COMP7035

Python for Data Analytics and Artificial Intelligence

Pandas

Renjie Wan, Wei Xue

10/30/2022

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

What we will learn?

Topic		**Hours**
I.	Python Fundamentals A. Program control and logic B. Data types and structures C. Function D. File I/O	12
II.	Numerical Computing and Data Visualization Tools and libraries such as A. NumPy B. Matplotlib C. Seaborn	9
III.	Exploratory Data Analysis (EDA) with Python Tools and libraries such as **A. Pandas** B. Sweetviz	9
IV.	Artificial Intelligence and Machine Learning with Python Tools and libraries such as A. Keras B. Scikit-learn	9

Contents of Today

- Pandas Series
- Pandas Dataframe: Creation, indexing, slicing

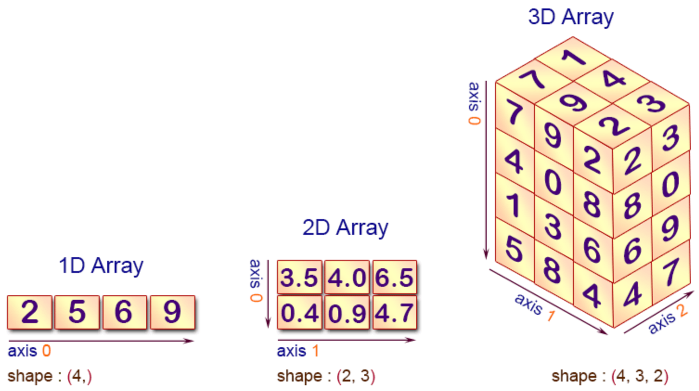
Why do we need Pandas

Q:*What is the most commonly used* ***software*** *for tabular data processing?*

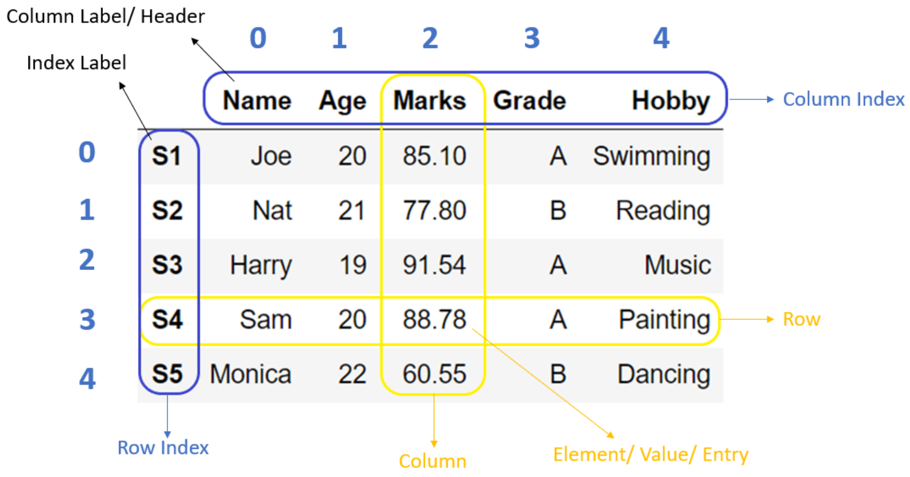
Pandas is a package mainly for working with tabular data.

Main Differences with Numpy Array

- Numpy arrays are designed to contain data of one type (e.g. Int, Float, ...)
- DataFrames can contain different types of data (Int, Float, String, ...)
- Different I/O functions, table operations, time series-specific functionalities...



Numpy Array



		0	1	2	3	4
		Name	Age	Marks	Grade	Hobby
0	S1	Joe	20	85.10	A	Swimming
1	S2	Nat	21	77.80	B	Reading
2	S3	Harry	19	91.54	A	Music
3	S4	Sam	20	88.78	A	Painting
4	S5	Monica	22	60.55	B	Dancing

Pandas Dataframe

Pandas Overview

- Pandas Objects
  - Series
  - Dataframe
- Pandas I/O Functions

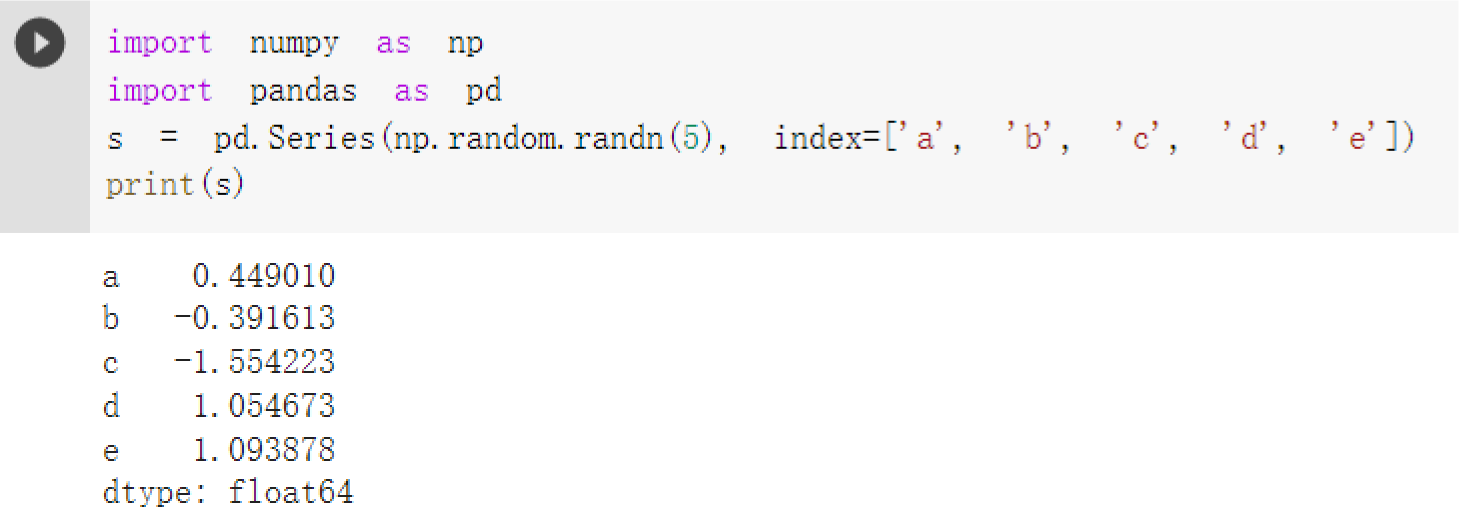
How to use Pandas

- Just import it!

```
import pandas as pd
```

Series

- Let us start with Series!
- A one-dimensional **labeled** array capable of holding **any mixture** data type
- Axis labels are collectively referred to as the index.
- Think "Series = Vector + labels"
- Create a series: s = pd.Series(data, index = index)

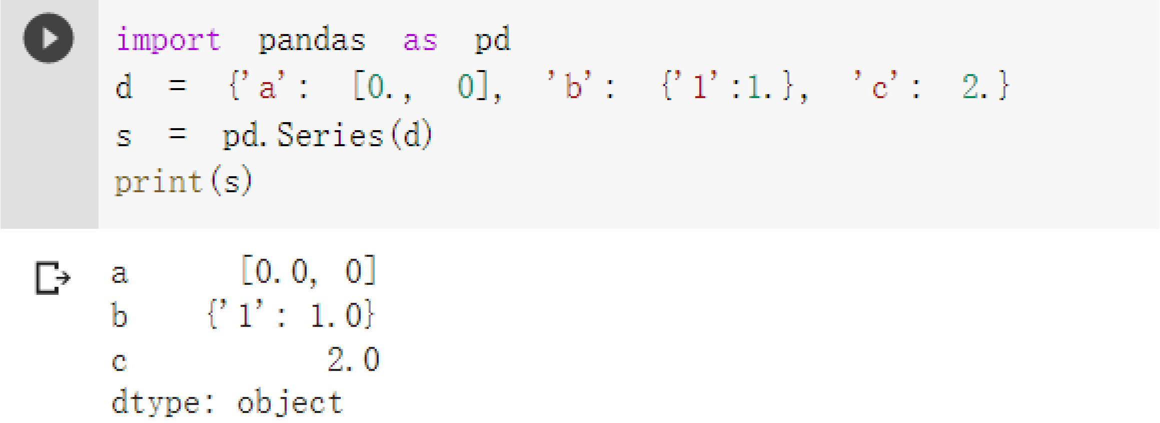


```
import numpy as np
import pandas as pd
s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
print(s)
```

```
a    0.449010
b   -0.391613
c   -1.554223
d    1.054673
e    1.093878
dtype: float64
```

Series

- Creating a series that supports mixed data types

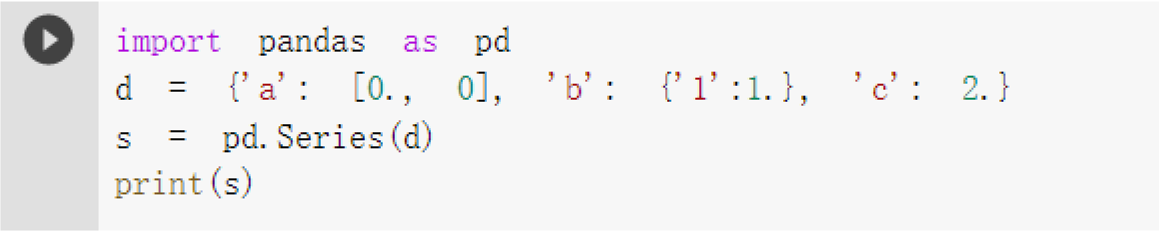


```
import pandas as pd
d = {'a': [0., 0], 'b': {'1':1.}, 'c': 2.}
s = pd.Series(d)
print(s)
```

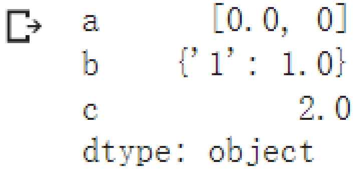
```
a       [0.0, 0]
b    {'1': 1.0}
c           2.0
dtype: object
```

Series

- Creating a series directly from a dictionary!



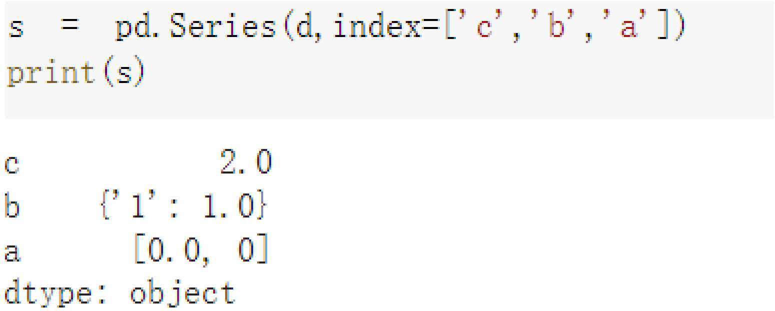
```
import  pandas  as  pd
d  =  {’a’:  [0.,  0],  ’b’:  {’1’:1.},  ’c’:  2.}
s  =  pd.Series(d)
print(s)
```



```
a       [0.0, 0]
b    {’1’: 1.0}
c           2.0
dtype: object
```

Index is constructed as sorted keys

- If you want to specify the order of index:

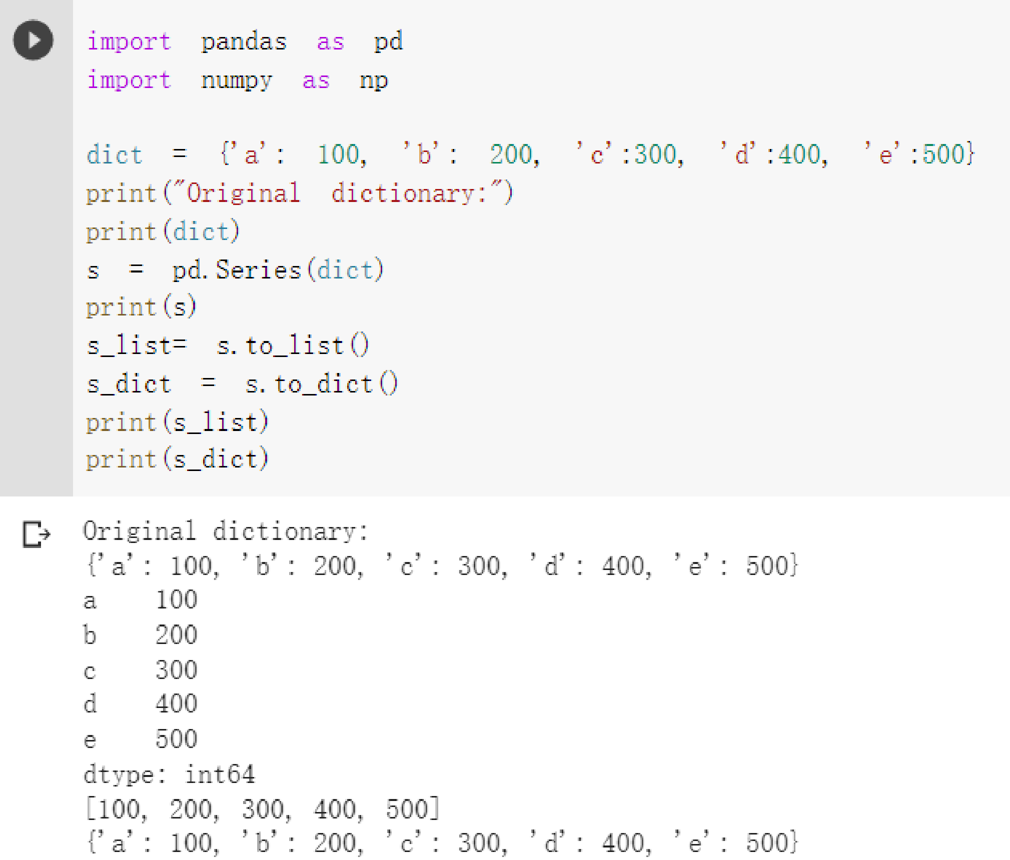


```
s  =  pd.Series(d,index=[’c’,’b’,’a’])
print(s)
```

```
c           2.0
b    {’1’: 1.0}
a       [0.0, 0]
dtype: object
```

Series

- Convert the created Series to other data types



```
import pandas as pd
import numpy as np

dict = {'a': 100, 'b': 200, 'c':300, 'd':400, 'e':500}
print("Original dictionary:")
print(dict)
s = pd.Series(dict)
print(s)
s_list= s.to_list()
s_dict = s.to_dict()
print(s_list)
print(s_dict)
```

```
Original dictionary:
{'a': 100, 'b': 200, 'c': 300, 'd': 400, 'e': 500}
a    100
b    200
c    300
d    400
e    500
dtype: int64
[100, 200, 300, 400, 500]
{'a': 100, 'b': 200, 'c': 300, 'd': 400, 'e': 500}
```

Use x.to_list() to convert to list
Use x.to_dict() to convert to the dict

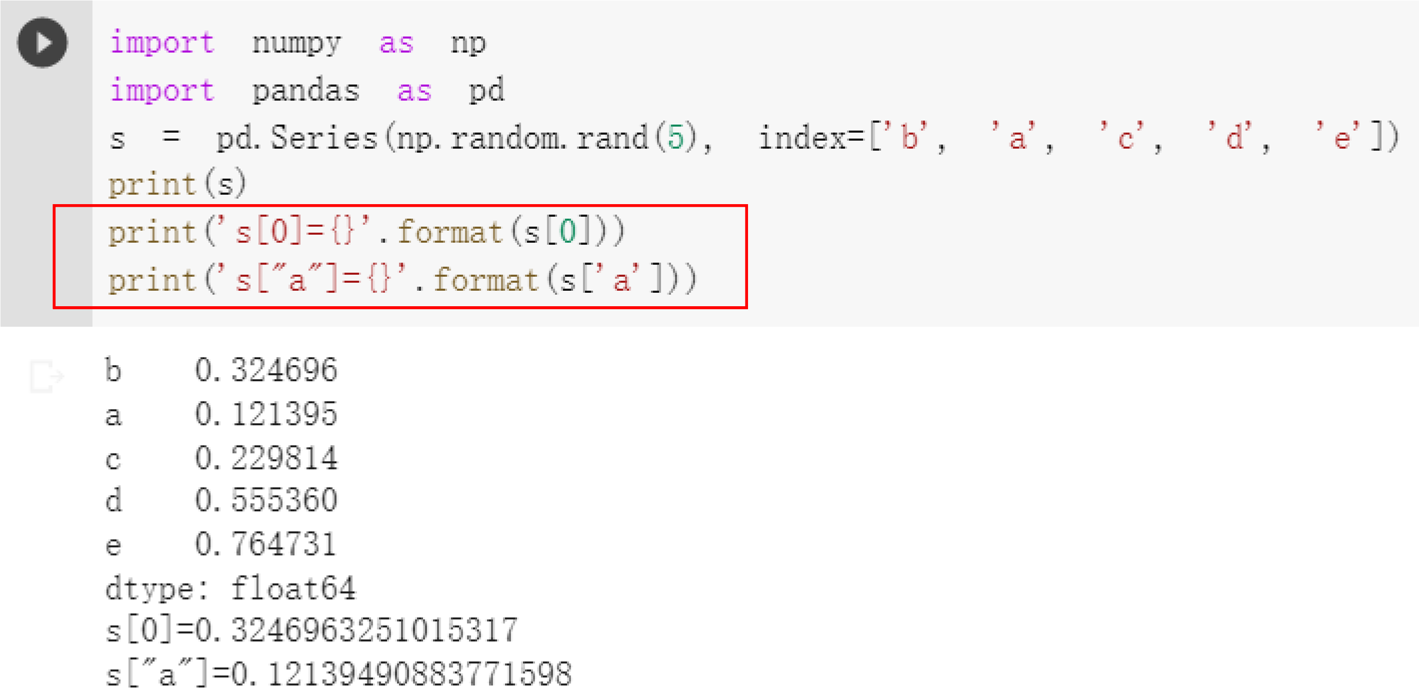
Series

- Exercise

1. Create a 5-D random numpy list **var_list**
2. use “uuid” to generate 5 random keys (use str(uuid.uuid4())), and store them into a list **key_list**
3. Create a dictionary **dict** from **var_list** and **key_list**
4. Create a Pandas Series from
   a) **var_list**
   b) **var_list** and **key_list**
   c) **dict**
5. Convert the Series back to the list and dictionary

Series

- Indexing: Just like you would for NumPy arrays/python lists!

