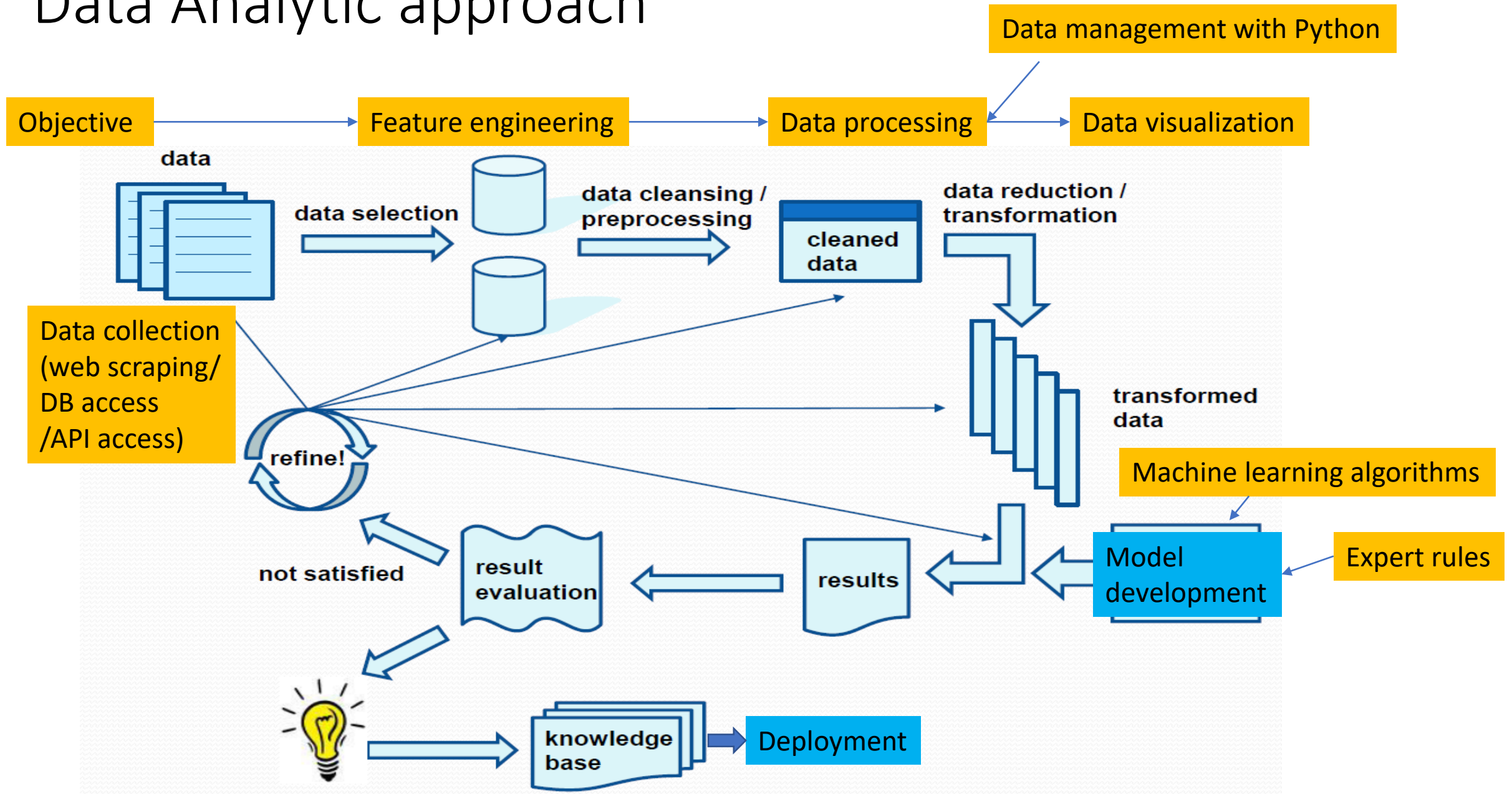


Data management with Python

Nov 2020



Data Analytic approach



Why we use Python

- Open source
- Easy to use
- Top 2 popular programming languages in the world
- Lots of users and developers with big user community
- Lots of useful packages
 - Numpy
 - Pandas
 - Matplotlib
 - Tensorflow
 - Beatifulsoup
 - Selenium
 - Scikit learn
 - And more...
- Most of the popular brokers have Python APIs nowadays
- Good support on data analytics tasks (use of different ML algorithms)

Python

Download Python

- <https://www.python.org/downloads/>
- IDE for Python
- Default: IDLE
- Others: Pycharm
<https://www.jetbrains.com/pycharm/download/#section=windows>
- Jupyter notebook
<https://www.anaconda.com/download/>

Step of uploading the notebook into jupyter

Files Running Clusters

Select items to perform actions on them.

