

# lab-2-numpy-pandas-typesofdata

January 24, 2024

## 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

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

#### 1.1.2 1.1) ndarray initialization

Construct using python list

```
[99]: # 1d ndarray from 1d python list
list_a1=[1,2,3.5]
arr_a1=np.array(list_a1)
arr_a1
```

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

```
[100]: # 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
```

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

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

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

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

or construct using some numpy classes and functions

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

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

```
[103]: np.ones((3,4),dtype=float) # fill with 1
```

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

```
[104]: np.full((4,),999) # fill with 999 as 1d dimension size 4
```

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

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

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

```
[106]: np.linspace(10,15,11) # Return evenly spaced numbers over a specified interval.↵  
    ↪with 11 numbers
```

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

```
[107]: np.random.choice(['a','b'],9) # random for 9 members
```

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

```
[108]: np.random.randn(10) # sampled from a standard normal distribution (a normal↵  
    ↪distribution with mean 0 and standard deviation 1).
```

```
[108]: array([-0.46945683, -0.40593038,  0.21175247, -0.4080758 ,  0.53054609,  
             0.07203192,  0.74650017, -1.88721041,  0.67727674, -1.3142227 ])
```

### 1.1.3 1.2) ndarray properties

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

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

```
[110]: arr_a.ndim # How many dimension are there for list_a
```

```
[110]: 2
```

```
[111]: arr_a.shape
```

```
[111]: (3, 4)
```

```
[112]: arr_a.dtype
```

```
[112]: dtype('int32')
```

```
[113]: arr_a.size
```

```
[113]: 12
```

### 1.1.4 1.3) Reshaping & Modification

from this original ndarray

```
[114]: arr_a # (3,4) row column
```

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

try to convert into 3D array

```
[115]: arr_a.reshape((2,2,3)) # reshape to 3d array with 2 layer 2 row 3 columns
```

```
[115]: 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

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

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

Would you like to try this?

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

```

-----
ValueError                                Traceback (most recent call last)
Cell In[117], 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: It didn't work because of the numbers in set. it can't fit within 5 columns. It can fit but it can't be equally distributed.

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

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

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

```
[119]: arr_a
```

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

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

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

```
[121]: arr_a[0].reshape((1,-1))
```

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

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

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

```
[123]: arr_a[:,0] # select all rows, elements from first column
arr_a[:,0].reshape((-1,1))
```

```
[123]: array([[1],
              [5],
              [9]])
```

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

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

```
[125]: np.concatenate([arr_a,arr_a]) # merge 2 array together
```

```
[125]: 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]])
```

```
[126]: np.concatenate([arr_a,arr_a],axis=1) # add in the second axis
```

```
[126]: 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

```
[127]: arr_a
```

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

try to access all element at the second row

```
[128]: arr_a[1]
```

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

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

```
[129]: arr_a[1][2]
```

```
[129]: 7
```

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

```
[130]: 7
```

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

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

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

```
[ ]:
```

```
[132]: arr_a[:2,1:] # from any row to row 1, start from second element
```

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

sometimes you may specify some row number using list within indexing

```
[133]: arr_a[[1,2,1]] # repeat row 2
```

```
[133]: array([[ 5,  6,  7,  8],  
            [ 9, 10, 11, 12],  
            [ 5,  6,  7,  8]])
```

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

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

### 1.1.6 1.5) Boolean slicing

based on this original array

```
[135]: arr_a
```

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

try to filter all elements which more than 5

```
[136]: arr_a>5
```

```
[136]: 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

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

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

Run the cell below and answer a question.

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

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

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

Ans: After getting boolean mask from the conditions (whether if it is between 6 - 9 or not), accessing the element with the boolean mask, it will filter all the element that's in the conditions.

