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] Numpy

1.0) import numpy library

```
import numpy as np
```

1.1) ndarray initialization

Construct using python list

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

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

```
# 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
```

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

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

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

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

or construct using some numpy classes and functions

```
np.zeros(5)
```

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

```
np.ones((3,4),dtype=float)
```

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

```
np.full((4,),999)
```

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

```
np.arange(3,10,2)
```

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

```
np.linspace(10,15,11)
```

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

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

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

```
np.random.randn(10)
```

```
array([ 0.67082814, -0.06298677,  0.2061625 , -2.0029053 ,  0.22687337,
       -0.41573847,  2.55240552,  1.30317491,  0.68781846, -0.70609415])
```

1.2) ndarray properties

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

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

```
arr_a.ndim
```

```
2
```

```
arr_a.shape
```

```
(3, 4)
```

```
arr_a.dtype
```

```
dtype('int64')
```

```
arr_a.size
```

```
12
```

1.3) Reshaping & Modification

from this original ndarray

```
arr_a
```

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

try to convert into 3D array

```
arr_a.reshape((2,2,3))
```

```
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

```
arr_a.reshape((-1,6))
```

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

Would you like to try this?

```
arr_a.reshape((-1,5))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-286d5aa6424c> in <cell line: 1>()
----> 1 arr_a.reshape((-1,5))

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

SEARCH STACK OVERFLOW

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

Ans: It did not work, the dimention cannot be shape to 5 which mean 5 columns because it would be 2 number left and not fulfill the whole row.

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

```
np.append(arr_a,13)
```

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

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

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

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

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

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

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

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

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

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

```
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]])
```

1.4) indexing & slicing

from this original array again

```
arr_a
```

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

try to access all element at the first row

```
arr_a[1]
```

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

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

```
arr_a[1][2]
```

```
7
```

```
arr_a[1,2]
```

```
7
```

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

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

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

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

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

sometimes you may specify some row number using list within indicing

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

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

1.5) Boolean slicing

based on this original array

```
arr_a
```

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

try to filter all elements which more than 5

```
arr_a>5
```

```
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

```
(arr_a>5)&(arr_a<10)
```

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

Run the cell below and answer a question.

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

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

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

Ans: The output is come from the result of (arr_a>5)&(arr_a<10) which filter out only the True answer which is number more than 5 but less than 10 and then use to create a subarray.

Try running the cell below.

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-34-78eb1746bbfd> 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()
```

SEARCH STACK OVERFLOW

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

Ans: The cell above is error because it use "and" which can only do the single Boolean evaluation and not suit in this code.

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

Ans: The "&" symbol should be use instead to make the code error-free.

1.6) Basic operations

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

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

This is some operations for only 1 array

```
np.sqrt(arr_b)
```

```
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

```
arr_a-arr_b
```

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

```
np.add(arr_a,arr_b)
```

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

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

```
arr_a*3
```

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

```
1+arr_a**2
```

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

Try to play with 2 arrays with different shape

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

```
arr_c-arr_d
```

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

1.7) Basic aggregations

```
arr_a
```

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

```
arr_a.sum()
```

```
78
```

```
arr_a.mean()
```

```
6.5
```

```
arr_a.min()
```

```
1
```

```
arr_a.max()
```

```
12
```

```
arr_a.std()
```

```
3.452052529534663
```

1.8) ndarray axis

```
arr_a
```

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

```
arr_a.sum(axis=0)
```

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

```
arr_a.sum(axis=1)
```

```
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: The value for row-wise summation is array([10, 26, 42]) and the value of column-wise summation is array([15, 18, 21, 24]) respectively.

