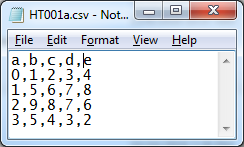
QF627 Programming and Computational Finance

S0304: Data Manipulation and Visualization

(part 2)

**Learning Outcomes:**

1. True /  False Both **Python** and **MATLAB** can **use one command** to import data from a CSV file, say **HT001a.csv** (see the screenshot), using the first row as column labels, and the first column as row labels and store the data in a variable with the name **data**.

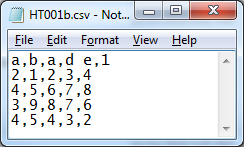


|  |
| --- |
| **Python** |
| **data=pd.read\_csv('Ht001a.csv',index\_col=0,header=0)** |
| **MATLAB** |
| **data=readtable('HT001a.csv','ReadRowNames',true,'ReadVariableNames',true)** |

1. If the filename is **newfolder\newfile.csv**, we need to use the string as follows:

|  |  |  |
| --- | --- | --- |
| **Python** | | |
| **Windows:** | Method 1: | **data=pd.read\_csv('newfolder\\newfile.csv')** |
| Method 2: | **data=pd.read\_csv(r'newfolder\newfile.csv')** |
| **Mac:** | **data=pd.read\_csv('newfolder/newfile.csv')** | |
| **MATLAB** | | |
| **Windows:** | **data=readtable('newfolder\newfile.csv')** | |
| **Mac:** | **data=readtable('newfolder\newfile.csv')** | |

1. True /  False **Python DataFrame** is a dict-like containers of Series. Duplicate **column** **labels** are not allowed. With the parameter **mangle\_dupe\_cols=True** (default value), duplicate columns will be specified as **X**, **X.1**, … **X.N**, rather than **X**, …, **X**. Numbers can be used as column names, e.g. **1**. However, duplicate row names are allowed. For example, for the CSV file **HT001b.csv**, the DataFrame **data** obtained from **pandas.read\_csv** has the following column labels and row labels.



|  |  |
| --- | --- |
| **data.columns** | **Index(['b','a.1','de','1'], dtype='object')** |
| **data.index** | **Int64Index(['2','4','3','4'], dtype='int64', name='a')** |

1. True /  False **MATLAB tables** do not allow duplicate column names (a.k.a. variable names) or row names. Column/Variable names must be a valid identifier. For example, **'1'** and **'d e'** cannot be used as a column/variable name. Function **readtable** will automatically convert them to **'x1'** and **'dE'**. However, **'1'** can be used as a row name. After loading **data** from **HT001b.csv** using **readtable**, column labels and row labels are as following:

|  |  |
| --- | --- |
| **data.Properties.VariableNames** | **{'b' 'a' 'dE' 'x1'}** |
| **data.Properties.RowNames** | **{'2' '4' '3' '4\_1'}** |

1. True /  False Both Python and MATLAB can use **position-based** **indexing/slicing** method to select data in **data**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Python** | | **MATLAB** | |
| **data** | | **data** | |
|  | |  | |
|  | **data.iloc[0,0]** |  | **data{1,1}** |
|  |  |
|  | **data.iloc[0:2,0:2]** |  | **data{1:2,1:2}** |
|  |  |
|  | **data.iloc[0,:]** |  | **data{1,:}** |
|  |  |
|  | **data.iloc[:,0]** |  | **data{:,1}** |
|  |  |
|  | **data.iloc[[2,0][0,1]]** |  | **data{[3,1][1,2]}** |
|  |  |

1. True /  False In Python, **data.values** returns the whole DataFrame as a 2D Numpy ndarray. In MATLAB, **data{:,:}** returns the whole table as a 2D array/matrix.
2. True /  False Both Python and MATLAB can use **label-based** **indexing/slicing** method to select data in **data**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Python** | | **MATLAB** | |
| **data** | | **data** | |
|  | |  | |
|  | **data.loc[2,'b'] or**  **data.loc['b'][2]** |  | **data{'2','b'}** |
|  |  |
|  | **data.loc[2:3,'b':'1']** | **Related image Related image Related image Related image** | |
|  |
|  | **data.loc[2,:]** |  | **data{'2',:}** |
|  |  |
|  | **data.loc[:,'a.1']** |  | **data{:,'a'}** |
|  |  |
|  | **data.loc[[2,3]['b','1']]** |  | **data{{'2','3'}{'b','1'}}** |
|  |  |