Step one

Upload New ↕

0 / Name ↓ Last Modified File size

Python_Command_Reference_stude

Step two

Upload Cancel

<input type="checkbox"/>	3D Objects	10 days ago
<input type="checkbox"/>	Contacts	10 days ago
<input type="checkbox"/>	Desktop	10 days ago
<input type="checkbox"/>	Documents	2 days ago
<input type="checkbox"/>	Download	9 months ago
<input type="checkbox"/>	Downloads	2 hours ago
<input type="checkbox"/>	Favorites	10 days ago
<input type="checkbox"/>	GA	5 months ago
<input type="checkbox"/>	IdeaProjects	9 months ago

Agenda

- Introduction to program trading with Python
- **Data management with Python**

Data management with Python

- Data types
- String
- Common Python data objects
 - List, dictionary, tuple, set
 - Array, series, dataframe
- List
- Use of for loop and if statement in Python
- List comprehension
- Dictionary
- Function
- Numpy Array
- Pandas Series and Dataframe
- Data management on Pandas objects

Data types in Python

- booleans
 - integers
 - floats
 - Strings
 - Datetime
-
- Use `type()` function to check data type
 - Use `int()/float()/str()/bool()` for the conversion of data types

String management

- Adding strings
 - Extract characters
 - Number of characters in the string
 - substitute and replace of character inside the string
 - split and join operation in string
-
- Note: string is **immutable** object in Python: You cannot change the content in the string after it is defined.

Adding strings

```
a="Python "
```

```
b="is cool"
```

```
c=a+b
```

```
print(c)
```

```
"Python is cool"
```

```
c=a*3+b
```

```
print(c)
```

```
"Python Python Python is cool"
```

Extract characters in string (Filtering by Position)

To extract characters from the string in Python:

string[A:B:C]

where

A-the start position of the string

B-The position after the end part of the string

C: The step of the jumping from the start position towards the end

Note:

- The first character starts with 0 in the extraction.
- A slice goes up to, but will not include, the value at the second index.
- While indexes start at 0 and go up, you can also use negative integers for the index. The integer value -1 refers to the last index in a list, the value -2 refers to the second-to-last index in a list, and so on.

Quiz one: Extract characters in string

Try the following statements under the IDE

```
letters='abcdefghijklmnopqrstuvwxyz'
```

```
len(letters)
```

```
print(letters)
```

```
letters[0]
```

```
letters[-1]
```

```
letters[0:2]
```

```
letters[0:4:2]
```

```
letters[::-1]
```

```
letters[:-1]
```

Quiz two

- Try the following statement:

`letters[0]="b"`

- It seems that the statement fails. What could be the reason behind ?
- How to achieve the task instead by using "+" and slicing operation?
(i.e. replace the first character with "b")

Use of replace or strip method in string

- You could in fact consider to use the following statement for the character replacement:

```
letters.replace("a","b")  
print(letters)
```

- To strip the characters in the string

```
letters.strip("a")
```

Split and join string data

- To split the string into small strings

```
data="1;2;4;6;8"
```

```
data.split(";")
```

- To join the

```
data="1;2;4;6;8"
```

```
data_list=data.split(";")
```

```
data_string=",".join(data_list)
```

Basic data objects in Python

- **List**

- Example: [1,2,3] or [[1,2,3],[4,5,6]] or [1,"a",True]

- **Tuples**

- Example:(1,2,3)

- **Dictionaries**

- Example: {"Peter":10156743,"Alice":2020202}

- **Sets**

- Example: {"Peter","Susan","Alice","Victor"}

key

value

NumPy and Pandas

- With the support of additional packages such as NumPY and Pandas the following data structures are available:
- NumPy
 - Ndarray
- Pandas
 - **Series**
 - **DataFrame**
 - Index
- Those are powerful and effective data structures which are built on top of the existing data structures available in Python and are very useful in the world of data analytic/machine learning

