Lab 1 - Basic Python Programming

January 30, 2025

1 Lab 1: Basic Python Programming

1.1 #### CPE232 Data Models

1.2 [1] Variable

1.2.1 1.1 Number Variable

[1]:
```
num = 100 #integer variable
num2 = 12.5 #float variable
print(num)
print(num2)

print(num + num2)   #addition
print(num - num2)   #subtraction
print(num * num2)   #multiplication
print( num / num2)  #division
```

```
100
12.5
112.5
87.5
1250.0
8.0
```

1.2.2 Answer

100 12.5 112.5 87.5 1250.0 8.0

1.2.3 1.2 String Variable

[2]:
```
#string variable
string = "Data Models"
print(string) #print complete string

print("Hello " + string)    #print concatenated string
print(string[0])            #print first character of the string
print(string[:4])           #print first to 4th character of the string
print(string[5:])           #print 6th to last character of the string
print(string[1:4])          #print 2nd to 4th character of the string
```

```
print(string * 2)            #print string 2 time
```

```
Data Models
Hello Data Models
D
Data
Models
ata
Data ModelsData Models
```

1.2.4 1.3 Boolean Variable

[3]:
```
#boolean variable
boolean = True
boolean2 = False

print(boolean)               #print boolean variable
print(not boolean)           #print opposite of boolean variable
print(boolean and boolean2) #print boolean and boolean2
print(boolean or boolean2)  #print boolean or boolean2
```

```
True
False
False
True
```

1.2.5 1.4 List Variable

[4]:
```
#list variable
list = ["Data",20,123.23,40,50]
another_list = ["Models",60]

print(list)                  #print complete list
print(list[0])               #print first element of the list
print(list[1:3])             #print 2nd to 3rd element of the list
print(list[2:])              #print 3rd to last element of the list
print(another_list)          #print complete another_list
print(another_list * 2)      #print another_list two times
print(list + another_list)  #print concatenated list

list[0] = "CPE232"           #change first element of the list
print(list)                  #print complete list
```

```
['Data', 20, 123.23, 40, 50]
Data
[20, 123.23]
[123.23, 40, 50]
['Models', 60]
['Models', 60, 'Models', 60]
```

```
['Data', 20, 123.23, 40, 50, 'Models', 60]
['CPE232', 20, 123.23, 40, 50]
```

1.2.6 1.5 Tuple Variable

[5]:
```
#tuple variable
tuple = ("Data",20,123.23,40,50)
another_tuple = ("Models",60)

print(tuple)                    #print complete tuple
print(tuple[0])                 #print first element of the tuple
print(tuple[1:3])               #print 2nd to 3rd element of the tuple
print(tuple[2:])                #print 3rd to last element of the tuple
print(tuple * 2)                #print tuple two times
print(tuple + another_tuple)    #print concatenated tuple
```

```
('Data', 20, 123.23, 40, 50)
Data
(20, 123.23)
(123.23, 40, 50)
('Data', 20, 123.23, 40, 50, 'Data', 20, 123.23, 40, 50)
('Data', 20, 123.23, 40, 50, 'Models', 60)
```

[6]:
```
tuple[0] = "CPE232"          #trying to change first element of the tuple but␣
 ↪it cannot be changed so it gives error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 1
----> 1 tuple[0] = "CPE232"

TypeError: 'tuple' object does not support item assignment
```

1.2.7 1.6 Dictionary Variable

[7]:
```
#dictionary variable
dictionary = {"name":"Alice","age":21}
another_dictionary = {}
another_dictionary["name"] = "Bob"
another_dictionary["age"] = 21

print(dictionary)                   #print complete dictionary
print(dictionary["name"])           #print value for specific key
print(dictionary.keys())            #print all the keys
print(dictionary.values())          #print all the values
print(dictionary.items())           #print all the items
print(another_dictionary)           #print complete another_dictionary
```

```
{'name': 'Alice', 'age': 21}
Alice
dict_keys(['name', 'age'])
dict_values(['Alice', 21])
dict_items([('name', 'Alice'), ('age', 21)])
{'name': 'Bob', 'age': 21}
```

1.3 [2] Control Flow

1.3.1 2.1 IF ... ELIF ... ELSE

[8]:
```
number = 123
number2 = 34

if number > number2:
    print("number is greater thanu number2")
elif number < number2:
    print("number is less than number2")
else:
    print("number is equal to number2")
```

```
number is greater thanu number2
```

1.4 [3] Loop

1.4.1 3.1 For Loop

[9]:
```
#for loops
for num in range(0,10):
    print(num)
```

```
0
1
2
3
4
5
6
7
8
9
```

[10]:
```
#for loop with list

list = ["Alice","Bob","Charlie","Daisy"]

for name in list:
    print(name)
```

```
Alice
Bob
Charlie
Daisy
```

