Session 3: Data type II & Numpy

Module BUSN9690

Business Statistics with Python

Frequently used operations

- Nested? For example
  - nested = ["Hello, All", 40, 34, [31, 120]]

- Operations may need to be done on a collection
  - Create
  - Access
  - Insert
  - Update
  - Remove

Other data type

- Four built-in data types in Python used to store collections of data are
  - List
  - Tuple
  - Set
  - Dictionary

- They are different in
  - Definition
  - Mutability
  - Indexing
  - Duplication

Data types: list, tuple, set and dictionary

Four built-in data types in Python used to store collections of data: list, tuple, set and dictionary

Data Type	Example	Verify
List **Definition: []**	list1 = ["apple", "peach", "mellon"] list2 = [1, 15, 23, 91, 93] list3 = [True, False, False] list4 = ["peach", 21, True, 9, "cat"]	>>>list4 = ["peach", 21, True, 9, "cat"] >>>type(list4)
Tuple **Definition: ()**	tuple1 = ("apple", "peach", "mellon") tuple2 = (1, 15, 23, 91, 93) tuple3 = (True, False, False) tuple4 = ("peach", 21, True, 9, "cat")	>>>tuple4 = ("peach", 21, True, 9, "cat") >>>type(tuple4)
Set **Definition: {} without keys**	set1 = {"apple", "peach", "mellon"} set2 = {1, 15, 23, 91, 93} set3 = {True, False, False} set4 = {"peach", 21, True, 9, "cat"}	>>>set4 = {"peach", 21, True, 9, "cat"} >>>type(set4)

Dictionary
Definition: {} with keys
	thisdict = {
	"course": "MSC Business Analytics",
	"module": "Business Statistics with Python",
	"year": 2021
	}
		>>> thisdict = { "course": "MSC Business Analytics",
		"module": "Business Statistics with Python",
		"year": 2021}
		>>>type(thisdict)

Differences between list, tuple, set, and dictionary

	List	Tuple	Set	Dictionary
Represented by	[ ]	( )	{ }	{ }
Allows duplicate elements?	Yes	Yes	No	Not on keys
Function used to create	list()	tuple()	set()	dict()
Indexing /Slicing	Yes	Yes	No	Yes
Mutable or not?	Mutable	Immutable	Mutable, but elements are not duplicated.	Mutable, but Keys are not duplicated.
Ordered	ordered	ordered	unordered	unordered

Duplication

	List	Tuple	Set	Dictionary
Allows duplicate elements?	Yes	Yes	No	No
Examples	>>>L1=[1,2,4,4] >>>L1 [1,2,4,4]	>>>T1=(1,2,4,4) >>>T1 (1,2,4,4)	>>>S1={1,2,4,4} >>>S1 {1,2,4}	>>>Dict1={"module":"busn9690","module":"busn9690","course":"df"} >>>Dict1 {"module":"busn9690","course":"df"} However, values with different keys can be duplicated >>>Dict1={"module1":"busn9690","module2":"busn9690","module3":"busn9690","course":"df"} >>>Dict1 {"module1":"busn9690","module2":"busn9690","module3":"busn9690","course":"df"}

Function

- Examples

  >>> l1=[1,2,3,"apple"] #l1 is a list

  >>> t1=(1,2,3,"apple") #t1 is a tuple

  >>> s1={1,2,3,"apple"} #s1 is a set

- You can use the following functions to create a data type
  - list(): to create a list
    - list(t1); list(s1) #convert t1 and s1 to a list, respectively
    - l2 = list(('apple', 'banana', 'cherry')) #create a new list l2
  - tuple(): to create a tuple
    - tuple(l1); tuple(s1) #convert l1 and s1 to a tuple, respectively
    - t2 = tuple(('apple', 'banana', 'cherry')) #create a new tuple t2
  - set(): to create a set
    - set(l1); set(t1) #convert l1 and t1 to a set, respectively
    - s2 = set(('apple', 'banana', 'cherry')) #create a new tuple t2
  - dict()
    - d2 = dict(module = "Python", mark = 63, StudentName = "Alex")

Functions

Function

```
>>>text = "I like it"


>>>print(list(text))
['I', ' ', 'l', 'i', 'k', 'e', ' ', 'i', 't']


>>> set(text)
{' ', 'l', 'k', 't', 'I', 'e', 'i'}
```

Question: what is the result from tuple(text)?