Data management in list

- Adding elements in the list
 - **append()**
 - insert()
 - extend()
- Deleting elements in the list
 - del
 - **remove()**
 - pop()
- Checking if the element is in the list
 - in
- Sort the elements in the list
 - sort
- To count the number of elements in the list
 - count()
- To check the number of elements in the list
 - **len()**

Adding/removing elements in list

Add elements in the list

- append element(s) in the list
 - Use **.append()** method (only one element)
 - Use **.extend()** method (multiple elements are supported)
 - Use + operator. (The + operator can combine two lists to create a new list value in the same way it combines two strings into a new string value.)
- Insert element(s) in the list with the position
 - Use **.insert(P,O)** method where P is the position of the list where you want to insert the list objects (Only one element is supported)

Remove elements in the list

- Remove elements in the list
 - Use **.remove()** method (only one element)
 - Use **del function** if you want to delete the element based on the position in the list
 - or use **.pop()** method if you want to remove the list element based on the position in the list and assign to other object

Example: Adding elements in list

```
In [1]: days=["1","2","3","4","5","6","7"]
```

```
print(days)  
days.append("8")  
print(days)
```

```
['1', '2', '3', '4', '5', '6', '7']  
['1', '2', '3', '4', '5', '6', '7', '8']
```

```
In [2]: days.insert(1,"9")  
print(days)
```

```
['1', '9', '2', '3', '4', '5', '6', '7', '8']
```

```
In [4]: days.append(["8","9"])  
print(days)
```

```
['1', '9', '2', '3', '4', '5', '6', '7', '8', ['8', '9'], ['8', '9']]
```

Example: Removing elements in list

Did not filter !

```
In [5]: days=["1","13","2","3","4","5","6","13","7"]  
days.remove("13")  
print(days)
```

```
['1', '2', '3', '4', '5', '6', '13', '7']
```

Sorting elements in the list

```
days1=[1,2,3,4,5,6,7]
```

```
days1.sort(reverse=True)
```

```
weekdays.sort()
```

Checking if the element is in the list

"Tuesday" in weekdays

"tuesday" in weekdays

1 in days1

1.0 in days1

To count the number of elements in the list

```
weekdays.count("Monday")
```

```
weekdays.count("monday")
```


Use of for loop and if statement

```
L1=[1,2,3,4,6.7,9,11,100]
L2=[]
for i in L1:
    if i >=10:
        L2.append(i)
print(L2)
```

Exercise

- To filter all strings with values="13" in the list by using the for loop and append method.

Comprehension

- A comprehension is a compact way of creating a python data structure from one or other iterators.
- Comprehension makes it possible for you to combine loops and conditional tests with a less verbose syntax.
- It include **list comprehension**, **dictionary comprehension**, **set comprehension** and **generator comprehension**

List Comprehension

- More flexible and powerful data management on list element is feasible with list comprehension

- General syntax:

[expression for item in iterable if condition]

- Example:

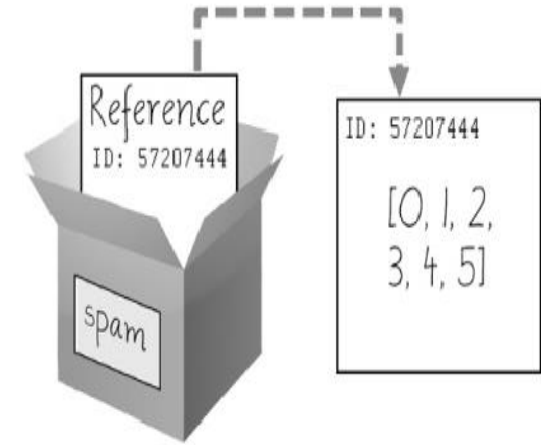
[number for number in range(1,6) if number%2==1]