1. (Python) Add a column of **1**s to **data** with column name **f1**.

|  |
| --- |
| **data['f1']=1** |
| **data.loc[:,'f1']=1** |

1. (Python) Add a column of integers **0, 1, 2, 3** to **data** with column name **f2** through a Python basic iterable, such as **range**, **list**, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **data['f2']**  **or**  **data.loc[:,'f2']** | **=** | Using **range** | **range(4)** |
| Using **list** | **[0,1,2,3]** |
| **[[0],[1],[2],[3]]** |

1. (Python) Add a column of integers **0, 1, 2, 3** to **data** with column name **f3** through a Numpy ndarray generated by **numpy.arange**.

|  |  |  |
| --- | --- | --- |
| **data['f3'] or**  **data.loc[:,'f3']** | **=** | **np.arange(4)** |
| **np.arange(4),reshape(4,1)** |

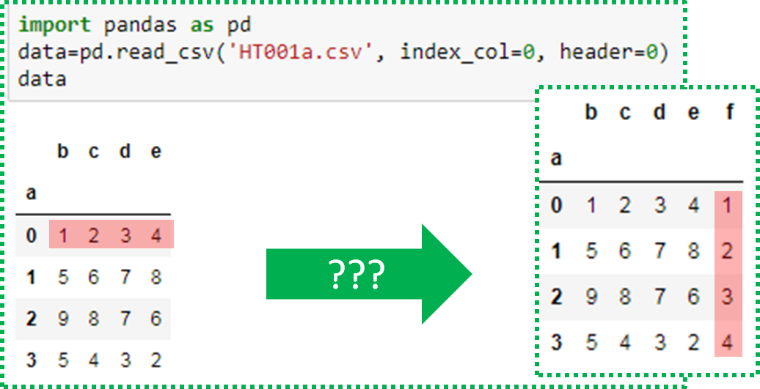
1. (Python) To add a column to **data** through a Series, we have the following results:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **data=pd.read\_csv('HT001b.csv', index\_col=0, header=0)**  **data** | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| **data['f4']= or data.loc[:,'f4']=** | | | | | | | | | | | | | | | | | |
| **pd.Series(range(3))** | | | | | | **pd.Series(range(4))** | | | | | | **pd.Series(range(5))** | | | | | |
|  | **b** | **a.1** | **d e** | **1** | **f4** |  | **b** | **a.1** | **d e** | **1** | **f4** |  | **b** | **a.1** | **d e** | **1** | **f4** |
| **a** |  | | | | | **a** |  | | | | | **a** |  | | | | |
| **2** | **1** | **2** | 3 | **4** | **2** | **2** | **1** | **2** | **3** | **4** | **2** | **2** | **1** | **2** | **3** | **4** | **2** |
| **4** | **5** | **6** | 7 | **8** | **NaN** | **4** | **5** | **6** | **7** | **8** | **NaN** | **4** | **5** | **6** | **7** | **8** | **4** |
| **3** | **9** | **8** | 7 | **6** | **NaN** | **3** | **9** | **8** | **7** | **6** | **3** | **3** | **9** | **8** | **7** | **6** | **3** |
| **4** | **5** | **4** | 3 | **2** | **NaN** | **4** | **5** | **4** | **3** | **2** | **NaN** | **4** | **5** | **4** | **3** | **2** | **4** |

