hw2-suchanat-ratanarueangrong

January 23, 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

[32]:
```
import numpy as np
```

1.1.2 1.1) ndarray initialization

Construct using python list

[33]:
```
# 1d ndarray from 1d python list
list_a1=[1,2,3.5]
arr_a1=np.array(list_a1)
arr_a1
```

[33]:
```
array([1. , 2. , 3.5])
```

[34]:
```
# 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
```

[34]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

[35]:
```
list_a3=[[[1,2],[2,3]],[[3,4],[4,5]]]
arr_a3=np.array(list_a3)
arr_a3
```

[35]:
```
array([[[1, 2],
        [2, 3]],
```

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

or construct using some numpy classes and functions

[36]:
```
np.zeros(5)
```

[36]:
```
array([0., 0., 0., 0., 0.])
```

[37]:
```
np.ones((3,4),dtype=float)
```

[37]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[38]:
```
np.full((4,),999)
```

[38]:
```
array([999, 999, 999, 999])
```

[39]:
```
np.arange(3,10,2)
```

[39]:
```
array([3, 5, 7, 9])
```

[40]:
```
np.linspace(10,15,11)
```

[40]:
```
array([10. , 10.5, 11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. ])
```

[41]:
```
np.random.choice(['a','b'],9)
```

[41]:
```
array(['b', 'a', 'a', 'b', 'a', 'b', 'a', 'a', 'b'], dtype='<U1')
```

[42]:
```
np.random.randn(10)
```

[42]:
```
array([-0.74840772,  0.27238636,  0.59450607,  0.50018668, -1.13362744,
        0.34105508,  0.43291744, -0.76043122, -1.31677157,  1.1144343 ])
```

1.1.3 1.2) ndarray properties

[43]:
```
list_a=[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
arr_a=np.array(list_a)
arr_a
```

[43]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[44]:
```
arr_a.ndim
```

[44]: 2

[45]:
```
arr_a.shape
```

[45]: (3, 4)

[46]:
```
arr_a.dtype
```

[46]: dtype('int64')

[47]:
```
arr_a.size
```

[47]: 12

1.1.4 1.3) Reshaping & Modification

from this original ndarray

[48]:
```
arr_a
```

[48]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to convert into 3D array

[49]:
```
arr_a.reshape((2,2,3))
```

[49]:
```
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

[50]:
```
arr_a.reshape((-1,6))
```

[50]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
```

Would you like to try this?

[51]:
```
arr_a.reshape((-1,5))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-51-286d5aa6424c> 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:

[70]:
```
#Answer
print("Because the array is not accompatible with the shape to resize,␣
 ↪\nresizing between new shape and old shape requires to have the same total␣
 ↪elements.")
```

```
Because the array is not accompatible with the shape to resize,
resizing between new shape and old shape requires to have the same total
elements.
```

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

[52]:
```
np.append(arr_a,13)
```

[52]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13])
```

[53]:
```
np.append(arr_a,arr_a[0])
```

[53]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,  1,  2,  3,  4])
```

[54]:
```
np.append(arr_a,arr_a[0].reshape((1,-1)),axis=0)
```

[54]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [ 1,  2,  3,  4]])
```

[55]:
```
np.append(arr_a,arr_a[:,0].reshape((-1,1)),axis=1)
```

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

[56]:
```
np.concatenate([arr_a,arr_a])
```

[56]:
```
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]])
```

[57]:
```
np.concatenate([arr_a,arr_a],axis=1)
```

[57]:
```
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

[58]:
```
arr_a
```

[58]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to access all element at the first row

[59]:
```
arr_a[1]
```

[59]:
```
array([5, 6, 7, 8])
```

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

[60]:
```
arr_a[1][2]
```

[60]:
```
7
```

[61]:
```
arr_a[1,2]
```

[61]:
```
7
```

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

[62]:
```
arr_a[1,1:]
```

[62]:
```
array([6, 7, 8])
```

[63]:
```
arr_a[:2,1:]
```

[63]:
```
array([[2, 3, 4],
       [6, 7, 8]])
```

sometimes you may specify some row number using list within indicing

[64]:
```
arr_a[[1,2,1],1:]
```

[64]:
```
array([[ 6,  7,  8],
       [10, 11, 12],
```

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

1.1.6 1.5) Boolean slicing

based on this original array

[65]:
```
arr_a
```

[65]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to filter all elements which more than 5

[71]:
```
arr_a>5
```

[71]:
```
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

[72]:
```
(arr_a>5)&(arr_a<10)
```

[72]:
```
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True, False, False, False]])
```

Run the cell below and answer a question.

[73]:
```
arr_a[(arr_a>5)&(arr_a<10)]
```

[73]:
```
array([6, 7, 8, 9])
```

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

Ans:

[79]:
```
#Answer
print("The array result will show the array index that is True from the␣
  ↪condition in arr_a (Condition: (arr_a>5)&(arr_a<10))")
```

```
The array result will show the array index that is True from the condition in
arr_a (Condition: (arr_a>5)&(arr_a<10))
```

Try running the cell below.