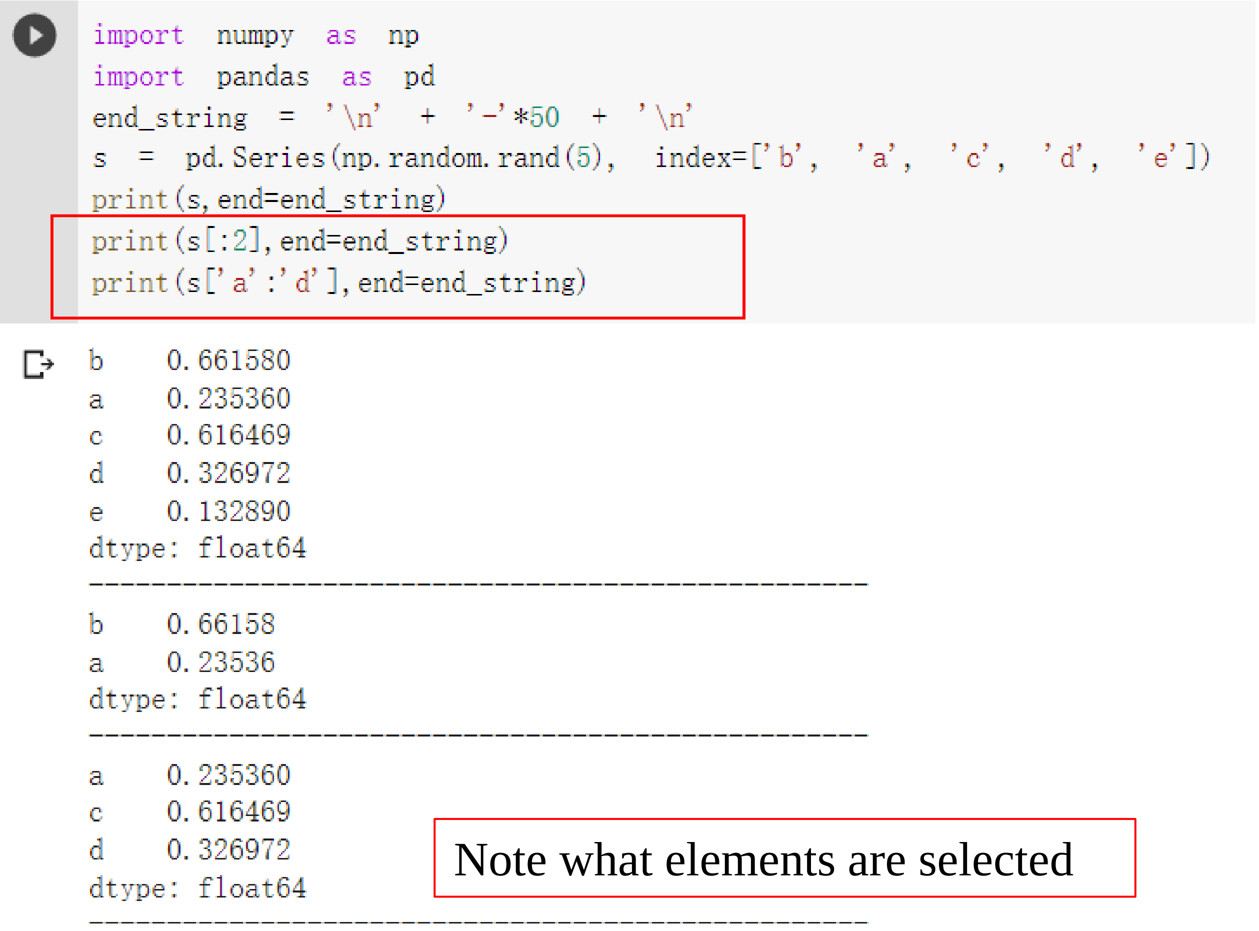


```python
import numpy as np
import pandas as pd
s = pd.Series(np.random.rand(5), index=['b', 'a', 'c', 'd', 'e'])
print(s)
print('s[0]={}'.format(s[0]))
print('s["a"]={}'.format(s['a']))
```

```
b    0.324696
a    0.121395
c    0.229814
d    0.555360
e    0.764731
dtype: float64
s[0]=0.3246963251015317
s["a"]=0.12139490883771598
```

Series

- Slicing: Also similar to python lists



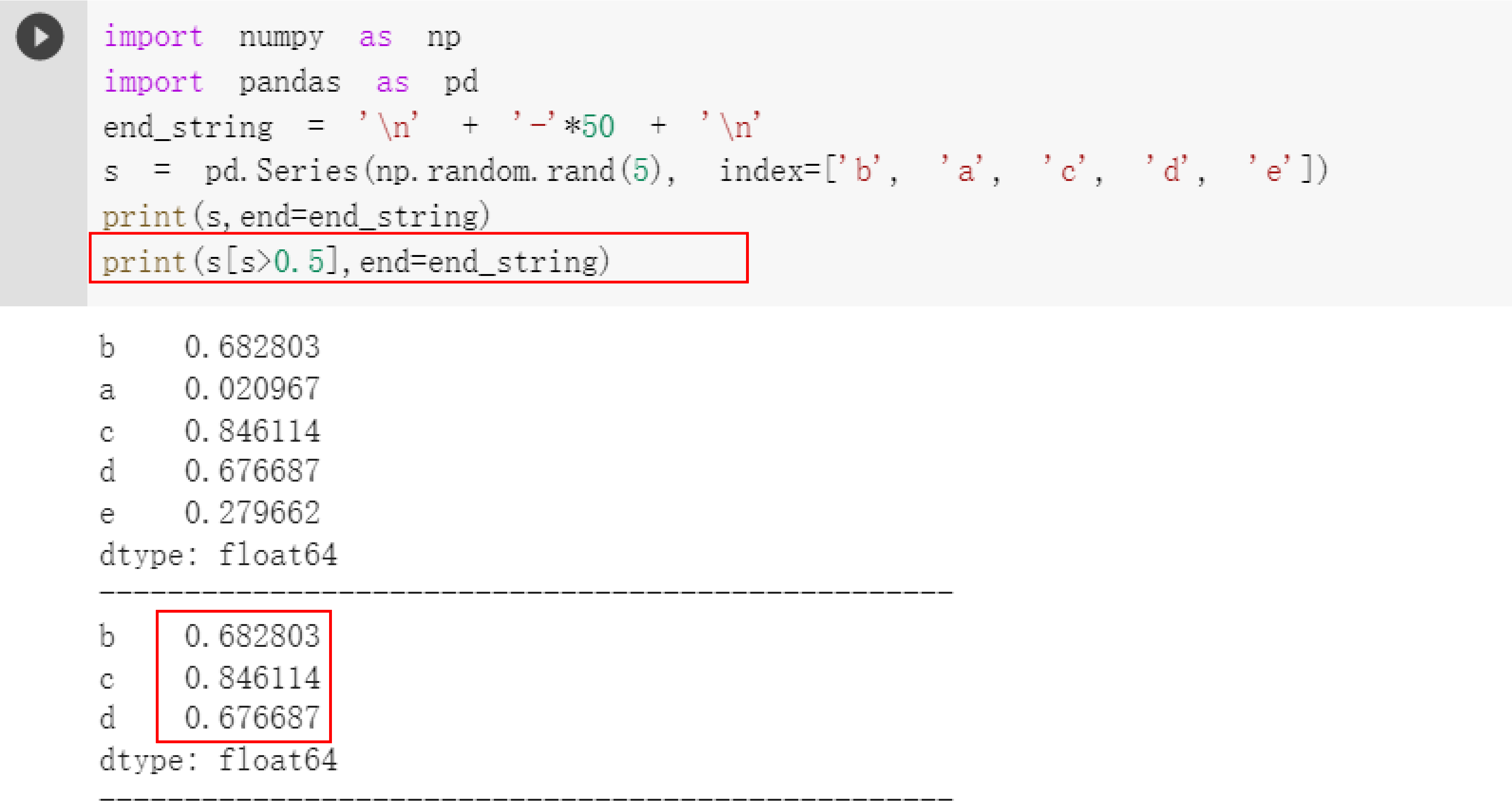
```
import numpy as np
import pandas as pd
end_string = '\n' + '-'*50 + '\n'
s = pd.Series(np.random.rand(5), index=['b', 'a', 'c', 'd', 'e'])
print(s, end=end_string)
print(s[:2], end=end_string)
print(s['a':'d'], end=end_string)
```

```
b    0.661580
a    0.235360
c    0.616469
d    0.326972
e    0.132890
dtype: float64
--------------------------------------------------
b    0.66158
a    0.23536
dtype: float64
--------------------------------------------------
a    0.235360
c    0.616469
d    0.326972
dtype: float64
--------------------------------------------------
```

Note what elements are selected

Series

- Slicing: picking elements under certain conditions

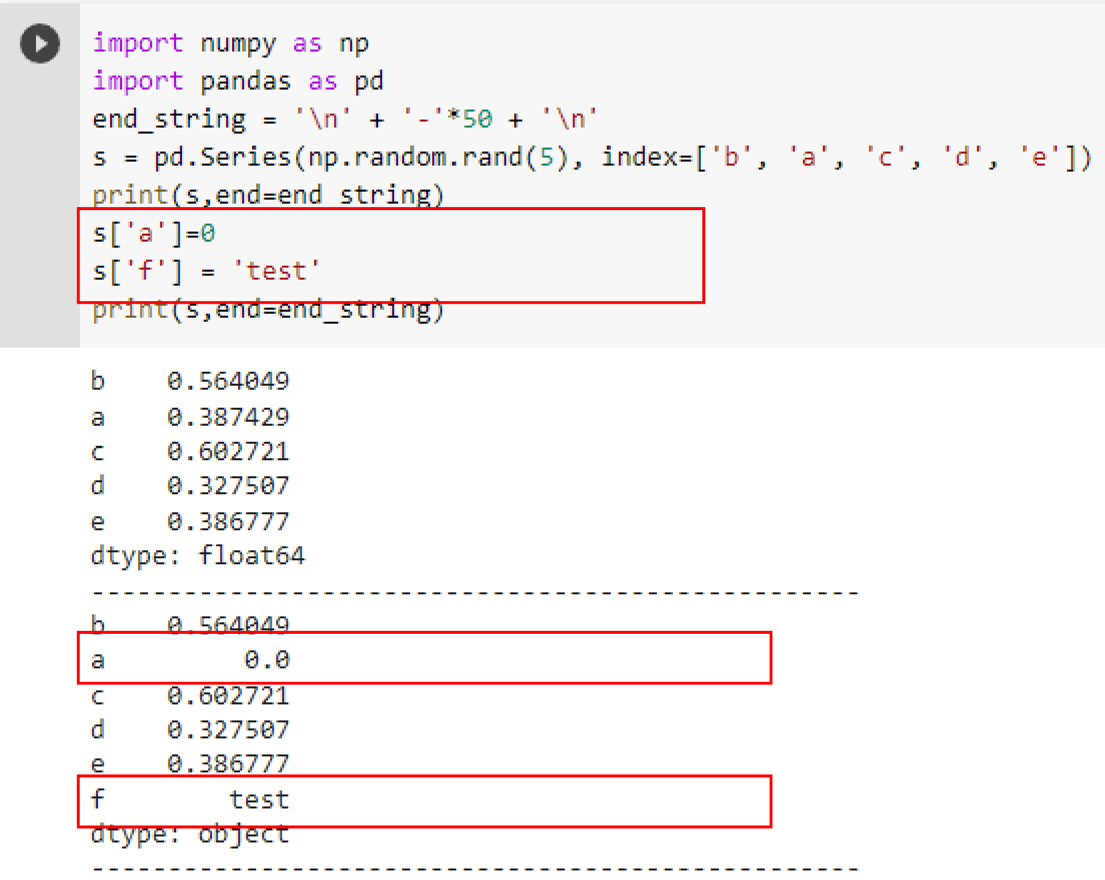


```
import numpy as np
import pandas as pd
end_string = ’\n’ + ’-’*50 + ’\n’
s = pd.Series(np.random.rand(5), index=[’b’, ’a’, ’c’, ’d’, ’e’])
print(s,end=end_string)
print(s[s>0.5],end=end_string)
```

```
b    0.682803
a    0.020967
c    0.846114
d    0.676687
e    0.279662
dtype: float64
--------------------------------------------------
b    0.682803
c    0.846114
d    0.676687
dtype: float64
--------------------------------------------------
```

Series

- Assign new values and indexes

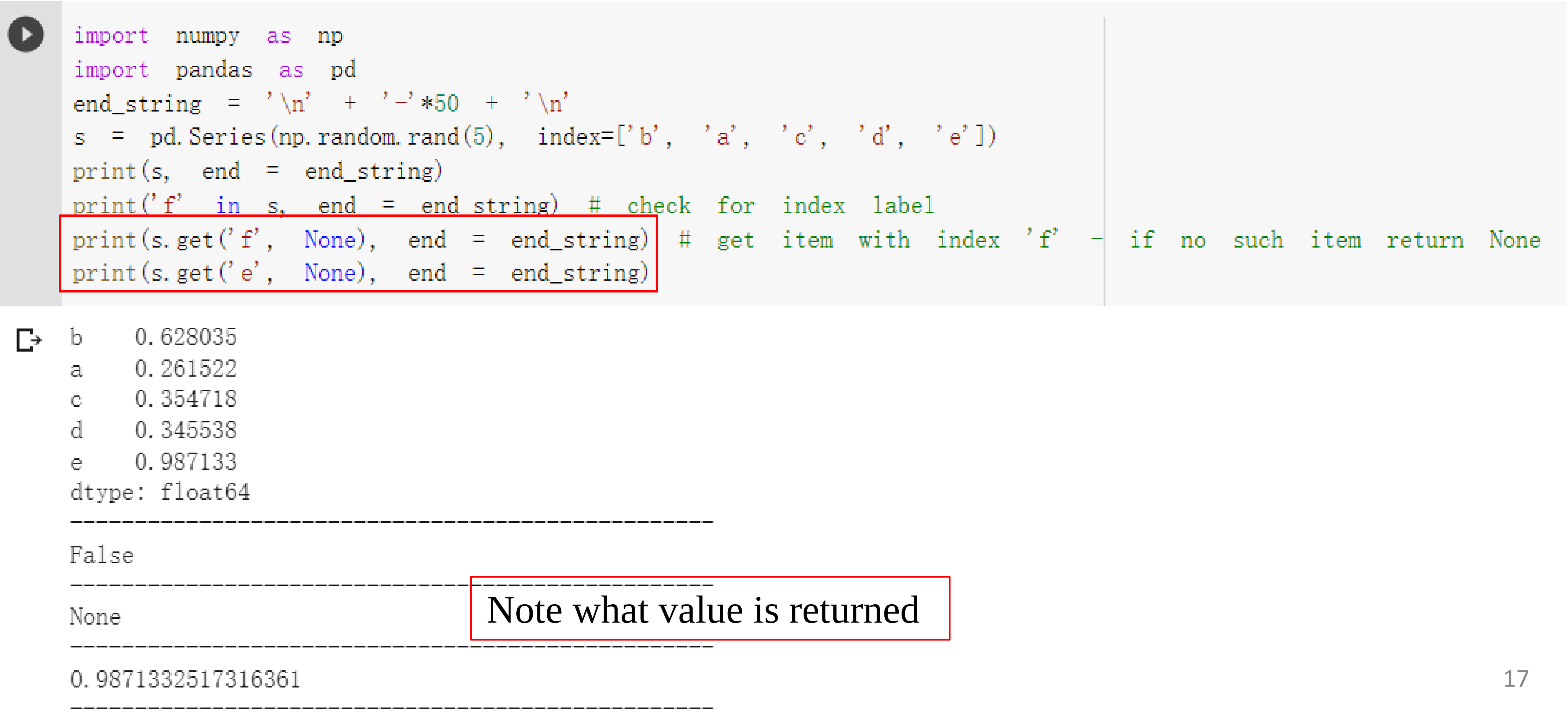


```
import numpy as np
import pandas as pd
end_string = '\n' + '-'*50 + '\n'
s = pd.Series(np.random.rand(5), index=['b', 'a', 'c', 'd', 'e'])
print(s,end=end_string)
s['a']=0
s['f'] = 'test'
print(s,end=end_string)
```

```
b    0.564049
a    0.387429
c    0.602721
d    0.327507
e    0.386777
dtype: float64
--------------------------------------------------
b    0.564049
a         0.0
c    0.602721
d    0.327507
e    0.386777
f        test
dtype: object
--------------------------------------------------
```

Series

- Operations
  - Get the element



```
import numpy as np
import pandas as pd
end_string = '\n' + '-'*50 + '\n'
s = pd.Series(np.random.rand(5), index=['b', 'a', 'c', 'd', 'e'])
print(s, end = end_string)
print('f' in s, end = end_string) # check for index label
print(s.get('f', None), end = end_string) # get item with index 'f' - if no such item return None
print(s.get('e', None), end = end_string)
```

```
b    0.628035
a    0.261522
c    0.354718
d    0.345538
e    0.987133
dtype: float64
--------------------------------------------------
False
--------------------------------------------------
None
--------------------------------------------------
0.9871332517316361
--------------------------------------------------
```

