COMP7035

Python for Data Analytics and Artificial Intelligence

Pandas

Renjie Wan

4/11/2024

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

What is Pandas?

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name “Pandas” has a reference to both “Panel Data”, and “Python Data Analysis” and was created by Wes McKinney in 2008.

Why Use Pandas?

- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.

What Can Pandas Do?

- Pandas gives you answers about the data.
- Is there a correlation between two or more columns?
  - What is average value?
  - Max value?
  - Min value?
- Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

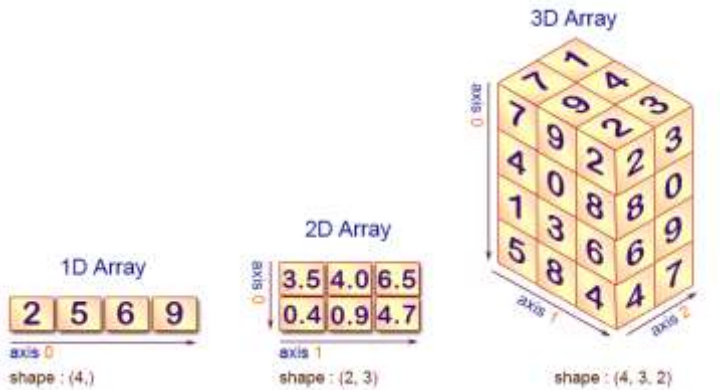
How to Use Pandas?

Just import it!

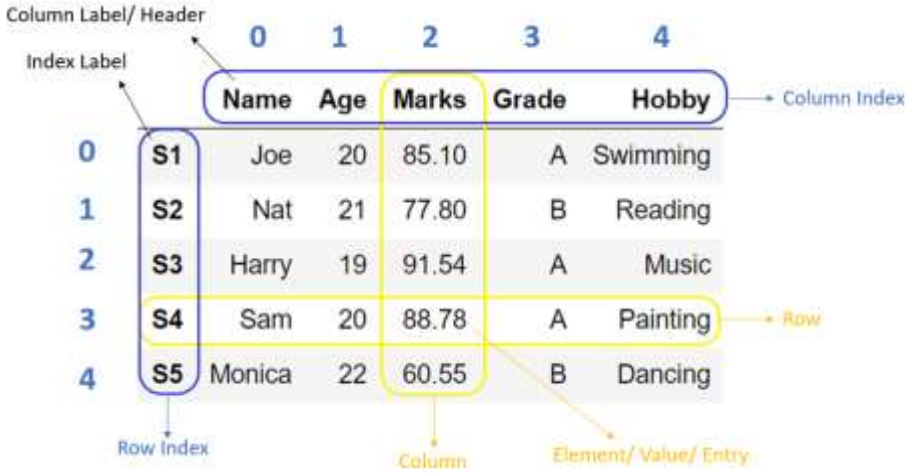
```
import pandas as pd
```

Unique property of Pandas

- NumPy arrays are designed to contain data of one type.
- Pandas can contain different types of data.
- Different I/O functions, table operations, time series-specific functionalities.



Numpy Array



Pandas DataFrames

Learning Roadmap in Pandas

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

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    0.397214
b    0.802672
c    0.907759
d   -1.030996
e    2.608806
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
```

- If you want to specify the order of index:

Index is constructed as sorted keys

```
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
  - Use $x.to\_list()$ to convert to list
  - Use $x.to\_dict()$ to convert to the dict

```
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)
```

Explanations:

1. The code imports the `pandas` and `numpy` libraries.
2. It creates a dictionary called `dict` with keys 'a', 'b', 'c', 'd', and 'e' associated with values 100, 200, 300, 400, and 500 respectively.
3. The original dictionary is printed.
4. A `pandas.Series` object `s` is created from the dictionary.
5. The series `s` is printed.
6. The series `s` is converted to a list (`s_list`) and to a dictionary (`s_dict`).
7. Both the list and dictionary representations are printed.