Example of using list comprehension

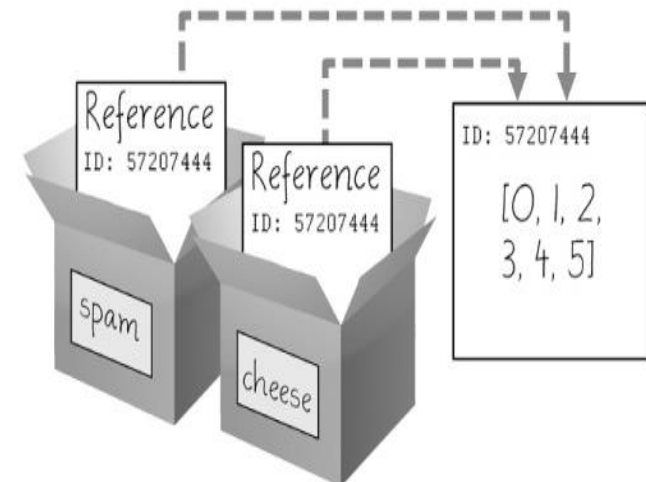
- Question:
 - Given the two list objects
 - L1=[1,2,3,4,5]
 - L2=[2,5]
 - Remove the list elements in L2 from L1
 - Use List Comprehension to achieve the task
- [i for i in L1 if i not in L2]

The concept of mutable and immutable data types-1/2

- When you create the list, you assign a reference to it in the spam variable.
- But the next line copies only the list reference in spam to cheese, not the list value itself.
- This means the values stored in spam and cheese now both refer to the same list.
- There is only one underlying list because the list itself was never actually copied.
- So when you modify the first element of cheese, you are modifying the same list that spam refers to.



spam=[0,1,2,3,4,5] stores a reference to a list, not the actual list



cheese=spam

The concept of mutable and immutable data types-2/2

- Data objects like list and dictionary are mutable objects
- Tuples, strings and integers are immutable objects
- For mutable objects the reference will be used when the objects are assigned to the variable
- For immutable objects the value of the objects will be simply assigned to the variable

Tuple

- Similar to lists, tuples are sequences of arbitrary items.
- It is immutable
- It uses less memory space than list
- Since it is immutable you could use it as the key in the dictionary object

```
empty_tuple=()
```

```
test1=('item1',)
```

```
#It is also called tuple unpacking
```

```
test2=("item1","item2","item3")
```

```
a,b,c=test2
```

```
#To convert a list into tuple
```

```
days=["1","2","3","4","5","6","7"]
```

```
t2=tuple(days)
```


Dictionary

Unique

↓
Key

Values

```
test1={"Monday":"Shopping","Tuesday":"Reading",  
       "Wednesday":"Sleeping","Thursday":"Working","Friday":"Reading"}  
print(test1["Monday"])
```

Shopping

Dictionary

- Add new key value pair into existing dictionary
- Update the values of the existing key in the dictionary
- Remove key-values pair in the dictionary
- How to use dictionary to store structural data (e.g. Excel)
- Refer to separate Ipython notebook for details.

Use of function in Python

```
def print_hello(name):  
    sentence="Hello"+" "+name  
    print(sentence)
```

```
print_hello("Peter")
```

Hello Peter

Numpy and Pandas

Reasons of using NumPy and Pandas for data analytic work

- Data structures (array, series and data frame) are more flexible and effective for big data analytic work
- Better support of vectorized operation (with universal function)
 - No need for loop and more productive in execution
- Better support of recycling work with broadcasting features
- Better support of data aggregation (with groupby)

Use of NumPy

- How to create array with NumPy
- How to create subset of array
- How to transform array
- How to combine arrays
- How to split array into separate array
- The concepts of universal function in NumPy
- The concepts of broadcasting in NumPy

Create array

- To import numpy
import numpy as np
- use array function in numpy
np.array(list structure)

Example:

```
import numpy as np
```

```
np.array([3.14,4,2,3])
```

```
np.array([[1,2,3],[4,5,6]])
```

Some useful attributes of array

- Each array has attributes **ndim** (the number of dimensions), **shape** (the size of each dimension), and **size** (the total size of the array):
- Example:

```
x2=np.array([[1,2,3],[4,5,6]])
```

x2.ndim

x2.shape


x2.size

Create subset of array

- Similar method used in list structure.
- Example

```
x2=np.array([list(range(i,i+3)) for i in [2,4,6]])
```

Row Column



```
x3=x2[:2,:2]
```

Quiz-1

Q1: Explain what it does

```
x2=np.array([list(range(i,i+3)) for i in [2,4,6]])  
x2[0,0]=10  
x3=x2[:2,:2]  
x3[0,0]=20
```

Q2: What is the value of x2 and x3 respectively ?

Q3: Now try the following commands

```
x4=x2[:2,:2].copy()  
x4[0,0]=100
```

What are the values of x4 and x2 now ?