Note what value is returned

Series

- Exercise

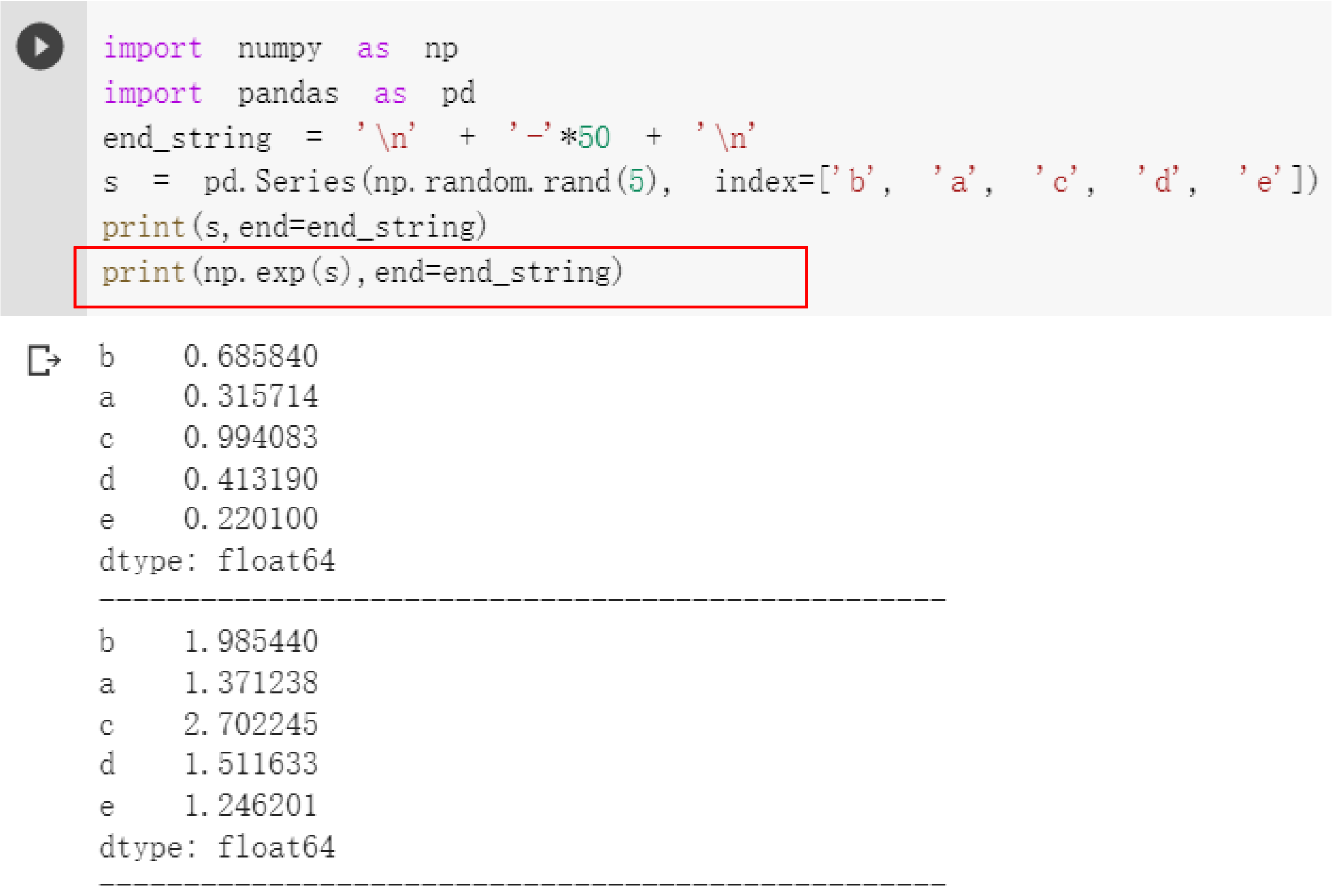
1. Create a 5-D random numpy list **var_list**
2. use “uuid” to generate 5 random keys (use str(uuid.uuid4())), and store them into a list **key_list**
3. Create a dictionary **dict** from **var_list** and **key_list**
4. Create a Pandas Series from
   - **a)** **var_list**
   - **b)** **var_list** and **key_list**
   - **c)** **dict**
5. Convert the Series back to the list and dictionary

----

6. Find out the elements larger than zero
7. Calculate the proportion of positive elements in the Series

Series

- Operations
  - Math calculations. Numpy operations can be applied to the Series.

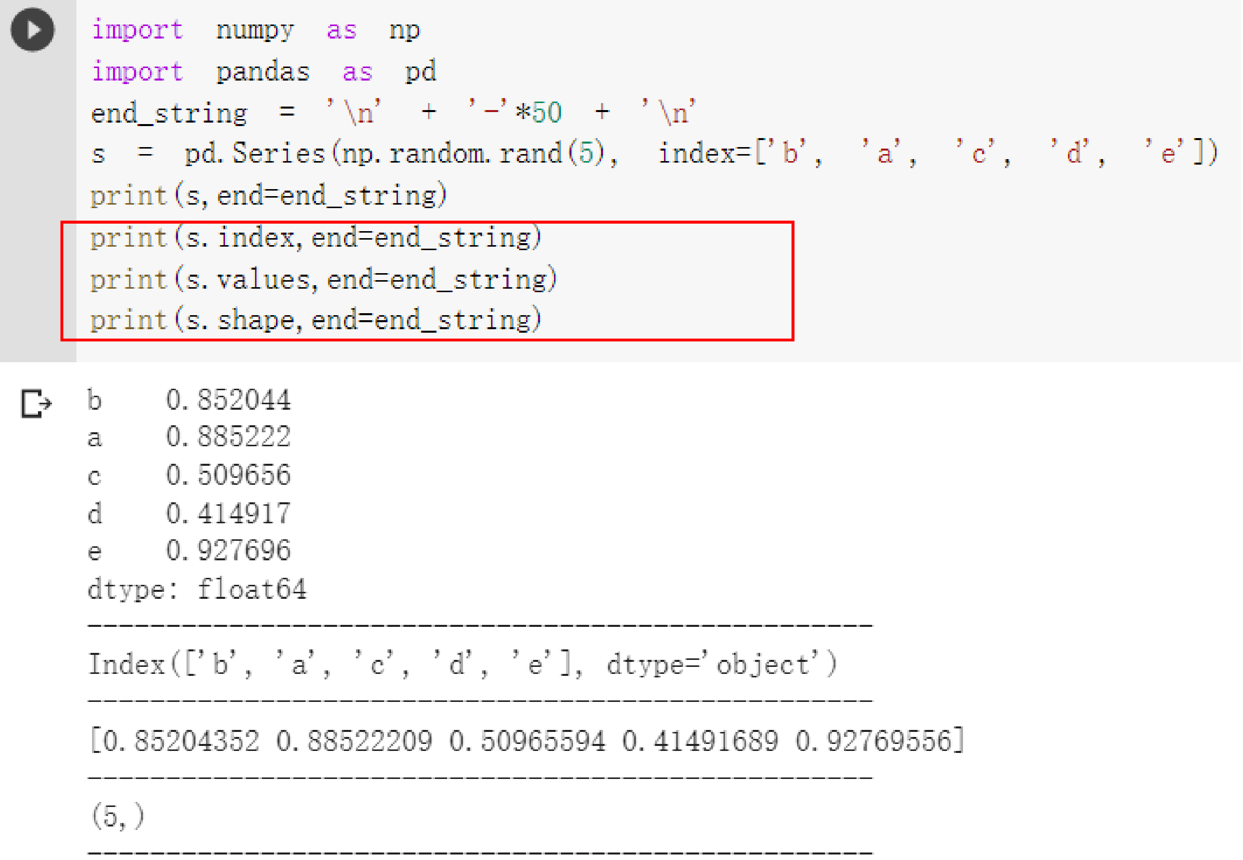


```
import numpy as np
import pandas as pd
end_string = '\n' + '-'*50 + '\n'
s = pd.Series(np.random.rand(5), index=['b', 'a', 'c', 'd', 'e'])
print(s,end=end_string)
print(np.exp(s),end=end_string)
```

```
b    0.685840
a    0.315714
c    0.994083
d    0.413190
e    0.220100
dtype: float64
--------------------------------------------------
b    1.985440
a    1.371238
c    2.702245
d    1.511633
e    1.246201
dtype: float64
--------------------------------------------------
```

Series

- Attributes
  - Get the index, value and shape

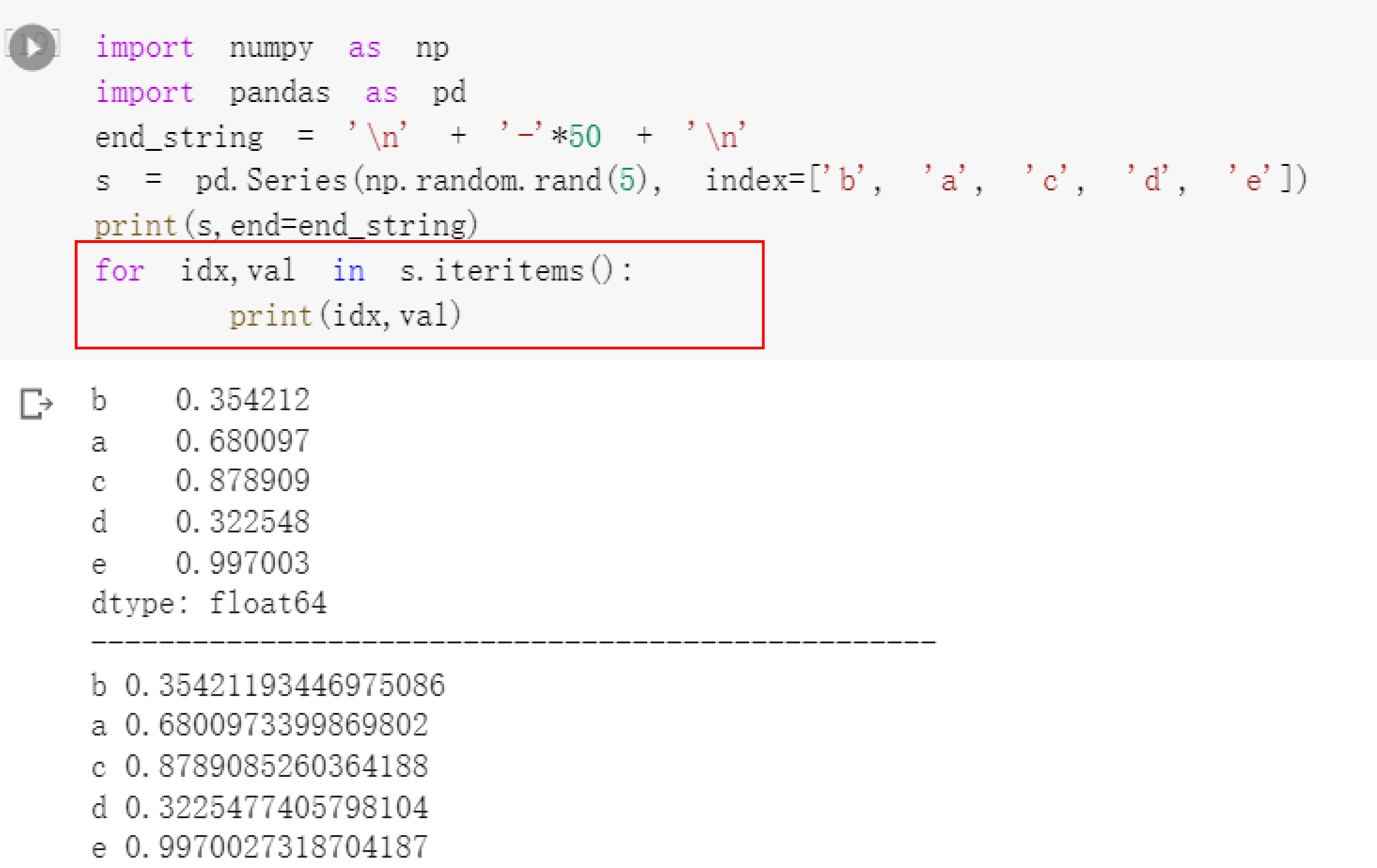


```
import numpy as np
import pandas as pd
end_string = ’\n’ + ’-’*50 + ’\n’
s = pd.Series(np.random.rand(5), index=[’b’, ’a’, ’c’, ’d’, ’e’])
print(s, end=end_string)
print(s.index, end=end_string)
print(s.values, end=end_string)
print(s.shape, end=end_string)
```

```
b    0.852044
a    0.885222
c    0.509656
d    0.414917
e    0.927696
dtype: float64
--------------------------------------------------
Index([’b’, ’a’, ’c’, ’d’, ’e’], dtype=’object’)
--------------------------------------------------
[0.85204352 0.88522209 0.50965594 0.41491689 0.92769556]
--------------------------------------------------
(5,)
--------------------------------------------------
```

Series

- Iteration



```python
import numpy as np
import pandas as pd
end_string = '\n' + '-'*50 + '\n'
s = pd.Series(np.random.rand(5), index=['b', 'a', 'c', 'd', 'e'])
print(s, end=end_string)
for idx, val in s.iteritems():
    print(idx, val)
```

```
b    0.354212
a    0.680097
c    0.878909
d    0.322548
e    0.997003
dtype: float64
--------------------------------------------------
b 0.35421193446975086
a 0.6800973399869802
c 0.8789085260364188
d 0.3225477405798104
e 0.9970027318704187
```

Series

- Exercise

1. Create a 5-D random numpy list **var_list**
2. use “uuid” to generate 5 random keys (use str(uuid.uuid4())), and store them into a list **key_list**
3. Create a dictionary **dict** from **var_list** and **key_list**
4. Create a Pandas Series **ps** from
    a) **var_list**
    b) **var_list** and **key_list**
    c) **dict**
5. Convert the Series back to the list and dictionary

----

6. Find out the elements larger than zero
7. Calculate the proportion of positive elements in the Series

----

8. Write down as many ways of forming a list that contains the values of Series elements
9. Calculate the proportion of elements that are larger than the mean value of the Series

Pandas Overview

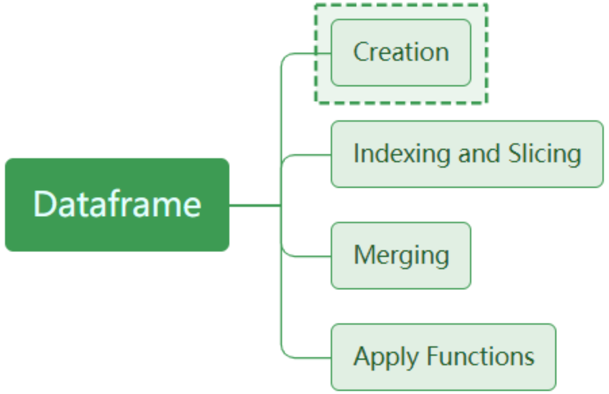
- Pandas Objects
  - Series
  - Dataframe
- Pandas I/O Functions

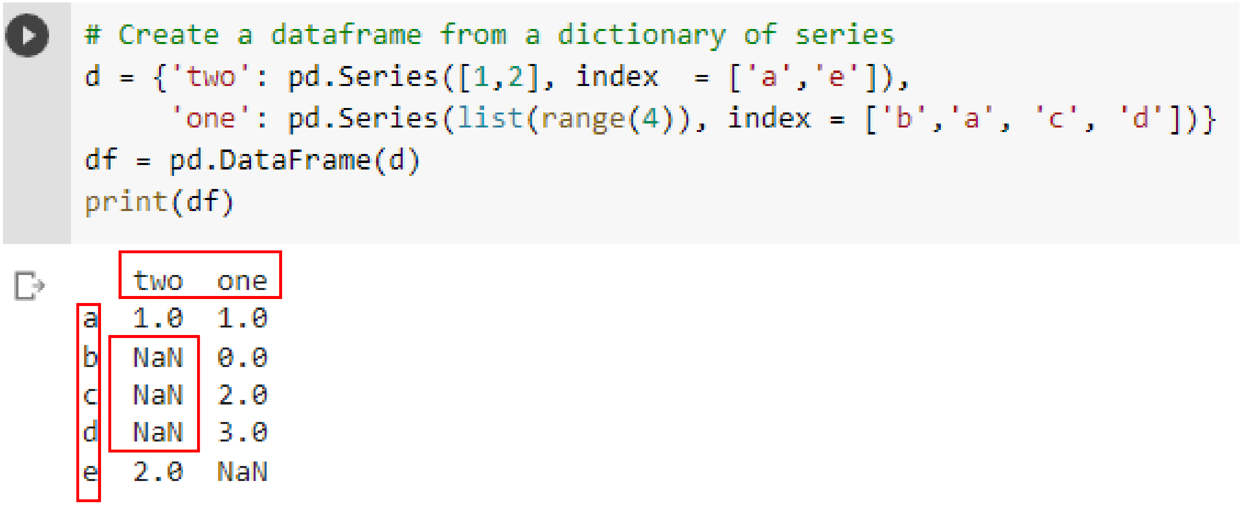
Dataframe

- A two-dimensional **labeled** data structure capable of holding **any mixture** data type
- Think Dataframe as spreadsheets
- Create a series: df = pd.Dataframe(data, index = index, columns = columns)

Dataframe

- Creating a Dataframe
  - From dict of series or dicts
    - **From dict of series**
    - **From dicts**

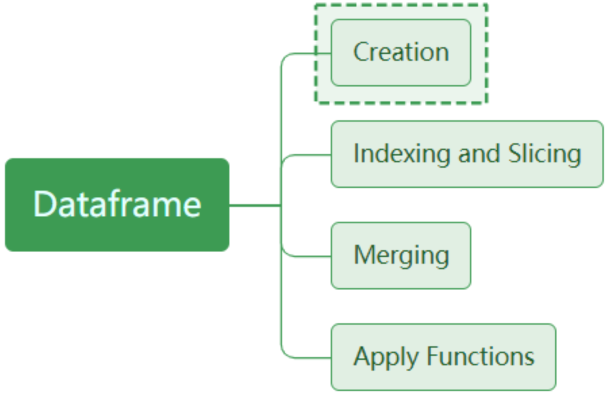




```
# Create a dataframe from a dictionary of series
d = {'two': pd.Series([1,2], index  = ['a','e']),
     'one': pd.Series(list(range(4)), index = ['b','a', 'c', 'd'])}
df = pd.DataFrame(d)
print(df)
```