Series

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

```
a    1.764052
b    0.400157
c    0.978738
d    2.240893
e    1.867558
```

```
import numpy as np
import pandas as pd

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

print('s[0]={}'.format(s[0]))
print('s["a"]={}'.format(s['a']))
```

```
s[0] = 1.764052

s["a"] = 1.764052
```

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.randn(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   -0.071105
a    2.106703
c    0.446140
d    0.837434
e    0.204917
dtype: float64
--------------------------------------------------
b   -0.071105
a    2.106703
dtype: float64
--------------------------------------------------
a    2.106703
c    0.446140
d    0.837434
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.randn(5), index=['b', 'a', 'c', 'd', 'e'])
print(s, end=end_string)
print(s[s>0.5], end=end_string)
```

```
b    0.832753
a   -0.781549
c    0.602902
d   -0.644796
e   -0.217810
dtype: float64
--------------------------------------------------
b    0.832753
c    0.602902
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.randn(5), index=['b', 'a', 'c', 'd', 'e'])
print(s, end=end_string)
s['a'] = 0
s['f'] = 'test'
print(s, end=end_string)
```

```
b    1.064368
a    0.141706
c   -0.018600
d   -0.920672
e   -0.035819
dtype: float64
--------------------------------------------------
b    1.064368
a         0.0
c     -0.0186
d   -0.920672
e   -0.035819
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.randn(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   -1.300723
a   -0.039006
c   -0.383477
d    0.744251
e    0.273632
dtype: float64
--------------------------------------------------
False
--------------------------------------------------
None
--------------------------------------------------
0.2736315719690892
--------------------------------------------------
```

Note what value is returned

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.randn(5), index=['b', 'a', 'c', 'd', 'e'])
print(s, end=end_string)
print(np.exp(s), end=end_string)
```

```
b    0.070831
a    1.001437
c   -2.216546
d    0.215791
e   -0.445475
dtype: float64
--------------------------------------------------
b    1.073400
a    2.722192
c    0.108985
d    1.240843
e    0.640520
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.randn(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   -0.468324
a   -0.319533
c    1.280803
d   -0.461655
e   -1.029969
dtype: float64
--------------------------------------------------
Index(['b', 'a', 'c', 'd', 'e'], dtype='object')
--------------------------------------------------
[-0.46832384 -0.31953284  1.28080299 -0.46165502 -1.02996893]
--------------------------------------------------
(5,)
--------------------------------------------------
```

Series

- Iteration

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

```
b   -1.168546
a   -0.648594
c   -0.907397
d    0.148649
e   -0.002735
dtype: float64
--------------------------------------------------
b -1.1685464442233768
a -0.6485944514706603
c -0.9073971630042718
d 0.1486489587437834
e -0.002734597592328171
```

Learning Roadmap in Pandas

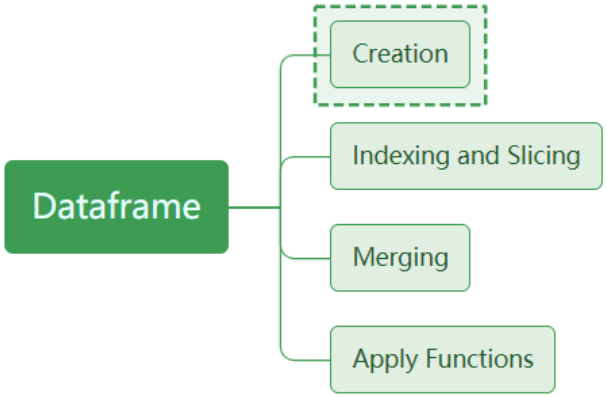
- Pandas Objects
  - Series
  - Dataframe
- Pandas I/O Functions