1. (Python) To add a column to **data** through a 1-row or 1-column DataFrame, we have the following results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **data=pd.read\_csv('HT001b.csv', index\_col=0, header=0)**  **data** | | | | | | |
|  | | | | | | |
| **data['f5']=pd.DataFrame([[i] for i in range(3)]) or**  **data.loc[:,'f5']=pd.DataFrame([[i] for i in range(3)])** | | | | | | |
| **data** |  | **b** | **a.1** | **d e** | **1** | **f5** |
| **a** |  | | | | |
| **2** | **1** | **2** | **3** | **4** | **2** |
| **4** | **5** | **6** | **7** | **8** | **NaN** |
| **3** | **9** | **8** | **7** | **6** | **NaN** |
| **4** | **5** | **4** | **3** | **2** | **NaN** |
| **data['f5']=pd.DataFrame([[i] for i in range(4)]) or**  **data.loc[:,'f5']=pd.DataFrame([[i] for i in range(4)])** | | | | | | |
| **data** |  | **b** | **a.1** | **d e** | **1** | **f5** |
| **a** |  | | | | |
| **2** | **1** | **2** | **3** | **4** | **2** |
| **4** | **5** | **6** | **7** | **8** | **NaN** |
| **3** | **9** | **8** | **7** | **6** | **3** |
| **4** | **5** | **4** | **3** | **2** | **NaN** |
| **data['f5']=pd.DataFrame([[i] for i in range(5)]) or**  **data.loc[:,'f5']=pd.DataFrame([[i] for i in range(5)])** | | | | | | |
| **data** |  | **b** | **a.1** | **d e** | **1** | **f5** |
| **a** |  | | | | |
| **2** | **1** | **2** | **3** | **4** | **2** |
| **4** | **5** | **6** | **7** | **8** | **4** |
| **3** | **9** | **8** | **7** | **6** | **3** |
| **4** | **5** | **4** | **3** | **2** | **4** |
| **data['f5']=pd.DataFrame([[i for i in range(4)]]) or**  **data.loc[:,'f5']=pd.DataFrame([[i for i in range(4)]])** | | | | | | |
| **Error** | | | | | | |

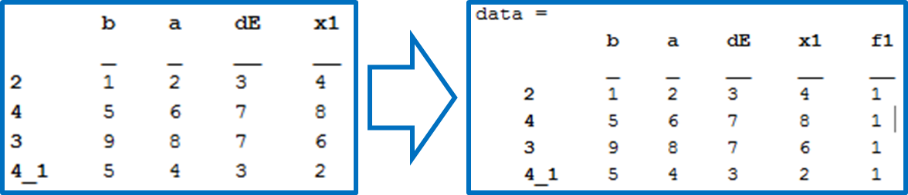
1. True /  False Homework question: (Python) Use one command to add a column to **data**, using the first row of **data**, and name this column **f**.



1. Complete the following diagram:

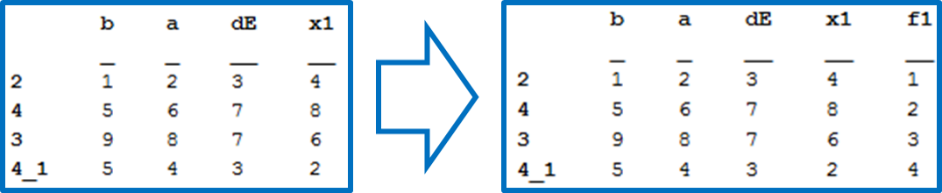
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DataFrame** | − | **values** | ⭢ | **ndarray (2D)** |
| ⏐ | **Hint** | | | ⏐ |
| * Selection by **[]** * Selection by **.loc** * Selection by **.iloc** | * Indexing |
| ⭣ | ⭣ |
| **Series** | − | **values** | ⭢ | **ndarray (1D)** |

1. (MATLAB) Add a column of **1**s to **data** with column name **f1**.