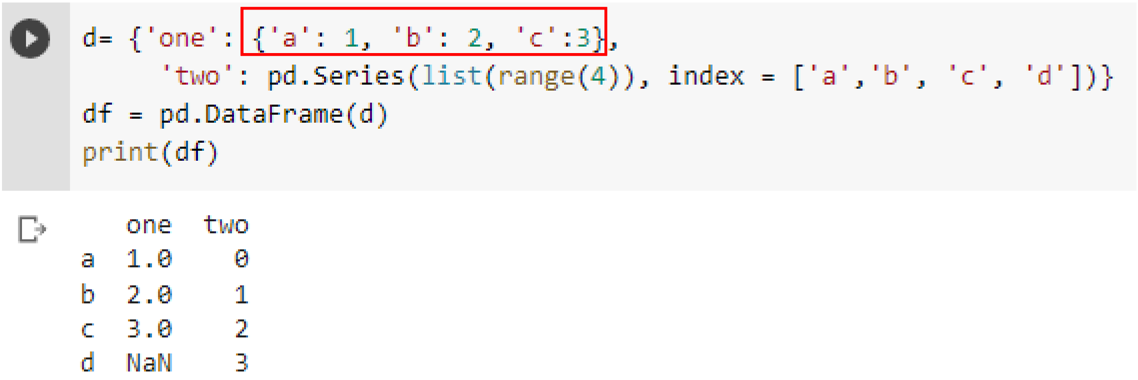
```
   two  one
a  1.0  1.0
b  NaN  0.0
c  NaN  2.0
d  NaN  3.0
e  2.0  NaN
```

Dataframe



- Creating a Dataframe
  - From dict of series or dicts
    - **From dict of series**
    - **From dicts**

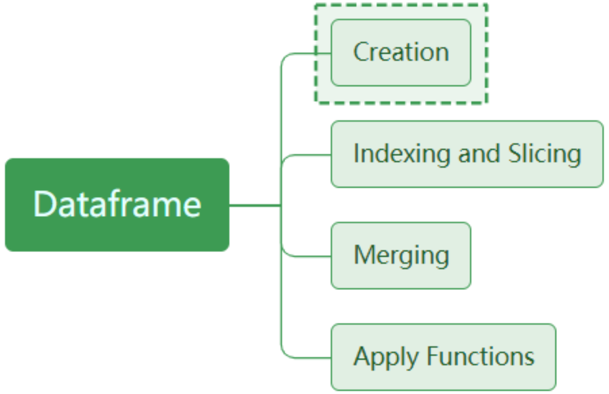
If there are any nested dicts, these will be first converted to Series.



```
d= {'one': {'a': 1, 'b': 2, 'c':3},
     'two': pd.Series(list(range(4)), index = ['a','b', 'c', 'd'])}
df = pd.DataFrame(d)
print(df)
```

```
   one  two
a  1.0    0
b  2.0    1
c  3.0    2
d  NaN    3
```

Dataframe

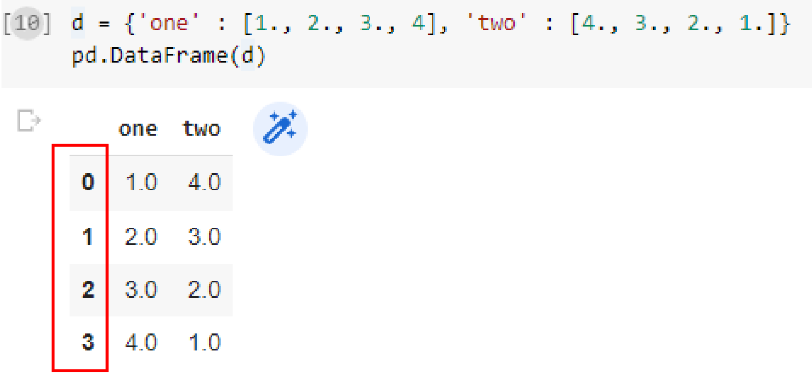


- Creating a Dataframe
  - **From dict of ndarray / lists**

➢ The ndarrays must all be the same length.

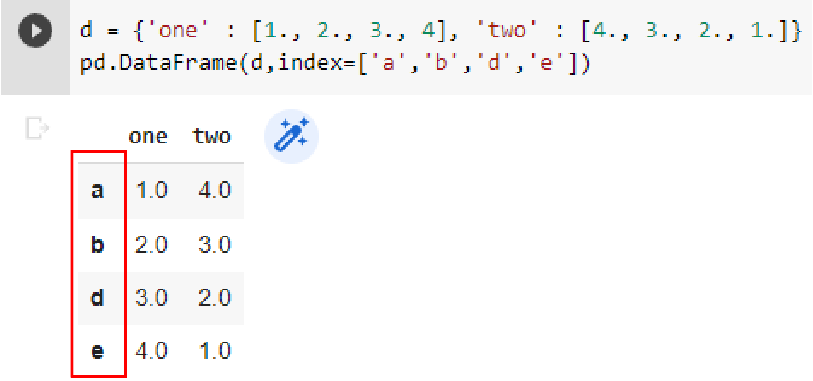
➢ If an index is passed, it must clearly also be the same length as the arrays.

➢ If no index is passed, the result will be range(len_array)



```
[10] d = {'one' : [1., 2., 3., 4], 'two' : [4., 3., 2., 1.]}
     pd.DataFrame(d)
```

```
   one  two
0  1.0  4.0
1  2.0  3.0
2  3.0  2.0
3  4.0  1.0
```

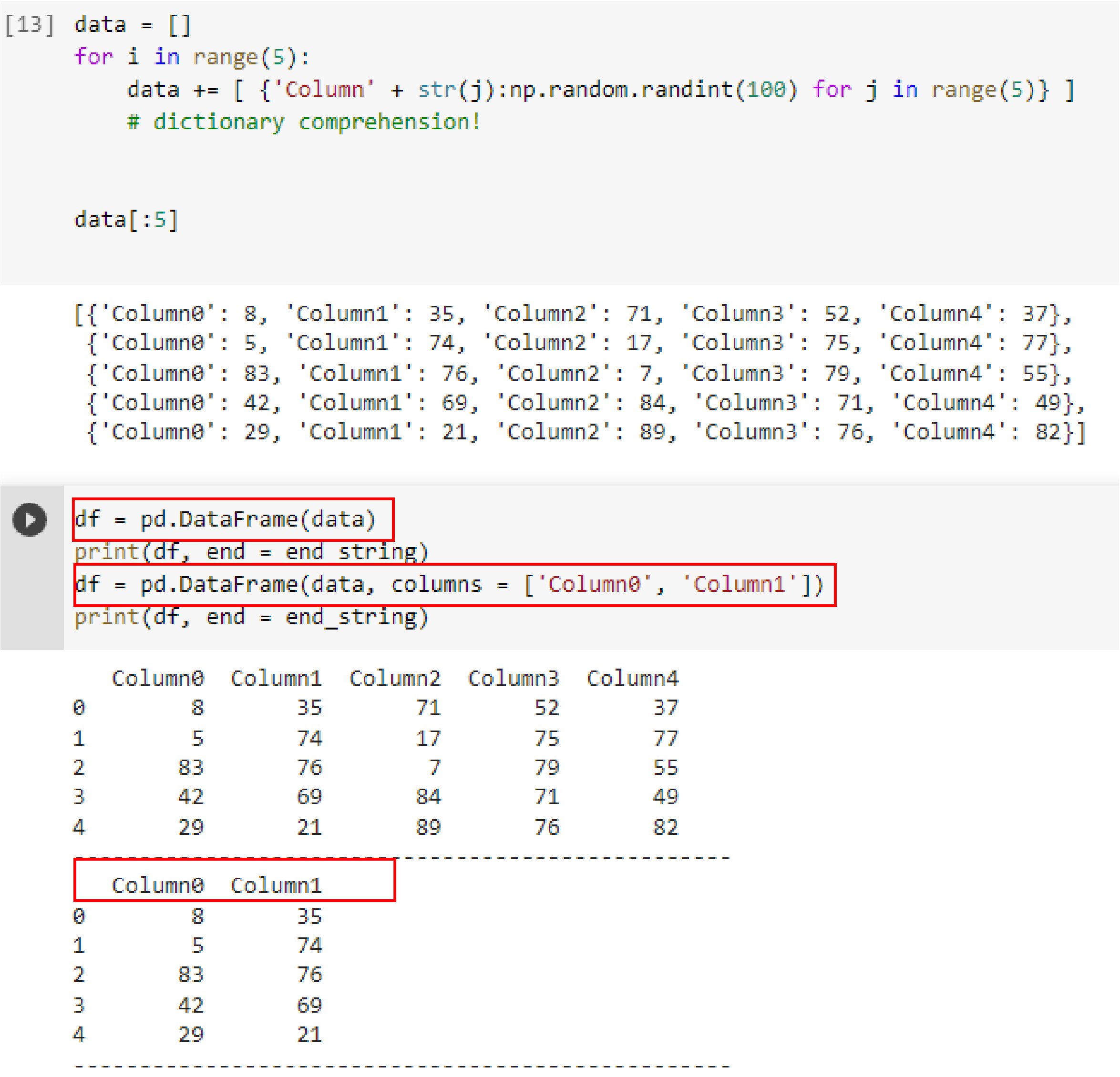


```
d = {'one' : [1., 2., 3., 4], 'two' : [4., 3., 2., 1.]}
pd.DataFrame(d,index=['a','b','d','e'])
```

```
   one  two
a  1.0  4.0
b  2.0  3.0
d  3.0  2.0
e  4.0  1.0
```

Dataframe

- Creating a Dataframe
  - **From a list of dicts**

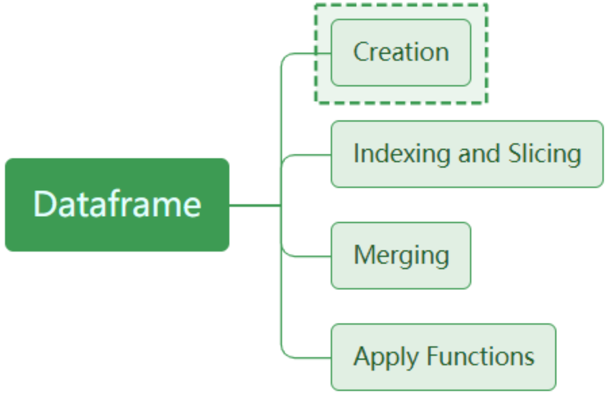


```
[13] data = []
     for i in range(5):
         data += [ {'Column' + str(j):np.random.randint(100) for j in range(5)} ]
         # dictionary comprehension!


     data[:5]
```

```
[{'Column0': 8, 'Column1': 35, 'Column2': 71, 'Column3': 52, 'Column4': 37},
 {'Column0': 5, 'Column1': 74, 'Column2': 17, 'Column3': 75, 'Column4': 77},
 {'Column0': 83, 'Column1': 76, 'Column2': 7, 'Column3': 79, 'Column4': 55},
 {'Column0': 42, 'Column1': 69, 'Column2': 84, 'Column3': 71, 'Column4': 49},
 {'Column0': 29, 'Column1': 21, 'Column2': 89, 'Column3': 76, 'Column4': 82}]
```

```
df = pd.DataFrame(data)
print(df, end = end_string)
df = pd.DataFrame(data, columns = ['Column0', 'Column1'])
print(df, end = end_string)
```

```
   Column0  Column1  Column2  Column3  Column4
0        8       35       71       52       37
1        5       74       17       75       77
2       83       76        7       79       55
3       42       69       84       71       49
4       29       21       89       76       82
-------------------------------------------------------
   Column0  Column1
0        8       35
1        5       74
2       83       76
3       42       69
4       29       21
-------------------------------------------------------
```



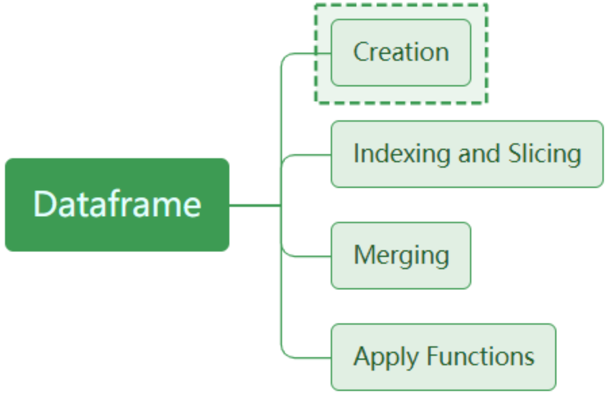
Dataframe
Creation
Indexing and Slicing
Merging
Apply Functions

Series

- Exercise

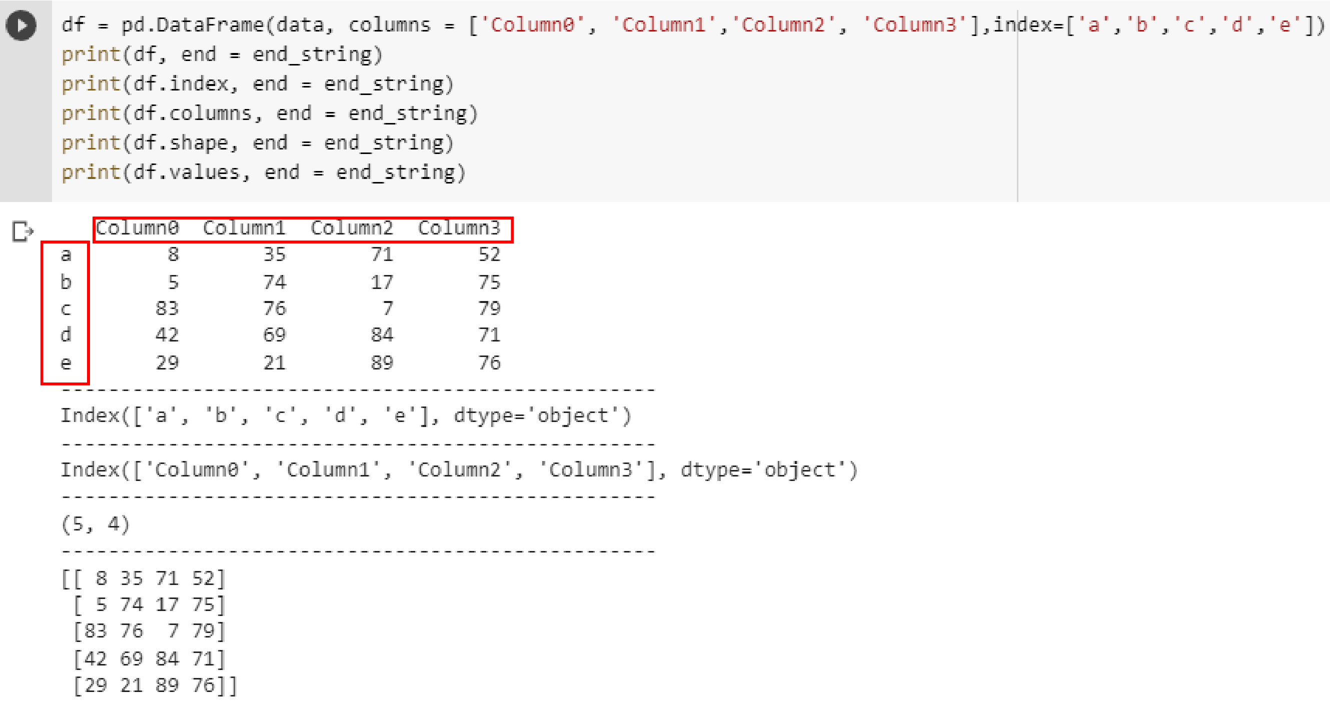
1. Write codes to create a random dict **x** which has 5 random keys and each key corresponds to a 6-D numpy array
2. Create a pandas dataframe using **x**
3. Create a pandas dataframe using a subset of **x**, in the subset of **x,** only keys that start with a digit are chosen.

