

How to ...



QF666
Programming and
Computational
Finance



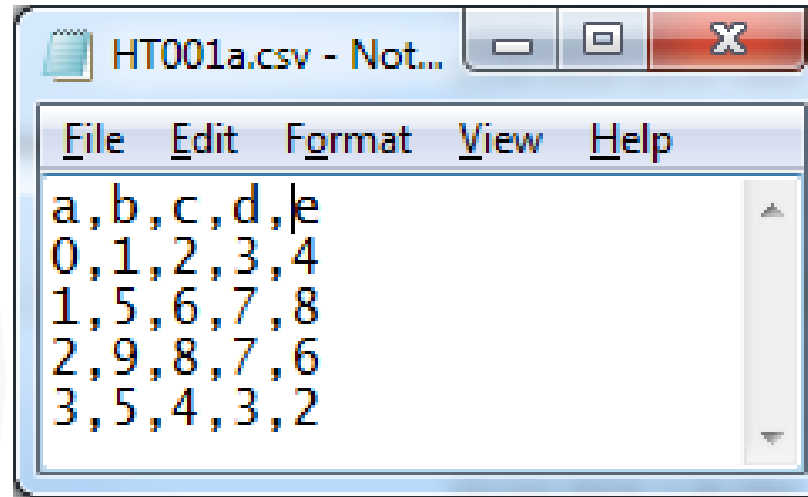
Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

```
import matplotlib.pyplot as plt  
import pandas as pd  
imprt numpy as np
```



Python
MATLAB

HT001: Use one command to import data from a CSV file, **HT001a.csv**, using the first row as column labels, and the first column as row labels. Name the data imported as **data**.



✓ Python: `pandas.read_csv`

```
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
```

? MATLAB: `readtable`

```
data=readtable('HT001a.csv', 'ReadRowNames',true, 'ReadVariableNames',true)
```



```
import pandas as pd  
data=pd.read_csv('HT001a.csv', index_col=0, header=0)  
data
```

a	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

Pandas DataFrame

a	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data=readtable('HT001a.csv', 'ReadRowNames',true, 'ReadVariableNames',true)
```

data =

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

MATLAB Table

099



readtable

R2018a

Create table from file

[collapse all in page](#)

Syntax

```
T = readtable(filename)
T = readtable(filename,Name,Value)
T = readtable(filename,opts)
T = readtable(filename,opts,Name,Value)
```

Description

`T = readtable(filename)` creates a table by reading column oriented data from a file.

[example](#)

`readtable` determines the file format from the file extension:

- `.txt`, `.dat`, or `.csv` for delimited text files
- `.xls`, `.xlsb`, `.xlsm`, `.xlsx`, `.xltm`, `.xltx`, or `.ods` for spreadsheet files

`readtable` creates one variable in `T` for each column in the file and reads variable names from the first row of the file. By default, the variables created are double when the entire column is numeric, or cell arrays of character vectors when any element in a column is not numeric.

.CSV

a table **T**



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

What if the file is newfolder\newfile.csv?

- ✓ `data=pd.read_csv('newfolder\\newfile.csv')`
- ✓ `data=pd.read_csv(r'newfolder\newfile.csv')`

(Mac) `data=pd.read_csv('newfolder/newfile.csv')`

100

- ✓ `data=readtable('newfolder\newfile.csv')`

(Mac) `data=readtable('newfolder/newfile.csv')`



Name-Value Pair Arguments

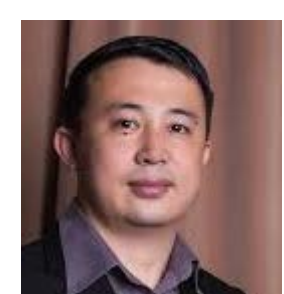
- ❑ **'ReadVariableNames'** – Read first row as variable names
- ❑ **'ReadRowNames'** – Read first column as row names
- ❑ **'DatetimeType'** – Type for imported date and time data
- ❑ **'Delimiter'** – Field delimiter character
- ❑ **'HeaderLines'** – Lines to skip
- ❑ **'Format'** – Column format
- ❑ **'EmptyValue'** – Returned value for empty numeric fields
- ❑ **'DurationType'** – Output data type of duration data

Spreadsheet Files Only

- ❑ **'Sheet'** – Worksheet to read
- ❑ **'Range'** – Portion of worksheet to read

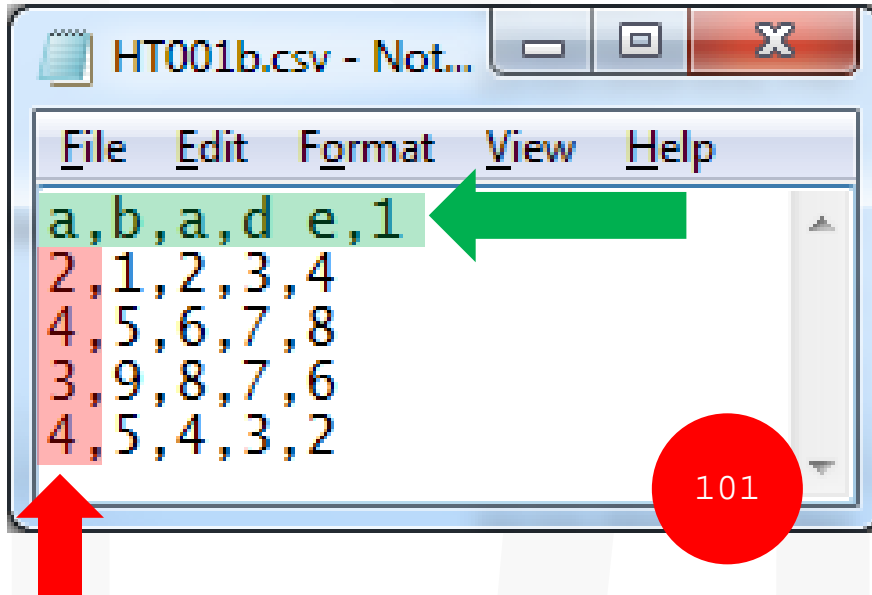


QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

What if the labels are as follows?



	b	a.1	d e	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

`mangle_dupe_cols=True`

`mangle_dupe_cols` : boolean, default True
Duplicate columns will be specified as 'X', 'X.1', ... 'X.N', rather than 'X'...'X'. Passing in False will cause data to be overwritten if there are duplicate names in the columns.

Remember DataFrame is a dict-like container of Series??

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

d e -> dE
1 -> x1 (column label)
4 -> 4_1 (row label)

no duplicate names



	b	a.1	d e	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

data.columns

Index(['b', 'a.1', 'd e', '1'], dtype='object')

data.index

Int64Index([2, 4, 3, 4], dtype='int64', name='a')

(continued)

101

Warning: Variable names were modified
data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

column labels

data.Properties.VariableNames

ans =
'b' 'a' 'dE' 'x1'

1x4

row labels

data.Properties.RowNames

ans =
'2'
'4'
'3'
'4_1'

4x1

LAB identifiers.

cell array

102



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

HT002: Use an appropriate **position-based indexing/slicing** method to select data in **data**.

	b	a.1	de	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

✓ `data.iloc[0,0]`

✗ `data[0][0]`

✗ `data[0,0]`

103

DataFrame \Rightarrow Number

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

✓ `data{1, 1}`

✓ `data(1, 1)`



position-based indexing

```
data{1,1}
```

```
ans =  
    1
```

1x1 **double**

103

```
data(1,1)
```

```
ans =  
      b  
      -  
2      1
```

1x1 **table**



(continued)

position-based slicing (1)

	b	a.1	dE	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

`data.iloc[0:2,0:2]`

DataFrame \Rightarrow DataFrame

	b	a.1
a		
2	1	2
4	5	6

103

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

- ✓ `data{1:2, 1:2}`
- ✓ `data(1:2, 1:2)`



position-based slicing (1)

```
data{1:2, 1:2}
```

```
ans =
```

1	2
5	6

2x2 **double**

103

```
data(1:2, 1:2)
```

```
ans =
```

	b	a
2	1	2
4	5	6

2x2 **table**



(continued)

position-based slicing (2)

(the 1st row)

`data.iloc[0,:]`

DataFrame \Rightarrow Series

	b	a.1	d e	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

b	1
a.1	2
d e	3
1	4
Name: 2, dtype: int64	

103

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

✓ `data{1, :}`

✓ `data(1, :)`



position-based slicing (2): the 1st row

```
data{1, :}
```

```
ans =  
      1      2      3      4
```

1x4 **double**

103

```
data(1, :)
```

```
ans =  
      b      a      dE      x1  
      —      —      —      —  
      2      1      2      3      4
```

1x4 **table**



(continued)

position-based slicing (3)

(the 1st column)

`data.iloc[:,0]`

DataFrame \Rightarrow Series

	b	a.1	dE	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

a		
2	1	
4	5	
3	9	
4	5	
Name: b, dtype: int64		

103

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

✓ `data[:, 1]`

✓ `data(:, 1)`



position-based slicing (3): the 1st column

```
data[:, 1]
```

```
ans =  
     1  
     5  
     9  
     5
```

4x1 **double**

103

```
data(:, 1)
```

```
ans =  
  
      b  
      _  
     2     1  
     4     5  
     3     9  
     4_1    5
```

4x1 **table**



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

Assume the table consists of numbers only.

`data.values` ↔ `data[:, :]`

⇒ Numpy 2D ndarray

⇒ 2D array

`data.Variables`

R2018a



```
data{:, :}
```

```
data{:, :}
```

```
ans =
```

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

4x4 **double**

104

```
data{:, :}
```

```
data{:, :}
```

```
ans =
```

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

4x4 **table**



(continued)

position-based fancy indexing

103

	b	a.1	dE	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

`data.iloc[[2,0],[0,1]]`

DataFrame → DataFrame

(Dr. Z: Fancy indexing is similar to slicing.)

	b	a.1
a		
3	9	8
2	1	2

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

✓ `data{[3 1], [1 2]}`

✓ `data([3 1], [1 2])`



fancy indexing

```
data{[3 1],[1 2]}
```

```
ans =  
     9     8  
     1     2
```

2x2 **double**

103

```
data([3 1],[1 2])
```

```
ans =  
      b  a  
     -- --  
     3  9  8  
     2  1  2
```

2x2 **table**



HT003: Use an appropriate label-based indexing/slicing method to select data in data.

	b	a.1	d e	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

✓ data.loc[2, 'b']
 ✓ data['b'][2]
 ✗ data[2, 'b']

105

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

✓ data{'2', 'b'}
 ✓ data('2', 'b')



label-based indexing

```
data{'2','b'}
```

```
ans =  
    1
```

1x1 **double**

105

```
data{'2','b'}
```

```
ans =  
      b  
      -  
2      1
```

1x1 **table**



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

(continued)

label-based slicing (1)

	b	a.1	d	e	1
a					
2	1	2	3	4	
4	5	6	7	8	
3	9	8	7	6	
4	5	4	3	2	

`data.loc[2:3, 'b': '1']`

105

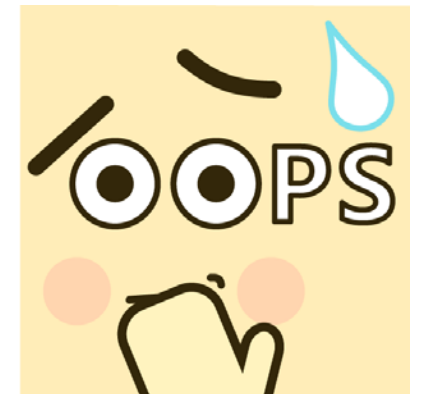
	b	a.1	d	e	1
a					
2	1	2	3	4	
4	5	6	7	8	
3	9	8	7	6	

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

NO
SOLUTION



https://www.mathworks.com/help/matlab/matlab_prog/access-data-in-a-table.html



(continued)

label-based slicing (2)

(row with label 2)

	b	a.1	d e	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

`data.loc[2, :]`

105

b	1
a.1	2
d e	3
1	4
Name: 2, dtype: int64	

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

✓ `data{'2', :}`
✓ `data('2', :)`





label-based slicing (2): row with label 2

```
data['2',:]
```

```
ans =
      1      2      3      4
```

1x4 **double**

105

```
data('2',:)
```

```
ans =
      b      a      dE      x1
      1      2      3      4
      2      1      2      3      4
```

1x4 **table**

(continued)

label-based slicing (3)

(column with label a.1)

	b	a.1	dE	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

`data.loc[:, 'a.1']`

105

a	
2	2
4	6
3	8
4	4

Name: a.1, dtype: int64

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

✓ `data[:, 'a']`
✓ `data(:, 'a')`



label-based slicing (3): column with label a