Reshape array

- Another useful type of operation is reshaping of arrays. The most flexible way of doing this is with the `reshape()` method. For example, if you want to put the numbers 1 through 9 in a 3x3 grid, you can do the following:

```
grid = np.arange(1, 10).reshape((3, 3))
```

To combine arrays

- The following methods could be used:

- **concatenate()**

- **vstack()**

- To perform vertical stacking

- **hstack()**

- To perform horizontal stacking

- Example:

```
x=np.array([1,2,3])
```

```
y=np.array([4,5,6])
```

```
z=np.concatenate([x,y])
```

```
z=np.vstack([x,y])
```

```
z=np.hstack([x,y])
```

Computation on NumPy Arrays: Universal Functions

With universal function the use of looping could be avoided (vectorized operation)

Example:

```
x=np.array([1,2,3])
```

```
y=np.array([4,5,6])
```

Try the following statements:

```
x+y
```

```
x-y
```

```
x>y
```

Computation on Arrays: Broadcasting-1/3

- We saw in the previous section how NumPy's universal functions can be used to vectorize operations and thereby remove slow Python loops.
- Another means of vectorizing operations is to use NumPy's broadcasting functionality.
- Broadcasting is simply a set of rules for applying binary ufuncs (addition, subtraction, multiplication, etc.) on arrays of different sizes.

Computation on Arrays: Broadcasting-2/3

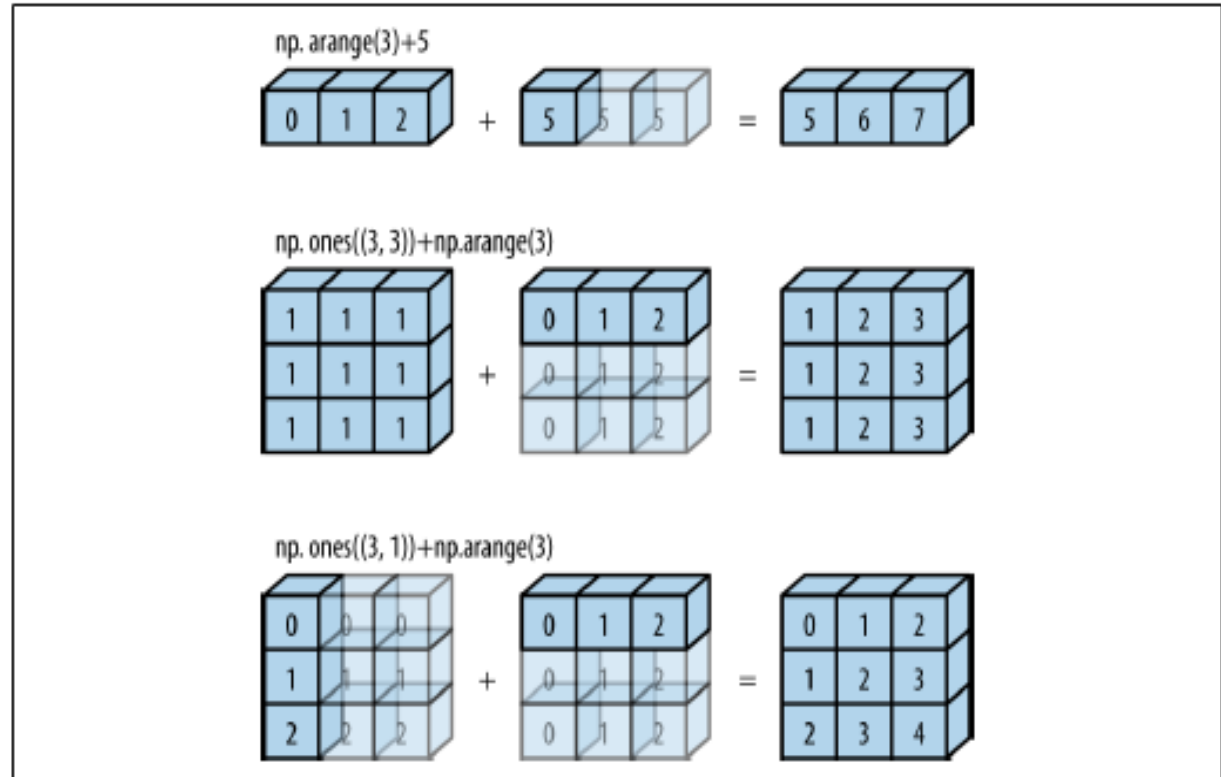
Example:

```
x1=np.array([0,1,2])
```

```
x1+5
```

```
x2=x1.reshape((3,1))
```

```
x1+x2
```



Computation on Arrays: Broadcasting-3/3

- Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded with ones on its leading (left) side.
- Rule 2: If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
- Rule 3: If in any dimension the sizes disagree and neither is equal to 1, an error is raised.