Dataframe



- Attributes

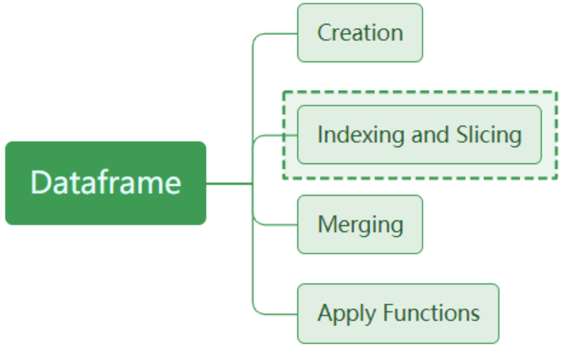
- df.index : the row index of df
- df.columns : the columns of df
- df.shape : the shape of the df
- df.values : numpy array of values



```
df = pd.DataFrame(data, columns = ['Column0', 'Column1','Column2', 'Column3'],index=['a','b','c','d','e'])
print(df, end = end_string)
print(df.index, end = end_string)
print(df.columns, end = end_string)
print(df.shape, end = end_string)
print(df.values, end = end_string)
```

```
   Column0  Column1  Column2  Column3
a        8       35       71       52
b        5       74       17       75
c       83       76        7       79
d       42       69       84       71
e       29       21       89       76
--------------------------------------------------
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
--------------------------------------------------
Index(['Column0', 'Column1', 'Column2', 'Column3'], dtype='object')
--------------------------------------------------
(5, 4)
--------------------------------------------------
[[ 8 35 71 52]
 [ 5 74 17 75]
 [83 76  7 79]
 [42 69 84 71]
 [29 21 89 76]]
```

Dataframe

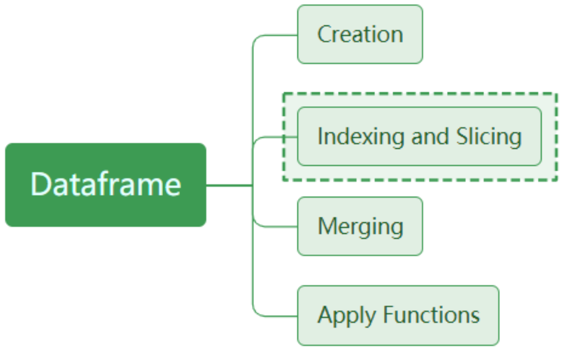


- Indexing and Slicing

3 methods [], iloc, loc

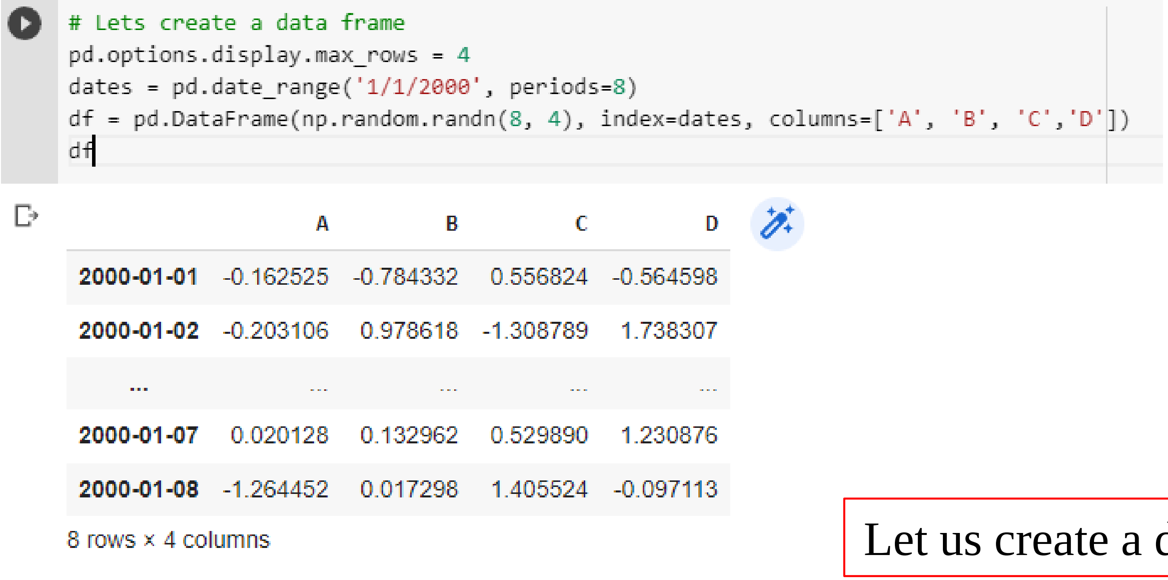
Operation	Syntax	Result
Select Column	df[col]	Series
Select Row by Label	df.loc[label]	Series
Select Row by Integer Location	df.iloc[idx]	Series
Select Columns	df[col_list]	DataFrame
Slice rows	df[5:10]	DataFrame
Select rows by boolean	df[mask]	DataFrame

Dataframe



- Simplest form of Indexing: []

Operation	Syntax	Result
Select Column	df[col]	Series
Select Columns	df[col_list]	DataFrame
Slice rows	df[5:10]	DataFrame
Select rows by boolean	df[mask]	DataFrame

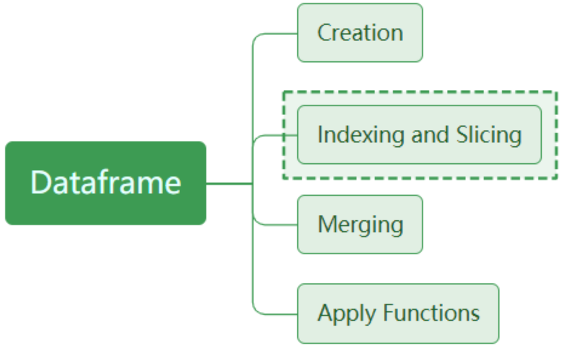


```python
# Lets create a data frame
pd.options.display.max_rows = 4
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
df
```

```
                   A          B          C          D
2000-01-01 -0.162525  -0.784332   0.556824  -0.564598
2000-01-02 -0.203106   0.978618  -1.308789   1.738307
   ...        ...        ...        ...        ...
2000-01-07  0.020128   0.132962   0.529890   1.230876
2000-01-08 -1.264452   0.017298   1.405524  -0.097113
8 rows × 4 columns
```

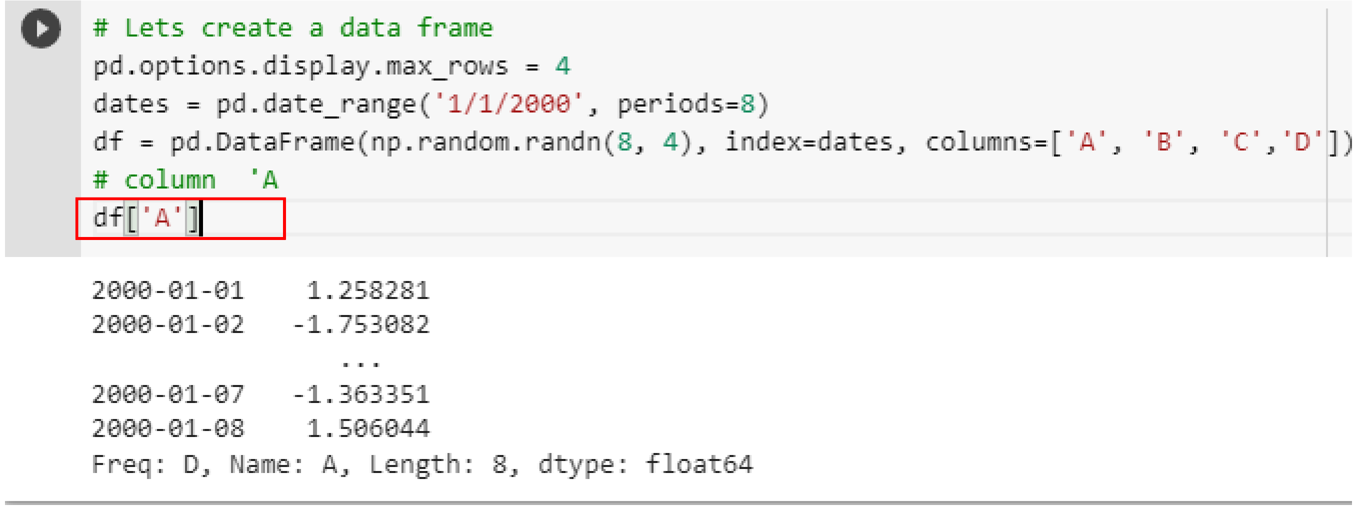
Let us create a dataframe first

Dataframe



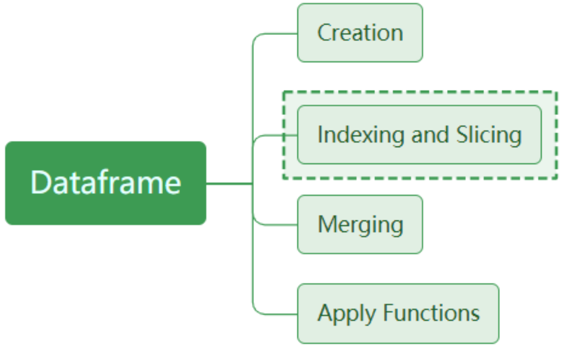
- Simplest form of Indexing: []

Operation	Syntax	Result
Select Column	df[col]	Series
Select Columns	df[col_list]	DataFrame
Slice rows	df[5:10]	DataFrame
Select rows by boolean	df[mask]	DataFrame



```
# Lets create a data frame
pd.options.display.max_rows = 4
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
# column  'A
df['A']
```

```
2000-01-01    1.258281
2000-01-02   -1.753082
                ...
2000-01-07   -1.363351
2000-01-08    1.506044
Freq: D, Name: A, Length: 8, dtype: float64
```



Dataframe

Creation

Indexing and Slicing

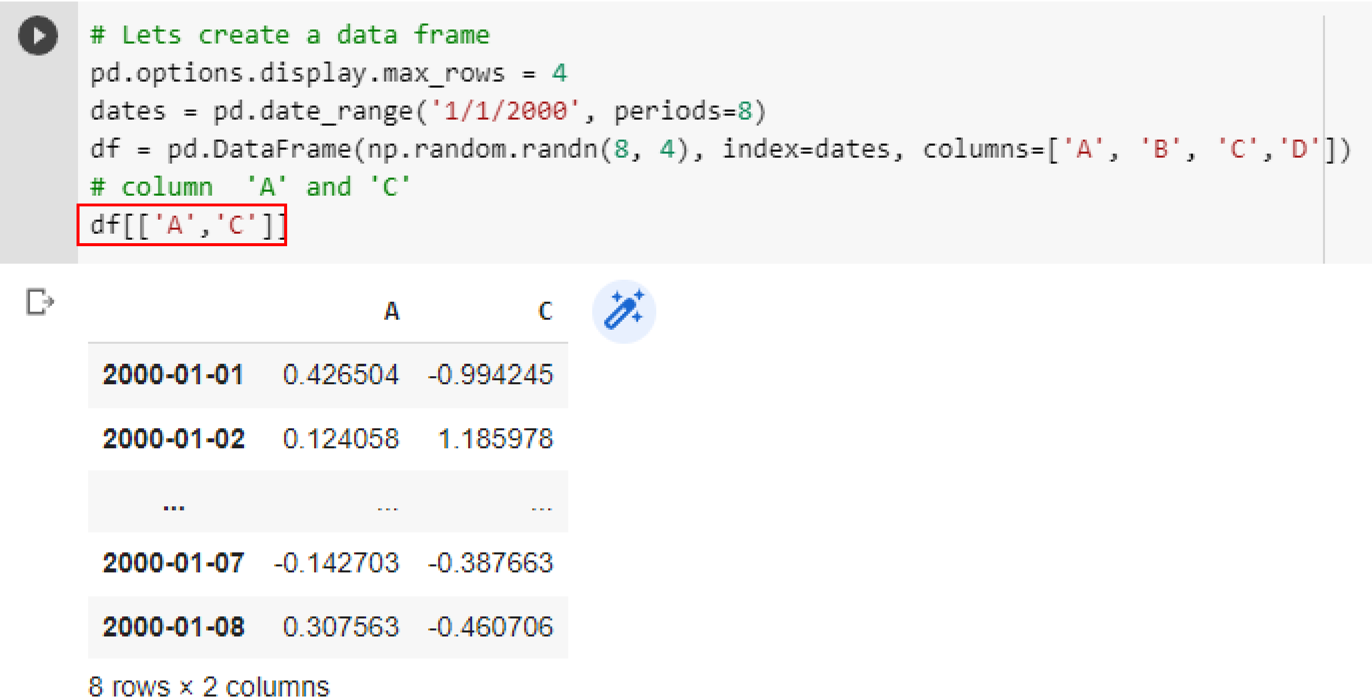
Merging

Apply Functions

Dataframe

- Simplest form of Indexing: []

Operation	Syntax	Result
Select Column	df[col]	Series
Select Columns	df[col_list]	DataFrame
Slice rows	df[5:10]	DataFrame
Select rows by boolean	df[mask]	DataFrame



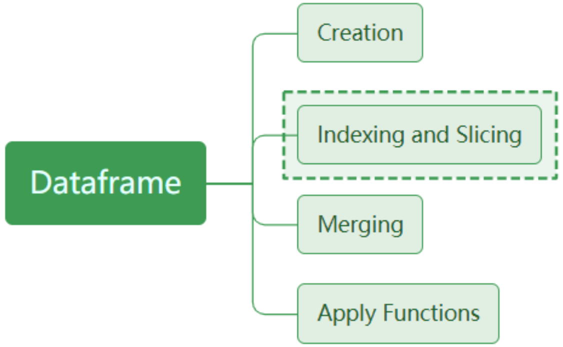
```
# Lets create a data frame
pd.options.display.max_rows = 4
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
# column  'A' and 'C'
df[['A','C']]
```

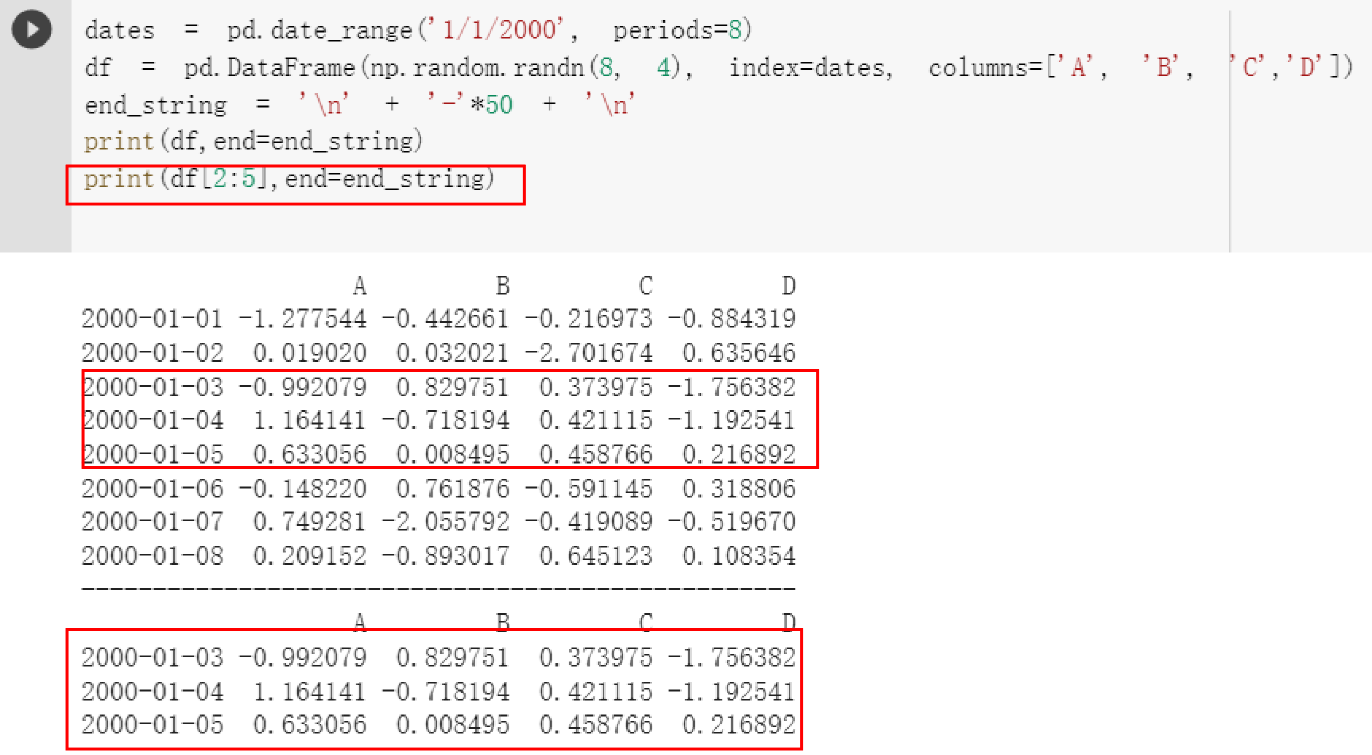
```
                   A          C
2000-01-01  0.426504  -0.994245
2000-01-02  0.124058   1.185978
       ...       ...        ...
2000-01-07 -0.142703  -0.387663
2000-01-08  0.307563  -0.460706
8 rows × 2 columns
```

Dataframe

- Simplest form of Indexing: []

Operation	Syntax	Result
Select Column	df[col]	Series
Select Columns	df[col_list]	DataFrame
Slice rows	df[5:10]	DataFrame
Select rows by boolean	df[mask]	DataFrame





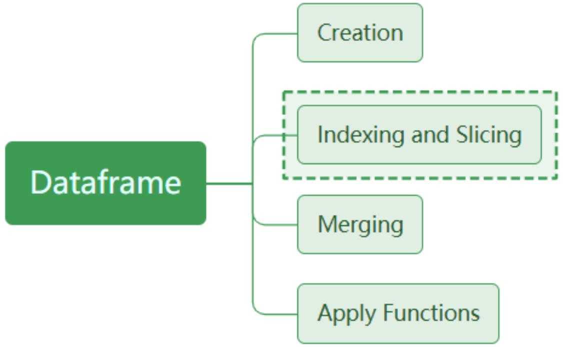
```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)
print(df[2:5], end=end_string)
```

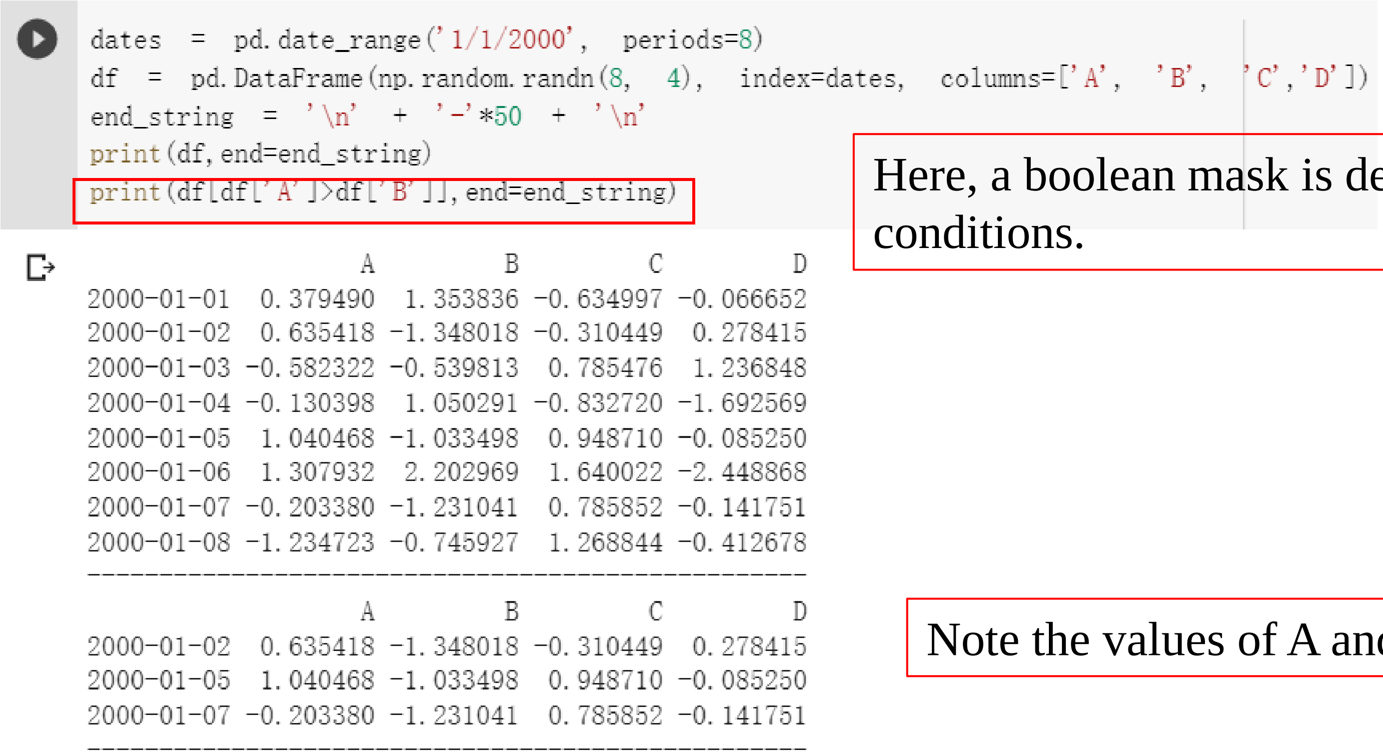
```
                   A         B         C         D
2000-01-01 -1.277544 -0.442661 -0.216973 -0.884319
2000-01-02  0.019020  0.032021 -2.701674  0.635646
2000-01-03 -0.992079  0.829751  0.373975 -1.756382
2000-01-04  1.164141 -0.718194  0.421115 -1.192541
2000-01-05  0.633056  0.008495  0.458766  0.216892
2000-01-06 -0.148220  0.761876 -0.591145  0.318806
2000-01-07  0.749281 -2.055792 -0.419089 -0.519670
2000-01-08  0.209152 -0.893017  0.645123  0.108354
--------------------------------------------------
                   A         B         C         D
2000-01-03 -0.992079  0.829751  0.373975 -1.756382
2000-01-04  1.164141 -0.718194  0.421115 -1.192541
2000-01-05  0.633056  0.008495  0.458766  0.216892
--------------------------------------------------
```

