]}
```

[67]:
```
df2 = pd.DataFrame(data=d2)
df2
```

[67]:

	Name	Age
0	Joe	20
1	Nat	21
2	Harry	19
3	Sam	20
4	Monica	22

Constructing DataFrame from a List

[65]:
```
marks_list = [85.10, 77.80, 91.54, 88.78, 60.55]
```

[66]:
```
df3 = pd.DataFrame(marks_list, columns=['Marks'])
df3
```

[66]:

	Marks
0	85.10
1	77.80
2	91.54
3	88.78
4	60.55

Creating DataFrame from file

[81]:
```
# Read csv file from path and store to df for create dataframe
df = pd.read_csv('nss15.csv')
```

[79]:
```
df
```

[79]:

	col1	col2
0	1	3
1	2	4

2.0.3 2.2) Viewing DataFrame information

(.shape, .head, .tail, .info, select column, .unique, .describe, select low with .loc and .iloc)

Check simple information

[69]:
```
# Check dimension by .shape
df.shape
```

[69]: (2, 2)

[70]:
```
# Display the first 5 rows by default
df.head()
```

[70]:

	col1	col2
0	1	3
1	2	4

[71]:
```
# Display the first 3 rows
df.head(3)
```

[71]:

	col1	col2
0	1	3
1	2	4

[72]:
```
# Display the last 5 rows by default
df.tail()
```

[72]:

	col1	col2
0	1	3
1	2	4

[73]:
```
# Overview information of dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   col1    2 non-null      int64
 1   col2    2 non-null      int64
dtypes: int64(2)
memory usage: 160.0 bytes
```

Select column, multiple column, with condition

[74]:
```
df.columns
```

[74]:
```
Index(['col1', 'col2'], dtype='object')
```

[83]:
```
#select single column
df['age']
```

[83]:
```
0     5
1    36
2    20
3    61
4    88
```