Use of Pandas-1/2

- In the previous section, we dove into detail on NumPy and its ndarray object, which provides efficient storage and manipulation of dense typed arrays in Python.
- Here we'll build on this knowledge by looking in detail at the data structures provided by the Pandas library.
- Pandas is a newer package built on top of NumPy, and provides an efficient implementation of a **DataFrame**.
- DataFrames are essentially multidimensional arrays with attached row and column labels, and often with heterogeneous types and/or missing data.
- As well as offering a convenient storage interface for labeled data, Pandas implements a number of powerful data operations familiar to users of both database frameworks and spreadsheet programs.

Use of Pandas-2/2

- As we saw, NumPy's ndarray data structure provides essential features for the type of clean, well-organized data typically seen in numerical computing tasks.
- While it serves this purpose very well, its limitations become clear when we need more flexibility (attaching labels to data, working with missing data, etc.) and when attempting operations that do not map well to element-wise broadcasting (groupings, pivots, etc.), each of which is an important piece of analyzing the less structured data available in many forms in the world around us.
- Pandas, and in particular its **Series** and **DataFrame** objects, builds on the NumPy array structure and provides efficient access to these sorts of “data munging” tasks that occupy much of a data scientist's time.

Data structure of Pandas

- We will focus on the following two data structures from Pandas:
- Series
- data frame

Data structure of Pandas-Series

- To create series object:

```
import pandas as pd
```

Method 1

```
pd.Series(list object,index=list object)
```

Method 2

```
pd.Series(dictionary object)
```

Example:

```
data = pd.Series([0.25, 0.5, 0.75, 1.0])
```

```
data.values
```

```
data.index
```

```
data = pd.Series([0.25, 0.5, 0.75, 1.0],index=['a', 'b', 'c', 'd'])
```

Data structure of Pandas-Series

Example:

```
population_dict = {'California': 38332521,  
                  'Texas': 26448193,  
                  'New York': 19651127,  
                  'Florida': 19552860,  
                  'Illinois': 12882135}  
population = pd.Series(population_dict)
```


Filtering methods in Pandas

- By position (with `iloc[]`)
- By Boolean series
- By index (with `loc[]`)

Filtering methods of Pandas (By position)

```
In [7]: import pandas as pd  
data = pd.Series([0.25,0.5,0.75,1.0],index=['a','b','c','d'])  
data
```