Indexing

	List	Tuple	Set	Dictionary
Indexing?	Yes	Yes	No	Yes
Examples	>>>List1=[1,2,4] >>>List1[1]	>>>T1=(1,2,4) >>>T1[1]	>>>S1={1,2, 4} >>>S1[1] An error will occur	>>>Dict1={"module1":"busn9690","module2":"busn9690","module3":"busn9690","course":“machine learning"} >>>Dict1["course"] #Note: different indexing method

Mutable

	List	Tuple	Set	Dictionary
Mutable or not?	Mutable	Immutable	Mutable, but elements are not duplicated.	Mutable. But Keys are not duplicated.
Examples	>>>list1=[1,2,3] >>>list1[1]=12 >>>list1 [1,12,3]	>>>tuple1=(1,2,3) >>>tuple1[1]=12 • Error will occur • Update/Add: You need to convert a tuple to a list and then change elements in the list or add an item into the list: list1=list(tuple1)	>>>S1={1,2,4,4} >>>S1[1] • Error will occur • Add or remove is allowed, e.g. • S1.add(999) • S1.remove(999)	>>>Dict1={"module":"busn9690","course":"df"} **Add** >>>Dict1[“grade"]=65 **Update** >>>Dict1["course"]="business analytics" >>>Dict1.update({"course":"Machine Learning"}) **Remove** >>>Dict1.pop("module") #you need to remove a key, instead of a value under a key

- Unpacking a tuple:
  - ❖ fruits = ("apple", "banana")
  - ❖ (green, yellow) = fruits
  - ❖ print(green)
  - ❖ print(yellow)

Order

	List	Tuple	Set	Dictionary
Ordered?	ordered	ordered	unordered	unordered

Examples	>>>l1=[1,2,3]
	>>>l2=[1,3,2]
	
	>>>l1==l2
	False
		>>>t1=(1,2,3)
		>>>t2=(1,3,2)
		
		>>>t1==t2
		False
			>>>s1={1,2,3}
			>>>s1={1,3,2}
			
			>>>s1 == s2
			True
				>>> Dict1={"module":"busn9690","course":"df"}
				>>> Dict2={"course":"df","module":"busn9690"}
				
				>>>Dict1==Dict2
				True

Can you give an example and verify that the dictionary type is unordered?

Functions

Functions

- In programming, a function is a set of statements organized together to perform a specific task. A function is normally written for the purpose that it will be used repeatedly, or because of its complexity, it is better self-contained in a sub program and called when needed.

- Python has two types of functions
  - Built-in functions: can be directly called in the program without defining them first. A built-in function may be created
    - by Python, e.g., print(), min(), max(), etc; or
    - by a third party, e.g., math.exp(), math.log()
  - User-defined functions: created by users

Built-in functions/methods, libraries

- Built-in functions in libraries created by a third party:
  - os: e.g., to show the current working directory using getcwd(), to change to a new working directory using chdir()
  - math: e.g., to find log(2.3), exp(1.2)
  - statistics: e.g., to find mean(), stdev(), variance(), etc
- Methods to import a library
  a) import statistics
     - usage: statistics.mean([1,2,3,4]) #format: library name, followed by a dot, and then by the function name
  b) import statistics as st
     - usage: st.mean([1,2,3,4]) #use the abbreviation to replace the name
  c) from statistics import *
     - usage: sqrt(9)
  d) from statistics import sqrt
     - usage: sqrt(9)

Content & deletion of a library

- what inside?
  - dir(math),
  - dir(statistics)

```
>>> dir(statistics)
['Counter', 'Decimal', 'Fraction', 'NormalDist', 'StatisticsError', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__
_loader__', '__name__', '__package__', '__spec__', '_coerce', '_convert', '_exact_ratio', '_fail_neg', '_find_lteq', '_find_rteq', '_i
sfinite', '_normal_dist_inv_cdf', '_ss', '_sum', 'bisect_left', 'bisect_right', 'erf', 'exp', 'fabs', 'fmean', 'fsum', 'geometric_mean
', 'groupby', 'harmonic_mean', 'hypot', 'itemgetter', 'log', 'math', 'mean', 'median', 'median_grouped', 'median_high', 'median_low',
'mode', 'multimode', 'numbers', 'pstdev', 'pvariance', 'quantiles', 'random', 'sqrt', 'stdev', 'tau', 'variance']
```

