Lab1_BasicPythonProgramming

January 24, 2025

1 Lab 1: Basic Python Programming

1.1 #### CPE232 Data Models

1.2 [1] Variable

1.2.1 1.1 Number Variable

[3]:
```
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 1.2 String Variable

[4]:
```
#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
```

1.2.3 1.3 Boolean Variable

[5]:
```
#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
```

1.2.4 1.4 List Variable

[6]:
```
#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]
```

1.2.5 1.5 Tuple Variable

[7]:
```
#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)
```

[8]:
```
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)
<ipython-input-8-e338f959f95c> in <cell line: 0>()
----> 1 tuple[0] = "CPE232"          #trying to change first element of the␣
 ↪tuple but it cannot be changed so it gives error

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

1.2.6 1.6 Dictionary Variable

[13]:
```
#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

[14]:
```
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

[15]:
```
#for loops
for num in range(0,10):
    print(num)
```

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

[16]:
```
#for loop with list

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

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

```
Alice
```

```
Bob
Charlie
Daisy
```

[17]:
```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
```

[18]:
```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

[19]:
```python
#while loop

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

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

```
Alice
Bob
Charlie
```

```
Daisy
```

[20]:
```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
```

[21]:
```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

[22]:
```python
#define function
def function_name (arg1, arg2):
    return arg1 + arg2

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

[22]: 3

[23]:
```
#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
```

[24]:
```
#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'>
```

[24]:
```
15
```

[25]:
```
#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

[26]:
```
with open("test.txt","w") as file:
    file.write("Hello World")
```

[27]:
```
with open("test.txt","r") as file:
    print(file.read())
```

```
Hello World
```

1.6.2 5.2 CSV File

[28]:
```
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"])
```

[29]:
```
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

[30]:
```
import numpy as np
```

ndarray initialization Construct using python list

[31]:
```
# 1d ndarray from 1d python list
list_a1=[1,2,3.5]
arr_a1=np.array(list_a1)
arr_a1
```

[31]:
```
array([1. , 2. , 3.5])
```

[32]:
```
# 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
```

[32]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

[33]:
```
list_a3=[[[1,2],[2,3]],[[3,4],[4,5]]]
arr_a3=np.array(list_a3)
arr_a3
```

[33]:
```
array([[[1, 2],
        [2, 3]],

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

or construct using some numpy classes and functions

[34]:
```
np.zeros(5)
```

[34]:
```
array([0., 0., 0., 0., 0.])
```

[35]:
```
np.ones((3,4),dtype=float)
```

[35]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[36]:
```
np.full((4,),999)
```

[36]:
```
array([999, 999, 999, 999])
```

[37]:
```
np.arange(3,10,2)
```

[37]:
```
array([3, 5, 7, 9])
```

[38]:
```
np.linspace(10,15,11)
```

[38]:
```
array([10. , 10.5, 11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. ])
```

[39]:
```
np.random.choice(['a','b'],9)
```

[39]:
```
array(['a', 'b', 'b', 'b', 'a', 'b', 'a', 'a', 'b'], dtype='<U1')
```

[40]:
```
np.random.randn(10)
```

[40]:
```
array([-0.53945589,  0.1947966 ,  0.37854543, -0.2788537 ,  0.42091922,
        0.11060606, -1.5991202 , -1.081086  ,  1.97417057,  0.2383722 ])
```

ndarray properties

[41]:
```
list_a=[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
arr_a=np.array(list_a)
arr_a
```

[41]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[42]:
```
arr_a.ndim
```

[42]:
```
2
```

[43]:
```
arr_a.shape
```

[43]:
```
(3, 4)
```

[44]:
```
arr_a.dtype
```

[44]:
```
dtype('int64')
```

[45]:
```
arr_a.size
```

[45]:
```
12
```

Reshaping & Modification from this original ndarray

[46]:
```
arr_a
```

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

try to convert into 3D array

[47]:
```
arr_a.reshape((2,2,3))
```

[47]:
```
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

[48]:
```
arr_a.reshape((-1,6))
```

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

Would you like to try this?

[49]:
```
arr_a.reshape((-1,5))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-49-286d5aa6424c> in <cell line: 0>()
----> 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: 1. Code `arr_a.reshape((-1, 5))` did not work because total number of elements in original array `arr_a` (with shape of 3x4 totaling 12 elements) cannot be evenly divided into new shape `(-1, 5)`. 2. The `-1` in `reshape` function allows size of that dimension to be calculated automatically but since 12 cannot be divided evenly by 5 `reshape` operation fails. This is because array must preserve total number of elements and new shape must be compatible with original element count.