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

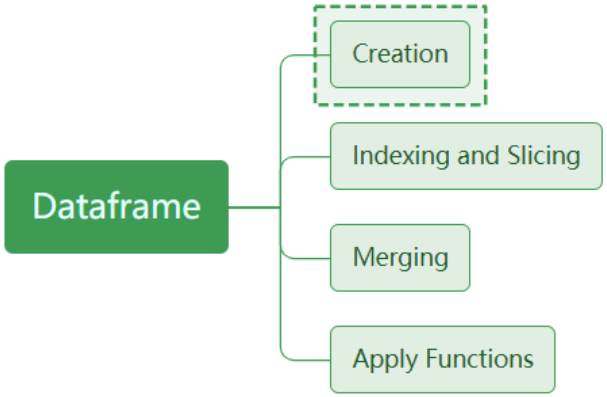


- Creating a Dataframe
  - From dict of series or dicts
    - **From dict of series**
    - **From dicts**

```
# 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



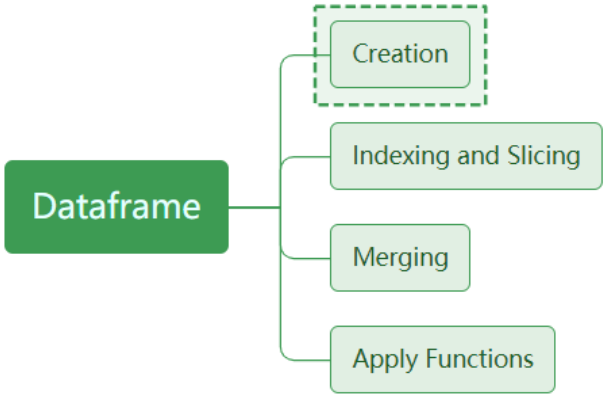
- Creating a Dataframe
  - From dict of series or dicts
    - **From dict of series**
    - **From dicts**

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)

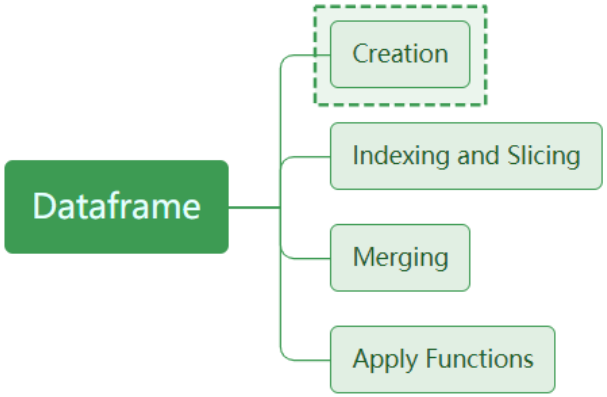
```
d = {'one': [1., 2., 3., 4.], 'two': [4., 3., 2., 1.]}
pd.DataFrame(d)
```

	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



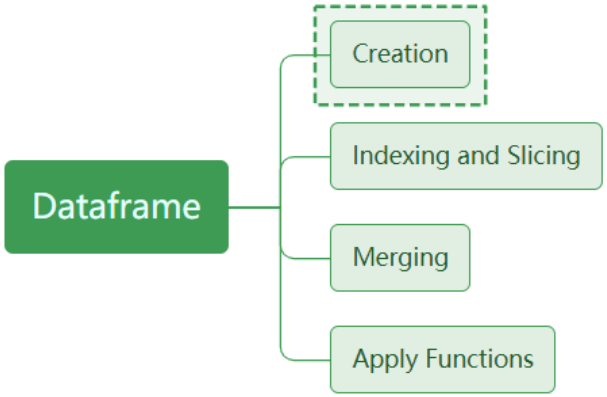
- Creating a Dataframe
  - **From a list of dicts**

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


data[:5]
```

```
[{'Column0': 21, 'Column1': 33, 'Column2': 55, 'Column3': 29, 'Column4': 5},
 {'Column0': 16, 'Column1': 39, 'Column2': 16, 'Column3': 8, 'Column4': 58},
 {'Column0': 48, 'Column1': 9, 'Column2': 10, 'Column3': 55, 'Column4': 46},
 {'Column0': 89, 'Column1': 15, 'Column2': 98, 'Column3': 84, 'Column4': 53},
 {'Column0': 98, 'Column1': 26, 'Column2': 40, 'Column3': 41, 'Column4': 36}]
```

Dataframe

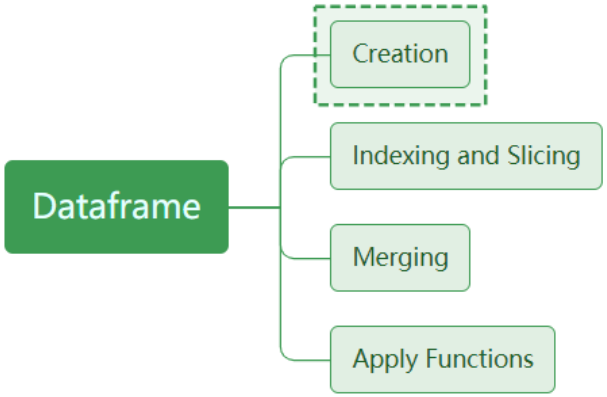


- Creating a Dataframe
  - **From a list of dicts**

```
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       21       33       55       29        5
1       16       39       16        8       58
2       48        9       10       55       46
3       89       15       98       84       53
4       98       26       40       41       36
--------------------------------------------------
   Column0  Column1
0       21       33
1       16       39
2       48        9
3       89       15
4       98       26
--------------------------------------------------
```