```
data[:, 'a']
```

```
ans =  
    2  
    6  
    8  
    4
```

4x1 **double**

105

```
data(:, 'a')
```

```
ans =  
  
      a  
    _  
    2    2  
    4    6  
    3    8  
    4_1   4
```

4x1 **table**



(continued)

label-based fancy indexing

	b	a.1	d e	1
a				
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4	5	4	3	2

`data.loc[[2,3],['b','1']]`

105

	b	1
a		
2	1	4
3	9	6

Warning: Variable names were modified to make them valid MATLAB identifiers.

data =

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

✓ `data({'2','3'},{'b','x1'})`

✓ `data(['2','3'],['b','x1'])`



label-based fancy indexing

```
data({'2','3'},{'b','x1'})
```

```
ans =
     1     4
     9     6
```

```
data({'2','3'},{'b','x1'})
```

```
ans =
      b    x1
     --    --
     2    1    4
     3    9    6
```

2x2 **double**

105

2x2 **table**



HT004: Use one command to add a column to **data** with a specific name.

	b	a.1	d	e	1
a					
2	1	2	3	4	
4	5	6	7	8	
3	9	8	7	6	
4	5	4	3	2	

- ✓ `data['f1']=1`
- ✓ `data.loc[:, 'f1']=1`

106

1. a number
2. an iterable
3. A Numpy array
4. a Series
5. a DataFrame

	b	a.1	d	e	1	f1
a						
2	1	2	3	4	1	
4	5	6	7	8	1	
3	9	8	7	6	1	
4	5	4	3	2	1	

- ✓ `data['f2']=range(4)`
- ✓ `data.loc[:, 'f2']=range(4)`
- ✓ `...=[0,1,2,3]`
- ✓ `...=[[0],[1],[2],[3]]`
- ✗ `...=[[0,1,2,3]]`

	b	a.1	d	e	1	f1	f2
a							
2	1	2	3	4	1	0	
4	5	6	7	8	1	1	
3	9	8	7	6	1	2	
4	5	4	3	2	1	3	

107



```
# add a column (3): a Numpy array
import numpy as np
np.arange(4)
```

```
array([0, 1, 2, 3])
```

```
np.arange(4).reshape(4,1)
```

```
array([[0],
       [1],
       [2],
       [3]])
```

```
np.arange(4).reshape(1,4)
```

```
array([[0, 1, 2, 3]])
```

✓ data['f3']=np.arange(4)
 ✓ data['f3']=np.arange(4).reshape(4,1)
 × data['f3']=np.arange(4).reshape(1,4)

1D array or 2D column array

1. a number
2. an iterable
3. A Numpy array
4. a Series
5. a DataFrame

	b	a.1	d e	1	f1	f2	f3
a							
2	1	2	3	4	1	0	0
4	5	6	7	8	1	1	1
3	9	8	7	6	1	2	2
4	5	4	3	2	1	3	3





```
# add a column (4): a Series (with different labels)
pd.Series(range(4))
```

```
0    0
1    1
2    2
3    3
dtype: int64
```

```
data['f4']=pd.Series(range(4))
data
```

	b	a.1	d e	1	f1	f2	f3	f4
a								
2	1	2	3	4	1	0	0	2.0
4	5	6	7	8	1	1	1	NaN
3	9	8	7	6	1	2	2	3.0
4	5	4	3	2	1	3	3	NaN

1. a number
2. an iterable
3. A Numpy array
4. a Series
5. a DataFrame

109

```
? data['f4']=pd.Series(range(3))
? data['f4']=pd.Series(range(4))
? data['f4']=pd.Series(range(5))
```



```
? data['f5']=pd.DataFrame([[i] for i in range(3)])
? data['f5']=pd.DataFrame([[i] for i in range(4)])
? data['f5']=pd.DataFrame([[i] for i in range(5)])
```

```
# add a column (5): a DataFrame
pd.DataFrame([[i] for i in range(4)])
```

	0
0	0
1	1
2	2
3	3

110

1. a number
2. an iterable
3. a Numpy array
4. a Series
5. a DataFrame

```
pd.DataFrame([[i for i in range(4)]], index=[2])
```

	0	1	2	3
2	0	1	2	3

```
data['f5']=pd.DataFrame([[i] for i in range(4)])
data
```

	b	a.1	d e	1	f1	f2	f3	f4	f5
a									
2	1	2	3	4	1	0	0	2.0	2.0
4	5	6	7	8	1	1	1	NaN	NaN
3	9	8	7	6	1	2	2	3.0	3.0
4	5	4	3	2	1	3	3	NaN	NaN

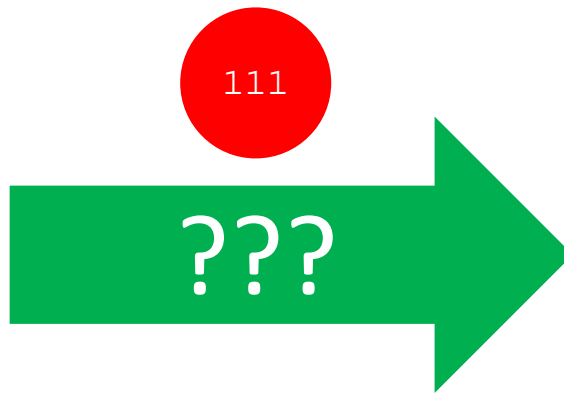
So, how to assign a row's value to a column in a DataFrame??

```
× data['f5']=pd.DataFrame([[i for i in range(4)]],index=[2])
```

Homework: Use one command to add a column to **data**, using the first row of **data**, and name this column **f**.

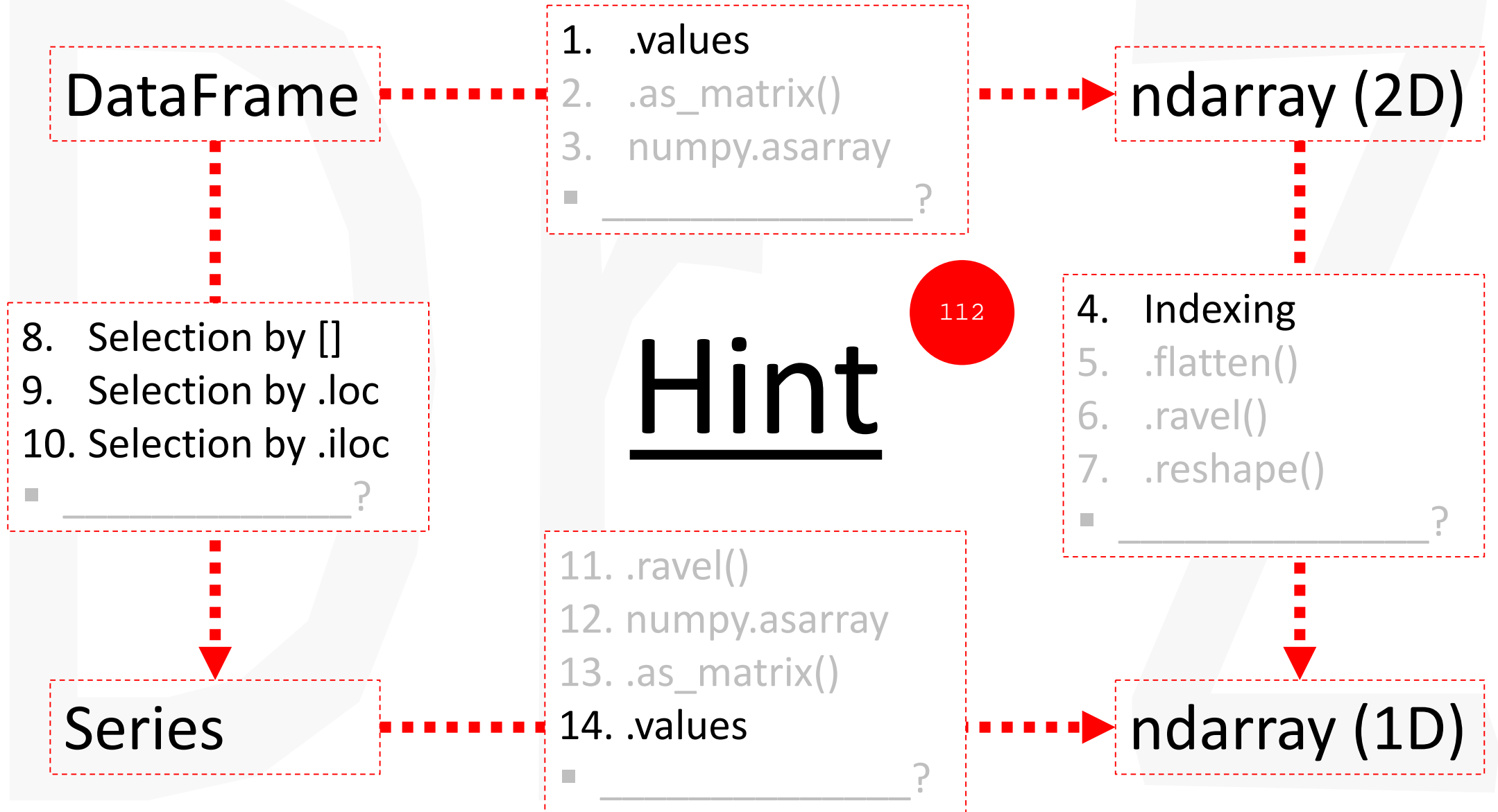
```
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
data
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2



	b	c	d	e	f
a					
0	1	2	3	4	1
1	5	6	7	8	2
2	9	8	7	6	3
3	5	4	3	2	4





HT004: Use one command to add a column to **data** with a specific name.

	b	a	dE	x1
2	1	2	3	4
4	5	6	7	8
3	9	8	7	6
4_1	5	4	3	2

113

× `data.f1=1`

⇒ `data.f1(:)=1`

✓ `data.f1(:,1)=1`

✓ `data{:,'f1'}=1`

× `data(:, 'f1')=1`

✓ `data(:, 'f1')={1}`

`data =`

	b	a	dE	x1	f1
2	1	2	3	4	1
4	5	6	7	8	1
3	9	8	7	6	1
4_1	5	4	3	2	1

1. a number
2. an array



1. a number
2. an array

- ✗ `data.f1=[1,2,3,4]`
- ✓ `data.f1=[1;2;3;4]`
- ? `data.f1={1;2;3;4}`
- ✗ `data.f1(:)=[1;2;3;4]`
- ✗ `data.f1(:)={1;2;3;4}`
- ✗ `data.f1(:)=1:4`
- ✓ `data.f1(:,1)=[1;2;3;4]`
- ✓ `data.f1(:,1)=1:4`
- ✓ `data{:,'f1'}=[1;2;3;4]`
- ? `data{:,'f1'}=1:4`

	b	a	dE	x1	f1
2	1	2	3	4	1
4	5	6	7	8	2
3	9	8	7	6	3
4_1	5	4	3	2	4

To assign to or create a variable in a table, the number of rows must match the height of the table.

In an assignment `A(:) = B`, the number of elements in A and B must be the same.

114

`[1;2;3;4] ⇔ (1:4)'`



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

Homework: Use one command to add a column to **data**, using the first row of **data**, and name this column **f**.

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
```

```
????????????????????
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	b	c	d	e	f
0	1	2	3	4	1
1	5	6	7	8	2
2	9	8	7	6	3
3	5	4	3	2	4

115

?



QF666
Programming and
Computational
Finance

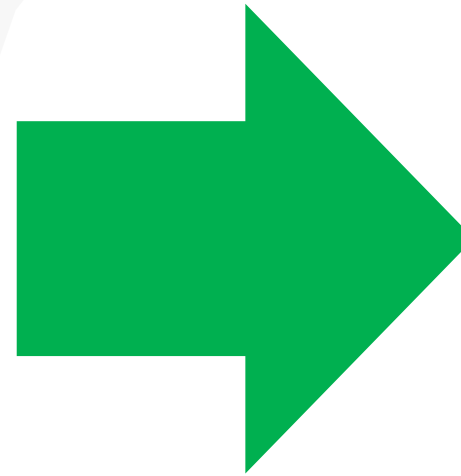


Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

HT005: Use one command to add a row to **data** with a specific name.

- ✓ `data.loc[rowname, :]=...`
- ✓ `data=data.append(...)`

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2



	b	c	d	e
a				
0	1.0	2.0	3.0	4.0
1	5.0	6.0	7.0	8.0
2	9.0	8.0	7.0	6.0
3	5.0	4.0	3.0	2.0
100	1.0	1.0	1.0	1.0

116

	b	c	d	e
a				
0	1.0	2.0	3.0	4.0
1	5.0	6.0	7.0	8.0
2	9.0	8.0	7.0	6.0
3	5.0	4.0	3.0	2.0
100	1.0	2.0	3.0	4.0

117



- ✓ `data.loc[rowname, :]=...`
- ✓ `data.append(...)`

HT005: Use one command
to add a row to `data` with
a specific name.