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

[50]:
```
np.append(arr_a,13)
```

[50]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13])
```

[51]:
```
np.append(arr_a,arr_a[0])
```

[51]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,  1,  2,  3,  4])
```

[52]:
```
np.append(arr_a,arr_a[0].reshape((1,-1)),axis=0)
```

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

[53]:
```
np.append(arr_a,arr_a[:,0].reshape((-1,1)),axis=1)
```

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

[54]:
```
np.concatenate([arr_a,arr_a])
```

[54]:
```
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]])
```

[55]:
```
np.concatenate([arr_a,arr_a],axis=1)
```

[55]:
```
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

[56]:
```
arr_a
```

[56]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to access all element at the first row

[57]:
```
arr_a[1]
```

[57]:
```
array([5, 6, 7, 8])
```

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

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

[58]:
```
7
```

[59]:
```
arr_a[1,2]
```

[59]:
```
7
```

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

[60]:
```
arr_a[1,1:]
```

[60]:
```
array([6, 7, 8])
```

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

[61]:
```
array([[2, 3, 4],
       [6, 7, 8]])
```

sometimes you may specify some row number using list within indicing

[62]:
```
arr_a[[1,2,1],1:]
```

[62]:
```
array([[ 6,  7,  8],
       [10, 11, 12],
       [ 6,  7,  8]])
```

Boolean slicing based on this original array

[63]:
```
arr_a
```

[63]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to filter all elements which more than 5

[64]:
```
arr_a>5
```

[64]:
```
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

[65]:
```
(arr_a>5)&(arr_a<10)
```

[65]:
```
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True, False, False, False]])
```

Run the cell below and answer a question.

[66]:
```
arr_a[(arr_a>5)&(arr_a<10)]
```

[66]:
```
array([6, 7, 8, 9])
```

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

Ans: - The output of `arr_a[(arr_a > 5) & (arr_a < 10)]` is an array filtered to include only elements in `arr_a` that are greater than 5 and less than 10, - The expression `(arr_a > 5) & (arr_a < 10)` creates a boolean mask to check if each element in `arr_a` satisfies both conditions. The resulting array contains only elements where the mask evaluates to `True`

Try running the cell below.

[67]:
```
arr_a[(arr_a>5) and (arr_a<10)]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-67-78eb1746bbfd> in <cell line: 0>()
----> 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 code `arr_a[(arr_a > 5) and (arr_a < 10)]` gives an error because `and` cannot be used for element-wise comparison in a NumPy array.
- `and` is Python logical operator that works only with single boolean values, not arrays. For element-wise comparisons in NumPy arrays, the `&` operator must be used, as it is specifically designed for boolean operations on arrays

---

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

Ans: `and` -> `&` = `arr_a[(arr_a>5) & (arr_a<10)]`

Basic operations

[68]:
```
list_b=[[1,2,3,4],[1,2,3,4],[1,2,3,4]]
arr_b=np.array(list_b)
arr_b
```

[68]:
```
array([[1, 2, 3, 4],
       [1, 2, 3, 4],
       [1, 2, 3, 4]])
```

This is some operations for only 1 array

[69]:
```
np.sqrt(arr_b)
```

[69]:
```
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

[70]:
```
arr_a-arr_b
```

[70]:
```
array([[0, 0, 0, 0],
       [4, 4, 4, 4],
       [8, 8, 8, 8]])
```

[71]:
```
np.add(arr_a,arr_b)
```

[71]:
```
array([[ 2,  4,  6,  8],
       [ 6,  8, 10, 12],
       [10, 12, 14, 16]])
```

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

[72]:
```
arr_a*3
```

[72]:
```
array([[ 3,  6,  9, 12],
       [15, 18, 21, 24],
       [27, 30, 33, 36]])
```