[2] Pandas

2.0) Series

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

```
pd.Series(np.random.randn(6))
```

```
0   -0.583343
1    0.145128
2   -0.361085
3    0.506660
4   -0.487421
5   -0.002738
dtype: float64
```

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

```
a   -0.628367
b    1.744706
c    0.366284
d   -0.427291
e   -0.475990
f   -0.065848
dtype: float64
```

2.1) Constructing Dataframe

Constructing DataFrame from a dictionary

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

```
df = pd.DataFrame(data=d)
df
```

	col1	col2
0	1	3
1	2	4

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

```
df2 = pd.DataFrame(data=d2)
df2
```

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

Constructing DataFrame from a List

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

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

	Marks
0	85.10
1	77.80
2	91.54
3	88.78
4	60.55

Creating DataFrame from file

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

```
df
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	boc
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	
...	...	...	...	...	...	...	...	...	
36008	151200344	11/12/2015	49.2646	M	13	Female	Other	53	
36009	151148927	11/11/2015	97.9239	M	10	Male	NaN	59	
36010	151149054	10/15/2015	5.6748	C	4	Male	White	62	
36011	151154817	11/24/2015	85.7374	S	1	Female	White	50	
36012	151246262	12/19/2015	16.5650	V	14	Male	NaN	64	

36013 rows × 12 columns

2.2) Viewing DataFrame information

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

Check simple information

```
# Check dimension by .shape
df.shape
```

```
(36013, 12)
```

```
# Display the first 5 rows by default
df.head()
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPar
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	3
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	3
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	9
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	3
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	7

```
# Display the first 3 rows
df.head(3)
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPar
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	3
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	3
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	9

```
# Display the last 5 rows by default
df.tail()
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	boc
36008	151200344	11/12/2015	49.2646	M	13	Female	Other	53	
36009	151148927	11/11/2015	97.9239	M	10	Male	NaN	59	
36010	151149054	10/15/2015	5.6748	C	4	Male	White	62	
36011	151154817	11/24/2015	85.7374	S	1	Female	White	50	
36012	151246262	12/19/2015	16.5650	V	14	Male	NaN	64	

```
# Overview information of dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36013 entries, 0 to 36012
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   caseNumber     36013 non-null  int64
 1   treatmentDate  36013 non-null  object
 2   statWeight     36013 non-null  float64
 3   stratum        36013 non-null  object
 4   age            36013 non-null  int64
 5   sex            36013 non-null  object
 6   race           22060 non-null  object
 7   diagnosis      36013 non-null  int64
 8   bodyPart       36013 non-null  int64
 9   disposition    36013 non-null  int64
 10  location       36013 non-null  int64
 11  product        36013 non-null  int64
dtypes: float64(1), int64(7), object(4)
memory usage: 3.3+ MB
```

Select column, multiple column, with condition

```
df.columns
```

```
Index(['caseNumber', 'treatmentDate', 'statWeight', 'stratum', 'age', 'sex',
       'race', 'diagnosis', 'bodyPart', 'disposition', 'location', 'product'],
      dtype='object')
```

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

```
0         5
1        36
2        20
3        61
4        88
         ..
36008    13
36009    10
36010     4
36011     1
36012    14
Name: age, Length: 36013, dtype: int64
```

```
df.age
```

```
0         5
1        36
2        20
3        61
4        88
```

```
          ..
36008    13
36009    10
36010     4
36011     1
36012    14
Name: age, Length: 36013, dtype: int64
```

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

	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
...	...	...	...	...
36008	11/12/2015	49.2646	13	Female
36009	11/11/2015	97.9239	10	Male
36010	10/15/2015	5.6748	4	Male
36011	11/24/2015	85.7374	1	Female
36012	12/19/2015	16.5650	14	Male

36013 rows × 4 columns

Viewing the unique value

```
df.race.unique()
```

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

Describe

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

```
count    36013.000000
mean        31.090662
std         25.987271
min          0.000000
25%         10.000000
50%         23.000000
75%         51.000000
max        104.000000
Name: age, dtype: float64
```

Select row with condition

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyP
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	
6	150713483	6/8/2015	15.7762	V	25	Male	Black	51	
7	150704114	6/14/2015	83.2157	S	53	Male	White	57	
...	...	...	...	...	...	...	...	...	
36006	151213194	12/2/2015	4.9655	C	8	Male	Other	60	
36007	151146425	11/2/2015	74.8813	L	20	Male	White	55	
36009	151148927	11/11/2015	97.9239	M	10	Male	NaN	59	
36010	151149054	10/15/2015	5.6748	C	4	Male	White	62	
36012	151246262	12/19/2015	16.5650	V	14	Male	NaN	64	