Dataframe

- Simplest form of Indexing: []

Operation	Syntax	Result
Select Column	df[col]	Series
Select Columns	df[col_list]	DataFrame
Slice rows	df[5:10]	DataFrame
Select rows by boolean	df[mask]	DataFrame





```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)
print(df[df['A']>df['B']], end=end_string)
```

```
                   A         B         C         D
2000-01-01  0.379490  1.353836 -0.634997 -0.066652
2000-01-02  0.635418 -1.348018 -0.310449  0.278415
2000-01-03 -0.582322 -0.539813  0.785476  1.236848
2000-01-04 -0.130398  1.050291 -0.832720 -1.692569
2000-01-05  1.040468 -1.033498  0.948710 -0.085250
2000-01-06  1.307932  2.202969  1.640022 -2.448868
2000-01-07 -0.203380 -1.231041  0.785852 -0.141751
2000-01-08 -1.234723 -0.745927  1.268844 -0.412678
--------------------------------------------------
                   A         B         C         D
2000-01-02  0.635418 -1.348018 -0.310449  0.278415
2000-01-05  1.040468 -1.033498  0.948710 -0.085250
2000-01-07 -0.203380 -1.231041  0.785852 -0.141751
--------------------------------------------------
```

Here, a boolean mask is defined by some conditions.

Note the values of A and B columns

Series

- Exercise

1. Write codes to create a random dict **x** which has 5 random keys and each key corresponds to a 6-D numpy array
2. Create a pandas dataframe using **x**
3. Create a pandas dataframe using a subset of **x**, in the subset of **x,** only keys that start with a digit are chosen.

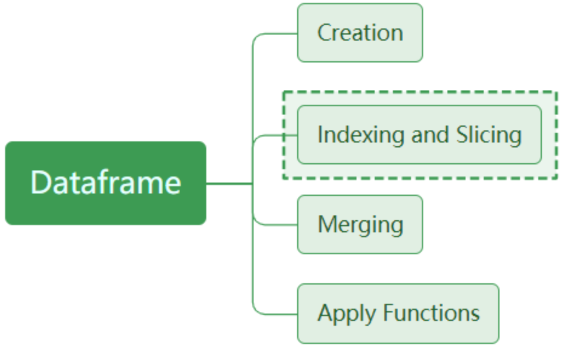
---

4. Create a new pandas dataframe using the codes in the previous slide.
5. Select rows whose attribute A is smaller than the mean of attribute C
6. Can you select the column B and C using [] indexing? Try it out and see what happens.

Dataframe

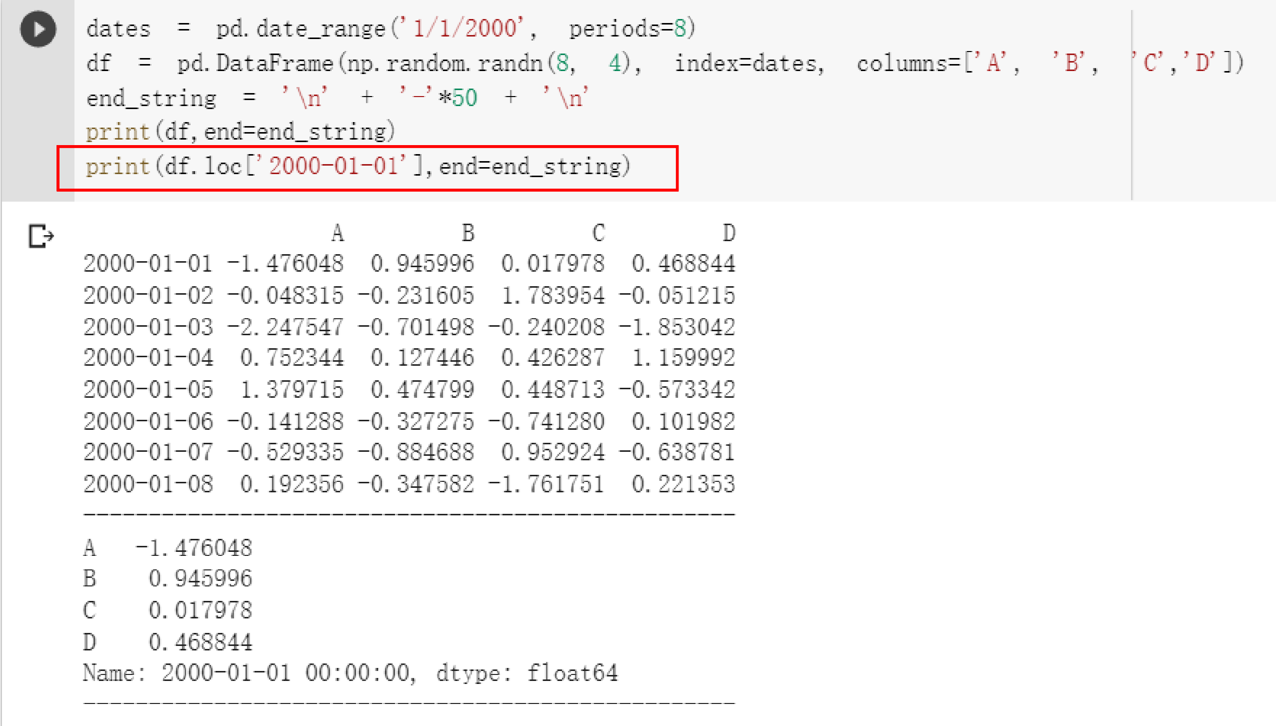
- Selecting by label .loc (string based)

Operation	Syntax	Result
Select Row by Label	df.loc[label]	Series or dataFrame



Allowed inputs:

1. A single label
2. A list of labels
3. A boolean array



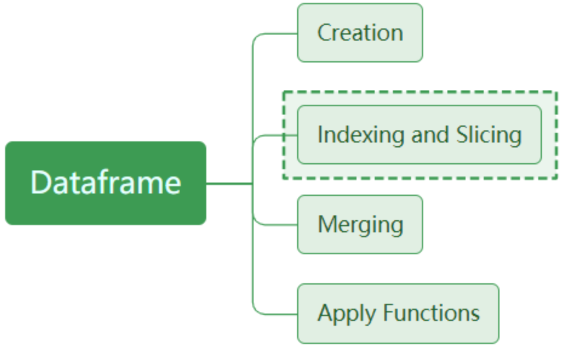
```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)
print(df.loc['2000-01-01'], end=end_string)
```

```
                   A         B         C         D
2000-01-01 -1.476048  0.945996  0.017978  0.468844
2000-01-02 -0.048315 -0.231605  1.783954 -0.051215
2000-01-03 -2.247547 -0.701498 -0.240208 -1.853042
2000-01-04  0.752344  0.127446  0.426287  1.159992
2000-01-05  1.379715  0.474799  0.448713 -0.573342
2000-01-06 -0.141288 -0.327275 -0.741280  0.101982
2000-01-07 -0.529335 -0.884688  0.952924 -0.638781
2000-01-08  0.192356 -0.347582 -1.761751  0.221353
--------------------------------------------------

A   -1.476048
B    0.945996
C    0.017978
D    0.468844
Name: 2000-01-01 00:00:00, dtype: float64
--------------------------------------------------
```

Dataframe

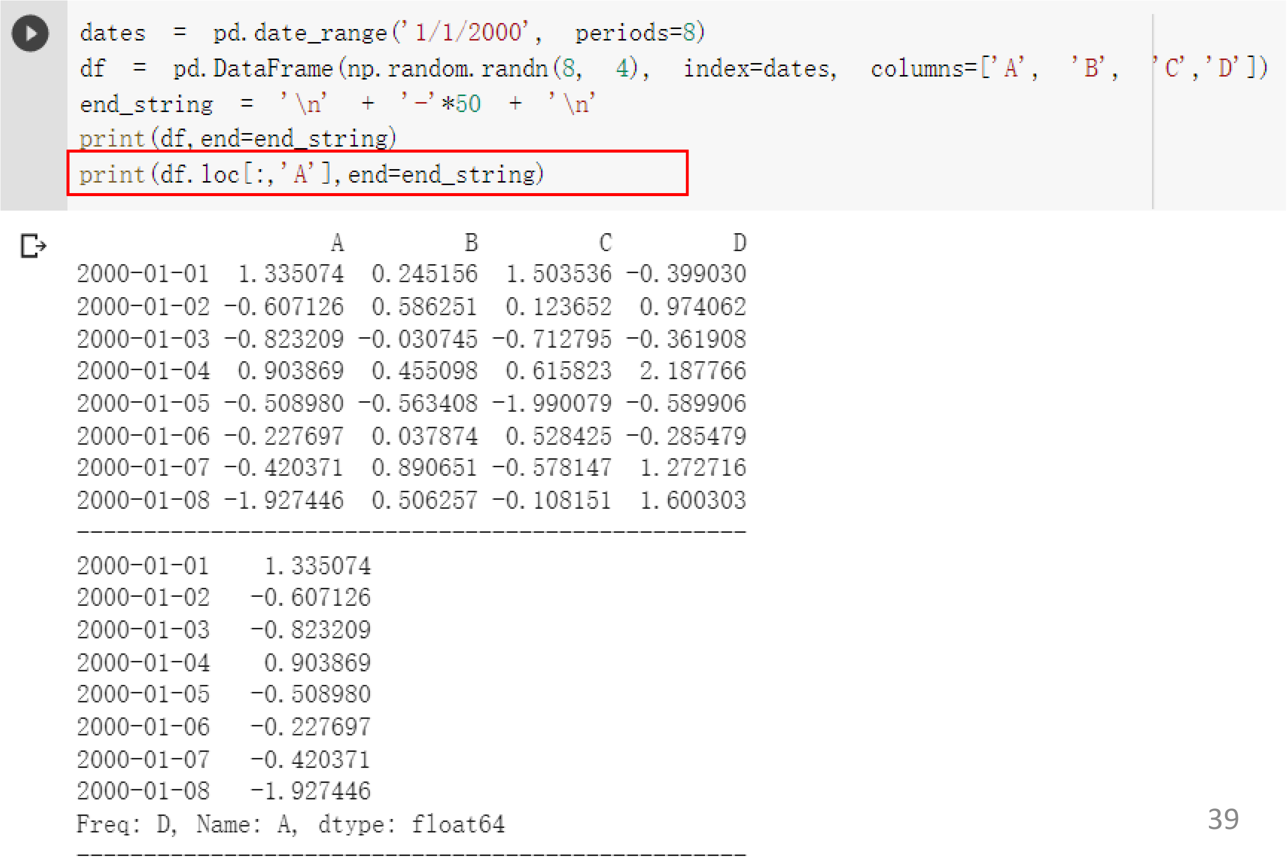
- Selecting by label .loc (string based)

Operation	Syntax	Result
Select Row by Label	df.loc[label]	Series or dataFrame



Allowed inputs:

1. A single label
2. A list of labels
3. A boolean array



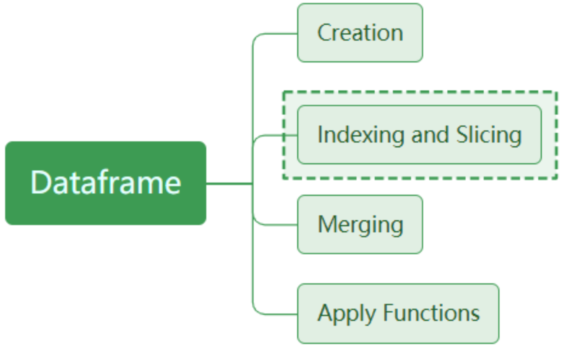
```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)
print(df.loc[:,'A'], end=end_string)
```

```
                   A         B         C         D
2000-01-01  1.335074  0.245156  1.503536 -0.399030
2000-01-02 -0.607126  0.586251  0.123652  0.974062
2000-01-03 -0.823209 -0.030745 -0.712795 -0.361908
2000-01-04  0.903869  0.455098  0.615823  2.187766
2000-01-05 -0.508980 -0.563408 -1.990079 -0.589906
2000-01-06 -0.227697  0.037874  0.528425 -0.285479
2000-01-07 -0.420371  0.890651 -0.578147  1.272716
2000-01-08 -1.927446  0.506257 -0.108151  1.600303
--------------------------------------------------
2000-01-01    1.335074
2000-01-02   -0.607126
2000-01-03   -0.823209
2000-01-04    0.903869
2000-01-05   -0.508980
2000-01-06   -0.227697
2000-01-07   -0.420371
2000-01-08   -1.927446
Freq: D, Name: A, dtype: float64
--------------------------------------------------
```

Dataframe

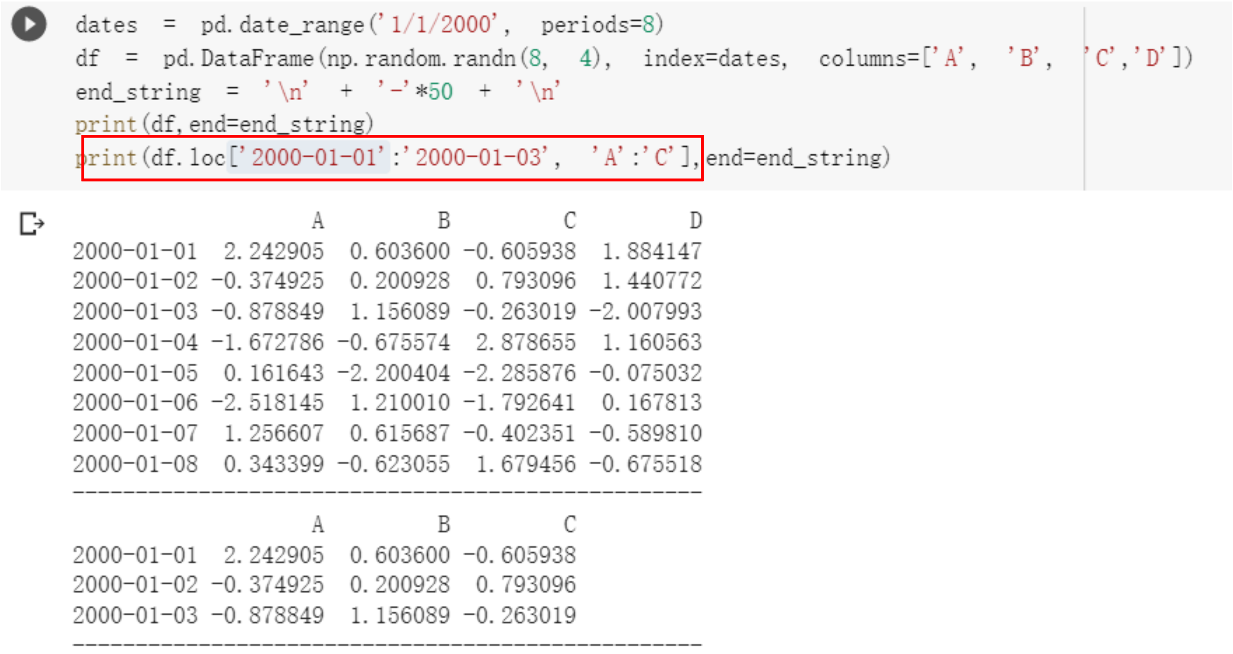
- Selecting by label .loc (string based)

Operation	Syntax	Result
Select Row by Label	df.loc[label]	Series or dataFrame



Allowed inputs:

1. A single label
2. A list of labels
3. A boolean array



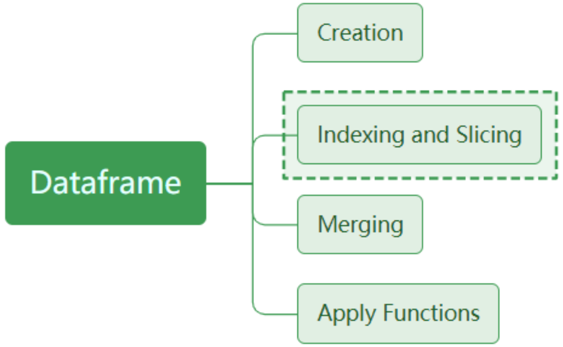
```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)
print(df.loc['2000-01-01':'2000-01-03', 'A':'C'],end=end_string)
```

```
                   A         B         C         D
2000-01-01  2.242905  0.603600 -0.605938  1.884147
2000-01-02 -0.374925  0.200928  0.793096  1.440772
2000-01-03 -0.878849  1.156089 -0.263019 -2.007993
2000-01-04 -1.672786 -0.675574  2.878655  1.160563
2000-01-05  0.161643 -2.200404 -2.285876 -0.075032
2000-01-06 -2.518145  1.210010 -1.792641  0.167813
2000-01-07  1.256607  0.615687 -0.402351 -0.589810
2000-01-08  0.343399 -0.623055  1.679456 -0.675518
--------------------------------------------------
                   A         B         C
2000-01-01  2.242905  0.603600 -0.605938
2000-01-02 -0.374925  0.200928  0.793096
2000-01-03 -0.878849  1.156089 -0.263019
--------------------------------------------------
```