Dataframe

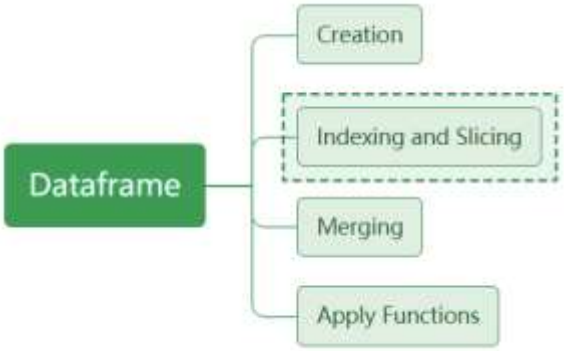


- Attributes
  - df.index : the row index of df
  - df.columns : the columns of df
  - df.shape : the shape of the df
  - df.values : numpy array of values

```
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       21       33       55       29
b       16       39       16        8
c       48        9       10       55
d       89       15       98       84
e       98       26       40       41
--------------------------------------------------------------------------------
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
--------------------------------------------------------------------------------
Index(['Column0', 'Column1', 'Column2', 'Column3'], dtype='object')
--------------------------------------------------------------------------------
(5, 4)
--------------------------------------------------------------------------------
[[21 33 55 29]
 [16 39 16  8]
 [48  9 10 55]
 [89 15 98 84]
 [98 26 40 41]]
--------------------------------------------------------------------------------
```

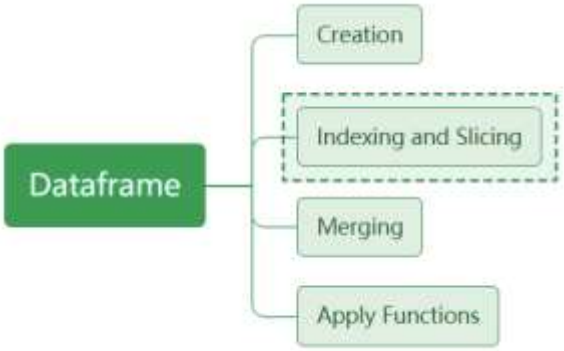
Dataframe



- 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

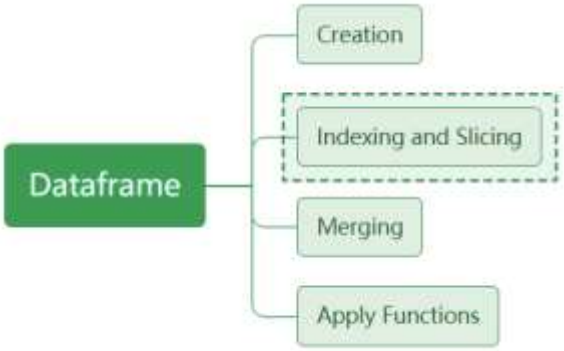
```
# Let's Create a data frame
pd.options.display.max_rows = 4
dates = pd.date_range('1/1/2000', periods=8)
df = pd.DataFrame(np.random.randn(8, 4), index = dates, columns = ['A', 'B', 'C', 'D'])
df
```

	A	B	C	D
2000-01-01	-0.997781	-0.215963	-0.605741	0.873614
2000-01-02	0.378836	0.596297	-0.699708	-0.678365
...	...	...	...	...
2000-01-07	0.269168	0.081180	-0.785234	-1.414546
2000-01-08	-1.424302	-0.508259	-0.710441	-0.370580

8 rows × 4 columns

Let us create a dataframe first

Dataframe



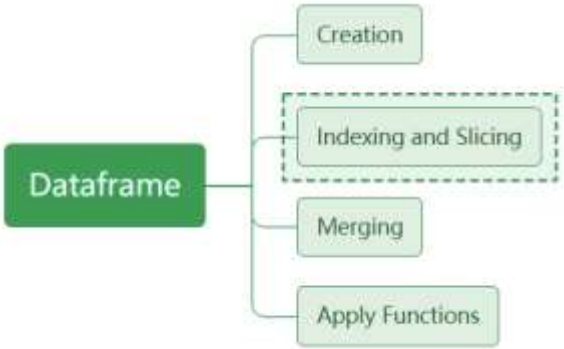
- Simplest form of Indexing: []