[11]:
```python
#continue in for loop

list = [1,23,7,"hello",True,1123,43,23,12]

for element in list:
    if type(element) != int:
        continue
    print(element)
```

```
1
23
7
1123
43
23
12
```

[12]:
```python
#break in for loop

list = [1,23,7,"hello",True,1123,43,23,12]

for element in list:
    if type(element) != int:
        break
    print(element)
```

```
1
23
7
```

1.4.2 3.2 While loop

[13]:
```python
#while loop

list = ["Alice","Bob","Charlie","Daisy"]
count = 0

while count < len(list):
    print(list[count])
    count += 1
```

```
Alice
Bob
Charlie
```

```
Daisy
```

[14]:
```python
#continue in while loop

list = [1,23,7,"hello",True,1123,43,23,12]
count = 0

while count < len(list):
    if type(list[count]) != int:
        count += 1
        continue
    print(list[count])
    count += 1
```

```
1
23
7
1123
43
23
12
```

[15]:
```python
#break in while loop

list = [1,23,7,"hello",True,1123,43,23,12]
count = 0

while count < len(list):
    if type(list[count]) != int:
        break
    print(list[count])
    count += 1
```

```
1
23
7
```

1.5 [4] Function

[16]:
```python
#define function
def function_name (arg1, arg2):
    return arg1 + arg2

#calling function
function_name(1,2)
```

[16]: 3

[17]:
```
#define function with default argument
def function_with_default_arg(arg1, arg2 = 10, arg3 = 20 , arg4 = 30):
    return arg1 + arg2 + arg3 + arg4

result_1 = function_with_default_arg(1)
result_2 = function_with_default_arg(1,2,5)
result_3 = function_with_default_arg(1,2,5,10)

print(result_1)
print(result_2)
print(result_3)
```

```
61
38
18
```

[18]:
```
#multiple agument
def function_with_multiple_arg(*args):
    print(args)
    print(type(args))
    sum = 0
    for num in args:
        sum += num

    return sum

function_with_multiple_arg(1,2,3,4,5)
```

```
(1, 2, 3, 4, 5)
<class 'tuple'>
```

[18]:
```
15
```

[19]:
```
#lambda function
lambda_function = lambda arg1, arg2: arg1 + arg2

print(lambda_function(1,2))
```

```
3
```

1.6 [5] File Handling

1.6.1 5.1 Text File

[20]:
```
with open("test.txt","w") as file:
    file.write("Hello World")
```

[21]:
```
with open("test.txt","r") as file:
    print(file.read())
```

```
Hello World
```

1.6.2 5.2 CSV File

[22]:
```
import csv

with open("test.csv","w",newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["Name","Surname"])
        writer.writerow(["Alice","Johnson"])
        writer.writerow(["Bob","Smith"])
```

[23]:
```
import csv

with open("test.csv","r") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

```
['Name', 'Surname']
['Alice', 'Johnson']
['Bob', 'Smith']
```

1.7 [4] Libraries

1.7.1 4.1 Numpy

import numpy library

[40]:
```
import numpy as np
print(np.__version__)
```

```
1.26.4
```

ndarray initialization Construct using python list

[41]:
```
# 1d ndarray from 1d python list
list_a1=[1,2,3.5]
arr_a1=np.array(list_a1)
arr_a1
```

[41]:
```
array([1. , 2. , 3.5])
```

[26]:
```
# 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
```

[26]:
```
array([[1, 2],
       [3, 4],
```

```
       [5, 6]])
```

[27]:
```
list_a3=[[[1,2],[2,3]],[[3,4],[4,5]]]
arr_a3=np.array(list_a3)
arr_a3
```

[27]:
```
array([[[1, 2],
        [2, 3]],

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

or construct using some numpy classes and functions

[28]:
```
np.zeros(5)
```

[28]:
```
array([0., 0., 0., 0., 0.])
```

[29]:
```
np.ones((3,4),dtype=float)
```

[29]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[30]:
```
np.full((4,),999)
```

[30]:
```
array([999, 999, 999, 999])
```

[31]:
```
np.arange(3,10,2)
```

[31]:
```
array([3, 5, 7, 9])
```

[32]:
```
np.linspace(10,15,11)
```

[32]:
```
array([10. , 10.5, 11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. ])
```

[33]:
```
np.random.choice(['a','b'],9)
```

[33]:
```
array(['b', 'a', 'b', 'b', 'b', 'a', 'a', 'a', 'b'], dtype='<U1')
```

[34]:
```
np.random.randn(10)
```

[34]:
```
array([ 0.20646622, -0.56481716, -0.65039786,  2.18961341,  0.11480774,
        1.29197527, -0.35786505,  0.02316607, -0.9317795 ,  0.64936108])
```

ndarray properties

[35]:
```
list_a=[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
arr_a=np.array(list_a)
arr_a
```