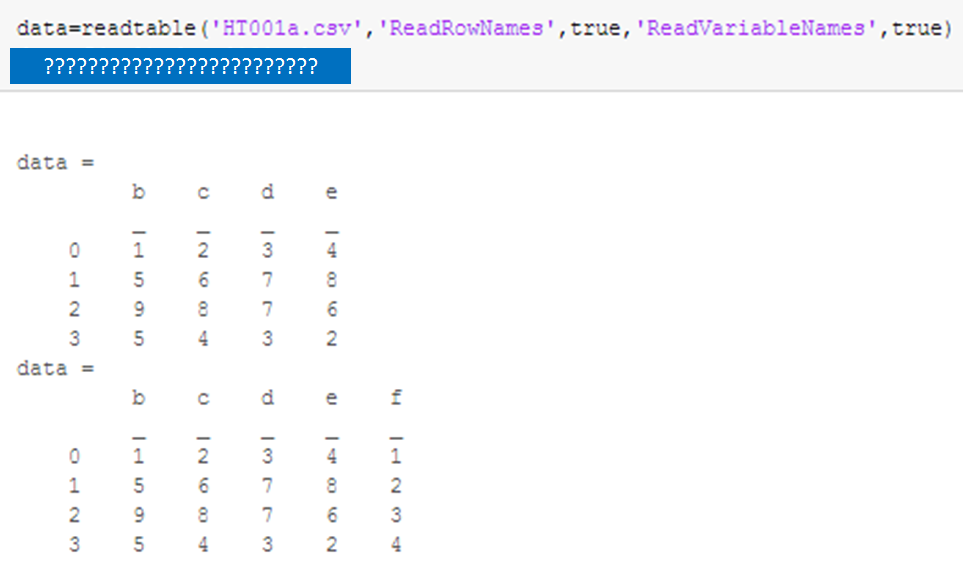
|  |  |  |
| --- | --- | --- |
| **data.f1(:,1)=1** | **data{:,'f1'}=1** | **data(:,'f1')={1}** |

1. (MATLAB) Add a column of integers **0, 1, 2, 3** to **data** with column name **f2** through a Python basic iterable, such as **range**, **list**, etc.



|  |  |
| --- | --- |
| **data.f1=[1;2;3;4]** | **data.f1(:,1)=[1;2;3;4]** |
| **data.f1(:,1)=1:4** | **data{:,'f1'}=[1;2;3;4]** |

1. True /  False Homework question: (MATLAB) Use one command to add a column to **data**, using the first row of **data**, and name this column **f**.



1. (Python) To **use one command** to add a row of **1**s to **data** with the row name **100**, we can use

|  |
| --- |
| **data.loc[100,:]=1 or** |
| **data=data.append(pd.DataFrame(1,**  **columns=data.columns,**  **index=[100])** |

1. (Python) To **use one command** to add a row of numbers, say **1**, **2**, **3** and **4**, to **data** with the row name **100**, we can use

|  |
| --- |
| **data.loc[100,:]=range(1,5) or** |
| **data=data.append(pd.DataFrame(np.arange(1,5),**  **columns=data.columns,**  **index=[100])** |

1. True /  False (Python) **pandas.DataFrame.append(*other*,…)** appends rows of ***other*** to the end of the dataframe and returns a new object. Columns not in this frame are added as new columns. Some examples are given as follows.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **import pandas as pd**  **data=pd.read\_csv('HT001a.csv', index\_col=0, header=0)**  **print(data)** | | | | | | |
|  | | | | | | |
| **data=data.append({k:v for (v,k) in enumerate(data.columns,1)}, ignore\_index=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** |
| **4** | | **1** | **2** | **3** | **4** |
| **data=data.append([{k:v for (v,k) in enumerate(data.columns,1)}], ignore\_index=False)** | | | | | | |
| **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** |
| **0** | | **1** | **2** | **3** | **4** |
| **data=data.append([{k:v for (v,k) in enumerate(data.columns,1)}], ignore\_index=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** |
| **4** | | **1** | **2** | **3** | **4** |
| **data=data.append(pd.Series(range(1,5),index=data.columns),ignore\_index=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** |
| **4** | | **1** | **2** | **3** | **4** |
| **data=data.append([pd.Series(range(1,5),index=data.columns)],ignore\_index=False)** | | | | | | |
| **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** |
| **0** | | **1** | **2** | **3** | **4** |
| **data=data.append([pd.Series(range(1,5),index=data.columns)],ignore\_index=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** |
| **4** | **1** | | **2** | **3** | **4** |

1. (MATLAB) To **use one command** to add a row of **1**s to **data** with the row name **100**, we can use **data{'100',:}=1 or data('100',:)={1}**.
2. (MATLAB) To **use one command** to add a row of numbers, say **1**, **2**, **3** and **4**, to **data** with the row name **100**, we can use **data{'100',:}=[1,2,3,4]** or **data{'100',:}=1:4**.
3. (Python) To **use one command** to delete column **b** in **data** we can use