Try running the cell below.

```
[139]: (arr_a>5).dtype
```

```
[139]: dtype('bool')
```

```
[140]: ((arr_a>5) and (arr_a<10)).dtype
```

```
-----
ValueError                                Traceback (most recent call last)
Cell In[140], line 1
----> 1 ((arr_a>5) and (arr_a<10)).dtype

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

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

```
-----
ValueError                                Traceback (most recent call last)
Cell In[141], 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: “and” works as a logical operator. It doesn't compare element-wise in the variable like bitwise operator (&). Moreover, there are different truth value in the resulted boolean mask. So, this masking operation failed because of different functionalities of “and” and “&”.

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

Ans: Instead, replace “and” with “&” bitwise operator.

### 1.1.7 1.6) Basic operations

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

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

This is some operations for only 1 array

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

```
[143]: 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

```
[144]: arr_a-arr_b
```

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

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

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

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

```
[146]: arr_a*3
```

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

```
[147]: 1+arr_a**2
```

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

Try to play with 2 arrays with different shape



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

```
[149]: arr_c-arr_d
```

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

### 1.1.8 1.7) Basic aggregations

```
[150]: arr_a
```

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

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

```
[151]: 78
```

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

```
[152]: 6.5
```

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

```
[153]: 1
```

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

```
[154]: 12
```

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

```
[155]: 3.452052529534663
```

### 1.1.9 1.8) ndarray axis

```
[156]: arr_a
```

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

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

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

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

```
[158]: 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: If parameter “axis=0”, it means that it calculates along the chosen axis (row). So it calculates each element in the column. In the same manner, axis = 1 means that we choose to calculate along the column. So, it calculates each element in the row. Summary: axis = 0 -> column-wise summation axis = 1 -> row-wise summation <https://i.stack.imgur.com/Z29Nn.jpg>

## 2 [2] Pandas

### 2.0.1 2.0) Series

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

```
[160]: pd.Series(np.random.randn(6)) # label
```

```
[160]: 0    -1.172753  
1     0.423567  
2    -0.398816  
3    -0.146362  
4    -0.534897  
5     0.800737  
dtype: float64
```

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

```
[161]: a     1.914202  
b    -2.984832  
c    -1.139446  
d    -2.455952  
e     0.247220  
f    -0.248478  
dtype: float64
```

### 2.0.2 2.1) Constructing Dataframe

Constructing DataFrame from a dictionary

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

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

```
[163]:    col1  col2
0      1      3
1      2      4
```

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

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

```
[165]:    Name  Age
0     Joe   20
1     Nat   21
2   Harry   19
3     Sam   20
4  Monica   22
```

Constructing DataFrame from a List

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

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

```
[167]:    Marks
0   85.10
1   77.80
2   91.54
3   88.78
4   60.55
```

Creating DataFrame from file

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

```
[169]: df
```

```
[169]:    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
2    150817487    8/2/2015    74.8813      L   20 Female  NaN
3    150717776    6/26/2015    15.7762      V   61  Male  NaN
4    150721694    7/4/2015    74.8813      L   88 Female  Other
...
```

334834	150739278	5/31/2015	15.0591	V	7	Male	NaN
334835	150733393	7/11/2015	5.6748	C	3	Female	Black
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN
334837	150823002	8/8/2015	97.9239	M	38	Female	White
334838	150723074	6/20/2015	49.2646	M	5	Female	White

	diagnosis	bodyPart	disposition	location	product
0	57	33	1	9	1267
1	57	34	1	1	1439
2	71	94	1	0	3274
3	71	35	1	0	611
4	62	75	1	0	1893
...	...	...	...	...	...
334834	59	76	1	1	1864
334835	68	85	1	0	1931
334836	71	79	1	0	3250
334837	59	82	1	1	464
334838	57	34	1	9	3273

[334839 rows x 12 columns]

## 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

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

```
[170]: (334839, 12)
```

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

```
[171]:
```

	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	
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	
4	150721694	7/4/2015	74.8813	L	88	Female	Other	

	diagnosis	bodyPart	disposition	location	product
0	57	33	1	9	1267
1	57	34	1	1	1439
2	71	94	1	0	3274
3	71	35	1	0	611
4	62	75	1	0	1893

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

```
[172]:
```

	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	
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	