✓ `data.loc[100, :]=1`

✓ `data.loc[100, :]=[1]`

✗ `data.loc[100, :]=[[1]]`

✓ `data.loc[100, :]=np.array(1)`

✓ `data.loc[100, :]=np.array([1])`

✓ `data.loc[100, :]=np.array([[1]])`





116



- ✓ `data.loc[rowname, :]=...`
- ✓ `data.append(...)`

HT005: Use one command
to add a row to `data` with
a specific name.

- ✓ `data.loc[100,:]=range(1,5)`
 - ✓ `data.loc[100,:]=[1, 2, 3, 4]`
 - ✗ `data.loc[100,:]=[[1, 2, 3, 4]]`
 - ✗ `data.loc[100,:]=[[1],[2],[3],[4]]`
 - ✓ `data.loc[100,:]=np.array([1,2,3,4])`
 - ✓ `data.loc[100,:]=np.array([[1,2,3,4]])`
 - ✗ `data.loc[100,:]=np.array([[1],[2],[3],[4]])`
- 
- 



Dr. Z: 1D for 1D (100%).

pandas.DataFrame.append

`DataFrame.append(other, ignore_index=False, verify_integrity=False, sort=None)` [\[source\]](#)

Append rows of *other* to the end of this frame, returning a new object. Columns not in this frame are added as new columns.

data=data.append(...)

Parameters:

other : *DataFrame or Series/dict-like object, or list of these*
The data to append.

ignore_index : *boolean, default False*
If True, do not use the index labels.

verify_integrity : *boolean, default False*
If True, raise `ValueError` on creating index with duplicates.

sort : *boolean, default None*
Sort columns if the columns of *self* and *other* are not aligned. The default sorting is deprecated and will change to not-sorting in a future version of pandas. Explicitly pass `sort=True` to silence the warning and sort. Explicitly pass `sort=False` to silence the warning and not sort.
New in version 0.23.0.

Returns:

appended : *DataFrame*





```
# a dictionary
%reset -f
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
#data=data.append({k:1 for (v,k) in enumerate(data.columns,1)},ignore_index=True)
data=data.append({k:v for (v,k) in enumerate(data.columns,1)},ignore_index=True)
data
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
4	1	2	3	4

118

The dictionary does not have index to use. **ignore_index=False** will cause an error.

a dictionary

Key ⇔ column names

Cannot specify row name.



```
# Error: a dictionary + "ignore_index=False"
%reset -f
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
#data=data.append({k:1 for (v,k) in enumerate(data.columns,1)})
data=data.append({k:v for (v,k) in enumerate(data.columns,1)})
data
```

ignore_index=False

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-97-cfeaf4369542> in <module>()
      4 data=pd.read_csv('HT001a.csv', index_col=0, header=0)
      5 #data=data.append({k:1 for (v,k) in enumerate(data.columns,1)})
----> 6 data=data.append({k:v for (v,k) in enumerate(data.columns,1)})
      7 data

C:\Continuum\anaconda3\lib\site-packages\pandas\core\frame.py in append(self, other, ignore_index, ver
ify_integrity, sort)
    6175         other = Series(other)
    6176         if other.name is None and not ignore_index:
-> 6177             raise TypeError('Can only append a Series if ignore_index=True'
    6178                             ' or if the Series has a name')
    6179

TypeError: Can only append a Series if ignore_index=True or if the Series has a name
```

(Dr. Z: The dictionary does not suggest any name.)

```
# a list of dictionaries
%reset -f
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
#data=data.append([{'k:1 for (v,k) in enumerate(data.columns,1)}])
data=data.append([{'k:v for (v,k) in enumerate(data.columns,1)}])
data
```

ignore_index=False

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

a list of
dictionaries

Key ⇔ column names

If using `ignore_index=True`, the row label will be 4.

Cannot specify row name.

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
0	1	2	3	4

118

118





```
# a Series
%reset -f
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
#data=data.append(pd.Series(1, index=data.columns),
#data=data.append(pd.Series([1,2,3,4], index=data.columns),
data=data.append(pd.Series(range(1,5), index=data.columns),
                  ignore_index=True)
```

data

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
4	1	2	3	4

The Series does not have an index to be used as row index. **ignore_index=False** will cause an error.

118

a Series

Series Index ↔ column names

Cannot specify row name.



```
# a list of Serieses
%reset -f
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
#data=data.append([pd.Series(1, index=data.columns)])
#data=data.append([pd.Series([1,2,3,4], index=data.columns)])
data=data.append([pd.Series(range(1,5), index=data.columns)])
data
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
0	1	2	3	4

ignore_index=False

a list of Series

Series Index ↔ column names

If using `ignore_index=True`, the row label will be 4.

Cannot specify row name.

118

118



```
# a DataFrame
%reset -f
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
#data=data.append(pd.DataFrame(1,
#data=data.append(pd.DataFrame([[i for i in range(1,5)]]),
data=data.append(pd.DataFrame(np.arange(1,5).reshape(1,4),
                             columns=data.columns,
                             index=[100]))
```

data

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
100	1	2	3	4

ignore_index=False

index=[100]

Index or array-like

a DataFrame

116

117

HT005: Use one command
to add a row to **data** with
a specific name.

If using **ignore_index=True**, the row label will be 4.



HT005: Use one command to add a row to **data** with a specific name.

✓ Add a column

~~✗ data.f1=1~~
 ➡ data.f1(:)=1
 ✓ data.f1(:,1)=1
 ✓ data{:', 'f1'}=1
~~✗ data(:, 'f1')=1~~
 ✓ data(:, 'f1')={1}

~~✗ data.f1=[1,2,3,4]~~
 ✓ data.f1=[1;2;3;4]
 ? data.f1={1;2;3;4}
~~✗ data.f1(:)=[1;2;3;4]~~
~~✗ data.f1(:)={1;2;3;4}~~
~~✗ data.f1(:)=1:4~~
 ✓ data.f1(:,1)=[1;2;3;4]
 ✓ data.f1(:,1)=1:4
 ✓ data{:', 'f1'}=[1;2;3;4]
 ? data{:', 'f1'}=1:4

✓ Add a row

~~✗ data.f1=1~~
~~✗ data.f1(:)=1~~
~~✗ data.f1(1,:)=1~~
 [✓ data{'4',:}=1]
~~✗ data('4',:)=1~~
 ✓ data('4',:)=1

~~✗ data.f1=[1,2,3,4]~~
~~✗ data.f1=[1;2;3;4]~~
~~✗ data.f1={1;2;3;4}~~
~~✗ data.f1(:)=[1;2;3;4]~~
~~✗ data.f1(:)={1;2;3;4}~~
~~✗ data.f1(:)=1:4~~
~~✗ data.f1(1,:)=1:4~~
~~✗ data.f1(1,:)=1:4~~
 ✓ data{'4',:}=[1,2,3,4]
 [✓ data{'4',:}=1:4]

119

120



HT006: Delete a row/column from data

1. Delete a column as deleting an item in a dictionary

```
del data['b']
```

 (in-place delete)

2. Use `pandas.DataFrame.drop`

```
axis=0, inplace=False
```

 (default)

```
0 or 'index', 1 or 'columns'
```

121

pandas.DataFrame.drop

`DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')`

Drop specified labels from rows or columns.

[source]

Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. When using a multi-index, labels on different levels can be removed by specifying the level.



```
%reset -f
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
del data['b'] #or del data[data.columns[0]]
print(data)
data.drop('c', axis=1, inplace=True)
print(data)
data.drop(data.columns[0], axis=1, inplace=True)
print(data)
```

121

122

```
del data['b'] or del data[data.columns[0]]
```

```
data.drop('c',
          axis=1,
          inplace=True)
```

```
data.drop(data.columns[0],
          axis=1,
          inplace=True)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	c	d	e
a			
0	2	3	4
1	6	7	8
2	8	7	6
3	4	3	2

	d	e
a		
0	3	4
1	7	8
2	7	6
3	3	2

	e
a	
0	4
1	8
2	6
3	2



HT006: Delete a row/column from data

```
pandas.DataFrame.drop
```

```
data.index
```

```
axis=0
```

```
inplace=True
```



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

```
%reset -f
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.drop(0, inplace=True)
print(data)
data.drop(data.index[0], inplace=True)
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
a				
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
a				
2	9	8	7	6
3	5	4	3	2

`data.drop(0, inplace=True)`

`data.drop(data.index[0],
inplace=True)`

123

124



HT006: Delete a row/column from `data`

- ✓ Delete a column using the “dot syntax”

```
data.b=[]
```

- ✓ Delete a column using the indexing (position-based and label-based)

```
data(:,1)=[]
```

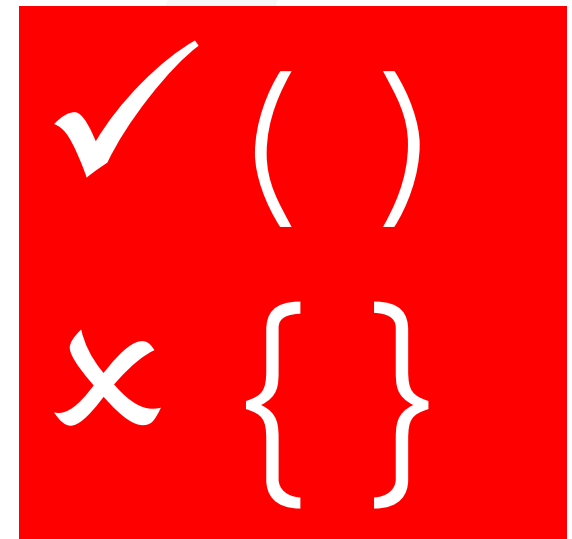
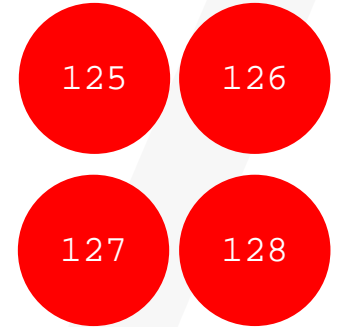
```
data(:, 'b')=[]
```

- ✓ Delete a row by the row number

```
data(1,:)=[]
```

- ✓ Delete a row by the row name

```
data('0',:)=[]
```



Delete a column using the dot syntax

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
data.b=[]
```

data.b= []

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	c	d	e
0	2	3	4
1	6	7	8
2	8	7	6
3	4	3	2

Delete a column using label-based indexing

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
data(:, 'b')=[]
```

data(:, 'b')= []

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	c	d	e
0	2	3	4
1	6	7	8
2	8	7	6
3	4	3	2

Delete a column using position-based indexing

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
data(:,1)=[]
```

data(:,1)= []

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	c	d	e
0	2	3	4
1	6	7	8
2	8	7	6
3	4	3	2

✓ Delete the column 'b'

125

126

✓ Delete the first column



QF666

Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

Delete a row by the row number

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
data(1,:)=[]
```

data(1,:)=[]

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	b	c	d	e
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

✓ Delete the first row

127

128

Delete a row by the row name

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
data('0',:)=[]
```

data('0',:)=[]

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```


	b	c	d	e
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

✓ Delete the row '0'



Q: How to swap two values in variables **a** and **b**?

????????????????????????????
?
? a=b
? b=a
?
?
????????????????????????????



>>> a=1 ?
>>> b=2 ?
>>> a=b ?
>>> b=a ?
>>> a,b ?
(2, 2) ?