**del data['b']** or **data.drop('b',axis=1.inplace=true)**.

1. (Python) To **use one command** to delete the **n**th column in **data** we can use

**del data[data.columns[n]]** or

**data.drop(data.columns[n],axis=1.inplace=true)**.

1. (Python) To **use one command** to delete row **b** in **data** we can use

**data.drop(0,inplace=true)**.

1. (Python) To **use one command** to delete the **n**th row in **data** we can use **data.drop(data.index[0],inplace=true)**.
2. (MATLAB) To **use one command** to delete a column **b** in **data** we can use

**data.b=[]** or **data(:,'b')=[]**.

1. (Python) To **use one command** to delete the **n**th column in **data** we can use **data(:,1)=[]**.
2. (Python) To **use one command** to delete a row **b** in **data** we can use

**data('b',:)=[]**.

1. (Python) To **use one command** to delete the **n**th row in **data** we can use **data(n,:)=[]**.
2. (Python) We can use the following code to swap the objects bounded by **a** and **b** respectively.

|  |  |
| --- | --- |
| **a=1**  **b=2** | |
| **#Solution 1**  **t=a**  **a=b**  **b=t** | **#Solution 2**  **a,b=b,a** |

1. (MATLAB) We can use the following code to swap the values stored in **a** and **b** respectively.

|  |  |
| --- | --- |
| **a=1**  **b=2** | |
| **%Solution 1**  **t=a**  **a=b**  **b=t** | **%Solution 2**  **[a,b]=deal(b,a)** |

1. (Python) To **use one command** to swap two columns (**b** and **c**) in **data**, we can use

|  |
| --- |
| **import pandas as pd**  **data=pd.read\_csv('HT001a.csv', index\_col=0, header=0)**  **print(data)** |
| **#Solution 1**  **data[:,['b','c']]=data[:,['c','b']].values** |
| **#Solution 2**  **data['b'], data['c']=data['c'].copy(), data['b'].copy()** |
| **#Solution 3**  **data['b'], data['c']=data['c'],data['b'].copy()** |
| **#Solution 4**  **data[['b','c']]=data[['c','b']]** |
| **#Solution 5**  **data[['b','c']]=data.loc[:,['c','b']]** |
| **#Solution 6**  **data[:,['b','c']]=data['c','b'].values** |

1. (Python) To **use one command** to swap two rows (**b** and **c**) in **data**, we can use

**data.loc[[b,c],:]=data.loc[[c,b],:].values,copy()**

1. (Python) To **use one command** to swap two rows (**m**th and **n**th) in **data**, we can use

**data.loc[[m,n],:]=data.loc[[n,m],:].values.copy()**

1. (MATLAB) To **use one command** (without using function **deal**) to swap two columns (**b** and **c**) in **data**, we can use

**data(:,{'b','c'})=data(:,{'c','b'})** or

**data{:,{'b','c'}}=data{:,{'c','b'}}**

1. (MATLAB) To **use one command** (without using function **deal**) to swap two rows (**b** and **c**) in **data**, we can use

**data({'b','c'},:)=data({'c','b'},:)** or

**data{{'b','c'},:}=data{{'c','b'},:}**

1. (MATLAB) To **use one command** (without using function **deal**) to swap two columns (**n**th and **m**th) in **data**, we can use

**data(:,[n,m])=data(:,[m,n])**  or

**data{:,[n,m]}=data{:,[m,n]}**

1. (MATLAB) To **use one command** (without using function **deal**) to swap two rows (**n**th and **m**th) in **data**, we can use

**data([n,m],:)=data([m,n],:)**  or

**data{[n,m],:}=data{[m,n],:}**

1. (MATLAB) To **use temp** (without using function **deal**) to swap two rows (**n**th and **m**th) in **data**, we can use

|  |
| --- |
| **temp=data(n,:);**  **data(n,:)=data(m,:);**  **data(m,:)=temp;** |