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

```
# Let's 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    0.688436
2000-01-02   -0.386200
                ...
2000-01-07   -0.424302
2000-01-08   -0.021676
Freq: D, Name: A, Length: 8, dtype: float64
```

Dataframe



- Simplest form of Indexing: []

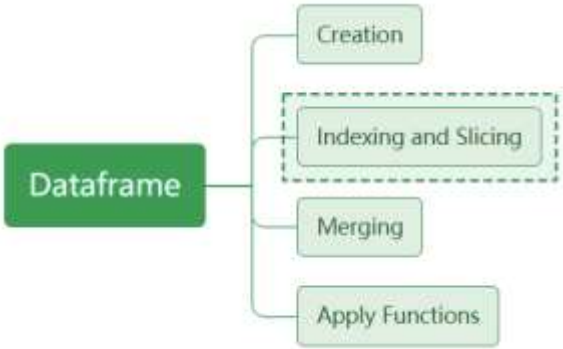
Operation	Syntax	Result
Select Column	df[col]	Series
Select Columns	df[col_list]	DataFrame
Slice rows	df[5:10]	DataFrame
Select rows by boolean	df[mask]	DataFrame

```
# Let's 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	-0.751194	0.425156
2000-01-02	0.887640	0.651765
...	...	...
2000-01-07	1.460495	-0.587518
2000-01-08	-1.423355	-0.151773

8 rows × 2 columns

Dataframe



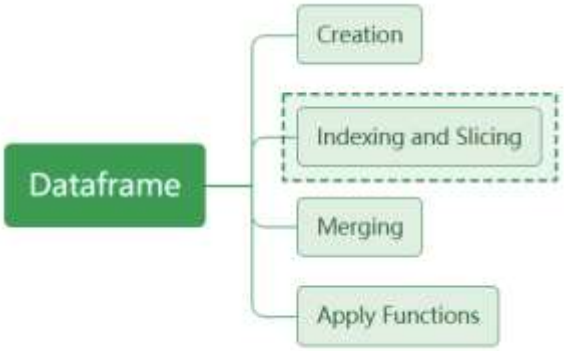
- Simplest form of Indexing: []

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)
pd.set_option('display.max_rows', 8) # Display 8 rows
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 -0.365233  0.142506 -1.732764  0.924292
2000-01-02  0.654392  0.427583 -1.056638 -0.238343
2000-01-03 -2.138162  0.270142 -0.939101  0.244022
2000-01-04  0.128641 -0.689967  0.420726 -1.298280
2000-01-05  0.284054  0.150972  0.133405  0.492983
2000-01-06 -0.210794 -0.645579  0.249993 -0.744907
2000-01-07  0.948135 -1.299271  1.309775  0.412123
2000-01-08  0.696241 -0.069187  0.217833 -0.245216
--------------------------------------------------
                   A         B         C         D
2000-01-03 -2.138162  0.270142 -0.939101  0.244022
2000-01-04  0.128641 -0.689967  0.420726 -1.298280
2000-01-05  0.284054  0.150972  0.133405  0.492983
--------------------------------------------------
```