[35]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[36]:
```
arr_a.ndim
```

[36]:
```
2
```

[37]:
```
arr_a.shape
```

[37]:
```
(3, 4)
```

[38]:
```
arr_a.dtype
```

[38]:
```
dtype('int64')
```

[39]:
```
arr_a.size
```

[39]:
```
12
```

Reshaping & Modification from this original ndarray

[42]:
```
arr_a
```

[42]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to convert into 3D array

[43]:
```
arr_a.reshape((2,2,3))
```

[43]:
```
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

[44]:
```
arr_a.reshape((-1,6))
```

[44]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
```

Would you like to try this?

[45]:
```
arr_a.reshape((-1,5))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[45], 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: The total number of elements in the array must evenly match the requested shape. If the division results in a non-integer value, it means the elements cannot be perfectly distributed, leading to an error.

In this case, since 12 divided by 5 gives 2.4, which is not a whole number, the reshape operation fails.

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

[46]:
```
np.append(arr_a,13)
```

[46]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13])
```

[47]:
```
np.append(arr_a,arr_a[0])
```

[47]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,  1,  2,  3,  4])
```

[48]:
```
np.append(arr_a,arr_a[0].reshape((1,-1)),axis=0)
```

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

[49]:
```
np.append(arr_a,arr_a[:,0].reshape((-1,1)),axis=1)
```

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

[50]:
```
np.concatenate([arr_a,arr_a])
```

[50]:
```
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]])
```

[51]:
```
np.concatenate([arr_a,arr_a],axis=1)
```

[51]:
```
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]])
```

indexing & slicing from this original array again

[52]:
```
arr_a
```

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

try to access all element at the first row

[53]:
```
arr_a[1]
```

[53]:
```
array([5, 6, 7, 8])
```

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

[54]:
```
arr_a[1][2]
```

[54]:
```
7
```

[55]:
```
arr_a[1,2]
```

[55]:
```
7
```

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

[56]:
```
arr_a[1,1:]
```

[56]:
```
array([6, 7, 8])
```

[57]:
```
arr_a[:2,1:]
```

[57]:
```
array([[2, 3, 4],
       [6, 7, 8]])
```

sometimes you may specify some row number using list within indicing

[58]:
```
arr_a[[1,2,1],1:]
```

[58]:
```
array([[ 6,  7,  8],
       [10, 11, 12],
       [ 6,  7,  8]])
```

Boolean slicing based on this original array

[59]:
```
arr_a
```

[59]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to filter all elements which more than 5

[60]:
```
arr_a>5
```

[60]:
```
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

[61]:
```
(arr_a>5)&(arr_a<10)
```

[61]:
```
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True, False, False, False]])
```

Run the cell below and answer a question.

[62]:
```
arr_a[(arr_a>5)&(arr_a<10)]
```

[62]:
```
array([6, 7, 8, 9])
```

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

Ans: The code selects elements from arr_a that satisfy both conditions: being greater than 5 and less than 10.

Try running the cell below.

[63]:
```
arr_a[(arr_a>5) and (arr_a<10)]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[63], 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: The and operator tries to determine the truth value of the entire array, which leads to ambiguity. This is not the correct way to apply conditions in NumPy.

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

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

[64]:
```
arr_a[(arr_a>5) & (arr_a<10)]
```

[64]:
```
array([6, 7, 8, 9])
```

Basic operations

[65]:
```
list_b=[[1,2,3,4],[1,2,3,4],[1,2,3,4]]
arr_b=np.array(list_b)
arr_b
```

[65]:
```
array([[1, 2, 3, 4],
       [1, 2, 3, 4],
       [1, 2, 3, 4]])
```

This is some operations for only 1 array

[66]:
```
np.sqrt(arr_b)
```

[66]:
```
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

[67]:
```
arr_a-arr_b
```

[67]:
```
array([[0, 0, 0, 0],
       [4, 4, 4, 4],
       [8, 8, 8, 8]])
```

[68]:
```
np.add(arr_a,arr_b)
```

[68]:
```
array([[ 2,  4,  6,  8],
       [ 6,  8, 10, 12],
       [10, 12, 14, 16]])
```

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

[69]:
```
arr_a*3
```

[69]:
```
array([[ 3,  6,  9, 12],
       [15, 18, 21, 24],
       [27, 30, 33, 36]])
```

[70]:
```
1+arr_a**2
```

[70]:
```
array([[  2,   5,  10,  17],
       [ 26,  37,  50,  65],
       [ 82, 101, 122, 145]])
```