1. (MATLAB) To **use** function **deal** to swap two rows (**n**th and **m**th) in **data**, we can use

|  |
| --- |
| **[data(n,:),data(m,:)]=deal(data(m,:),data(n,:))** |

1. (Python) **Use one command** to compute and return the square root of every cell in **data** as a new dataframe, we can use

**data.apply(np.sqrt)** or **np.sqrt(data)**

1. (Python) **Use one command** to compute the sum of every column in **data**, we can use

**data.apply(np.sum,axis=0)**. The return value is a  DataFrame /  Series.

1. (Python) **Use one command** to compute the sum of every row in **data**, we can use

**data.apply(np.sum,axis=1)**. The return value is a  DataFrame /  Series.

1. (Python) **Use one command** **with the library function pandas.DataFrame.apply** to use the following formula to compute and return a number on every column in **data**:

**data.apply(lambda x:x[0]\*\*2+np.sum(x),axis=0)**. The command returns a  DataFrame /  Series.

1. (Python) **Use one command with the library function pandas.DataFrame.apply** to use the following formula to compute and return a list on every row in **data**:

**data.apply(lambda x=[x['d']\*\*2,x['c']\*\*2+np.sum(x)],axis=1)**. The return value is a  DataFrame /  Series.

1. (Python) Continued from 144, with the parameter **result\_type='expand'**, the return value is a  DataFrame /  Series. List-like results will be expanded to columns of DataFrame.
2. (Python) Continued from 144, if the function returns a Series, the return value is a

DataFrame /  Series. Similar to 145, the resulting column names will be the Series index.

1. True /  False (MATLAB) **B=rowfun(func,A)** applies the function **func** to rows in table **A**. The number of parameters in the function **func** should be the same as the number of columns in **A**. Each parameter denotes a column in **A**. The return value **B** is a table. To return a numeric vector instead of a table, use **B=rowfun(func,A,Name,Value)**.
2. True /  False (MATLAB) **B=varfun(func,A)** applies the function **func** to columns/variables in table **A**. The function **func** is a one-variable function. The variable denotes the whole column. The return value **B** is a table. To return a numeric vector instead of a table, use **B=varfun(func,A,Name,Value)**.
3. True /  False (Python) Complete the following table for basic operations on two rows/columns of **data**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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 |

1. (Python) **Use one command** to add the a row (e.g. the 1st row) to every row in **data** with **data** ***op*** **row-Series**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **import pandas as pd**  **data=pd.read\_csv('HT001a.csv', index\_col=0, header=0)**  **print(data)** | | | | | |
|  | | | | | |
| **data+data.iloc[0,:]** |  | **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** |
| **import pandas as pd**  **data=pd.read\_csv('HT001b.csv', index\_col=0, header=0)**  **print(data)** | | | | | |
|  | | | | | |
| **data+data.iloc[0,:]** |  | **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** |

1. (Python) **Use one command** to add a vector to every row in **data** with **data** ***op*** **1D-array** (or **list**)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **import pandas as pd**  **data=pd.read\_csv('HT001a.csv', index\_col=0, header=0)**  **print(data)** | | | | | |
|  | | | | | |
| **data+range(4)**  or  **data+np.arange(4)** |  | **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** |

1. (Python) **Use one command** to add a column (e.g. the 1st column) to every row in **data** without changing the **id** of **data**.

|  |
| --- |
| **data+np.tile(data.iloc[:,0].values.reshape(4,1),4)** |
| **data+np.tile(data.iloc[:,[0]].values,4)** |
| **data.apply(lambdax: x+data.iloc[:,0].values,axis=0)** |
|  |

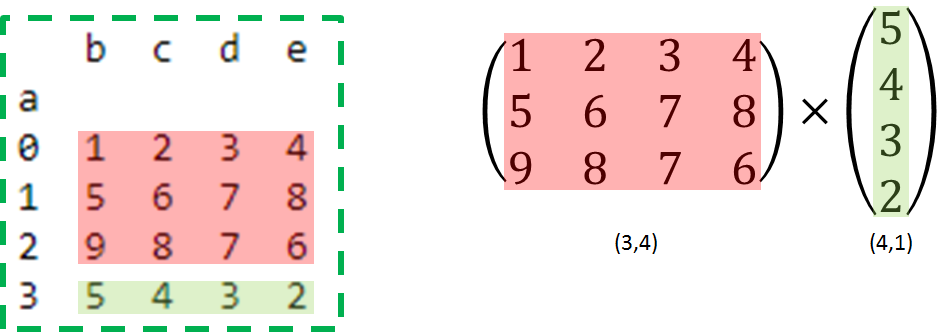
1. (Python) Numpy array arithmetic operations: **array1** ***op*** **array2**