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

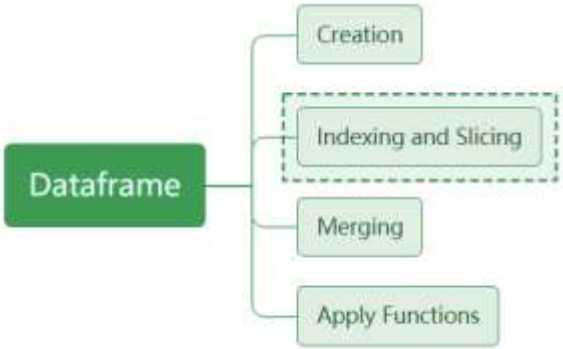
```
dates = pd.date_range('1/1/2000', periods=8)
pd.set_option('display.max_rows', 8) # Display 8 rows
df = pd.DataFrame(np.random.randn(8, 4), index = dates, columns = ['A', 'B', 'C', 'D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)
print(df[df['A']>df['B']], end=end_string)
```

```
                   A         B         C         D
2000-01-01 -0.002477  0.126967 -0.710448 -0.489816
2000-01-02 -1.584115 -2.476514  0.633584 -1.145928
2000-01-03  1.094663 -0.243002 -0.167205 -0.581259
2000-01-04 -1.000275  0.896312 -0.696041  0.756301
2000-01-05 -0.737790 -2.288674  0.632861  0.406432
2000-01-06  0.594861  0.419736  0.135544  0.960401
2000-01-07  0.224134  0.942081 -1.039566  0.731860
2000-01-08  0.342149 -0.854481  0.980198 -0.378723
--------------------------------------------------
                   A         B         C         D
2000-01-02 -1.584115 -2.476514  0.633584 -1.145928
2000-01-03  1.094663 -0.243002 -0.167205 -0.581259
2000-01-05 -0.737790 -2.288674  0.632861  0.406432
2000-01-06  0.594861  0.419736  0.135544  0.960401
2000-01-08  0.342149 -0.854481  0.980198 -0.378723
--------------------------------------------------
```

Here, a boolean mask is defined by some conditions.

Note the values of A and B columns

Dataframe



- Selecting by label .loc (string based)

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

- Allowed inputs:

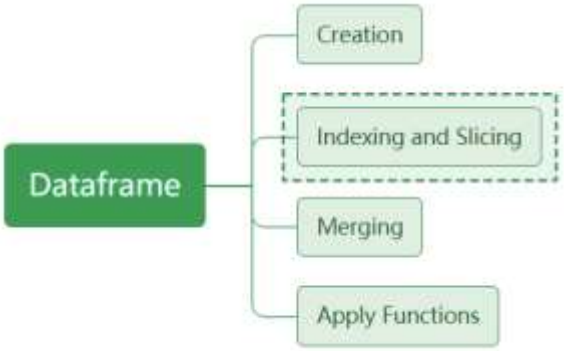
1. A single label
2. A list of labels
3. A boolean array

```
dates = pd.date_range('1/1/2000', periods=8)
pd.set_option('display.max_rows', 8) # Display 8 rows
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 -0.808969  0.121777 -0.164984 -0.060074
2000-01-02 -1.331889  1.188068 -0.330459 -1.237809
2000-01-03  0.765298 -1.460486 -0.309483 -1.097138
2000-01-04  0.065466  0.163511 -0.185552  0.006477
2000-01-05  2.312284  0.363372  1.006344 -0.849812
2000-01-06  0.532343 -0.188493  1.437959  1.014004
2000-01-07  1.875510 -0.191363 -1.410239  0.442029
2000-01-08  0.064507 -0.024586 -0.514740 -0.098022
--------------------------------------------------

A   -0.808969
B    0.121777
C   -0.164984
D   -0.060074
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:

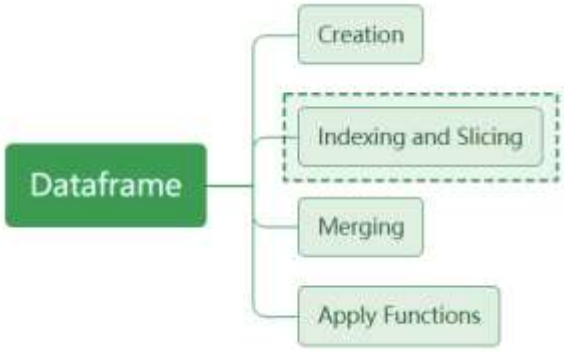
1. A single label
2. A list of labels
3. A boolean array

```
dates = pd.date_range('1/1/2000', periods=8)
pd.set_option('display.max_rows', 8) # Display 8 rows
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  0.146305 -1.038181  1.166563  1.354891
2000-01-02  0.102130  0.356554  0.556163 -1.360177
2000-01-03 -1.168610 -1.506452  1.969748  0.294921
2000-01-04  0.858679 -0.551758  0.209976 -0.485688
2000-01-05  0.199495 -2.199328 -0.874934 -0.013026
2000-01-06 -0.333012 -0.626984 -0.411459 -0.566501
2000-01-07 -0.138827 -0.747938 -0.103765 -1.891941
2000-01-08  1.307394  0.062616 -0.483316  0.688899
--------------------------------------------------