(continued)

✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

✓

✓

temp=a

✓

a=b

✓

b=temp

✓

✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

129

>>> a=1 ✓

>>> b=2 ✓

>>> temp=a ✓

>>> a=b ✓

>>> b=temp ✓

>>> a,b ✓

<2, 1> ✓

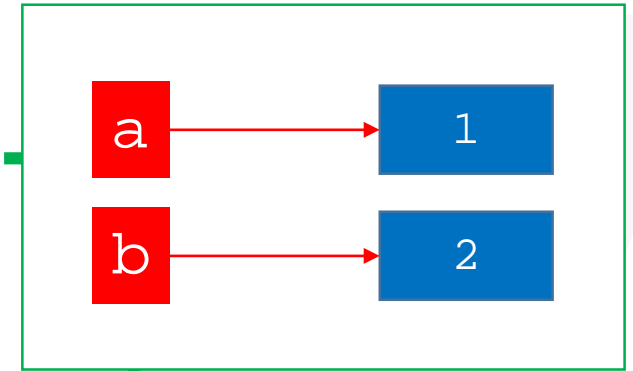


QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

(continued)



`expr3, expr4=expr1, expr2`

`>>> a=1`

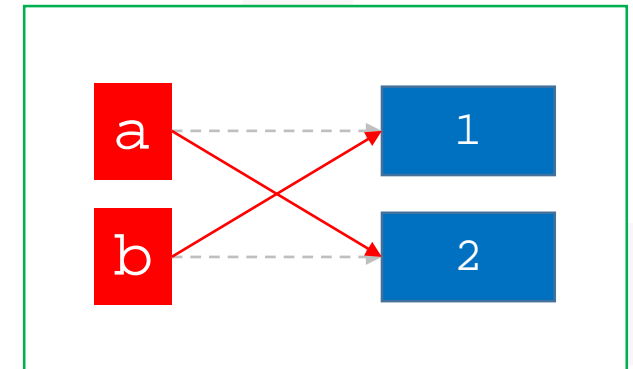
`>>> b=2`

`>>> a, b=b, a`

`>>> a, b`

`(2, 1)`

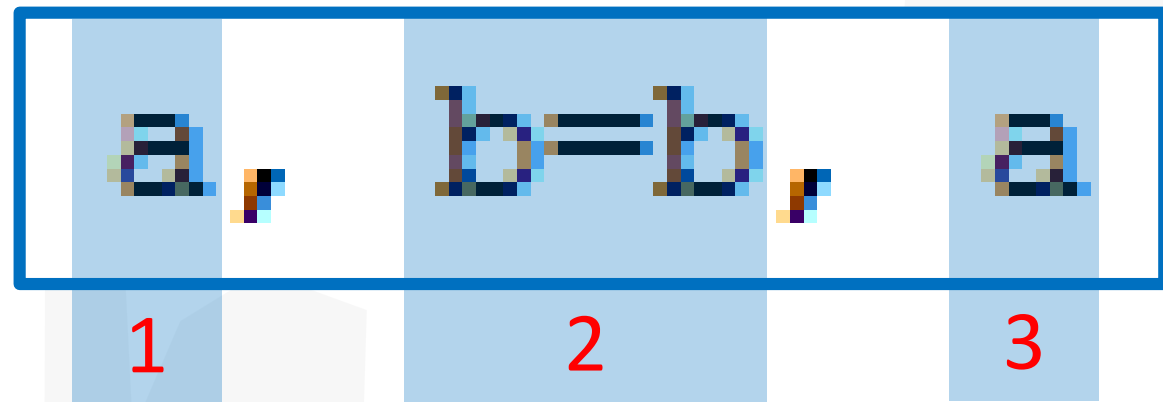

129





```
>> a=1;
>> b=2;
>> temp=a;
>> a=b;
>> b=temp;
>> a
a =
    2


>> b
b =
    1
```



130

```
>> a=1;
>> b=2;
>> [a,b]=deal(b,a)
a =
    2

b =
    1
```



HT007: Swap two rows/columns in data

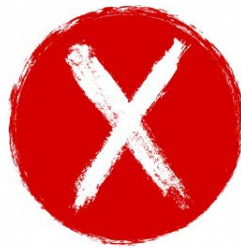
```
# A wrong answer to swap two columns
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
temp=data['b']
data['b']=data['c']
data['c']=temp
print(data)
```

1. Select data via []
2. Select data via .loc or .iloc

temp does not work!

data['b'] and data['c'] are views. A view is something like a “pointer”. Assignment to a view is function to use the pointer to assign values.

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
	b	c	d	e
a				
0	2	2	3	4
1	6	6	7	8
2	8	8	7	6
3	4	4	3	2



????





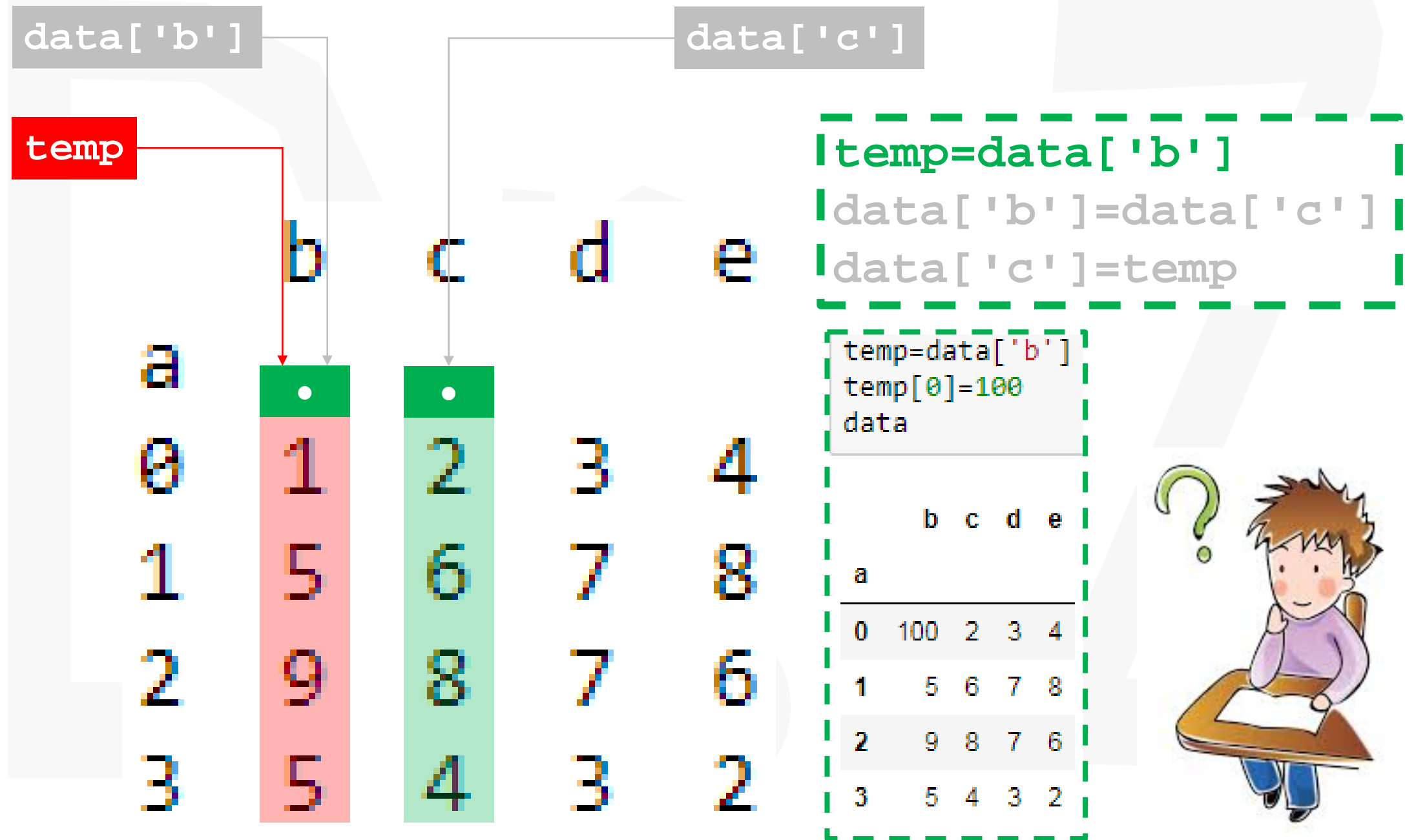
```
# Another wrong answer to swap to columns  
import pandas as pd  
data=pd.read_csv('HT001a.csv', index_col=0, header=0)  
print(data)  
data['b'], data['c']=data['c'], data['b']  
print(data)
```

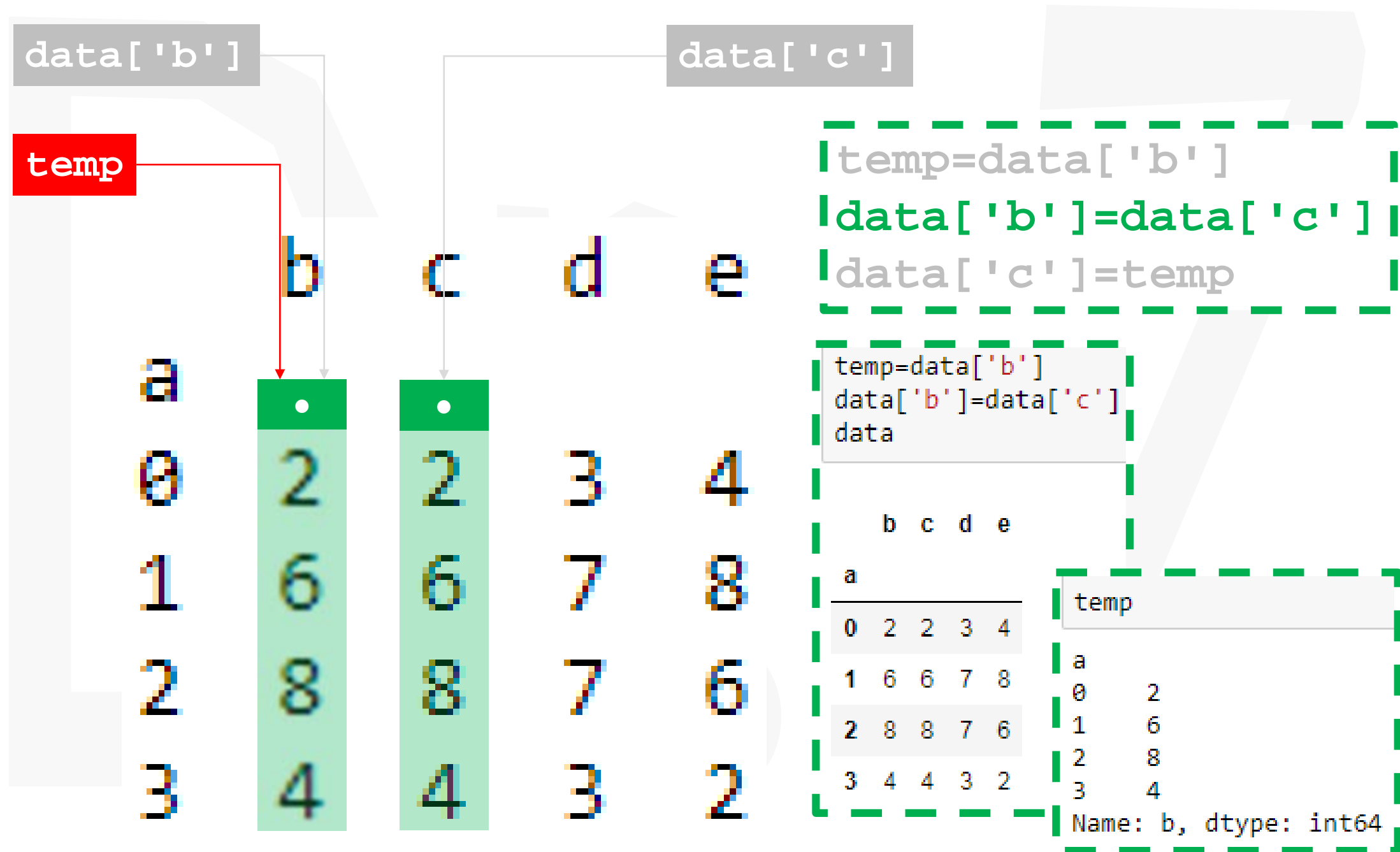
	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