|  |  |  |
| --- | --- | --- |
| **array1** | **array2** | **array1 + array2** |
| (3,) 1D array | (3,) 1D array | Error /  (**3**,) 1D array |
| (3,) 1D array | (4,) 1D array | Error /  (,) 1D array |
| (3,4) 2D array | (3,4) 2D array | Error /  (**3**, **4**) 2D array |
| (3,3) 2D array | (3,4) 2D array | Error /  (,) 2D array |
| (3,3) 2D array | (4,3) 2D array | Error /  (,) 2D array |
| (3,4) 2D array | (3,1) 2D array | Error /  (**3**, **4**) 2D array |
| (3,4) 2D array | (1,4) 2D array | Error /  (**3**, **4**) 2D array |
| (3,4) 2D array | (4,) 1D array | Error /  (**3**, **4**) 2D array |
| (3,4) 2D array | (3,) 1D array | Error /  (,) 2D array |

1. (Python) Matrix multiplication using Numpy 1D/2D arrays.

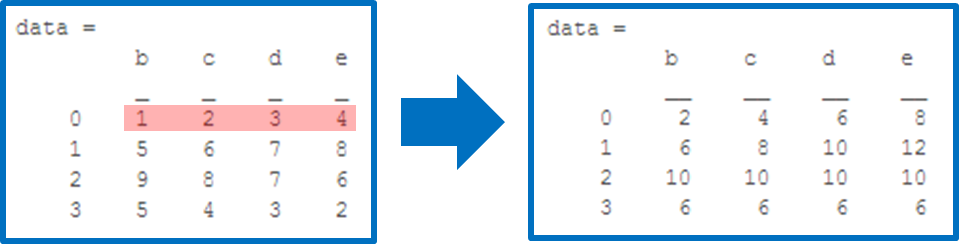
|  |  |  |
| --- | --- | --- |
| Shape of **A** | Shape of **B** | Shape of **A@B** |
| **(M, N)** | **(N, P)** | **(M,P)** |
| **(M, N)** | **(N, 1)** | **(M,1)** |
| **(M, N)** | **(N, )** | **(M,)** |
| **(1, N)** | **(N, P)** | **(1,P)** |
| **(N, )** | **(N, P)** | **(P,)** |
| **(1, N)** | **(N, 1)** | **(1,1)** |
| **(N, )** | **(N, )** | **(     )** |

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 finding the number of rows) in **data** as the second 1-column matrix.



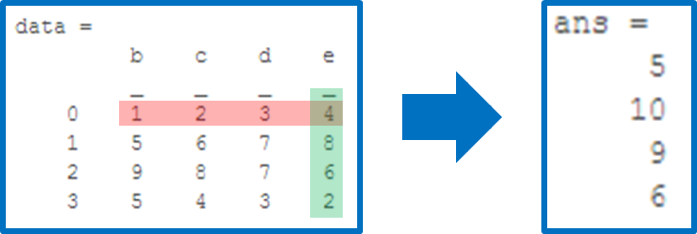
|  |
| --- |
| **data.values[:3]@data.values[-1]** |
| **data.values[:3]@data.values[-1:].T** |
| **data.values[:3]@data.values[[-1]].T** |

1. True /  False (MATLAB) MATLAB tables do not support arithmetic operations. We can use the dot syntax or **{}**-indexing/slicing to obtain arrays. A 1D array is a 1-row 2D array. Matrix dimensions must agree or one is a scalar when applying arithmetic operation. There is no auto “broadcasting” except for the scalar.
2. (MATLAB) **Use one command** to add the first row of **data** to every row in **data**.