Dataframe

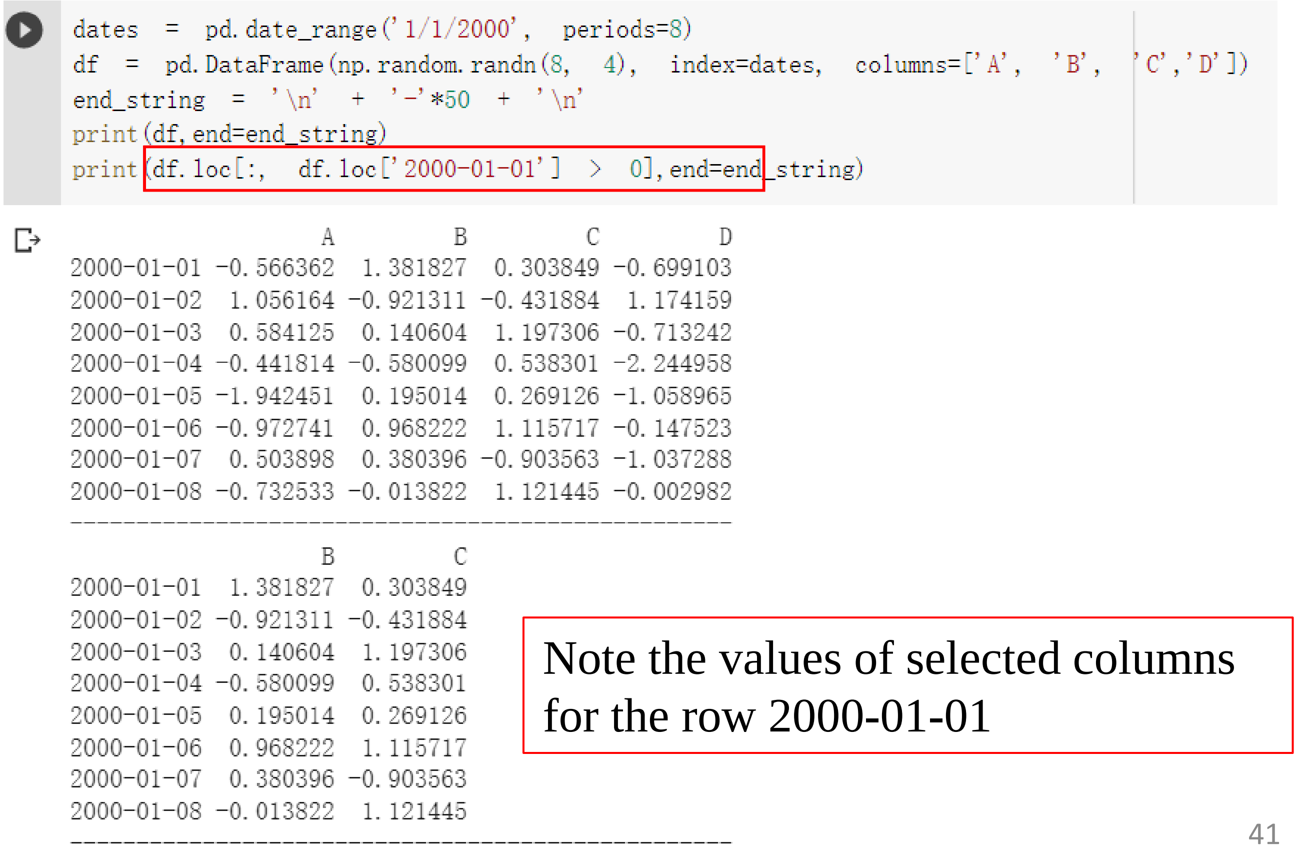
- Selecting by label .loc (string based)

Operation	Syntax	Result
Select Row by Label	df.loc[label]	Series or dataFrame



Allowed inputs:

1. A single label
2. A list of labels
3. A boolean array



```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)
print(df.loc[:, df.loc['2000-01-01'] > 0], end=end_string)
```

```
                   A         B         C         D
2000-01-01 -0.566362  1.381827  0.303849 -0.699103
2000-01-02  1.056164 -0.921311 -0.431884  1.174159
2000-01-03  0.584125  0.140604  1.197306 -0.713242
2000-01-04 -0.441814 -0.580099  0.538301 -2.244958
2000-01-05 -1.942451  0.195014  0.269126 -1.058965
2000-01-06 -0.972741  0.968222  1.115717 -0.147523
2000-01-07  0.503898  0.380396 -0.903563 -1.037288
2000-01-08 -0.732533 -0.013822  1.121445 -0.002982
--------------------------------------------------
                   B         C
2000-01-01  1.381827  0.303849
2000-01-02 -0.921311 -0.431884
2000-01-03  0.140604  1.197306
2000-01-04 -0.580099  0.538301
2000-01-05  0.195014  0.269126
2000-01-06  0.968222  1.115717
2000-01-07  0.380396 -0.903563
2000-01-08 -0.013822  1.121445
--------------------------------------------------
```

Note the values of selected columns for the row 2000-01-01

Series

- Exercise

1. Write codes to create a random dict **x** which has 5 random keys and each key corresponds to a 6-D numpy array
2. Create a pandas dataframe using **x**
3. Create a pandas dataframe using a subset of **x**, in the subset of **x,** only keys that start with a digit are chosen.

---

4. Create a new pandas dataframe using the codes in the previous slide.
5. Select rows whose attribute A is smaller than the mean of attribute C
6. Can you select the column B and C using [] indexing? Try it out and see what happens.

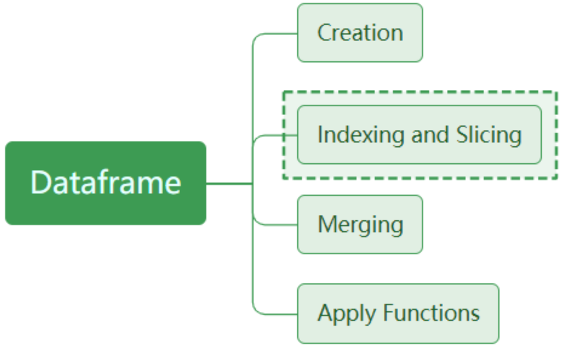
---

7. Now try to select the column B and C using the newly learned method.

Dataframe

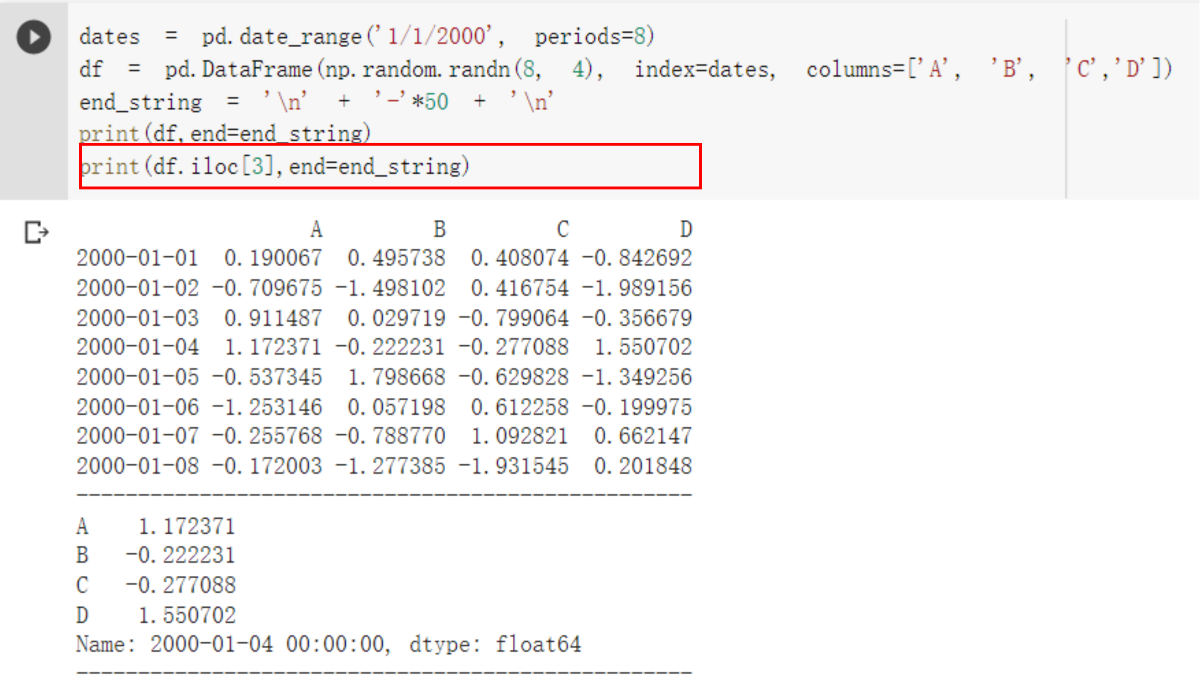
- Selecting by position .iloc (index based)

Operation	Syntax	Result
Select Row by Integer Location	df.iloc[idx]	Series/Dataframe



Allowed inputs:

1. An integer
2. A list of integers
3. A slice
4. A boolean array



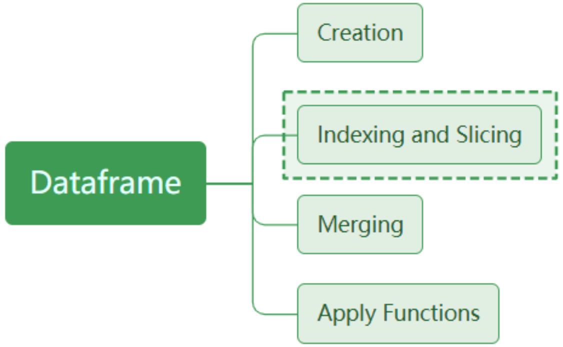
```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)
print(df.iloc[3], end=end_string)
```

```
                   A         B         C         D
2000-01-01  0.190067  0.495738  0.408074 -0.842692
2000-01-02 -0.709675 -1.498102  0.416754 -1.989156
2000-01-03  0.911487  0.029719 -0.799064 -0.356679
2000-01-04  1.172371 -0.222231 -0.277088  1.550702
2000-01-05 -0.537345  1.798668 -0.629828 -1.349256
2000-01-06 -1.253146  0.057198  0.612258 -0.199975
2000-01-07 -0.255768 -0.788770  1.092821  0.662147
2000-01-08 -0.172003 -1.277385 -1.931545  0.201848
--------------------------------------------------
A    1.172371
B   -0.222231
C   -0.277088
D    1.550702
Name: 2000-01-04 00:00:00, dtype: float64
--------------------------------------------------
```

Dataframe

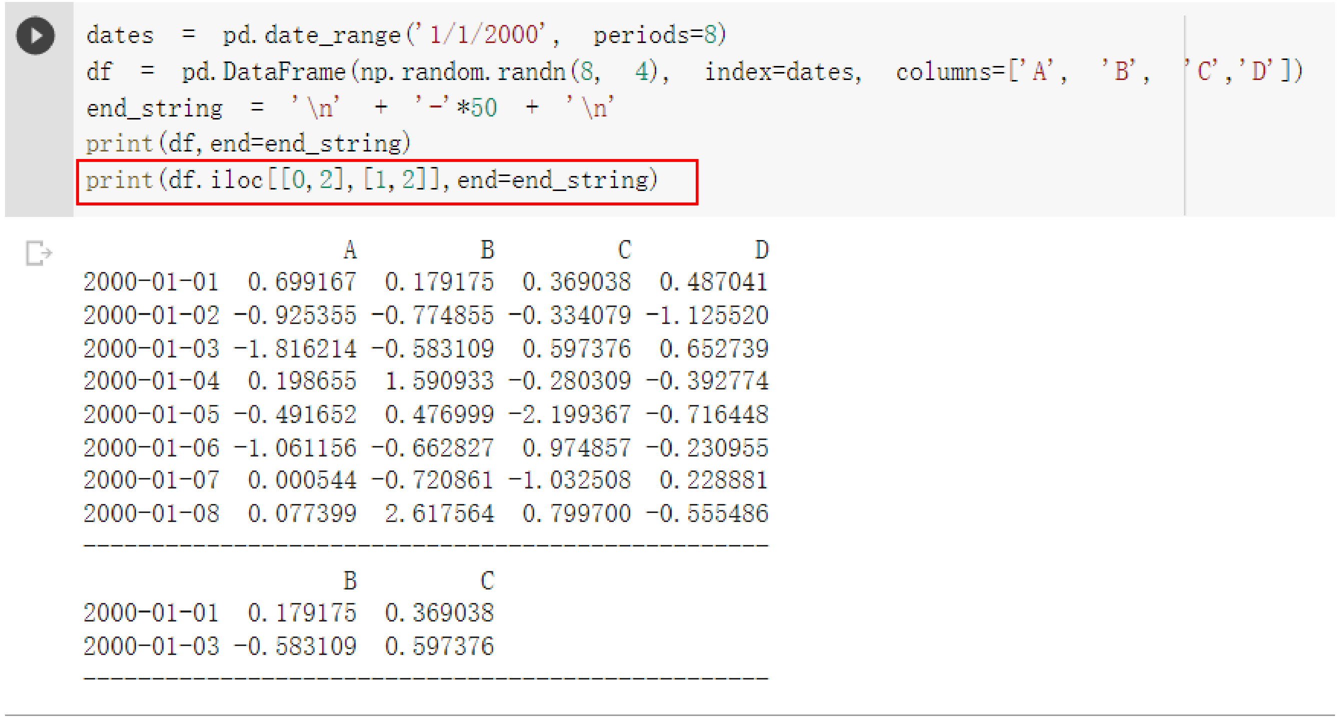
- Selecting by position .iloc (index based)

Operation	Syntax	Result
Select Row by Integer Location	df.iloc[idx]	Series/Dataframe


Dataframe
Creation
Indexing and Slicing
Merging
Apply Functions

Allowed inputs:

1. An integer
2. A list of integers
3. A slice
4. A boolean array



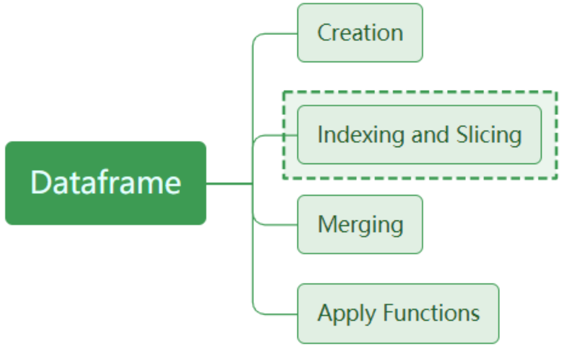
```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)
print(df.iloc[[0, 2], [1, 2]], end=end_string)
```

```
                   A         B         C         D
2000-01-01  0.699167  0.179175  0.369038  0.487041
2000-01-02 -0.925355 -0.774855 -0.334079 -1.125520
2000-01-03 -1.816214 -0.583109  0.597376  0.652739
2000-01-04  0.198655  1.590933 -0.280309 -0.392774
2000-01-05 -0.491652  0.476999 -2.199367 -0.716448
2000-01-06 -1.061156 -0.662827  0.974857 -0.230955
2000-01-07  0.000544 -0.720861 -1.032508  0.228881
2000-01-08  0.077399  2.617564  0.799700 -0.555486
--------------------------------------------------
                   B         C
2000-01-01  0.179175  0.369038
2000-01-03 -0.583109  0.597376
--------------------------------------------------
```

Dataframe

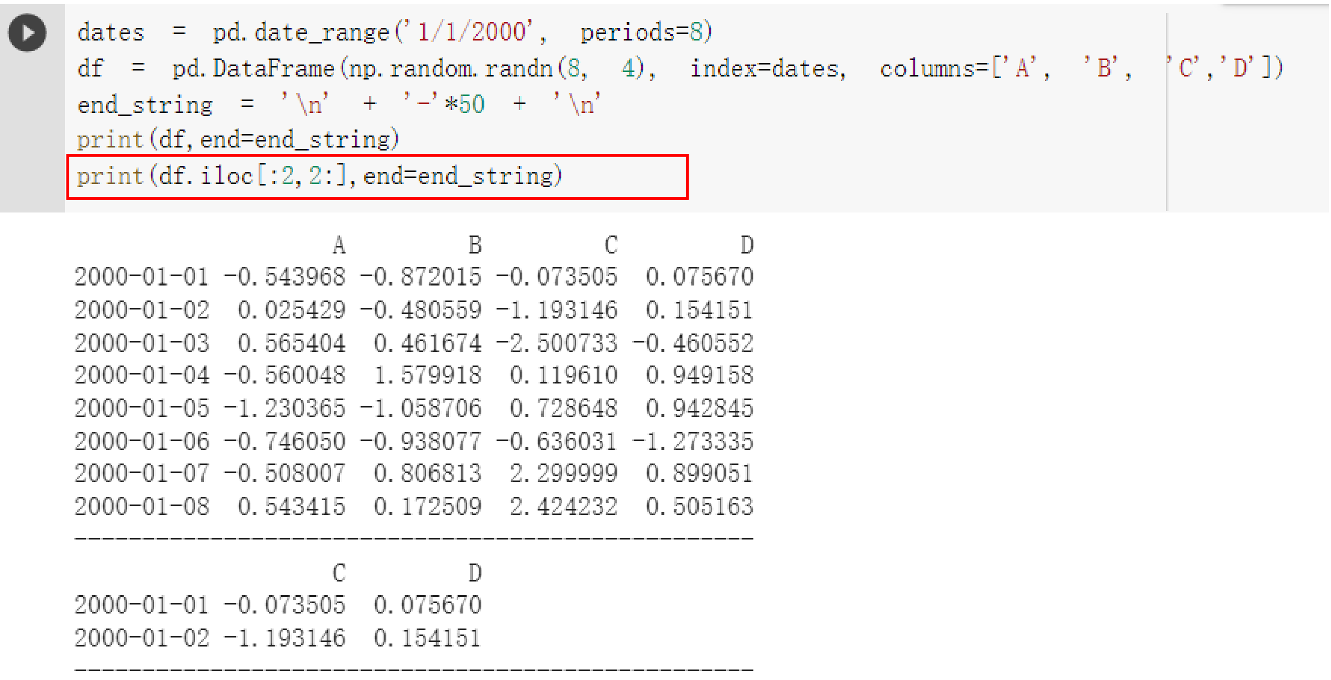
- Selecting by position .iloc (index based)