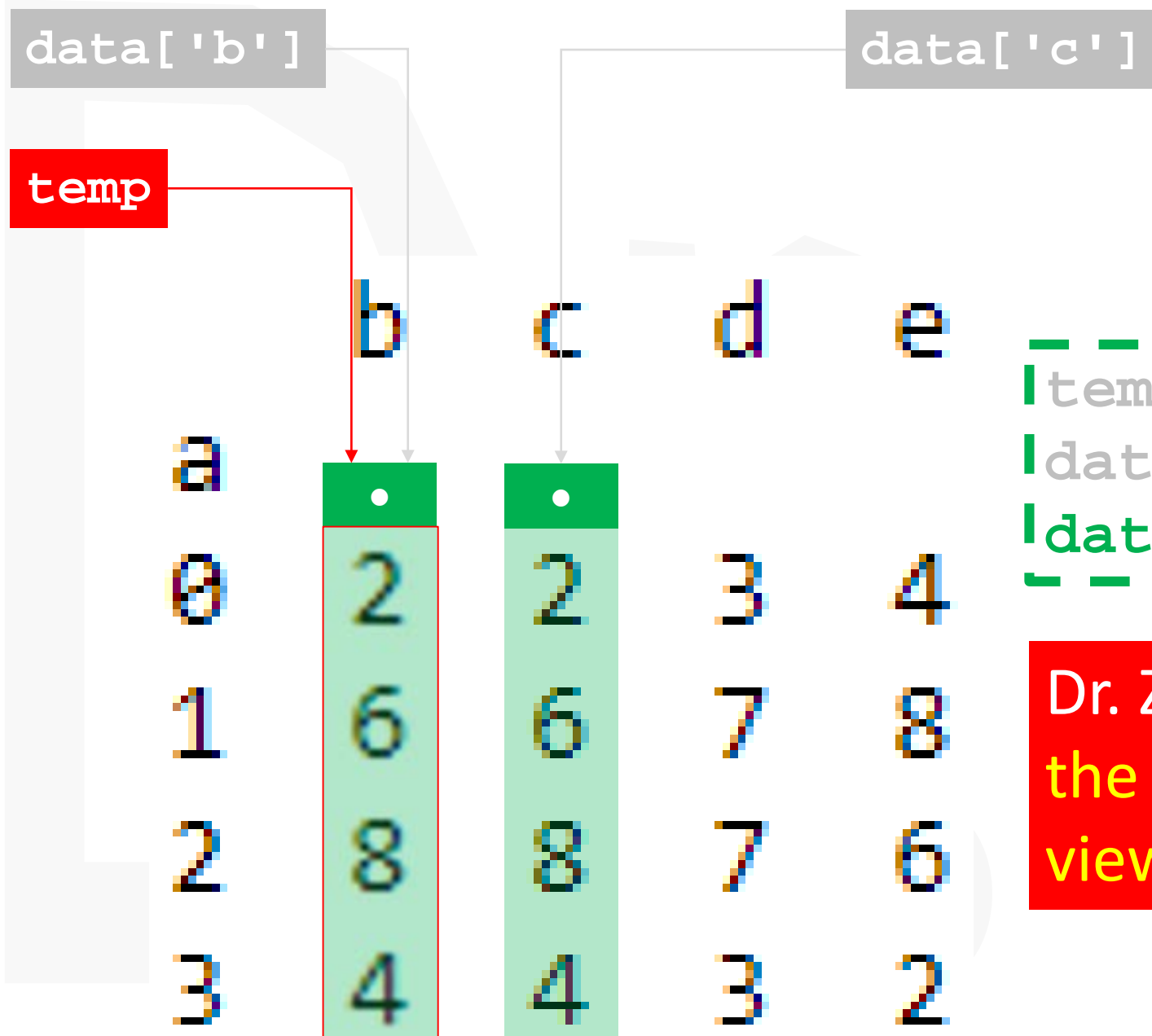


	b	c	d	e
a				
0	2	2	3	4
1	6	6	7	8
2	8	8	7	6
3	4	4	3	2

????







```
temp=data['b']  
data['b']=data['c']  
data['c']=temp
```

Dr. Z's interpretation on
the assignment to a
view



```
data=[[1,2],[3,4]]  
print(data)  
data[0]=data[1]  
print(data)  
data[0][0]=100  
print(data)
```

```
[[1, 2], [3, 4]]  
[[3, 4], [3, 4]]  
[[100, 4], [100, 4]]
```



```
import numpy as np  
x=[[1,2],[3,4]]  
data=np.array(x)  
print(data)  
data[0]=data[1]  
print(data)  
data[0][0]=100  
print(data)
```

```
[[1 2]  
 [3 4]]  
[[3 4]  
 [3 4]]  
[[100  4]  
 [ 3  4]]
```

← copy

```
import pandas as pd  
x=[[1,2],[3,4]]  
data=pd.DataFrame(x)  
print(data)  
data.iloc[0]=data.iloc[1]  
print(data)  
data.iloc[0][0]=100  
print(data)
```

```
   0  1  
0  1  2  
1  3  4  
   0  1  
0  3  4  
1  3  4  
   0  1  
0 100  4  
1   3  4
```

← copy



```
# swap two columns (1)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
temp=data['b'].copy()
data['b']=data['c']
data['c']=temp
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
a				
0	2	1	3	4
1	6	5	7	8
2	8	9	7	6
3	4	5	3	2

```
# swap two columns (2)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data['b'], data['c']=data['c'].copy(), data['b'].copy()
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
a				
0	2	1	3	4
1	6	5	7	8
2	8	9	7	6
3	4	5	3	2

.copy()

Series.copy(deep=True)

DataFrame.copy(deep=True)



```
# swap two columns (3)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data['b'], data['c']=data['c'], data['b'].copy()
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
	b	c	d	e

a				
0			3	4
1			7	8
2			7	6
3			3	2

Q: Will this code work?

131



```
# swap two columns (?)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data['b'], data['c']=data['c'].copy(), data['b']
print(data)
```

	b	c	d	e
a	1	2	3	4
0	5	6	7	8
1	9	8	7	6
2	5	4	3	2
3	b	c	d	e

a				
0			3	4
1	????		7	8
2			7	6
3			3	2

Q: Will this code work?



```
# swap two columns (4)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data[['b', 'c']]=data[['c', 'b']]#.copy()
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
a				
0	2	1	3	4
1	6	5	7	8
2	8	9	7	6
3	4	5	3	2

[] fancy indexing
does not match
labels.

Interesting...

131





```
# swap two columns (5)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data[['b','c']]=data.loc[:,['c','b']] #.copy()
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
	b	c	d	e
a				
0	2	1	3	4
1	6	5	7	8
2	8	9	7	6
3	4	5	3	2

[] fancy indexing
does not match
labels.

Interesting...

131



```
# Another wrong answer to swap to columns
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.loc[:,['b','c']]=data.loc[:,['c','b']]
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

1. Select data via []
2. Select data via
.loc or .iloc

.loc and .iloc
will match labels

Q: How to remove
the label?




```
# Another wrong answer to swap to columns
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.loc[:,['b','c']]=data[['c','b']]
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

1. Select data via []
2. Select data via
.loc or .iloc

.loc and .iloc
will match labels

Q: How to remove
the label?



```
# swap two columns (6)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.loc[:,['b','c']]=data.loc[:,['c','b']].values#.copy()
print(data)
```

	b	c	d	e
a	1	2	3	4
0	5	6	7	8
1	9	8	7	6
2	5	4	3	2

	b	c	d	e
a	2	1	3	4
0	6	5	7	8
1	8	9	7	6
2	4	5	3	2

131

1. Select data via []
2. Select data via
.loc or .iloc

.loc and .iloc
will match labels

DataFrame.values
will remove the labels.



```
# swap two columns (7)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.loc[:,['b','c']]=data[['c','b']].values#.copy()
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
a				
0	2	1	3	4
1	6	5	7	8
2	8	9	7	6
3	4	5	3	2

131

1. Select data via []
2. Select data via
.loc or .iloc

.loc and .iloc
will match labels

DataFrame.values
will remove the labels.



Summary:

Use one command to swap two columns using column labels in data.

```
1. data['b'], data['c']=data['c'].copy(), data['b'].copy()
```

```
2. data['b'], data['c']=data['c'], data['b'].copy()
```

```
3. data[['b','c']]=data[['c','b']].copy()
```

```
4. data[['b','c']]=data.loc[:,['c','b']].copy()
```

```
5. data.loc[:,['b','c']]=data.loc[:,['c','b']].values.copy()
```

```
6. data.loc[:,['b','c']]=data[['c','b']].values.copy()
```

131

Dr. Z: ⇒ Better to use **.copy()**.



How about rows?

Use one command to swap two rows in data.

1. Label-based indexing/slicing

```
data.loc[[0,1],:] = data.loc[[1,0],:].values.copy()
```



2. Position-based indexing/slicing

```
data.iloc[[0,1],:] = data.iloc[[1,0],:].values.copy()
```

Dr. Z: \Rightarrow Better to use `.copy()`.





```
# swap two rows (1)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.loc[[0,1],:]=data.loc[[1,0],:].values#.copy()
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
a				
0	5	6	7	8
1	1	2	3	4
2	9	8	7	6
3	5	4	3	2



133

132

```
# swap two rows (2)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.iloc[[0,1],:]=data.iloc[[1,0],:].values#.copy()
print(data)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b	c	d	e
a				
0	5	6	7	8
1	1	2	3	4
2	9	8	7	6
3	5	4	3	2



Use one command

- ✓ `data(:, {'b', 'c'}) = data(:, {'c', 'b'})`
- ✓ `data[:, {'b', 'c'}] = data[:, {'c', 'b'}]`

134

✗ `data(:, {'b', 'c'}) = data[:, {'c', 'b'}]`

Right hand side of an assignment into a table must be another table or a cell array.

✗ `data[:, {'b', 'c'}] = data(:, {'c', 'b'})`

The following error occurred converting from table to double:



Swap two columns (1)

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
data(:,{'b','c'})=data(:,{'c','b'})
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	b	c	d	e
0	2	1	3	4
1	6	5	7	8
2	8	9	7	6
3	4	5	3	2

134

Swap two columns (2)

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
data(:,{'b','c'})=data(:,{'c','b'})
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	b	c	d	e
0	2	1	3	4
1	6	5	7	8
2	8	9	7	6
3	4	5	3	2



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

How about rows?

✓ label-based indexing

Use one command

- ✓ `data({'0','1'},:)=data({'1','0'},:)`
- ✓ `data[{'0','1'},:]=data[{'1','0'},:]`

135



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

Swap two rows, label-based (1)

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
data({'0','1'},:)=data({'1','0'},:)
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	b	c	d	e
0	5	6	7	8
1	1	2	3	4
2	9	8	7	6
3	5	4	3	2

135

Swap two rows, label-based (2)

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
data({'0','1'},:)=data({'1','0'},:)
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	b	c	d	e
0	5	6	7	8
1	1	2	3	4
2	9	8	7	6
3	5	4	3	2

Slide
248/307



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

How about using position-based indexing?

➤ Swap two columns

✓ `data(:, [1, 2]) = data(:, [2, 1])`

✓ `data{:, [1, 2]} = data{:, [2, 1]}`

136

➤ Swap two rows

✓ `data([1, 2], :) = data([2, 1], :)`

✓ `data{[1, 2], :} = data{[2, 1], :}`

137

Use one command



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance



Use temp

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
temp=data(1,:);
data(1,:)=data(2,:);
data(2,:)=temp
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	b	c	d	e
0	5	6	7	8
1	1	2	3	4
2	9	8	7	6
3	5	4	3	2



138

Use deal

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
[data(1,:),data(2,:)]=deal(data(2,:), data(1,:))
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
data =
```

	b	c	d	e
0	5	6	7	8
1	1	2	3	4
2	9	8	7	6
3	5	4	3	2



139

pandas.DataFrame.apply

```
DataFrame.apply(func, axis=0, broadcast=None, raw=False, reduce=None,  
result_type=None, args=(), **kwargs) [source]
```

Apply a function along an axis of the DataFrame.

Objects passed to the function are Series objects whose index is either the DataFrame's index (`axis=0`) or the DataFrame's columns (`axis=1`). By default (`result_type=None`), the final return type is inferred from the return type of the applied function. Otherwise, it depends on the `result_type` argument.





```
#A Numpy universal function
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.apply(np.sqrt)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

140

	b	c	d	e
a				
0	1.000000	1.414214	1.732051	2.000000
1	2.236068	2.449490	2.645751	2.828427
2	3.000000	2.828427	2.645751	2.449490
3	2.236068	2.000000	1.732051	1.414214

1. Using a Numpy universal function, **fun**, (in the case same as **np.fun(data)**)

Dr. Z: How about `math.sqrt`?



```
#A reducing function, axis=0
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.apply(np.sum,axis=0)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

b	20
c	20
d	20
e	20

dtype: int64

2. Using a reducing
function on
either axis

141

axis=0



```
#A reducing function, axis=1
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.apply(np.sum,axis=1)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

a	
0	10
1	26
2	30
3	14

dtype: int64

142

2. Using a reducing
function on
either axis

axis=1



```
#using a user-defined function, axis=0, return a single number
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.apply(lambda x: x[0]**2+np.sum(x),axis=0)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

b	21
c	24
d	29
e	36

dtype: int64

143

3. Using a user-defined function

axis=0

The function returns a single value.

Dr. Z: Can I use `sum()`?



```
#using a user-defined function, axis=1, return a list
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.apply(lambda x: [x['d']**2, x['c']**2+np.sum(x)],axis=1)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

a		
0	[9, 14]	
1	[49, 62]	
2	[49, 94]	
3	[9, 30]	

dtype: object

3. Using a user-defined function

axis=1

The function returns a list.



```
#using a user-defined function, axis=1, return a list, result_type='expand'  
import pandas as pd  
import numpy as np  
data=pd.read_csv('HT001a.csv', index_col=0, header=0)  
print(data)  
data.apply(lambda x: [x['d']**2, x['c']**2+sum(x)],  
           axis=1, result_type='expand')
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

0	1
---	---

a	
0	9 14
1	49 62
2	49 94
3	9 30

3. Using a user-defined function

145

The function returns a list.

Passing **result_type='expand'** will expand list-like results to columns of a DataFrame.



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

```
#using a user-defined function, axis=1, return a Series
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
data.apply(lambda x: pd.Series([x['d']**2, x['c']**2+sum(x)],
                                index=['d**2', 'c**2+sum']),
           axis=1)
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

d**2 c**2+sum

a		
0	9	14
1	49	62
2	49	94
3	9	30

3. Using a user-defined function

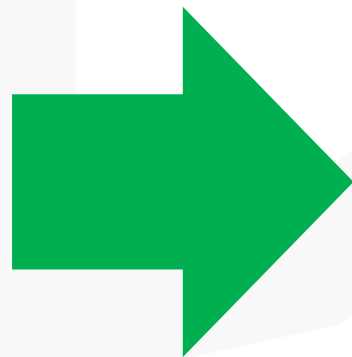
146

The function returns a list.