[73]:
```
1+arr_a**2
```

[73]:
```
array([[  2,   5,  10,  17],
       [ 26,  37,  50,  65],
       [ 82, 101, 122, 145]])
```

Try to play with 2 arrays with different shape

[74]:
```
arr_c=np.array([1,2,3])
arr_d=np.array([[3],[5],[8]])
```

[75]:
```
arr_c-arr_d
```

[75]:
```
array([[-2, -1,  0],
       [-4, -3, -2],
       [-7, -6, -5]])
```

Basic aggregations

[76]:
```
arr_a
```

[76]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

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

[77]:
```
78
```

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

[78]:
```
6.5
```

[79]:
```
arr_a.min()
```

[79]:
```
1
```

[80]:
```
arr_a.max()
```

[80]:
```
12
```

[81]:
```
arr_a.std()
```

[81]:
```
3.452052529534663
```

ndarray axis

[82]:
```
arr_a
```

[82]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[83]:
```
arr_a.sum(axis=0)
```

[83]:
```
array([15, 18, 21, 24])
```

[84]:
```
arr_a.sum(axis=1)
```

[84]:
```
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 argument `axis` in NumPy functions specifies dimension along which the operation is performed. - `axis=0`: Performs column-wise summation (adds values across rows). - `axis=1`: Performs row-wise summation (adds values across columns).

Example: - `arr_a.sum(axis=0)` adds the elements along the vertical direction (column-wise). - `arr_a.sum(axis=1)` adds the elements along the horizontal direction (row-wise).

1.7.2 4.2 Pandas

Series

[85]:
```
import pandas as pd
import numpy as np
```

[86]:
```
pd.Series(np.random.randn(6))
```

[86]:
```
0    0.614745
1    1.389935
2   -0.151511
3    0.425487
4    0.923433
5    2.108643
dtype: float64
```

[87]:
```
pd.Series(np.random.randn(6), index=['a','b','c','d','e','f'])
```

[87]:
```
a    0.463649
b   -0.569441
c    0.765237
d   -2.676614
e    0.201931
f    0.006861
dtype: float64
```

Constructing Dataframe Constructing DataFrame from a dictionary

[88]:
```
d = {'col1':[1,2], 'col2': [3,4]}
```

[89]:
```
df = pd.DataFrame(data=d)
df
```

[89]:

	col1	col2
0	1	3
1	2	4

[90]:
```
d2 = {'Name':['Joe','Nat','Harry','Sam','Monica'],
      'Age': [20,21,19,20,22]}
```

[91]:
```
df2 = pd.DataFrame(data=d2)
df2
```

[91]:

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

Constructing DataFrame from a List

[92]:
```
marks_list = [85.10, 77.80, 91.54, 88.78, 60.55]
```

[93]:
```
df3 = pd.DataFrame(marks_list, columns=['Marks'])
df3
```

[93]:

	Marks
0	85.10
1	77.80
2	91.54
3	88.78
4	60.55

Creating DataFrame from file

[95]:
```
# Read csv file from path and store to df for create dataframe
df = pd.read_csv('nss15.csv')
```

[96]:
```
df
```

[96]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
0	150733174	7/11/2015	15.7762	V	5.0	Male	NaN	57.0	33.0	1.0	9.0	1267.0
1	150734723	7/6/2015	83.2157	S	36.0	Male	White	57.0	34.0	1.0	1.0	1439.0
2	150817487	8/2/2015	74.8813	L	20.0	Female	NaN	71.0	94.0	1.0	0.0	3274.0
3	150717776	6/26/2015	15.7762	V	61.0	Male	NaN	71.0	35.0	1.0	0.0	611.0
4	150721694	7/4/2015	74.8813	L	88.0	Female	Other	62.0	75.0	1.0	0.0	1893.0
…	…	…	…	…	…	…	…	…	…	…	…	…
234085	150968928	9/22/2015	15.7762	V	23.0	Male	Black	64.0	92.0	1.0	1.0	1141.0
234086	150965850	9/24/2015	83.2157	S	37.0	Female	NaN	64.0	31.0	1.0	1.0	4014.0
234087	150971407	9/26/2015	5.6748	C	13.0	Male	White	71.0	79.0	1.0	9.0	1211.0
234088	151026924	10/6/2015	5.6748	C	1.0	Male	White	53.0	75.0	1.0	1.0	4057.0
234089	15100638	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

[234090 rows x 12 columns]

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

Check simple information

[97]:
```
# Check dimension by .shape
df.shape
```

[97]:
```
(234090, 12)
```

[98]:
```
# Display the first 5 rows by default
df.head()
```

[98]:
```
   caseNumber treatmentDate  statWeight stratum   age     sex   race  \
0   150733174     7/11/2015     15.7762       V   5.0    Male    NaN
1   150734723      7/6/2015     83.2157       S  36.0    Male  White
2   150817487      8/2/2015     74.8813       L  20.0  Female    NaN
3   150717776     6/26/2015     15.7762       V  61.0    Male    NaN
4   150721694      7/4/2015     74.8813       L  88.0  Female  Other