  

	diagnosis	bodyPart	disposition	location	product
0	57	33	1	9	1267
1	57	34	1	1	1439
2	71	94	1	0	3274

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

```
[173]:
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	\
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	

  

	diagnosis	bodyPart	disposition	location	product
334834	59	76	1	1	1864
334835	68	85	1	0	1931
334836	71	79	1	0	3250
334837	59	82	1	1	464
334838	57	34	1	9	3273

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 334839 entries, 0 to 334838
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---  -
0   caseNumber      334839 non-null  int64
1   treatmentDate   334839 non-null  object
2   statWeight      334839 non-null  float64
3   stratum         334839 non-null  object
4   age             334839 non-null  int64
5   sex             334837 non-null  object
6   race            205014 non-null  object
7   diagnosis       334839 non-null  int64
8   bodyPart        334839 non-null  int64
```

```
9   disposition      334839 non-null  int64
10  location          334839 non-null  int64
11  product           334839 non-null  int64
dtypes: float64(1), int64(7), object(4)
memory usage: 30.7+ MB
```

Select column, multiple column, with condition

```
[175]: df.columns
```

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

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

```
[176]: 0         5
1        36
2        20
3        61
4        88
      ..
334834    7
334835    3
334836   38
334837   38
334838    5
Name: age, Length: 334839, dtype: int64
```

```
[177]: df.age
```

```
[177]: 0         5
1        36
2        20
3        61
4        88
      ..
334834    7
334835    3
334836   38
334837   38
334838    5
Name: age, Length: 334839, dtype: int64
```

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

```
[178]:      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
...
334834    5/31/2015      15.0591     7    Male
334835    7/11/2015       5.6748     3  Female
334836    7/24/2015      15.7762    38    Male
334837    8/8/2015      97.9239    38  Female
334838    6/20/2015      49.2646     5  Female
```

[334839 rows x 4 columns]

Viewing the unique value

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

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

Describe

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

```
[180]: count      334839.000000
mean         31.385451
std          26.105098
min           0.000000
25%          10.000000
50%          23.000000
75%          51.000000
max          107.000000
Name: age, dtype: float64
```

Select row with condition

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

```
[181]:      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
...      ...      ...      ...      ...      ...
```

334824	150607827	5/27/2015	5.6748	C	1	Male	White
334825	150600190	5/28/2015	80.8381	S	5	Male	NaN
334833	150747217	7/24/2015	83.2157	S	2	Male	NaN
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN

	diagnosis	bodyPart	disposition	location	product
0	57	33	1	9	1267
1	57	34	1	1	1439
3	71	35	1	0	611
6	51	33	4	9	1138
7	57	30	1	0	5040
...	...	...	...	...	...
334824	71	36	1	1	1807
334825	56	94	1	0	1936
334833	62	75	1	1	1301
334834	59	76	1	1	1864
334836	71	79	1	0	3250

[182501 rows x 12 columns]

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

```
[182]: caseNumber treatmentDate statWeight stratum age sex race \
8 150736558 7/16/2015 83.2157 S 98 Male Black
63 150418623 1/12/2015 15.0591 V 97 Male Other
97 150700375 6/28/2015 83.2157 S 85 Male NaN
131 150940801 9/14/2015 15.7762 V 96 Male NaN
177 160110774 12/19/2015 85.7374 S 81 Male White
...
334616 160104368 12/30/2015 74.8813 L 86 Male Other
334677 151115099 11/4/2015 16.5650 V 83 Male NaN
334699 150633387 5/29/2015 74.8813 L 84 Male NaN
334701 150515945 4/27/2015 97.9239 M 86 Male NaN
334785 150733286 7/11/2015 15.7762 V 86 Male White
```

	diagnosis	bodyPart	disposition	location	product
8	59	76	1	1	1807
63	62	75	4	1	4076
97	59	92	1	0	478
131	62	75	1	5	1807
177	59	82	1	1	3278
...	...	...	...	...	...
334616	71	31	4	1	4078
334677	63	82	1	9	3223
334699	53	83	1	0	1842