```
          ..
18001     6
18002    79
18003    16
18004     4
18005    33
Name: age, Length: 18006, dtype: int64
```

[82]:
```
df.age
```

[82]:
```
0         5
1        36
2        20
3        61
4        88
          ..
18001     6
18002    79
18003    16
18004     4
18005    33
Name: age, Length: 18006, dtype: int64
```

[84]:
```
#select multiple column
df[['treatmentDate','statWeight','age','sex']]
```

[84]:

	treatmentDate	statWeight	age	sex
0	7/11/2015	15.7762	5	Male
1	7/6/2015	83.2157	36	Male
2	8/2/2015	74.8813	20	Female
3	6/26/2015	15.7762	61	Male
4	7/4/2015	74.8813	88	Female
…	…	…	…	…
18001	1/3/2015	5.6748	6	Female
18002	4/17/2015	97.9239	79	Male
18003	4/15/2015	83.2157	16	Male
18004	4/15/2015	83.2157	4	Male
18005	1/22/2015	49.2646	33	M

[18006 rows x 4 columns]

Viewing the unique value

[85]:
```
df.race.unique()
```

[85]:
```
array([nan, 'White', 'Other', 'Black', 'Asian', 'American Indian'],
      dtype=object)
```

Describe

[86]:
```
df['age'].describe()
```

[86]:
```
count    18006.000000
mean        31.035155
std         25.818940
min          1.000000
25%         10.000000
50%         23.000000
75%         51.000000
max        104.000000
Name: age, dtype: float64
```

Select row with condition

[87]:
```
#select by condition
df[df['sex'] == 'Male']
```

[87]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	\
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	
1	150734723	7/6/2015	83.2157	S	36	Male	White	
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	
6	150713483	6/8/2015	15.7762	V	25	Male	Black	
7	150704114	6/14/2015	83.2157	S	53	Male	White	
…	…	…	…	…	…	…	…	
17995	150142379	1/16/2015	15.0591	V	37	Male	White	
17999	150318885	2/23/2015	15.7762	V	47	Male	NaN	
18002	150448416	4/17/2015	97.9239	M	79	Male	White	
18003	150439048	4/15/2015	83.2157	S	16	Male	NaN	
18004	150439159	4/15/2015	83.2157	S	4	Male	White	

	diagnosis	bodyPart	disposition	location	product
0	57.0	33.0	1.0	9.0	1267.0
1	57.0	34.0	1.0	1.0	1439.0
3	71.0	35.0	1.0	0.0	611.0
6	51.0	33.0	4.0	9.0	1138.0
7	57.0	30.0	1.0	0.0	5040.0
…	…	…	…	…	…
17995	53.0	35.0	1.0	1.0	4057.0
17999	59.0	75.0	1.0	0.0	713.0
18002	64.0	35.0	1.0	1.0	1615.0
18003	57.0	93.0	1.0	4.0	5040.0
18004	53.0	76.0	1.0	5.0	1679.0

[9883 rows x 12 columns]

[88]:
```
#select by multiple condition
df[(df['sex'] == 'Male') & (df['age'] > 80)]
```

[88]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
8	150736558	7/16/2015	83.2157	S	98	Male	Black	59.0	76.0	1.0	1.0	1807.0
63	150418623	1/12/2015	15.0591	V	97	Male	Other	62.0	75.0	4.0	1.0	4076.0
97	150700375	6/28/2015	83.2157	S	85	Male	NaN	59.0	92.0	1.0	0.0	478.0
131	150940801	9/14/2015	15.7762	V	96	Male	NaN	62.0	75.0	1.0	5.0	1807.0
177	160110774	12/19/2015	85.7374	S	81	Male	White	59.0	82.0	1.0	1.0	3278.0
…	…	…	…	…	…	…	…	…	…	…	…	…
17768	150965066	9/21/2015	15.7762	V	91	Male	NaN	62.0	75.0	4.0	1.0	1842.0
17797	151214248	12/3/2015	74.8813	L	92	Male	White	71.0	30.0	1.0	0.0	1842.0
17843	150834600	8/12/2015	15.7762	V	85	Male	NaN	71.0	85.0	4.0	1.0	1439.0
17857	150749325	7/12/2015	15.7762	V	84	Male	NaN	62.0	75.0	1.0	5.0	1842.0
17974	150526714	5/9/2015	15.0591	V	88	Male	NaN	57.0	31.0	1.0	1.0	1842.0

[314 rows x 12 columns]

Select row with .iloc

[90]:
```
# select row by .iloc
df.iloc[10:15]
```

[90]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
10	150734952	7/4/2015	15.7762	V	20	Male	Black	59.0	82.0	1.0	1.0	1894.0
11	150821622	7/20/2015	83.2157	S	20	Female	White	57.0	36.0	1.0	9.0	1267.0
12	150713631	7/4/2015	15.7762	V	11	Male	NaN	60.0	88.0	1.0	0.0	3274.0
13	150666343	6/27/2015	15.7762	V	26	Female	White					
14	150748843	7/16/2015	37.6645	L	33	Male	Asian					

```
13       62.0      75.0          1.0       1.0   1807.0
14       53.0      93.0          1.0       1.0   4057.0
```

[89]:
```
# select column by .iloc
df.iloc[:,[0,1,2,3,4]]
```

[89]:
```
       caseNumber treatmentDate  statWeight stratum  age
0       150733174     7/11/2015     15.7762       V    5
1       150734723      7/6/2015     83.2157       S   36
2       150817487      8/2/2015     74.8813       L   20
3       150717776     6/26/2015     15.7762       V   61
4       150721694      7/4/2015     74.8813       L   88
…             …            …           …      …   …
18001   150116636      1/3/2015      5.6748       C    6
18002   150448416     4/17/2015     97.9239       M   79
18003   150439048     4/15/2015     83.2157       S   16
18004   150439159     4/15/2015     83.2157       S    4
18005   150534711     1/22/2015     49.2646       M   33