Returning a **Series** inside the function is similar to passing `result_type='expand'`. The resulting column names will be the Series index.

Homework Question:

```
dataset01.csv - Notepad
File Edit Format View Help
S,K,r,q,sigma,T,c
1403,1350,0.0534,0.0118,0.26,0.102777778,80.828
1403,1375,0.0534,0.0118,0.267,0.102777778,66.084
1403,1400,0.0534,0.0118,0.231,0.102777778,45.894
1403,1425,0.0534,0.0118,0.213,0.102777778,30.955
1403,1450,0.0534,0.0118,0.198,0.102777778,19.224
```



	S	K	r	q	sigma	T	c	BS
0	1403	1350	0.0534	0.0118	0.260	0.102778	80.828	80.827847
1	1403	1375	0.0534	0.0118	0.267	0.102778	66.084	66.084173
2	1403	1400	0.0534	0.0118	0.231	0.102778	45.894	45.894142
3	1403	1425	0.0534	0.0118	0.213	0.102778	30.955	30.955446
4	1403	1450	0.0534	0.0118	0.198	0.102778	19.224	19.224057





Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

1. Use one command with the Pandas library function **pandas.read_csv** to load data from the CSV file, `dataset01.csv`, using the first row as column names. Name the data as **data**.
2. Define a function, **option_BS**, which computes and returns the European call option price using the following formula:

$$c = S \cdot e^{-q \cdot T} \cdot \Phi(d_1) - K \cdot e^{-r \cdot T} \cdot \Phi(d_2)$$

where

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r - q + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} \text{ and } d_2 = d_1 - \sigma\sqrt{T}$$

3. Use one command with the Pandas library function **pandas.DataFrame.apply** to compute the European call option price for each row of **data** and add the results to **data** as a new column, and name this column as **BS**.

HT008: Apply a function to each row/column of data

rowfun

R2018b

Apply function to table or timetable rows

147

Syntax

```
B = rowfun(func,A)
B = rowfun(func,A,Name,Value)
```

In **rowfun**, the number of parameters in the function should be the same as the number of columns in **A**. Each parameter denotes a column in **A**.

B is a table. To return a numeric vector instead of a table, use 'OutputFormat', 'uniform'.

varfun

Apply function to table or timetable variables

148

Syntax

```
B = varfun(func,A)
B = varfun(func,A,Name,Value)
```

In **varfun**, the function is a one-variable function. The variable denotes the whole column.





`rowfun(1)`

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
rowfun(@(x1,x2,x3,x4) x1+x2+x3+x4, data)
data(:, 'row_sum')=rowfun(@(x1,x2,x3,x4) x1+x2+x3+x4, data)
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
ans =
```

	Var1
0	10
1	26
2	30
3	14

```
data =
```

	b	c	d	e	row_sum
0	1	2	3	4	10
1	5	6	7	8	26
2	9	8	7	6	30
3	5	4	3	2	14

147

`data(:, 'row_sum') =`

Right hand side of an
assignment into a table
must be another table or
a cell array.



rowfun (2)

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
rowfun(@(x1,x2,x3,x4) x1+x2+x3+x4, data, 'OutputFormat', 'uniform')
data(:, 'row_sum')=rowfun(@(x1,x2,x3,x4) x1+x2+x3+x4, data, ...
    'OutputFormat', 'uniform')
```

data =

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

ans =

10
26
30
14

data =

	b	c	d	e	row_sum
0	1	2	3	4	10
1	5	6	7	8	26
2	9	8	7	6	30
3	5	4	3	2	14

147

data{ :, 'row_sum' } =



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

varfun (1)

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
varfun(@(x) sum(x), data)
data('column_mean',:)=varfun(@(x) sum(x), data)
```

data =

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

ans =

Fun_b	Fun_c	Fun_d	Fun_e
20	20	20	20

data =

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
column_mean	20	20	20	20

data('column_mean',:)=

148



varfun (2)

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
varfun(@(x) sum(x), data, 'OutputFormat', 'uniform')
data{'column_mean',:}=varfun(@(x) sum(x), data, 'OutputFormat', 'uniform')
```

data =

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

ans =

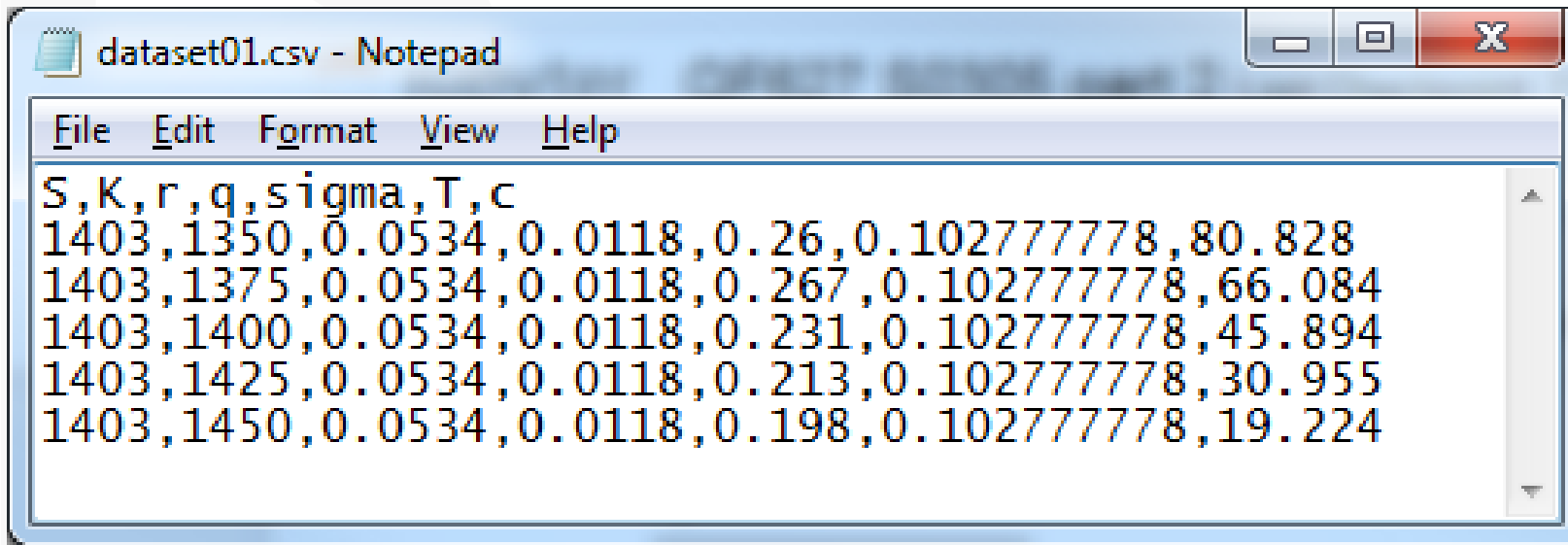
20 20 20 20

data =

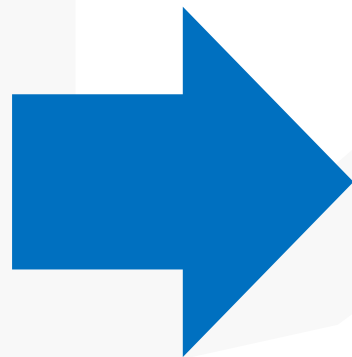
	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
column_mean	20	20	20	20

data{'column_mean',:}=

Homework Question:



```
dataset01.csv - Notepad
File Edit Format View Help
S,K,r,q,sigma,T,c
1403,1350,0.0534,0.0118,0.26,0.102777778,80.828
1403,1375,0.0534,0.0118,0.267,0.102777778,66.084
1403,1400,0.0534,0.0118,0.231,0.102777778,45.894
1403,1425,0.0534,0.0118,0.213,0.102777778,30.955
1403,1450,0.0534,0.0118,0.198,0.102777778,19.224
```



data =							
S	K	r	q	sigma	T	c	BS
1403	1350	0.0534	0.0118	0.26	0.10278	80.828	80.828
1403	1375	0.0534	0.0118	0.267	0.10278	66.084	66.084
1403	1400	0.0534	0.0118	0.231	0.10278	45.894	45.894
1403	1425	0.0534	0.0118	0.213	0.10278	30.955	30.955
1403	1450	0.0534	0.0118	0.198	0.10278	19.224	19.224



1. Load data from the CSV file, `dataset01.csv`, using the first row as column names. Name the data as **data**.
2. Define a function, **option_BS**, which computes and returns the European call option price using the following formula:

$$c = S \cdot e^{-q \cdot T} \cdot \Phi(d_1) - K \cdot e^{-r \cdot T} \cdot \Phi(d_2)$$

where

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r - q + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} \text{ and } d_2 = d_1 - \sigma\sqrt{T}$$

3. Use one command to compute the European call option price for each row of **data** and add the results to **data** as a new column, and name this column as **BS**.



HT009: Basic operations on two rows/columns of **data**

1. row(s) op row(s) (with the same label)
2. row(s) op row(s) (with different labels)
3. column(s) op column(s) (with the same label)
4. column(s) op column(s) (with different labels)
5. row(s) op column(s) (??? What operation???)

op: +, −, *, /, ** (or ^)



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

HT009: Basic operations on two rows/columns of data

- ✓ A row/column of a DataFrame can be a DataFrame, a Series, a Numpy 2D array or a Numpy 1D array.
- ✓ Rows/Columns of a DataFrame can be a DataFrame or a Numpy 2D array.

149

	DataFrame	Series	Numpy 2D array	Numpy 1D array
DataFrame	element-wise, aligned by labels	broadcasting, align DataFrame's column labels and Series' labels	element-wise, size must agree	use array as a row, broadcasting, element-wise, size must agree
Series		element-wise, aligned by labels	N.A.	element-wise, size must agree
Numpy 2D array			broadcasting, element-wise, size must agree	broadcasting
Numpy 1D array				element-wise, size must agree



```
#DataFrame op row-Series (DataFrame's column labels same as Series' label)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(id(data))
print(data)
print(data.iloc[0,:])
data=data+data.iloc[0,:]
print(id(data))
data
```

data=

150

a DataFrame op a row-Series

(row)

$(N,) \Rightarrow (1, N)$

Use one command to
add the first row to
every row in data.

a DataFrame (original row labels)

168589128

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

different id

b	1
c	2
d	3
e	4

broadcasting

Name: 0, dtype: int64
168589744

	b	c	d	e
0	1	2	3	4
1	1	2	3	4
2	1	2	3	4
3	1	2	3	4

	b	c	d	e
a				
0	2	4	6	8
1	6	8	10	12
2	10	10	10	10
3	6	6	6	6




```
#DataFrame op row-Series (DataFrame's column labels same as Series' Label)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(id(data))
print(data)
print(data.iloc[0,:])
data.values[:,:]=data+data.iloc[0,:]
print(id(data))
data
```

× data.values=
✓ data.values[:,:]=

168590080

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

same id

broadcasting

b	1
c	2
d	3
e	4

Name: 0, dtype: int64

168590080

	b	c	d	e
0	1	2	3	4
1	1	2	3	4
2	1	2	3	4
3	1	2	3	4

Use one command to
add the first row to
every row in data.

	b	c	d	e
a				
0	2	4	6	8
1	6	8	10	12
2	10	10	10	10
3	6	6	6	6

```
#assignment: DataFraem->2D array
%reset -f
import pandas as pd
import numpy as np
x=np.ones((3,3))
print(x)
df=pd.DataFrame(np.zeros((3,3)))
print(df)
x[:,:]=df #or df.values ✓
x
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

      0      1      2
0  0.0  0.0  0.0
1  0.0  0.0  0.0
2  0.0  0.0  0.0
```

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



```
#DataFrame op Series (DataFrame's column labels same as Series' label)
import pandas as pd
data=pd.read_csv('HT001b.csv', index_col=0, header=0)
print(data)
print(data.iloc[0,:])
data+data.iloc[0,:]
```

a DataFrame op a row-Series

	b	a.1	d	e	1
a					
2	1	2	3	4	
4	5	6	7	8	
3	9	8	7	6	
4	5	4	3	2	

b	1
a.1	2
d e	3
1	4

broadcasting

Name: 2, dtype: int64

	b	a.1	d	e	1
a					
2	2	4	6	8	
4	6	8	10	12	
3	10	10	10	10	
4	6	6	6	6	

	b	a.1	d	e	1
2	1	2	3	4	
4	1	2	3	4	
3	1	2	3	4	
4	1	2	3	4	

(row)

$(N,) \Rightarrow (1, N)$

Use one command to
add the first row to
every row in **data**.

a DataFrame (original row labels)

```
#DataFrame op 1D array (or list)
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
#arr=range(4)
arr=np.arange(4)
print(arr)
data+arr
```

a DataFrame op 1D array

(or list)

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
	[0 1 2 3]			

broadcasting

```
[[0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]]
```

a DataFrame (original row labels)

	b	c	d	e
a				
0	1	3	5	7
1	5	7	9	11
2	9	9	9	9
3	5	5	5	5

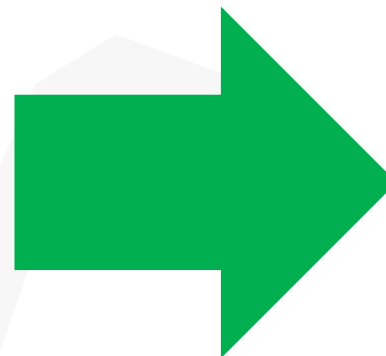




Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

	b	a.1	d	e	1
a					
2	1	2	3	4	
4	5	6	7	8	
3	9	8	7	6	
4	5	4	3	2	

152



	b	a.1	d	e	1
a					
2	2	3	4	5	
4	10	11	12	13	
3	18	17	16	15	
4	10	9	8	7	

Use one command to add the first column to every column in data.