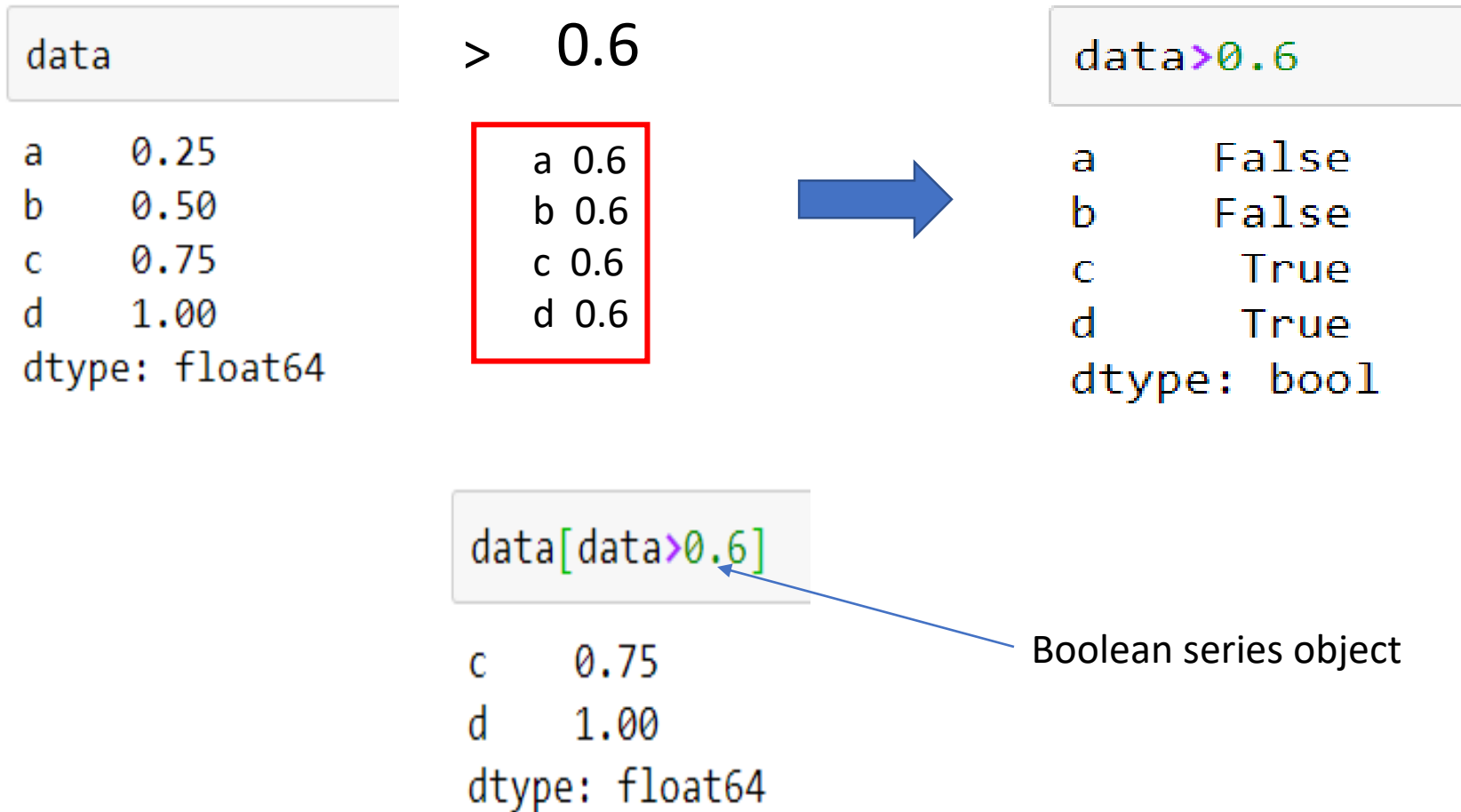
```
Out[7]: a    0.25  
       b    0.50  
       c    0.75  
       d    1.00  
       dtype: float64
```



```
In [8]: data.iloc[0:2]
```

```
Out[8]: a    0.25  
       b    0.50  
       dtype: float64
```

Filtering methods of Pandas (By Boolean series)



Filtering methods of Pandas (By index)

```
data.loc[["a", "c"]]
```

```
a    0.25  
c    0.75  
dtype: float64
```

To filter and extract the items with index="a" and "c"

```
data.loc["b":]
```

```
b    0.50  
c    0.75  
d    1.00  
dtype: float64
```

To filter items with index values starting from "b"

Basic methods/functions/attributes used

- `.index`
 - To get the index of the series/data frame object
- `.values`
 - To get the values of the series/data frame object
- `.describe()`
 - To show the overall statistics of the data frame
- `.head()`
 - To show the first few lines of the data frame
- `.tail()`
 - To show the last few lines of the data frame
- `len()`
 - To determine the number of rows in the data frame
- `sort_values(ascending=True/False)`
 - To sort the values

Common data management tasks on data frame or series objects

- Create the objects (Series or data frame objects)
- Import/export csv/Excel files
- Determine the number of rows and columns in the data frame
- Basic methods on data management work (e.g. head(), describe etc)
- Add/remove columns in the data frame object
- Extract/revise the column names and the index of the data frame object
- Filter data frame objects
- How to handle missing values
- How to sort values in columns
- Join data frame objects
- Multiple indexing in data frame
- Support datetime format in data frame objects
- Aggregate data frame columns

Deal with missing values in Pandas-1/2

- To assign missing values none or NaN could be used
- To check if there exists missing values, you could use isnull() method
- Or if you want to filter out missing values, you could use notnull() method

- Example

```
a=pd.Series({"a":1,"b":2,"c":None,"d":10})
```

```
a[a.isnull()].index
```

```
a[a.notnull()]
```