19642 rows × 12 columns

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyP
8	150736558	7/16/2015	83.2157	S	98	Male	Black	59	
63	150418623	1/12/2015	15.0591	V	97	Male	Other	62	
97	150700375	6/28/2015	83.2157	S	85	Male	NaN	59	
131	150940801	9/14/2015	15.7762	V	96	Male	NaN	62	
177	160110774	12/19/2015	85.7374	S	81	Male	White	59	
...	...	...	...	...	...	...	...	...	
35873	150559445	5/27/2015	80.8381	S	87	Male	White	57	
35881	150857901	4/26/2015	83.2157	S	100	Male	NaN	59	
35884	150938743	9/14/2015	15.7762	V	87	Male	NaN	71	
35911	150759868	7/29/2015	83.2157	S	84	Male	Black	59	
35957	150920074	9/6/2015	97.9239	M	84	Male	NaN	53	

671 rows × 12 columns

Select row with .iloc

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPa
10	150734952	7/4/2015	15.7762	V	20	Male	Black	59	
11	150821622	7/20/2015	83.2157	S	20	Female	White	57	
12	150713631	7/4/2015	15.7762	V	11	Male	NaN	60	
13	150666343	6/27/2015	15.7762	V	26	Female	White	62	
14	150748843	7/16/2015	37.6645	L	33	Male	Asian	53	

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

	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
...	...	...	...	...	...
36008	151200344	11/12/2015	49.2646	M	13
36009	151148927	11/11/2015	97.9239	M	10
36010	151149054	10/15/2015	5.6748	C	4
36011	151154817	11/24/2015	85.7374	S	1
36012	151246262	12/19/2015	16.5650	V	14

36013 rows × 5 columns

Select column and row with .loc

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

	treatmentDate	statWeight	stratum	age	sex	race	diagnosis
0	7/11/2015	15.7762	V	5	Male	NaN	57
1	7/6/2015	83.2157	S	36	Male	White	57
2	8/2/2015	74.8813	L	20	Female	NaN	71
3	6/26/2015	15.7762	V	61	Male	NaN	71
4	7/4/2015	74.8813	L	88	Female	Other	62
5	7/2/2015	5.6748	C	1	Female	White	71
6	6/8/2015	15.7762	V	25	Male	Black	51

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

	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
...	...	...
35925	6/18/2015	83
35952	9/21/2015	81
35957	9/6/2015	84
35993	12/11/2015	84
35994	11/2/2015	82

2138 rows × 2 columns

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

Ans: the .iloc use numbers of rows and columns but .loc use a specific information like the name to select and sort the data.

[3] Various Types of Data

3.0) HTML

```
from bs4 import BeautifulSoup


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)



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)
```

```
Sample Blog
<h2 class="article-title">Article 1: Introduction to Web Scraping</h2>
```

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

Ans: The code error because there are no table to be print from the code earlier so it return as 'NoneType' instead.

3.1) XML

```
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")
```

```
#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')


#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)
```

```
name: Chanon
email: chanon@mail.com
age: 22
gender: M
<data><student name="Chanon"><email>chanon@mail.com</email><age>22</age><gender>M</gender></student></data>
```

```
#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)
```

```
[{'Name': 'Chanon', 'Email': 'chanon@mail.com', 'Age': '22', 'Gender': 'M'}]
```

[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]

```
new_data = ET.SubElement(root, "student", name="Pitchayapat")

email = ET.SubElement(new_data, 'email')
email.text = "pitchayapat.ware@kmutt.ac.th"

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

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

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)
```

```
[{'Name': 'Chanon', 'Email': 'chanon@mail.com', 'Age': '22', 'Gender': 'M'}, {'Name': 'Pitchayapat', 'Email': 'pitchayapat.ware@kmutt.ac
```

```
#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)
```

```
#reading json file
with open('json_file.json', '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', '
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:

```
with open('json_file.json', 'r') as file:
    data = json.load(file)
```