[84]:
```
arr_a[(arr_a>5) and (arr_a<10)]
arr_a.all()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-84-1628e1a79dda> in <cell line: 1>()
----> 1 arr_a[(arr_a>5) and (arr_a<10)]
      2 arr_a.all()

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:

[85]:
```
print("Python cannot understand whether it should apply the condition to each␣
 ↪element or assess the entire array's truth value.")
```

```
Python cannot understand whether it should apply the condition to each element
or assess the entire array's truth value.
```

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

Ans:

[86]:
```
arr_a[np.logical_and(arr_a > 5, arr_a < 10)]
```

[86]:
```
array([6, 7, 8, 9])
```

1.1.7 1.6) Basic operations

[87]:
```
list_b=[[1,2,3,4],[1,2,3,4],[1,2,3,4]]
arr_b=np.array(list_b)
arr_b
```

[87]:
```
array([[1, 2, 3, 4],
       [1, 2, 3, 4],
       [1, 2, 3, 4]])
```

This is some operations for only 1 array

[88]:
```
np.sqrt(arr_b)
```

[88]:
```
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

[89]:
```
arr_a-arr_b
```

[89]:
```
array([[0, 0, 0, 0],
       [4, 4, 4, 4],
       [8, 8, 8, 8]])
```

[90]:
```
np.add(arr_a,arr_b)
```

[90]:
```
array([[ 2,  4,  6,  8],
       [ 6,  8, 10, 12],
       [10, 12, 14, 16]])
```

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

[91]:
```
arr_a*3
```

[91]:
```
array([[ 3,  6,  9, 12],
       [15, 18, 21, 24],
       [27, 30, 33, 36]])
```

[92]:
```
1+arr_a**2
```

[92]:
```
array([[  2,   5,  10,  17],
       [ 26,  37,  50,  65],
       [ 82, 101, 122, 145]])
```

Try to play with 2 arrays with different shape

[93]:
```
arr_c=np.array([1,2,3])
arr_d=np.array([[3],[5],[8]])
```

[94]:
```
arr_c-arr_d
```

[94]:
```
array([[-2, -1,  0],
       [-4, -3, -2],
       [-7, -6, -5]])
```

1.1.8 1.7) Basic aggregations

[95]:
```
arr_a
```

[95]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[96]:
```
arr_a.sum()
```

[96]:
```
78
```

[97]:
```
arr_a.mean()
```

[97]:
```
6.5
```

[98]:
```
arr_a.min()
```

[98]:
```
1
```

[99]:
```
arr_a.max()
```

[99]:
```
12
```

[100]:
```
arr_a.std()
```

[100]:
```
3.452052529534663
```

1.1.9 1.8) ndarray axis

[101]:
```
arr_a
```

[101]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[102]:
```
arr_a.sum(axis=0)
```

[102]:
```
array([15, 18, 21, 24])
```

[103]:
```
arr_a.sum(axis=1)
```

[103]:
```
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:

[134]:
```
#Answer
print("axis=0: will made a summation of y axis, but axis=1: will made a␣
 ↪summation of x axis.")
```

```
axis=0: will made a summation of y axis, but axis=1: will made a summation of x
axis.
```

2 [2] Pandas

2.0.1 2.0) Series

[104]:
```
import pandas as pd
import numpy as np
```

[105]:
```
pd.Series(np.random.randn(6))
```

[105]:
```
0   -0.744515
1   -1.588241
2   -0.949407
3   -0.913417
4   -0.591042
5   -0.279706
dtype: float64
```

[106]:
```
pd.Series(np.random.randn(6), index=['a','b','c','d','e','f'])
```

[106]:
```
a    0.023469
b    0.285485
c   -0.806646
d   -0.398081
e    1.178044
f    1.210881
dtype: float64
```

2.0.2 2.1) Constructing Dataframe

Constructing DataFrame from a dictionary

[107]:
```
d = {'col1':[1,2], 'col2': [3,4]}
```

[108]:
```
df = pd.DataFrame(data=d)
df
```

[108]:

	col1	col2
0	1	3
1	2	4

[109]:
```
d2 = {'Name':['Joe','Nat','Harry','Sam','Monica'],
      'Age': [20,21,19,20,22]}
```

[110]:
```
df2 = pd.DataFrame(data=d2)
df2
```

[110]:

	Name	Age
0	Joe	20

```
1     Nat   21
2   Harry   19
3     Sam   20
4  Monica   22
```

Constructing DataFrame from a List

[111]:
```
marks_list = [85.10, 77.80, 91.54, 88.78, 60.55]
```

[112]:
```
df3 = pd.DataFrame(marks_list, columns=['Marks'])
df3
```

[112]:
```
   Marks
0  85.10
1  77.80
2  91.54
3  88.78
4  60.55
```

Creating DataFrame from file

[113]:
```
# Read csv file from path and store to df for create dataframe
df = pd.read_csv('nss15.csv')
```

[114]:
```
df
```

[114]:
```
        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