Deal with missing values in Pandas=2/2

- use `fillna()` method to impute missing values
- Example
- To impute missing values with the mean calculation
`a.fillna(a.mean(),inplace=True)`

Aggregation and grouping

- Simple aggregation is available by using the aggregation method directly on the series or data frame object

- Example:

`a.sum()`

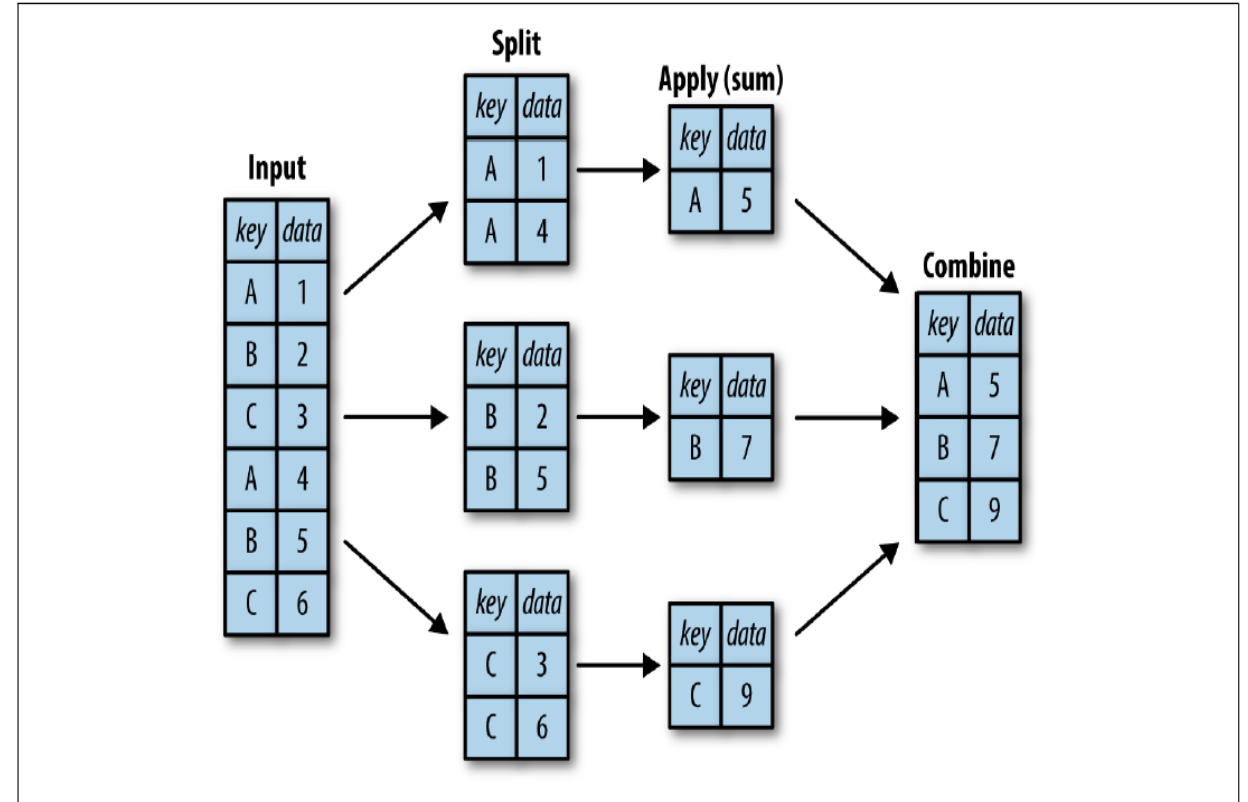
`a.median()`

#If you would like to aggregate the data per row instead of column

`a.median(axis="columns")`

GroupBy: Split, Apply, Combine

- The split step involves breaking up and grouping a DataFrame depending on the value of the specified key.
- The apply step involves computing some function, usually an aggregate, transformation, or filtering, within the individual groups.
- The combine step merges the results of these operations into an output array.



Example: GroupBy

```
df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],  
                  'data': range(6)}, columns=['key', 'data'])  
df.groupby("key").sum()
```


Other data management tasks with Pandas

- Convert data column to datetime
 - Handle missing values
 - Concat data frames
 - Join data frames
-
- Refer to separate Ipython notebook