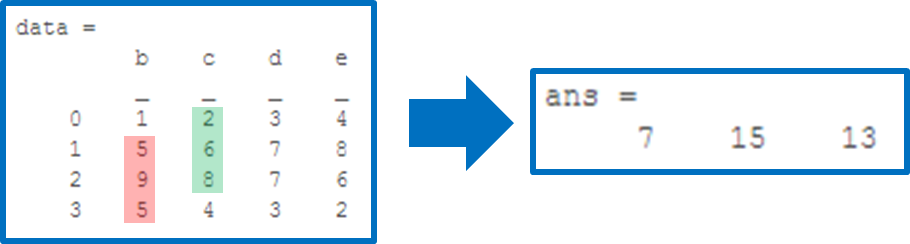
**data{:,:}=data{:,:}+repmat(data{1:},size(data,1),1)**

1. (MATLAB) **Use one command** to add the first row and the last column (without using the size of **data**) of **data** elementwise to every row in **data**.



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

1. (MATLAB) **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.

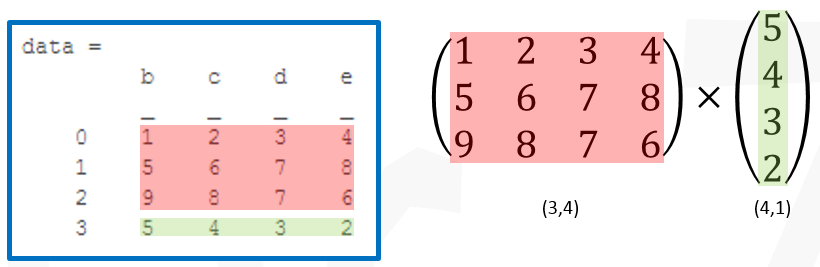


**transpose(data{2:end,1}+data{1:end-1.2})**

1. (MATLAB) MATLAB basic matrix operations.

|  |  |  |  |
| --- | --- | --- | --- |
| matrix multiplication | X**\***Y | elementwise multiplication | X**.\***Y |
| matrix power | X**^**Y | elementwise power | X**.^**Y |
| matrix right division | X**/**Y | elementwise divide | X**./**Y |

1. (MATLAB) **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.



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

1. (Python) **Use one command** to sort rows in **data** in-place by the 3rd column in **data** in ascending order.

**data.sort\_values(by=data.column[2],axis=0,inplace=True)** or

**I=data,argsort(data.values,axis=0)**

**data.values[I[:,2],:]**

1. (Python) **Use one command** to sort columns in **data** in-place by the 3rd row in **data** in ascending order.

**data.sort\_values(by=data.index[2],axis=1,inplace=True)** or

**I=data,argsort(data.values,axis=0)**

**data.values[:,I[2,:]]**

1. (MATLAB) **sortrows(X,COL)** sorts the matrix **X** based on the columns specified in vector **COL**. To use the matrix **X** based on rows specified in vector **ROW**, we can use

**sortrows(X)**

1. (MATLAB) **sort** works on matrices but not on tables. **sortrows** works on both matrices and tables. Therefore,

(1) to sort rows in **data** by the 3rd column, we can **use one command**:

|  |
| --- |
| **data=sortrows(data,3)** |

(2) to sort columns in **data** by the 3rd row, we need to use two commands:

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

1. (MATLAB) When assigning values from one matrix to another matrix, e.g. A(J,K,…)=B(M,N,…), the following must be true:

* The number of subscripts specified for B, not including trailing subscripts equal to 1, does not exceed ndims(B). For example, if B is a 2x3x4 3D array, ndims(B)=3, B(2,2:end,1:2,**1**,**1**) is a valid indexing.
* 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 in 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.

In view of the above, in a 3x3 matrix **x**, we can use the following command to copy the 3rd column of **x** and paste it to the 2nd row of **x**.

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

1. True /  False (Python) Slicing of Numpy arrays are views. In the assignment **slicing\_1**=**slicing\_2**, it is a good practice to use **.copy()** on **slicing\_2**. For example, in a 3x3 2D array **x**, to copy the 3rd column of **x** and paste it to the 2nd row of **x**, it will be nice to use **x[1,:]=x[:,2].copy()**.