- × `data+data.iloc[:,0]`
- ? `data+data.iloc[:,0].values`
- × `data+data.iloc[:,0].values.reshape(4,1)`
- ✓ `data+np.tile(data.iloc[:,0].values.reshape(4,1),4)`
- ✓ `data+np.tile(data.iloc[:,[0]].values,4)`
- ✓ `data.apply(lambda x: x+data.iloc[:,0].values, axis=0)`

numpy.tile

```
#Examples of numpy.tile  
import numpy as np  
print(np.tile([1,2],2))  
print(np.tile([1,2],(2,3)))
```

```
[1 2 1 2]  
[[1 2 1 2 1 2]  
 [1 2 1 2 1 2]]
```



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

Numpy Array Arithmetic Operations

153

$$1 \leq M_1 < M_2; 1 \leq N_1 < N_2$$

- ✓ $(M_1,)$ 1D array op $(M_1,)$ 1D array $\Rightarrow (M_1,)$ 1D array
 $\times (M_1,)$ 1D array op $(M_2,)$ 1D array
- ✓ (M_1, N_1) 2D array op (M_1, N_1) 2D array $\Rightarrow (M_1, N_1)$ 2D array
 $\times (M_1, N_1)$ 2D array op (M_1, N_2) 2D array
 $\times (M_1, N_1)$ 2D array op (M_2, N_1) 2D array
- ✓ $(M_1, 1)$ 2D array op $(1, N_1)$ 2D array $\Rightarrow (M_1, N_1)$ 2D array
- ✓ (M_1, N_1) 2D array op $(M_1, 1)$ 2D array $\Rightarrow (M_1, N_1)$ 2D array
- ✓ (M_1, N_1) 2D array op $(1, N_1)$ 2D array $\Rightarrow (M_1, N_1)$ 2D array
- ✓ (M_1, N_1) 2D array op $(N_1,)$ 1D array $\Rightarrow (M_1, N_1)$ 2D array
 $\times (M_1, N_2)$ 2D array op $(M_1,)$ 1D array





Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

#Test Numpy Array Arithmetic Operations

```
import numpy as np
```

```
A=[(3,), (3,), (3,4), (3,3), (3,3), (3,4), (3,4), (3,4), (3,4)]
```

```
B=[(3,), (4,), (3,4), (3,4), (4,3), (3,1), (1,4), (4,), (3,)]
```

```
for i in range(9):
```

```
    print("-----", A[i], "+", B[i], "-----")
```

```
    array1=np.zeros(A[i])
```

```
    array2=np.zeros(B[i])
```

```
    try:
```

```
        r=array1+array2
```

```
    except:
```

```
        print("Error.")
```

```
    else:
```

```
        print(r.shape)
```

```
----- (3,) + (3,) -----  
(3,)
```

```
----- (3,) + (4,) -----  
Error.
```

```
----- (3, 4) + (3, 4) -----  
(3, 4)
```

```
----- (3, 3) + (3, 4) -----  
Error.
```

```
----- (3, 3) + (4, 3) -----  
Error.
```

```
----- (3, 4) + (3, 1) -----  
(3, 4)
```

```
----- (3, 4) + (1, 4) -----  
(3, 4)
```

```
----- (3, 4) + (4,) -----  
(3, 4)
```

```
----- (3, 4) + (3,) -----  
Error.
```

#What is the output? (DataFrame+1D array)

```
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data.iloc[[0],:])
print(data.iloc[:,3].values)
data.iloc[[0],:]+data.iloc[:,3].values
```

	b	c	d	e
a				
0	1	2	3	4

[4 8 6 2]

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

DataFrame+1D array

	b	c	d	e
a				
0	5	10	9	6





```
#What is the output? (Series + Series, different labels)
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data.iloc[0,:])
print(data.iloc[:,3])
data.iloc[0,:]+data.iloc[:,3]
```

```
b    1
c    2
d    3
e    4
Name: 0, dtype: int64
```

```
a
0    4
1    8
2    6
3    2
Name: e, dtype: int64
```

```
b    NaN
c    NaN
d    NaN
e    NaN
0    NaN
1    NaN
2    NaN
3    NaN
dtype: float64
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

Series+Series



```
# How to calculate ...
import pandas as pd
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
print(data.iloc[1:,[0]])
print(data.iloc[: -1,1].values)
data.iloc[1:,[0]]+data.iloc[: -1,1].values
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

	b
a	
1	5
2	9
3	5

[2 6 8]

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-100-34a5128885b7> in <module>()
      5 print(data.iloc[1:,[0]])
      6 print(data.iloc[: -1,1].values)
----> 7 data.iloc[1:,[0]]+data.iloc[: -1,1].values
```

DataFrame+1D array

calculate \Rightarrow use array (label free)

Matrix Multiplication using Numpy 1D/2D arrays

(M, N) 2D array
or
(N,) 1D array used
as (1,N) 2D array



(N, P) 2D array
or
(N,) 1D array used
as (N,1) 2D array

A	B	A @ B
(M, N)	(N, P)	(M, P)
(M, N)	(N, 1)	(M, 1)
(M, N)	(N,)	(M,)
(1, N)	(N, P)	(1, P)
(N,)	(N, P)	(P,)
(1, N)	(N, 1)	(1, 1)
(N,)	(N,)	()

154

```
A=np.array([[1,2,3]])
B=np.array([[1],[2],[3]])
A@B
```

```
array([[14]])
```

```
A=np.array([1,2,3])
B=np.array([1,2,3])
A@B
```

```
14
```



	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 8 & 7 & 6 \end{pmatrix} \times \begin{pmatrix} 5 \\ 4 \\ 3 \\ 2 \end{pmatrix}$$

(3,4) (4,1)

155

Use one command to compute the matrix multiplication, using the first 3 rows in `data` as the first matrix, and using the last row (without finding the number of rows) as the second 1-column matrix.

- ✓ `data.values[:3] @ data.values[-1]`
- ✓ `data.values[:3] @ data.values[-1:].T`
- ✓ `data.values[:3] @ data.values[[-1]].T`



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

HT009: Basic operations on two rows/columns of data

✓ MATLAB tables do not support arithmetic operations.

HAHAHA....

HAHAHA....

✓ Using the dot syntax (\Rightarrow 1-column 2D array) or $\{\}$ -indexing, we obtain MATLAB arrays.

- ✓ 1D arrays are row matrices (or 1-row 2D arrays).
- ✓ Matrix dimensions must agree or one is a scalar.
- ✓ Manual broadcasting

156

Wow~~~



```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

1. Add the first row to every row.
2. "help repmat"
3. "help size"

Use one command

```
data =
```

	b	c	d	e
0	2	4	6	8
1	6	8	10	12
2	10	10	10	10
3	6	6	6	6

157

```
data{:,:}=data{:,:}+repmat(data{1,:),size(data,1),1)
```

numpy.tile(A, (M, N)) \Leftrightarrow **repmat(A, M, N)**

or **repmat(A, [M, N])**

<https://www.mathworks.com/help/matlab/ref/repmat.html>

<https://www.mathworks.com/help/matlab/ref/size.html>





```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

- ✓ Use one command to add the first row and last column (without using the size of `data`) of `data` elementwise and return the result in a column array.
- ✓ “help transpose”
- ✓ “help end”

```
ans =  
  
      5  
     10  
      9  
      6
```

158

```
transpose(data{1, :})+data{:, end}
```



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

- ✓ Use one command to add elements in the first column from the second row to the last row (without using the size of `data`) and elements in the second column from the first row to the second to the last row (without finding the size of `data`) element wise and return the result in a row array.

```
ans =
```

7	15	13
---	----	----

159

```
transpose(data{2:end,1}+data{1:end-1,2})
```


MATLAB Operators

$X * Y$: matrix multiplication

$X .* Y$: element-wise multiplication

$X ^ Y$: matrix power

$X . ^ Y$: element-wise power

X / Y : matrix right division

$X ./ Y$: element-wise divide

160



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 8 & 7 & 6 \end{pmatrix} \times \begin{pmatrix} 5 \\ 4 \\ 3 \\ 2 \end{pmatrix}$$

(3,4) (4,1)

Use one command to compute the matrix multiplication, using the first 3 rows in `data` as the first matrix, and using the last row (without using the size of `data`) as the second 1-column matrix.

161

```
data{1:3, :} * transpose(data{end, :})
```



HT010: Sort **data** using a row/column

pandas.DataFrame.sort

DataFrame.sort(columns=None, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last', **kwargs)

DEPRECATED: Use DataFrame.sort_values()

Sort DataFrame either by labels (along either axis) or by the values in column(s)

DEPRECATED

numpy.sort

numpy.sort(a, axis=-1, kind='quicksort', order=None)

Return a sorted **copy** of an array.

numpy.ndarray.sort

ndarray.sort(axis=-1, kind='quicksort', order=None)

Sort an array, **in-place**.

numpy.argsort

numpy.argsort(a, axis=-1, kind='quicksort', order=None)

Returns the indices that would sort an array.

pandas.DataFrame.sort_values

DataFrame.sort_values(by, axis=0, ascending=True, **inplace=False**, kind='quicksort', na_position='last') [\[source\]](#)

Sort by the values along either axis

by : str or list of str

Name or list of names to sort by.

- if axis is 0 or 'index' then by may contain index levels and/or column labels
 - if axis is 1 or 'columns' then by may contain column levels and/or index labels
- Changed in version 0.23.0: Allow specifying index or column level names.

axis : {0 or 'index', 1 or 'columns'}, default 0
Axis to be sorted

Returns:

index_array : ndarray, int

Array of indices that sort a along the specified axis. In other words, **a[index_array]** yields a sorted a.

[\[source\]](#)



```
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
print(data.sort_values(by='b',axis=0))
print(data.sort_values(by=['b','c'],axis=0))
print(np.sort(data,axis=0))
print(data)
data.values.sort(axis=0)
print(data)
```

- ❑ `np.sort` sort **every** row/column.
- ❑ `np.ndarray.sort` sort, **inplace, every** row/column.

	b	c	d	e
a	1	2	3	4
0	5	6	7	8
1	9	8	7	6
2	5	4	3	2
3	9	8	7	6



`pandas.DataFrame.sort_values:`
Sort data using rows/columns.

Use one command to sort rows
in ascending order using the 3rd
column?

	b	c	d	e
a	1	2	3	2
0	5	4	3	4
1	9	8	7	6
2	5	6	7	8
3	9	8	7	8

	b	c	d	e
a	1	2	3	4
0	5	6	7	8
1	9	8	7	6
2	5	4	3	2
3	9	8	7	6

inplace





```
#Use one command with pandas.DataFrame.sort_values
#to sort data using the 3rd row
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
print(id(data))
data.sort_values(by=data.index[2],axis=1,inplace=True)
```

163

```
print(data)
print(id(data))
```

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
158374880				
	e	d	c	b

	b	c	d	e
a				
0	4	3	2	1
1	8	7	6	5
2	6	7	8	9
3	2	3	4	5
158374880				

```
#Use one command with pandas.DataFrame.sort_values
#to sort data using the 3rd column
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
print(id(data))
data.sort_values(by=data.columns[2],axis=0,inplace=True)
print(data)
print(id(data))
```

162

	b	c	d	e
a				
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2
158373872				

	b	c	d	e
a				
0	1	2	3	4
3	5	4	3	2
1	5	6	7	8
2	9	8	7	6
158373872				

Use one command with
pandas.DataFrame.sort_values
to sort data using the 3rd row/column,
in-place (i.e. with the same id).