   diagnosis  bodyPart  disposition  location  product
0       57.0      33.0          1.0       9.0   1267.0
1       57.0      34.0          1.0       1.0   1439.0
2       71.0      94.0          1.0       0.0   3274.0
3       71.0      35.0          1.0       0.0    611.0
4       62.0      75.0          1.0       0.0   1893.0
```

[99]:
```
# Display the first 3 rows
df.head(3)
```

[99]:
```
   caseNumber treatmentDate  statWeight stratum   age     sex   race  \
0   150733174     7/11/2015     15.7762       V   5.0    Male    NaN
1   150734723      7/6/2015     83.2157       S  36.0    Male  White
2   150817487      8/2/2015     74.8813       L  20.0  Female    NaN

   diagnosis  bodyPart  disposition  location  product
0       57.0      33.0          1.0       9.0   1267.0
1       57.0      34.0          1.0       1.0   1439.0
2       71.0      94.0          1.0       0.0   3274.0
```

[100]:
```
# Display the last 5 rows by default
df.tail()
```

[100]:
```
        caseNumber treatmentDate  statWeight stratum   age     sex   race  \
234085   150968928     9/22/2015     15.7762       V  23.0    Male  Black
234086   150965850     9/24/2015     83.2157       S  37.0  Female    NaN
234087   150971407     9/26/2015      5.6748       C  13.0    Male  White
234088   151026924     10/6/2015      5.6748       C   1.0    Male  White
```

```
234089    15100638           NaN         NaN     NaN   NaN     NaN    NaN

        diagnosis  bodyPart  disposition  location  product
234085       64.0      92.0          1.0       1.0   1141.0
234086       64.0      31.0          1.0       1.0   4014.0
234087       71.0      79.0          1.0       9.0   1211.0
234088       53.0      75.0          1.0       1.0   4057.0
234089        NaN       NaN          NaN       NaN      NaN
```

[101]:
```
# Overview information of dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 234090 entries, 0 to 234089
Data columns (total 12 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   caseNumber      234090 non-null  int64
 1   treatmentDate   234089 non-null  object
 2   statWeight      234089 non-null  float64
 3   stratum         234089 non-null  object
 4   age             234089 non-null  float64
 5   sex             234087 non-null  object
 6   race            143318 non-null  object
 7   diagnosis       234089 non-null  float64
 8   bodyPart        234089 non-null  float64
 9   disposition     234089 non-null  float64
 10  location        234089 non-null  float64
 11  product         234089 non-null  float64
dtypes: float64(7), int64(1), object(4)
memory usage: 21.4+ MB
```

Select column, multiple column, with condition

[102]:
```
df.columns
```

[102]:
```
Index(['caseNumber', 'treatmentDate', 'statWeight', 'stratum', 'age', 'sex',
       'race', 'diagnosis', 'bodyPart', 'disposition', 'location', 'product'],
      dtype='object')
```

[103]:
```
#select single column
df['age']
```

[103]:
```
0          5.0
1         36.0
2         20.0
3         61.0
4         88.0
```