[18006 rows x 5 columns]
```

Select column and row with .loc

[91]:
```
# select column and low by .loc
df.loc[:6,'treatmentDate':'diagnosis']
```

[91]:
```
  treatmentDate  statWeight stratum  age     sex   race  diagnosis
0     7/11/2015     15.7762       V    5    Male    NaN       57.0
1      7/6/2015     83.2157       S   36    Male  White       57.0
2      8/2/2015     74.8813       L   20  Female    NaN       71.0
3     6/26/2015     15.7762       V   61    Male    NaN       71.0
4      7/4/2015     74.8813       L   88  Female  Other       62.0
5      7/2/2015      5.6748       C    1  Female  White       71.0
6      6/8/2015     15.7762       V   25    Male  Black       51.0
```

[92]:
```
# select row by condition
df.loc[df['age']>80, ['treatmentDate', 'age']]
```

[92]:
```
      treatmentDate  age
4          7/4/2015   88
8         7/16/2015   98
39         5/3/2015   88
46        4/15/2015   91
63        1/12/2015   97
…               …   …
17852     5/16/2015   86
17857     7/12/2015   84
```

```
17957     10/3/2015   85
17974      5/9/2015   88
17998     1/16/2015   91

[1043 rows x 2 columns]
```

[Q6] What is the difference between .iloc and .loc?

Answer loc gets rows (and/or columns) with particular labels. Where as, iloc gets rows (and/or columns) at integer locations.

3 [3] Various Types of Data

3.0.1 3.0) HTML

[93]:
```
from bs4 import BeautifulSoup
```

[94]:
```
html_temp = """
<!DOCTYPE html>
<html>
<head>
    <title>Sample Blog</title>
</head>
<body>
    <h2 class="article-title">Article 1: Introduction to Web Scraping</h2>
    <p class="article-content">This is an introduction to web scraping using␣
 ↪BeautifulSoup.</p>
    <h2 class="article-title">Article 2: Advanced Web Scraping Techniques</h2>
    <p class="article-content">Learn advanced techniques for web scraping with␣
 ↪Python.</p>
</body>
</html>
"""

with open('html_file.html', 'w') as file:
    file.write(html_temp)
```

[98]:
```
with open('html_file.html') as html_file:
    html_content = html_file.read()

# Parse the HTML content
soup = BeautifulSoup(html_content, 'html.parser')

print(soup.title.text)
print(soup.h2)
print(soup.p.text)
```

```
Sample Blog
```

```
<h2 class="article-title">Article 1: Introduction to Web Scraping</h2>
This is an introduction to web scraping using BeautifulSoup.
```

[Q7] Explain why the code above gives an error? Fix the code so that it runs without error.

Ans: There is no table tah in HTML code provide, change to print P tag.

3.0.2 3.1) XML

[146]:
```python
import xml.etree.ElementTree as ET

#writing new xml file
root = ET.Element("data")
student = ET.SubElement(root, "student", name = "Chanon")

email = ET.SubElement(student, 'email')
email.text = "chanon@mail.com"

age = ET.SubElement(student, 'age')
age.text = "21"

gender = ET.SubElement(student, 'gender')
gender.text = "M"

tree = ET.ElementTree(root)
tree.write("xml_file.xml")
```

[147]:
```python
#modifying existing xml file
tree = ET.parse('xml_file.xml')
root = tree.getroot()

for student in root:
    for element in student:
        if element.tag == "age":
            element.text = "22"

tree.write('xml_file.xml')
```

[148]:
```python
# #reading XML file
# tree = ET.parse('xml_file.xml')
# root = tree.getroot()

# for student in root:
#     print(f'name: {student.attrib["name"]}')
#     for element in student:
#         print(f'{element.tag}: {element.text}')

# # Print the entire XML content
```

```
# xml_content = ET.tostring(root, encoding='utf-8').decode('utf-8')
# print(xml_content)
```

[151]:
```
#convert XML to List of Dictionary
data_list = []
for line in root:
    name = line.attrib.get('name')
    email = line.find('email').text
    age = line.find('age').text
    gender = line.find('gender').text