- The same function can be found in different libraries
  - e.g., sqrt() can be found in both math and statistics
- Exit from a library
  - Import math
  - del math

Some commonly used built-in functions

Function in R	Description
abs(x)	absolute value: $\|x\|$
math.sqrt(x)	square root: $\sqrt{x}$
math.ceil(x)	math.ceil(3.475) is 4, math.ceil$(-1.9) = -1$
math.floor(x)	math.floor(3.475) = 3, math.floor$(-1.9) = -2$
math.trunc(x)	math.trunc(5.99) is 5, math.trunc$(-1.9) = -1$
round(x, n)	Round a number to only two decimals: round(3.475, 2) is 3.48
math.cos(x), math.sin(x), math.tan(x)	also math.acos(x), math.cosh(x), math.acosh(x), etc.
math.log(x)	natural logarithm, i.e, ln(x)
math.log10(x)	common logarithm
math.exp(x)	e^x

Other useful built functions

- Random number generator

```
>>>import random
>>>print(random.randrange(1, 10)) #return a random integer between 1 and 10
```

- Factorial, i.e., to find $n! = 1 \times 2 \times \cdots \times n$

```
>>>import math
>>>math.factorial(5)        #return value 1 × 2 × 3 × 4 × 5
```

- Questions:

1) Type print(random.randrange(1, 10)) in the Python console for several times, are the results the same?
2) Can you use command from math import *, and then find a random number between 1 and 100, and then find 10!, respectively?

Numpy

Most often used data manipulation

- Creation
  - To create an array, a table
- Insertion
  - To insert a new row/column
- Deletion
  - To delete an existing row/column
- Update
  - To update an element in a row
- Join
  - To join two tables/arrays
- Sorting
  - To sort the records/table by a given column

Numpy

- What is Numpy?
  - Numpy is a Python library used for working with arrays

- Why Numpy?
  - NumPy allows for efficient operations on the data structures often used in machine learning: vectors, matrices, and tensors, and is faster than lists
  - NumPy is the foundation of the Python machine learning stack.

- Installation:
  - conda install numpy
  - Please see https://numpy.org/install/

Create a NumPy ndarray Object

- 0-d arrays, i.e., scales. Each element in an array is a 0-d array
  - import numpy as np #load the library NumPy into Python
  - arr0 = np.array(12) #0-d array
  - arr1 = np.array([1, 2, 3, 4, 5]) #1-d array
  - arr2 = np.array([[1, 2, 3], [4, 5, 6]]) #2-d array
  - arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]) #3-d array
  - print(arr0)
  - print(arr1)
  - print(arr2)
  - print(arr3)
- To check the number of dimensions of your array
  - print(arr0.ndim)
  - print(arr1.ndim)
  - print(arr2.ndim)
  - print(arr3.ndim)

Create a NumPy ndarray from a list, a tuple

- From a list or a tuple
  - import numpy as np
  - arr_L1 = np.array([1,2,3,4,5,6,7,8,9]) #from a list
  - arr_T1 = np.array((1,2,3,4,5,6,7,8,9)) #from a tuple
  - arr2=np.array([[77, 88, 99] , [31,42,63] , [11,22,33]]) #from a list
  - print(arr_L1)
  - print(arr_T1)
  - print(arr2)
- Check the data type of elements in Numpy Array
  - print(arr1.dtype)

NumPy Array Indexing

- To fine elements
  - 1-d array:
    - print(arr1[1]) # to find the second element
    - print(arr1[-2]) #to find the 2nd last element
  - 2-d array: to find the element in the 2nd row and 3rd column
    - print(arr2[1,2])
- To find a slice of elements on 1-d array
  - import numpy as np
  - arr1 = np.array([1, 2, 3, 4, 5, 6, 7,8,9])
  - arr2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
  - print(arr1[2:7]) #to find the elements between the third and the 7th positions
  - print(arr1[1:7:2]) #to return every other element from index 1 to index 7
  - print(arr2[0:2, 1:4])

numpy.arange()