334701	57	79	1	0	4074
334785	71	87	4	1	4076

[6379 rows x 12 columns]

Select row with .iloc

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

```
[183]:
```

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

  

	diagnosis	bodyPart	disposition	location	product
10	59	82	1	1	1894
11	57	36	1	9	1267
12	60	88	1	0	3274
13	62	75	1	1	1807
14	53	93	1	1	4057

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

```
[184]:
```

	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
...	...	...	...	...	...
334834	150739278	5/31/2015	15.0591	V	7
334835	150733393	7/11/2015	5.6748	C	3
334836	150819286	7/24/2015	15.7762	V	38
334837	150823002	8/8/2015	97.9239	M	38
334838	150723074	6/20/2015	49.2646	M	5

[334839 rows x 5 columns]

Select column and row with .loc

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

```
[185]: 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
```

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

```
[186]: 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
...
334701  4/27/2015    86
334784  7/7/2015     82
334785  7/11/2015    86
334815  10/28/2015   85
334819  1/13/2015    85
```

```
[20422 rows x 2 columns]
```

[Q6] What is the difference between .iloc and .loc? Ans: different access approach. “.iloc” indexing based on integer. “.loc” indexing based on label.

### 3 [3] Various Types of Data

#### 3.0.1 3.0) HTML

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

```
[188]: 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)

```

```

[189]: 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.table.text)

```

Sample Blog

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

```

-----
AttributeError                                Traceback (most recent call last)
Cell In[189], line 9
      7 print(soup.title.text)
      8 print(soup.h2)
----> 9 print(soup.table.text)

AttributeError: 'NoneType' object has no attribute 'text'

```

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

Ans: There is no table tag in the html file. To fix the code, you can just comment the “print(soup.table.text)” line. Or, you can add table tag. So, the code can runs.

```

[190]: 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.table.text)

```

Sample Blog

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

### 3.0.2 3.1) XML

```
[191]: 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")
```

```
[192]: #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')
```

```
[193]: #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>

```

```

[194]: #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]

```

[195]: #Add you own code here
tree = ET.parse('xml_file.xml')
root = tree.getroot()

data_list = []

for student in root.findall('student'):
    name = student.get('name')
    email = student.find('email').text
    age = student.find('age').text
    gender = student.find('gender').text

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

new_data = {'Name': 'Panyawut', 'Email': 'gansahy@hotmail.com', 'Age': '20',
↪'Gender': 'M'}
data_list.append(new_data)
print(data_list)
root.clear()

for student_data in data_list:
    student_element = ET.Element('student', name=student_data['Name'])

    email_element = ET.SubElement(student_element, 'email')
    email_element.text = student_data['Email']

```

```

age_element = ET.SubElement(student_element, 'age')
age_element.text = student_data['Age']

gender_element = ET.SubElement(student_element, 'gender')
gender_element.text = student_data['Gender']

root.append(student_element)

tree.write('xml_file.xml')

```

```

[{'Name': 'Chanon', 'Email': 'chanon@mail.com', 'Age': '22', 'Gender': 'M'},
{'Name': 'Panyawut', 'Email': 'gansahy@hotmail.com', 'Age': '20', 'Gender':
'M'}]

```

### 3.0.3 3.2) JSON

```

[196]: #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)

```

```

[197]: #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', '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:

```
[198]: #write your own code here
for person in data['people']:
    person['job'] = "Computer Engineering"

print(json.dumps(data, indent=2))

with open('json_file.json', 'w') as file:
    json.dump(data, file, indent=2)
```

```
{
  "people": [
    {
      "name": "Alice",
      "age": 30,
      "city": "New York",
      "job": "Computer Engineering"
    },
    {
      "name": "Bob",
      "age": 25,
      "city": "San Francisco",
      "job": "Computer Engineering"
    },
    {
      "name": "Charlie",
      "age": 35,
      "city": "Los Angeles",
      "job": "Computer Engineering"
    }
  ]
}
```