    # data_list.append({"Name":name, "Email":email, "Age":age, "Gender":gender})

print(data_list)
```

```
[]
```

[Q8] Add your own data including Name, Email, Age and Gender to the XML file and put it in the existing data_list [You should show the data_list and XML file by reading the file]

[152]:
```
#Add you own code here
# data_list.append({"Name":'Ratchanon', "Email":'bill.ratchanon@gmail.com',␣
 ↪"Age":'19', "Gender":'M'})

root = ET.Element("data")
student = ET.SubElement(root, "student", name = "Chanon")

email = ET.SubElement(student, 'email')
email.text = "chanon@mail.com"

age = ET.SubElement(student, 'age')
age.text = "21"

gender = ET.SubElement(student, 'gender')
gender.text = "M"

tree = ET.ElementTree(root)
student = ET.SubElement(root, "student", name = "Ratchanon")

email = ET.SubElement(student, 'email')
email.text = "bill.ratchanon@gmail.com"

age = ET.SubElement(student, 'age')
age.text = "19"

gender = ET.SubElement(student, 'gender')
gender.text = "M"
```

```
tree = ET.ElementTree(root)

for line in root:
    name = line.attrib.get('name')
    email = line.find('email').text
    age = line.find('age').text
    gender = line.find('gender').text
    data_list.append({"Name":name, "Email":email, "Age":age, "Gender":gender})

xml_content = ET.tostring(root, encoding='utf-8').decode('utf-8')
print(xml_content)
print(data_list)
```

```
<data><student name="Chanon"><email>chanon@mail.com</email><age>21</age><gender>
M</gender></student><student name="Ratchanon"><email>bill.ratchanon@gmail.com</e
mail><age>19</age><gender>M</gender></student></data>
[{'Name': 'Chanon', 'Email': 'chanon@mail.com', 'Age': '21', 'Gender': 'M'},
{'Name': 'Ratchanon', 'Email': 'bill.ratchanon@gmail.com', 'Age': '19',
'Gender': 'M'}]
```

3.0.3 3.2) JSON

[113]:
```
#writing new json file
import json

# Data to be written to the JSON file
data_to_write = {
    "people": [
        {"name": "Alice", "age": 30, "city": "New York"},
        {"name": "Bob", "age": 25, "city": "San Francisco"},
        {"name": "Charlie", "age": 35, "city": "Los Angeles"}
    ]
}

# Open the file in write mode and write the data
with open('json_file.json', 'w') as json_file:
    json.dump(data_to_write, json_file, indent=2)
```

[114]:
```
#reading json file
with open('json_file', 'r') as file:
    # Load JSON data
    data = json.load(file)

print(data)

people = data['people']
```

```
# Print information about each person
for person in people:
    print(f"Name: {person['name']}, Age: {person['age']}, City:␣
 ↪{person['city']}")
```

```
{'people': [{'name': 'Alice', 'age': 30, 'city': 'New York'}, {'name': 'Bob',
'age': 25, 'city': 'San Francisco'}, {'name': 'Charlie', 'age': 35, 'city': 'Los
Angeles'}]}
Name: Alice, Age: 30, City: New York
Name: Bob, Age: 25, City: San Francisco
Name: Charlie, Age: 35, City: Los Angeles
```

[Q9] write a code to modify the existing json file so each person have a "job" data and print the result

Ans:

[117]:
```
#write your own code here

for person in data['people']:
  person['job'] = 'Owner'

for person in people:
    print(f"Name: {person['name']} Job: {person['job']}, Age: {person['age']},␣
 ↪City: {person['city']}")
```

```
Name: Alice Job: Owner, Age: 30, City: New York
Name: Bob Job: Owner, Age: 25, City: San Francisco
Name: Charlie Job: Owner, Age: 35, City: Los Angeles
```