```
           …
234085    23.0
234086    37.0
234087    13.0
234088     1.0
234089     NaN
Name: age, Length: 234090, dtype: float64
```

[104]:
```
df.age
```

[104]:
```
0          5.0
1         36.0
2         20.0
3         61.0
4         88.0
           …
234085    23.0
234086    37.0
234087    13.0
234088     1.0
234089     NaN
Name: age, Length: 234090, dtype: float64
```

[105]:
```
#select multiple column
df[['treatmentDate','statWeight','age','sex']]
```

[105]:

	treatmentDate	statWeight	age	sex
0	7/11/2015	15.7762	5.0	Male
1	7/6/2015	83.2157	36.0	Male
2	8/2/2015	74.8813	20.0	Female
3	6/26/2015	15.7762	61.0	Male
4	7/4/2015	74.8813	88.0	Female
…	…	…	…	…
234085	9/22/2015	15.7762	23.0	Male
234086	9/24/2015	83.2157	37.0	Female
234087	9/26/2015	5.6748	13.0	Male
234088	10/6/2015	5.6748	1.0	Male
234089	NaN	NaN	NaN	NaN

[234090 rows x 4 columns]

Viewing the unique value

[106]:
```
df.race.unique()
```

[106]:
```
array([nan, 'White', 'Other', 'Black', 'Asian', 'American Indian'],
      dtype=object)
```

Describe

[107]:
```
df['age'].describe()
```

[107]:
```
count    234089.000000
mean         31.323274
std          26.077750
min           0.000000
25%          10.000000
50%          23.000000
75%          51.000000
max         106.000000
Name: age, dtype: float64
```

Select row with condition

[108]:
```
#select by condition
df[df['sex'] == 'Male']
```

[108]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	\
0	150733174	7/11/2015	15.7762	V	5.0	Male	NaN	
1	150734723	7/6/2015	83.2157	S	36.0	Male	White	
3	150717776	6/26/2015	15.7762	V	61.0	Male	NaN	
6	150713483	6/8/2015	15.7762	V	25.0	Male	Black	
7	150704114	6/14/2015	83.2157	S	53.0	Male	White	
…	…	…	…	…	…	…	…	
234081	150953450	9/19/2015	5.6748	C	7.0	Male	White	
234084	150906288	8/27/2015	5.6748	C	14.0	Male	White	
234085	150968928	9/22/2015	15.7762	V	23.0	Male	Black	
234087	150971407	9/26/2015	5.6748	C	13.0	Male	White	
234088	151026924	10/6/2015	5.6748	C	1.0	Male	White	

	diagnosis	bodyPart	disposition	location	product
0	57.0	33.0	1.0	9.0	1267.0
1	57.0	34.0	1.0	1.0	1439.0
3	71.0	35.0	1.0	0.0	611.0
6	51.0	33.0	4.0	9.0	1138.0
7	57.0	30.0	1.0	0.0	5040.0
…	…	…	…	…	…
234081	57.0	80.0	4.0	0.0	3286.0
234084	57.0	33.0	1.0	9.0	1329.0
234085	64.0	92.0	1.0	1.0	1141.0
234087	71.0	79.0	1.0	9.0	1211.0
234088	53.0	75.0	1.0	1.0	4057.0

[127729 rows x 12 columns]

[109]:
```
#select by multiple condition
df[(df['sex'] == 'Male') & (df['age'] > 80)]
```

[109]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race
8	150736558	7/16/2015	83.2157	S	98.0	Male	Black
63	150418623	1/12/2015	15.0591	V	97.0	Male	Other
97	150700375	6/28/2015	83.2157	S	85.0	Male	NaN
131	150940801	9/14/2015	15.7762	V	96.0	Male	NaN
177	160110774	12/19/2015	85.7374	S	81.0	Male	White
…	…	…	…	…	…	…	…
233609	150117332	1/2/2015	78.5926	S	82.0	Male	NaN
233867	150822235	7/25/2015	15.7762	V	93.0	Male	NaN
233932	151029869	6/17/2015	74.8813	L	85.0	Male	NaN
233985	151015969	10/6/2015	83.2157	S	82.0	Male	NaN
234052	150827615	7/30/2015	15.7762	V	85.0	Male	NaN

	diagnosis	bodyPart	disposition	location	product
8	59.0	76.0	1.0	1.0	1807.0
63	62.0	75.0	4.0	1.0	4076.0
97	59.0	92.0	1.0	0.0	478.0
131	62.0	75.0	1.0	5.0	1807.0
177	59.0	82.0	1.0	1.0	3278.0
…	…	…	…	…	…
233609	59.0	76.0	4.0	9.0	3223.0
233867	57.0	79.0	4.0	5.0	1679.0
233932	71.0	36.0	4.0	1.0	4076.0
233985	62.0	75.0	1.0	5.0	1807.0
234052	59.0	33.0	1.0	1.0	1884.0

[4407 rows x 12 columns]

Select row with .iloc

[110]:
```
# select row by .iloc
df.iloc[10:15]
```

[110]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race
10	150734952	7/4/2015	15.7762	V	20.0	Male	Black
11	150821622	7/20/2015	83.2157	S	20.0	Female	White
12	150713631	7/4/2015	15.7762	V	11.0	Male	NaN
13	150666343	6/27/2015	15.7762	V	26.0	Female	White
14	150748843	7/16/2015	37.6645	L	33.0	Male	Asian

	diagnosis	bodyPart	disposition	location	product
10	59.0	82.0	1.0	1.0	1894.0
11	57.0	36.0	1.0	9.0	1267.0
12	60.0	88.0	1.0	0.0	3274.0

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

[111]:
```
# select column by .iloc
df.iloc[:,[0,1,2,3,4]]
```

[111]:
```
        caseNumber treatmentDate  statWeight stratum   age
0        150733174     7/11/2015     15.7762       V   5.0
1        150734723      7/6/2015     83.2157       S  36.0
2        150817487      8/2/2015     74.8813       L  20.0
3        150717776     6/26/2015     15.7762       V  61.0
4        150721694      7/4/2015     74.8813       L  88.0
…              …            …           …       …   …
234085   150968928     9/22/2015     15.7762       V  23.0
234086   150965850     9/24/2015     83.2157       S  37.0
234087   150971407     9/26/2015      5.6748       C  13.0
234088   151026924     10/6/2015      5.6748       C   1.0
234089    15100638           NaN         NaN     NaN   NaN