[115]:
```
# Check dimension by .shape
df.shape
```

[115]:
```
(334839, 12)
```

[116]:
```
# Display the first 5 rows by default
df.head()
```

[116]:
```
   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
```

[117]:
```
# Display the first 3 rows
df.head(3)
```

[117]:
```
   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
```

[118]:
```
# Display the last 5 rows by default
df.tail()
```

[118]:

	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

[119]:
```
# 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

[120]:
```
df.columns
```

[120]:
```
Index(['caseNumber', 'treatmentDate', 'statWeight', 'stratum', 'age', 'sex',
       'race', 'diagnosis', 'bodyPart', 'disposition', 'location', 'product'],
      dtype='object')
```

[121]:
```
#select single column
df['age']
```

[121]:
```
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
```

[122]:
```
df.age
```

[122]:
```
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
```

[123]:
```
#select multiple column
df[['treatmentDate','statWeight','age','sex']]
```

[123]:

	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

[124]:
```
df.race.unique()
```

[124]:
```
array([nan, 'White', 'Other', 'Black', 'Asian', 'American Indian'],
      dtype=object)
```

Describe

[125]:
```
df['age'].describe()
```

[125]:
```
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

[126]:
```
#select by condition
df[df['sex'] == 'Male']
```

[126]:

	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]
```

[127]:
```
#select by multiple condition
df[(df['sex'] == 'Male') & (df['age'] > 80)]
```

[127]:
```
        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

[128]:
```
# select row by .iloc
df.iloc[10:15]
```

[128]:
```
    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
```

[129]:
```
# select column by .iloc
df.iloc[:,[0,1,2,3,4]]
```

[129]:
```
        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

[130]:
```
# select column and low by .loc
df.loc[:6,'treatmentDate':'diagnosis']
```

[130]:
```
  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
```

[137]:
```
# select row by condition
df.loc[df['age']>80, ['treatmentDate', 'age']]
```

[137]:
```
        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?

[138]:
```
#Answer
print("The .iloc need to be input by the index array in Dataframe. However, .
 ↪loc can be use with condition, index, and column name.")
```

```
The .iloc need to be input by the index array in Dataframe. However, .loc can be
use with condition, index, and column name.
```

3 [3] Various Types of Data

3.0.1 3.0) HTML

[139]:
```
from bs4 import BeautifulSoup
```

[140]:
```
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)
```

[156]:
```
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)
<ipython-input-156-3bdb3b6cec9b> in <cell line: 10>()
      8 print(soup.h2)
      9
---> 10 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:

[292]:
```
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("\nBecause the HTML File does not have an table tag, so it cannot access␣
 ↪table structure.")
```

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

Because the HTML File does not have an table tag, so it cannot access table
structure.
```

3.0.2 3.1) XML

[374]:
```
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")
```

[375]:
```
#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')
```

[376]:
```
#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>

[377]:
```
#convert XML to List of Dictionary
data_list = []
for student_element in root:
  name = student_element.attrib.get('name')
  email = student_element.find('email').text
  age = student_element.find('age').text
  gender = student_element.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]

[378]:
```
import xml.etree.ElementTree as ET

# Load existing XML file or create a new one if it doesn't exist
try:
    tree = ET.parse('xml_file.xml')
    root = tree.getroot()
except FileNotFoundError:
    root = ET.Element('data_list')
    tree = ET.ElementTree(root)

# Create a new person element with the "name" attribute
new_person = ET.Element('person', name='Suchanat')

# Add sub-elements for the new person
email = ET.Element('email')
email.text = 'nutrock123123@gmail.com'
new_person.append(email)

age = ET.Element('age')
age.text = '19'
new_person.append(age)

gender = ET.Element('gender')
gender.text = 'Male'
new_person.append(gender)

# Append the new person to the root
root.append(new_person)

# Save the updated XML file
```

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

# Reload the XML file to reflect the changes
tree = ET.parse('xml_file.xml')
root = tree.getroot()


# Extract data from XML and append to the list
for person_element in root.iter('person'):
    name = person_element.attrib.get('name')
    email = person_element.find('email').text
    age = person_element.find('age').text
    gender = person_element.find('gender').text
    data_list.append({"Name": name, "Email": email, "Age": age, "Gender":␣
 ↪gender})

print(data_list)

# 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'},
{'Name': 'Suchanat', 'Email': 'nutrock123123@gmail.com', 'Age': '19', 'Gender':
'Male'}]
<data><student name="Chanon"><email>chanon@mail.com</email><age>22</age><gender>
M</gender></student><person name="Suchanat"><email>nutrock123123@gmail.com</emai
l><age>19</age><gender>Male</gender></person></data>
```

3.0.3 3.2) JSON

[209]:
```
#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)
```

[210]:
```
#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:

[379]:
```
job = ['Owner', 'Data Engineer', 'Data Scientist']
for i,person in enumerate(people):
  person['job'] = job[i]
  print(f"Name: {person['name']}, Age: {person['age']}, City: {person['city']},␣
 ↪Job: {person['job']}")
```

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

[ ]:
```

```