```
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
I=np.argsort(data.values,axis=0)
print(I)
print(data.values[I[:,2],:])
print(data)
```

	b	c	d	e	
a	0	1	2	3	4
1	5	6	7	8	
2	9	8	7	6	
3	5	4	3	2	

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

	b	c	d	e	
a	0	1	2	3	4
1	5	6	7	8	
2	9	8	7	6	
3	5	4	3	2	

Numpy ndarray:
sort rows using
the 3rd column.

162



? Sort data
? Use one command
? In-place/same id

```
import pandas as pd
import numpy as np
data=pd.read_csv('HT001a.csv', index_col=0, header=0)
print(data)
I=np.argsort(data.values,axis=1)
print(I)
print(data.values[:,I[2,:]])
print(data)
```

	b	c	d	e	
a	0	1	2	3	4
1	5	6	7	8	
2	9	8	7	6	
3	5	4	3	2	

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

	b	c	d	e	
a	0	1	2	3	4
1	5	6	7	8	
2	9	8	7	6	
3	5	4	3	2	

Numpy ndarray:
sort columns
using the 3rd row.

163



? Sort data
? Use one command
? In-place/same id



sort

(total row/column sort)



sortrows

(sort rows)

`sort` and `sortrows` are functions on matrices.
`sort` does not work on tables, `sortrows`
works on tables.





Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

```
>> help sort
```

```
sort      Sort in ascending or descending order.
```

```
For vectors, sort(X) sorts the elements of X in ascending order.  
For matrices, sort(X) sorts each column of X in ascending order.  
For N-D arrays, sort(X) sorts along the first non-singleton  
dimension of X. When X is a cell array of strings, sort(X) sorts  
the strings in ASCII dictionary order.
```

```
Y = sort(X,DIM,MODE)
```

```
has two optional parameters.
```

```
DIM selects a dimension along which to sort.
```

```
MODE selects the direction of the sort
```

```
    'ascend' results in ascending order
```

```
    'descend' results in descending order
```

```
The result is in Y which has the same shape and type as X.
```

```
[Y,I] = sort(X,DIM,MODE) also returns an index matrix I.
```

```
If X is a vector, then Y = X(I).
```

```
If X is an m-by-n matrix and DIM=1, then
```

```
    for j = 1:n, Y(:,j) = X(I(:,j),j); end
```




```
x=[1 2 3;3 1 2;2 1 3]
```

```
x =  
     1     2     3  
     3     1     2  
     2     1     3
```

```
[y,I]=sort(x,1,'ascend')
```

```
[y,I]=sort(x,1,'ascend')
```

```
y =  
     1     1     2  
     2     1     3  
     3     2     3
```

↓ ↓ ↓

```
I =  
     1     2     2  
     3     3     1  
     2     1     3
```

x(I(:,2),:) sort rows of x using the second column

```
x(I(:,2),:)
```

```
ans =  
     3     1     2  
     2     1     3  
     1     2     3
```

↓

sort a matrix

```
[y,I]=sort(x,2,'ascend')
```

```
[y,I]=sort(x,2,'ascend')
```

```
y =  
     1     2     3  
     1     2     3  
     1     2     3
```

→ → →

```
I =  
     1     2     3  
     2     3     1  
     2     1     3
```

x(:,I(:,2)) sort columns of|x| using the second row

```
x(:,I(:,2))
```

```
ans =  
     2     3     1  
     1     2     3  
     1     3     2
```

→



```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
[y,I]=sort(data{:,:},1,'ascend')
```

```
y =
```

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

```
I =
```

	1	1	1	4
2	4	4	4	1
4	2	2	2	3
3	3	3	3	2

sort rows of data using the 3rd column

```
data(I(:,3),:)
```

```
ans =
```

	b	c	d	e
0	1	2	3	4
3	5	4	3	2
1	5	6	7	8
2	9	8	7	6

sort data

```
[y,I]=sort(data{:,:},2,'ascend')
```

```
y =
```

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

```
I =
```

	1	2	3	4
1	1	2	3	4
4	4	3	2	1
4	4	3	2	1

sort columns of data using the 3rd row

```
data(:,I(3,:))
```

```
ans =
```

	e	d	c	b
0	4	3	2	1
1	8	7	6	5
2	6	7	8	9
3	2	3	4	5



```
>> help sortrows
```

```
sortrows Sort rows in ascending order.
```

```
Y = sortrows(X) sorts the rows of the matrix X in ascending order as a group. X is a 2-D numeric or char matrix. For a char matrix containing strings in each row, this is the familiar dictionary sort. When X is complex, the elements are sorted by ABS(X). Complex matches are further sorted by ANGLE(X). X can be any numeric or char class. Y is the same size and class as X.
```

```
sortrows(X,COL) sorts the matrix based on the columns specified in the vector COL. If an element of COL is positive, the corresponding column in X will be sorted in ascending order; if an element of COL is negative, the corresponding column in X will be sorted in descending order. For example, sortrows(X,[2 -3]) sorts the rows of X first in ascending order for the second column, and then by descending order for the third column.
```

```
sortrows(X) ⇔ sortrows(X,1:size(X,2))
```

```
[Y,I] = sortrows(X) and [Y,I] = sortrows(X,COL) also returns an index matrix I such that Y = X(I,:).
```



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

```
>> x=[1 3 2;2 1 3;1 2 3]
x =
     1     3     2
     2     1     3
     1     2     3

>> sortrows(x)
ans =
     1     3     2
     2     1     3
     1     2     3
```

↓ ↓ ↓

```
>> sortrows(x, [1,-2])
ans =
     1     3     2
     1     2     3
     2     1     3
```

```
>> sortrows(x, 2)
ans =
     2     1     3
     1     2     3
     1     3     2
```

sortrows(x, 2) sort **x**
using the second column.

Q: How to use **sortrows**
to sort **x** using the second
row?
[Hint: **transpose(x)**]



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

```
>> x=[1 3 2;2 1 3;1 2 3]
```

```
x =
```

1	3	2
2	1	3
1	2	3

```
>> transpose(sortrows(transpose(x),2))
```

```
ans =
```

3	1	2
1	2	3
2	1	3

Use **sortrows** to sort **x**
using the second row?

⇒ How to use `sortrows` to sort data?

Use `sortrows` to sort data using the 3rd column

```
% slide 300 (1)
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
% slide 300 (2)
sortrows(data,3)
```

```
ans =
```

	b	c	d	e
0	1	2	3	4
3	5	4	3	2
1	5	6	7	8
2	9	8	7	6

```
% slide 300 (3)
data
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
% slide 300 (4)
data=sortrows(data,3)
```

```
data =
```

	b	c	d	e
0	1	2	3	4
3	5	4	3	2
1	5	6	7	8
2	9	8	7	6

165

Use `sortrows` to sort data using a row?

Use one command?



Use sortrows to sort data using the 3rd row

```
data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)
```

```
data =
```

	b	c	d	e
0	1	2	3	4
1	5	6	7	8
2	9	8	7	6
3	5	4	3	2

```
[y,I]=sortrows(transpose(data{:,:}),3);  
data(:,I)
```

```
ans =
```

	e	d	c	b
0	4	3	2	1
1	8	7	6	5
2	6	7	8	9
3	2	3	4	5

Indexing on Assignment

When assigning values from one matrix to another matrix, you can use any of the styles of indexing covered in this section. Matrix assignment statements also have the following requirement.

In the assignment $A(J, K, \dots) = B(M, N, \dots)$, subscripts J, K, M, N , etc. may be scalar, vector, or array, provided that all of the following are true:

- The number of subscripts specified for B , not including trailing subscripts equal to 1, does not exceed `ndims(B)`.
- The number of nonscalar subscripts specified for A equals the number of nonscalar subscripts specified for B . For example, $A(5, 1:4, 1, 2) = B(5:8)$ is valid because both sides of the equation use one nonscalar subscript.
- The order and length of all nonscalar subscripts specified for A matches the order and length of nonscalar subscripts specified for B . For example, $A(1:4, 3, 3:9) = B(5:8, 1:7)$ is valid because both sides of the equation (ignoring the one scalar subscript 3) use a 4-element subscript followed by a 7-element subscript.

Q: How about A?

x =

1	2	3
4	5	6
7	8	9

166

$x(2, :) = x(:, 3)$

x =

1	2	3
3	6	9
7	8	9



Caution: Numpy Array (Slicings are Views)

```
>>> import numpy as np
>>> x=np.arange(1,10).reshape(3,3)
>>> x
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> x[1,:]=x[:,2]
>>> x
array([[1, 2, 3],
       [3, 9, 9],
       [7, 8, 9]])
```

```
>>> import numpy as np
>>> x=np.arange(1,10).reshape(3,3)
>>> x
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> x[1,:]=x[:,2].copy()
>>> x
array([[1, 2, 3],
       [3, 6, 9],
       [7, 8, 9]])
```

(Dr. Z: Is it because the assignment is from right to left?)(NO)

167





```
>>> import numpy as np
>>> x=np.arange(1,10).reshape(3,3)
>>> x
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
>>> x[:,1]=x[2,:].copy()
```

```
>>> x
array([[1, 7, 3],
       [4, 8, 6],
       [7, 9, 9]])
```

```
>>> import numpy as np
>>> x=np.arange(1,10).reshape(3,3)
>>> x
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> x[:,1]=x[2,:].copy()
>>> x
array([[1, 7, 3],
       [4, 8, 6],
       [7, 9, 9]])
```

(Dr. Z: it seems not.)



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance

```

[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
***** x[0,:]=x[:,8] *****
[[ 8 18 28 38 48 58 68 78 88 98]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
***** x[1,:]=x[:,8] *****
[[ 0  1  2  3  4  5  6  7  8  9]
 [ 8 88 28 38 48 58 68 78 88 98]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]

```

```

***** x[2,:]=x[:,8] *****
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [ 8 18 88 38 48 58 68 78 88 98]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
***** x[3,:]=x[:,8] *****
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [ 8 18 28 88 48 58 68 78 88 98]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
***** x[4,:]=x[:,8] *****
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [ 8 18 28 38 88 58 68 78 88 98]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]

```



```
***** x[5,:]=x[:,8] *****
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [ 8 18 28 38 48 88 68 78 88 98]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
***** x[6,:]=x[:,8] *****
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [ 8 18 28 38 48 58 88 78 88 98]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
***** x[7,:]=x[:,8] *****
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [ 8 18 28 38 48 58 68 88 88 98]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
```

```
***** x[8,:]=x[:,8] *****
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [ 8 18 28 38 48 58 68 78 88 98]
 [90 91 92 93 94 95 96 97 98 99]]
***** x[9,:]=x[:,8] *****
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [ 8 18 28 38 48 58 68 78 88 98]]
```

Dissection of the MATLAB code:

```

* 1 - clear;
  2 - clc;
  3 - figure('Name','Figure 1', 'units','inch','position',[1.5,1.5,12,8]);
* 4 - data=readtable('CC3.SI.csv');
1  5 - data(data.Volume==0,:)=[];
  6 - X=datetime(data.Date);
* 7 - Y=data.AdjClose;
2  8 - plot(X,Y,'k-','LineWidth',1);
3  9 - hold on;
4 10 - ave15=round(movmean(Y,15,'Endpoints','discard'),3);
5 11 - ave15(1:35)=[];
* 12 - ave50=round(movmean(Y,50,'Endpoints','discard'),3);
6 13 - daxis=X(50:end);
 14 - paxis=Y(50:end);
* 15 - plot(daxis,ave15,'b-');
 16 - plot(daxis,ave50,'c-');
7 17 - x=ave15-ave50;
8 18 - x(x>0)=1;
 19 - x(x<=0)=0;
9 20 - y=diff(x); %size is reduced by 1
10 21 - idxSell=find(y<0)+1;
    22 - idxBuy=find(y>0)+1;
    23 - plot(daxis(idxBuy),paxis(idxBuy), ...
*      'y.','MarkerSize',20,'Linewidth',1);
    25 - plot(daxis(idxSell),paxis(idxSell), ...
    26 - 'r.','MarkerSize',20,'Linewidth',1);
11 27 - legend('Adj Close', '15d', '50d', 'crossSell', 'crossBuy');
    28 - xlabel('Date');
* 29 - axis tight
    30 - set(gca,'XTickLabelRotation',30)

```

- ✓ readtable
- ✓ MATLAB table data selection
- ✓ plot
- ✓ **hold on**
- ✓ movmean (D.N.T.)
- ✓ round
- ✓ array comparison operations
- ✓ boolean index/logical array
- ✓ delete elements
- ✓ assignment
- ✓ array arithmetic operations
- ✓ **diff**
- ✓ find
- ✓ add legend
- ✓ add xlabel and ylabel
- ✓ add title



QF666
Programming and
Computational
Finance



Dr. Zhao Yibao
Senior Lecturer
Of Quantitative
Finance