[234090 rows x 5 columns]
```

Select column and row with .loc

[112]:
```
# select column and low by .loc
df.loc[:6,'treatmentDate':'diagnosis']
```

[112]:
```
  treatmentDate  statWeight stratum   age     sex   race  diagnosis
0     7/11/2015     15.7762       V   5.0    Male    NaN       57.0
1      7/6/2015     83.2157       S  36.0    Male  White       57.0
2      8/2/2015     74.8813       L  20.0  Female    NaN       71.0
3     6/26/2015     15.7762       V  61.0    Male    NaN       71.0
4      7/4/2015     74.8813       L  88.0  Female  Other       62.0
5      7/2/2015      5.6748       C   1.0  Female  White       71.0
6      6/8/2015     15.7762       V  25.0    Male  Black       51.0
```

[113]:
```
# select row by condition
df.loc[df['age']>80, ['treatmentDate', 'age']]
```

[113]:
```
       treatmentDate   age
4          7/4/2015  88.0
8         7/16/2015  98.0
39         5/3/2015  88.0
46        4/15/2015  91.0
63        1/12/2015  97.0
…               …   …
233985    10/6/2015  82.0
234047     8/4/2015  83.0
```

```
234049     6/16/2015  85.0
234052     7/30/2015  85.0
234069     8/29/2015  83.0

[14171 rows x 2 columns]
```

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

Ans: - `.iloc` is used for indexing by numeric position - `.loc` is used for indexingg by label or name

---

[119]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[120]:
```
!sudo apt-get install -y texlive-xetex texlive-fonts-recommended␣
 ↪texlive-plain-generic > /dev/null 2>&1
```

[ ]:
```
!apt-get install pandoc > /dev/null 2>&1
```

[ ]:
```
!jupyter nbconvert --output-dir='/content/' --to pdf "/content/drive/MyDrive/
 ↪Colab Notebooks/Lab1_BasicPythonProgramming.ipynb"
```