Try to play with 2 arrays with different shape

[71]:
```
arr_c=np.array([1,2,3])
arr_d=np.array([[3],[5],[8]])
```

[72]:
```
arr_c-arr_d
```

[72]:
```
array([[-2, -1,  0],
       [-4, -3, -2],
       [-7, -6, -5]])
```

Basic aggregations

[73]:
```
arr_a
```

[73]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[74]:
```
arr_a.sum()
```

[74]:
```
78
```

[75]:
```
arr_a.mean()
```

[75]:
```
6.5
```

[76]:
```
arr_a.min()
```

[76]:
```
1
```

[77]:
```
arr_a.max()
```

[77]:
```
12
```

[78]:
```
arr_a.std()
```

[78]:
```
3.452052529534663
```

ndarray axis

[79]:
```
arr_a
```

[79]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[80]:
```
arr_a.sum(axis=0)
```

[80]:
```
array([15, 18, 21, 24])
```

[81]:
```
arr_a.sum(axis=1)
```

[81]:
```
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 axis argument specifies the direction of the operation: - axis=0 applies the operation column-wise. - axis=1 applies the operation row-wise.

1.7.2 4.2 Pandas

Series

[82]:
```
import pandas as pd
import numpy as np
```

[83]:
```
pd.Series(np.random.randn(6))
```

[83]:
```
0    0.962286
1    2.084423
2   -0.597233
3    0.332573
4   -0.471398
5   -2.299910
dtype: float64
```

[84]:
```
pd.Series(np.random.randn(6), index=['a','b','c','d','e','f'])
```

[84]:
```
a    1.675389
b    0.422551
c   -0.502763
d   -0.010505
e    1.790058
f   -0.149318
dtype: float64
```

Constructing Dataframe Constructing DataFrame from a dictionary

[85]:
```
d = {'col1':[1,2], 'col2': [3,4]}
```

[86]:
```
df = pd.DataFrame(data=d)
df
```

[86]:

	col1	col2
0	1	3
1	2	4

[87]:
```
d2 = {'Name':['Joe','Nat','Harry','Sam','Monica'],
      'Age': [20,21,19,20,22]}
```

[88]:
```
df2 = pd.DataFrame(data=d2)
df2
```

[88]:

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

Constructing DataFrame from a List

[89]:
```
marks_list = [85.10, 77.80, 91.54, 88.78, 60.55]
```

[90]:
```
df3 = pd.DataFrame(marks_list, columns=['Marks'])
df3
```

[90]:

	Marks
0	85.10
1	77.80
2	91.54
3	88.78
4	60.55

Creating DataFrame from file

[91]:
```
# Read csv file from path and store to df for create dataframe
df = pd.read_csv('nss15.csv')
```

[92]:
```
df
```

[92]:

	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]
```

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

Check simple information

[93]:
```
# Check dimension by .shape
df.shape
```

[93]:
```
(334839, 12)
```

[94]:
```
# Display the first 5 rows by default
df.head()
```

[94]:
```
   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
```

[95]:
```
# Display the first 3 rows
df.head(3)
```

[95]:
```
   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
```

[96]:
```
# Display the last 5 rows by default
df.tail()
```

[96]:
```
        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
```

[97]:
```
# 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

[98]:
```
df.columns
```

[98]:
```
Index(['caseNumber', 'treatmentDate', 'statWeight', 'stratum', 'age', 'sex',
       'race', 'diagnosis', 'bodyPart', 'disposition', 'location', 'product'],
```

```
      dtype='object')
```

[99]:
```
#select single column
df['age']
```

[99]:
```
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
```

[100]:
```
df.age
```

[100]:
```
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
```

[101]:
```
#select multiple column
df[['treatmentDate','statWeight','age','sex']]
```

[101]:

	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

[102]:
```
df.race.unique()
```

[102]:
```
array([nan, 'White', 'Other', 'Black', 'Asian', 'American Indian'],
      dtype=object)
```

Describe

[103]:
```
df['age'].describe()
```

[103]:
```
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

[104]:
```
#select by condition
df[df['sex'] == 'Male']
```

[104]:

	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]
```

[105]:
```
#select by multiple condition
df[(df['sex'] == 'Male') & (df['age'] > 80)]
```

[105]:
```
        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

[106]:
```
# select row by .iloc
df.iloc[10:15]
```

[106]:
```
    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
```

[107]:
```
# select column by .iloc
df.iloc[:,[0,1,2,3,4]]
```

[107]:
```
        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

[108]:
```
# select column and low by .loc
df.loc[:6,'treatmentDate':'diagnosis']
```

[108]:
```
  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
```

[109]:
```
# select row by condition
df.loc[df['age']>80, ['treatmentDate', 'age']]
```

[109]:
```
       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: .iloc is used for integer-based indexing, while .loc is used for label-based indexing in Pandas.