- numpy.arange(): Create a Numpy Array of evenly spaced numbers in Python
- Syntax:
  - numpy.arange([start,]stop, [step,]dtype=**None**)
  - # If step is not specified, step=1
- Example 1:
  - import numpy as np
  - arr1= np.arange(5, 30, 2) # Start = 5, Stop = 30, Step Size = 2
  - print(arr1) # it will output: [5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
- Example 2:
  - import numpy as np
  - arr1 = np.array([1.1, 2.1, 3.1])
  - newarr1 = arr1.astype('i') # convert the elements in arr1 into integer type
  - print(newarr1) #it will output: [1 2 3]

Array shape and reshape

- Shape: the number of elements in each dimension.
  - import numpy as np
  - arr1 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
  - print(arr1.shape) #output: (2, 4)
- From 1-d to 2-d
  - import numpy as np
  - arr1 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
  - newarr2 = arr1.reshape(4, 3)
  - print(newarr2)

```
#output: [[ 1  2  3]
         [ 4  5  6]
         [ 7  8  9]
         [10 11 12]]
```

- Flattening the arrays
  - import numpy as np
  - arr2 = np.array([[1, 2, 3], [4, 5, 6]])
  - newarr1 = arr2.reshape(-1)
  - print(newarr1)
- You can also reshape from 1-d to 3-d, or flat from 3-d to 1-d

Joining NumPy Arrays

- Joining means putting contents of two or more arrays in a single array
  - Join two 1-d arrays
    - import numpy as np
    - arr1 = np.array([1, 2, 3])
    - arr2 = np.array([4, 5, 6])
    - arr = np.concatenate((arr1, arr2))
    - print(arr) #[1 2 3 4 5 6]
  - Join two 2-d arrays
    - import numpy as np
    - arr1 = np.array([[1, 2], [3, 4]])
    - arr2 = np.array([[5, 6], [7, 8]])
    - arr = np.concatenate((arr1, arr2), axis=1)
    - print(arr)

- Joining Arrays Using Stack Functions
  - import numpy as np
  - arr1 = np.array([1, 2, 3])
  - arr2 = np.array([4, 5, 6])
  - arr = np.stack((arr1, arr2), axis=1)
  - print(arr)

- There are other joining methods

Array splitting

- Joining means putting contents of two or more arrays in a single array
  - import numpy as np
  - arr1 = np.array([1, 2, 3, 4, 5, 6])
  - arr2 = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
  - newarr1 = np.array_split(arr1, 3)
  - newarr2 = np.array_split(arr2, 3)
  - newarr3 = np.array_split(arr2, 3,axis=1)
  - print(newarr1) #[array([1, 2]), array([3, 4]), array([5, 6])]
  - print(newarr2) #[array([[1, 2], [3, 4]]), array([[5, 6],[7, 8]]), array([[9, 10], [11, 12]])]
  - print(newarr3) #array([[1], [3], [5], [7], [9], [11]]), array([[2], [4], [6],[8], [10], [12]]), array([], shape=(6, 0), dtype=int32)]

Searching Arrays

- To find a certain value in an array, and return the indexes that get a match, using keyword where().
  - import numpy as np
  - arr = np.array([1, 2, 3, 4, 5, 4, 4])
  - x = np.where(arr == 4)
  - print(x)

- The searchsorted() method is assumed to be used on sorted arrays.
  - import numpy as np
  - arr = np.array([6, 7, 8, 9])
  - x = np.searchsorted(arr, 7)
  - print(x)) #output: 1

Sorting Arrays

- Sort an array using sort()
  - import numpy as np
  - arr1 = np.array([6, 8, 7, 9])
  - arr2 = np.array(['banana', 'cherry', 'apple'])
  - arr3 = np.array([[3, 2, 4], [5, 0, 1]])
  - x1 = np.sort(arr1)
  - x2 = np.sort(arr2)
  - x3 = np.sort(arr3)
  - print(x1)) #output: [6,7,8,9]
  - print(x2)) #output: ['apple','banana', 'cherry']
  - print(x3)) #output: [[2,3,4],[0, 1,5]]