Operation	Syntax	Result
Select Row by Integer Location	df.iloc[idx]	Series/Dataframe



Allowed inputs:

1. An integer
2. A list of integers
3. A slice
4. A boolean array



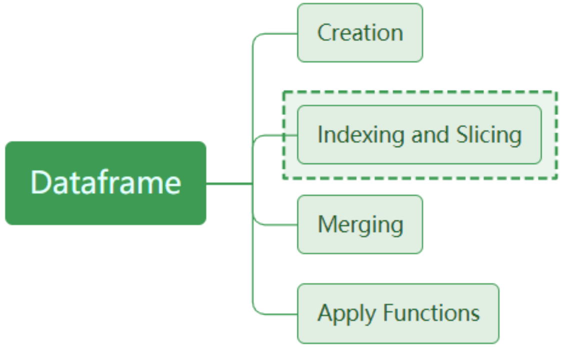
```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)
print(df.iloc[:2,2:], end=end_string)
```

```
                   A         B         C         D
2000-01-01 -0.543968 -0.872015 -0.073505  0.075670
2000-01-02  0.025429 -0.480559 -1.193146  0.154151
2000-01-03  0.565404  0.461674 -2.500733 -0.460552
2000-01-04 -0.560048  1.579918  0.119610  0.949158
2000-01-05 -1.230365 -1.058706  0.728648  0.942845
2000-01-06 -0.746050 -0.938077 -0.636031 -1.273335
2000-01-07 -0.508007  0.806813  2.299999  0.899051
2000-01-08  0.543415  0.172509  2.424232  0.505163
--------------------------------------------------
                   C         D
2000-01-01 -0.073505  0.075670
2000-01-02 -1.193146  0.154151
--------------------------------------------------
```

Dataframe

- Selecting by position .iloc (index based)

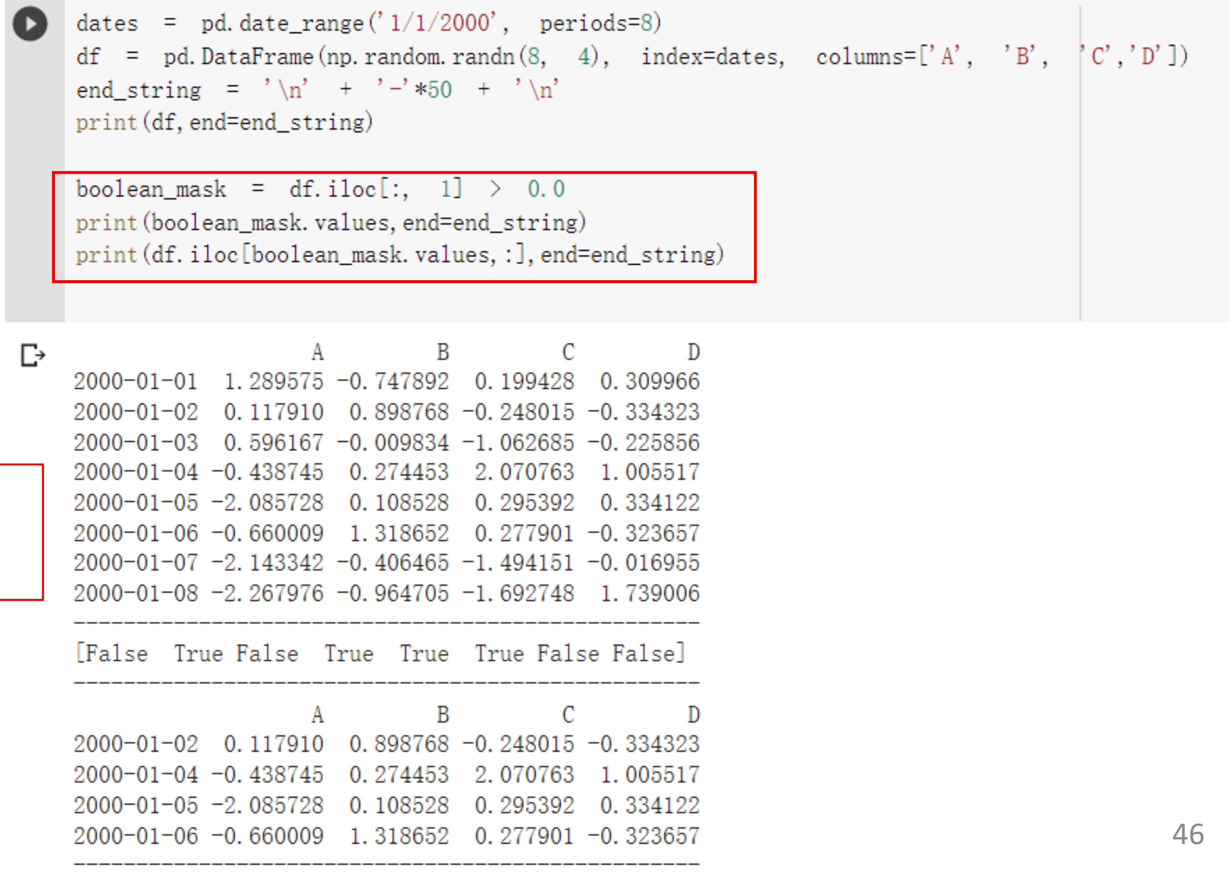
Operation	Syntax	Result
Select Row by Integer Location	df.iloc[idx]	Series/Dataframe



Allowed inputs:

1. An integer
2. A list of integers
3. A slice
4. A boolean array

Selecting the dates with positive values in column B



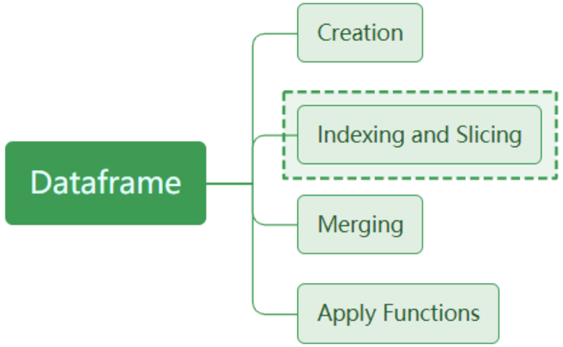
```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)

boolean_mask = df.iloc[:, 1] > 0.0
print(boolean_mask.values, end=end_string)
print(df.iloc[boolean_mask.values, :], end=end_string)
```

```
                   A         B         C         D
2000-01-01  1.289575 -0.747892  0.199428  0.309966
2000-01-02  0.117910  0.898768 -0.248015 -0.334323
2000-01-03  0.596167 -0.009834 -1.062685 -0.225856
2000-01-04 -0.438745  0.274453  2.070763  1.005517
2000-01-05 -2.085728  0.108528  0.295392  0.334122
2000-01-06 -0.660009  1.318652  0.277901 -0.323657
2000-01-07 -2.143342 -0.406465 -1.494151 -0.016955
2000-01-08 -2.267976 -0.964705 -1.692748  1.739006
--------------------------------------------------
[False  True False  True  True  True False False]
--------------------------------------------------
                   A         B         C         D
2000-01-02  0.117910  0.898768 -0.248015 -0.334323
2000-01-04 -0.438745  0.274453  2.070763  1.005517
2000-01-05 -2.085728  0.108528  0.295392  0.334122
2000-01-06 -0.660009  1.318652  0.277901 -0.323657
--------------------------------------------------
```

Dataframe

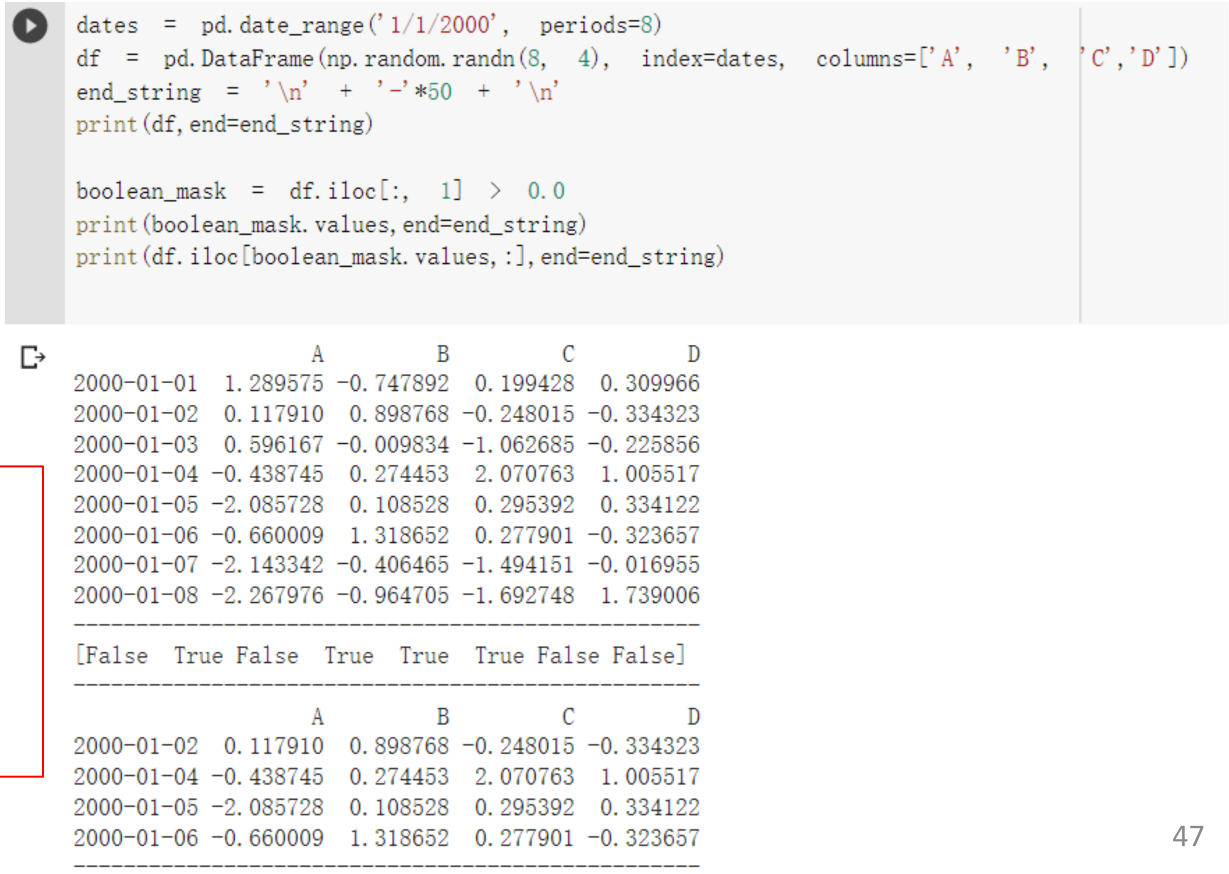
- Selecting by position .iloc (index based)

Operation	Syntax	Result
Select Row by Integer Location	df.iloc[idx]	Series/Dataframe



Allowed inputs:

1. An integer
2. A list of integers
3. A slice
4. A boolean array



```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)

boolean_mask = df.iloc[:, 1] > 0.0
print(boolean_mask.values, end=end_string)
print(df.iloc[boolean_mask.values, :], end=end_string)
```

```
                   A         B         C         D
2000-01-01  1.289575 -0.747892  0.199428  0.309966
2000-01-02  0.117910  0.898768 -0.248015 -0.334323
2000-01-03  0.596167 -0.009834 -1.062685 -0.225856
2000-01-04 -0.438745  0.274453  2.070763  1.005517
2000-01-05 -2.085728  0.108528  0.295392  0.334122
2000-01-06 -0.660009  1.318652  0.277901 -0.323657
2000-01-07 -2.143342 -0.406465 -1.494151 -0.016955
2000-01-08 -2.267976 -0.964705 -1.692748  1.739006
--------------------------------------------------
[False  True False  True  True  True False False]
--------------------------------------------------
                   A         B         C         D
2000-01-02  0.117910  0.898768 -0.248015 -0.334323
2000-01-04 -0.438745  0.274453  2.070763  1.005517
2000-01-05 -2.085728  0.108528  0.295392  0.334122
2000-01-06 -0.660009  1.318652  0.277901 -0.323657
--------------------------------------------------
```

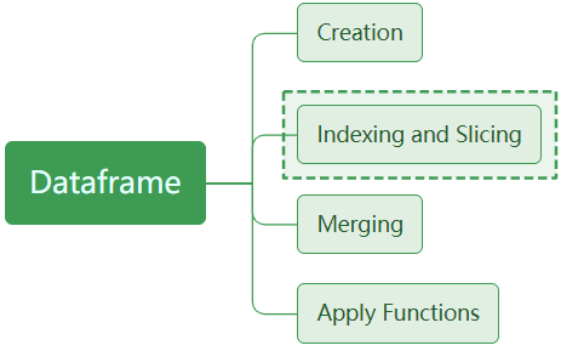
Selecting the dates with positive values in column B

Recap: How to achieve this by using .loc?

Dataframe

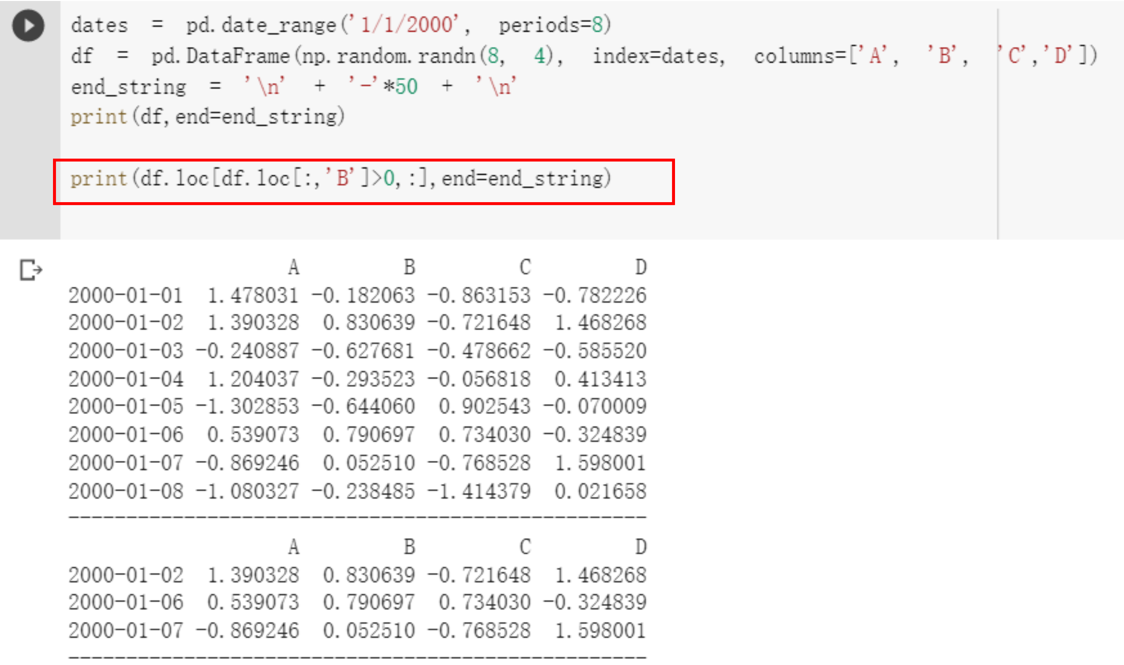
- Selecting by position .iloc (index based)

Operation	Syntax	Result
Select Row by Integer Location	df.iloc[idx]	Series/Dataframe


Dataframe
Creation
Indexing and Slicing
Merging
Apply Functions

Allowed inputs:

1. An integer
2. A list of integers
3. A slice
4. A boolean array



```
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C','D'])
end_string = '\n' + '-'*50 + '\n'
print(df,end=end_string)

print(df.loc[df.loc[:,'B']>0,:],end=end_string)
```

```
                   A         B         C         D
2000-01-01  1.478031 -0.182063 -0.863153 -0.782226
2000-01-02  1.390328  0.830639 -0.721648  1.468268
2000-01-03 -0.240887 -0.627681 -0.478662 -0.585520
2000-01-04  1.204037 -0.293523 -0.056818  0.413413
2000-01-05 -1.302853 -0.644060  0.902543 -0.070009
2000-01-06  0.539073  0.790697  0.734030 -0.324839
2000-01-07 -0.869246  0.052510 -0.768528  1.598001
2000-01-08 -1.080327 -0.238485 -1.414379  0.021658
--------------------------------------------------
                   A         B         C         D
2000-01-02  1.390328  0.830639 -0.721648  1.468268
2000-01-06  0.539073  0.790697  0.734030 -0.324839
2000-01-07 -0.869246  0.052510 -0.768528  1.598001
--------------------------------------------------
```

Selecting the dates with positive values in column B

Recap: How to achieve this by using .loc?

Series

- Exercise

1. Write codes to create a random dict **x** which has 5 random keys and each key corresponds to a 6-D numpy array
2. Create a pandas dataframe using **x**
3. Create a pandas dataframe using a subset of **x**, in the subset of **x,** only keys that start with a digit are chosen.

---

4. Create a new pandas dataframe using the codes in the previous slide.
5. Select rows whose attribute A is smaller than the mean of attribute C
6. Can you select the column B and C using [] indexing? Try it out and see what happens.

---

7. Now try to select the column B and C using the newly learned method.

---

8. Select the upper right half of the dataframe to create a new dataframe **"ur_df"**, find out the largest value in the last row of the **ur_df.**