2000-01-01    0.146305
2000-01-02    0.102130
2000-01-03   -1.168610
2000-01-04    0.858679
2000-01-05    0.199495
2000-01-06   -0.333012
2000-01-07   -0.138827
2000-01-08    1.307394
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:

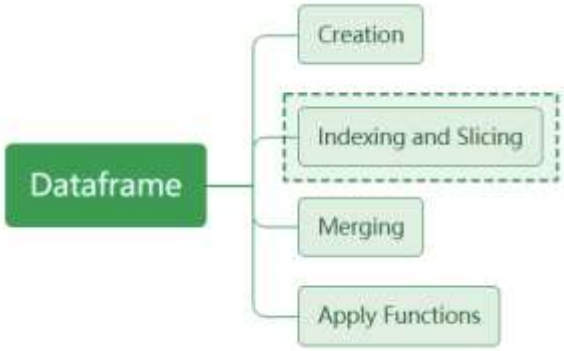
1. A single label
2. A list of labels
3. A boolean array

```
dates = pd.date_range('1/1/2000', periods=8)
pd.set_option('display.max_rows', 8) # Display 8 rows
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  0.741744  1.693977 -1.580422 -2.045799
2000-01-02 -1.458813 -1.472824 -0.771616  0.398972
2000-01-03 -0.976595  0.614056  0.898801 -0.858106
2000-01-04  0.063748  0.164629  0.547925 -1.365495
2000-01-05  0.524413 -1.657953 -1.095177  0.322759
2000-01-06 -0.511534  1.324298 -1.074259  0.804883
2000-01-07 -0.860723 -0.912653 -1.602376  0.445360
2000-01-08  1.154511  1.696445  1.400582  0.084790
--------------------------------------------------

                   A         B         C
2000-01-01  0.741744  1.693977 -1.580422
2000-01-02 -1.458813 -1.472824 -0.771616
2000-01-03 -0.976595  0.614056  0.898801
--------------------------------------------------
```

Dataframe



- Selecting by label .loc (string based)

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

- Allowed inputs:

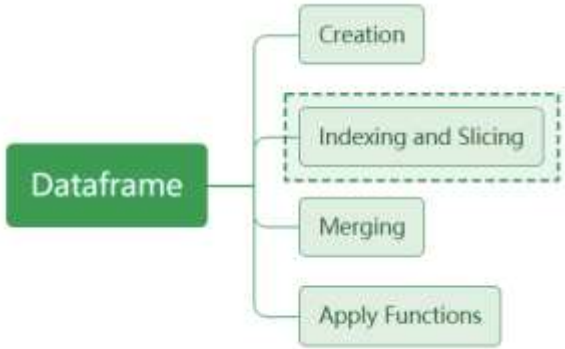
1. A single label
2. A list of labels
3. A boolean array

```
dates = pd.date_range('1/1/2000', periods=8)
pd.set_option('display.max_rows', 8) # Display 8 rows
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  1.081394 -0.029973  0.488412  0.216628
2000-01-02  0.920326 -1.969853 -1.261477  1.092389
2000-01-03  0.956746  1.900567 -0.580356  0.524056
2000-01-04 -0.537152 -1.637717 -2.510196 -0.133187
2000-01-05  1.036558  0.000195 -0.600056  0.321222
2000-01-06  0.374361 -1.075754  1.702924 -1.042571
2000-01-07 -0.888033  0.150886 -0.628813 -0.790897
2000-01-08  0.184880  0.281528  0.077296  1.200287
--------------------------------------------------
                   A         C         D
2000-01-01  1.081394  0.488412  0.216628
2000-01-02  0.920326 -1.261477  1.092389
2000-01-03  0.956746 -0.580356  0.524056
2000-01-04 -0.537152 -2.510196 -0.133187
2000-01-05  1.036558 -0.600056  0.321222
2000-01-06  0.374361  1.702924 -1.042571
2000-01-07 -0.888033 -0.628813 -0.790897
2000-01-08  0.184880  0.077296  1.200287
--------------------------------------------------
```

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

Dataframe



- Selecting by position .iloc (index based)

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

- Allowed inputs:

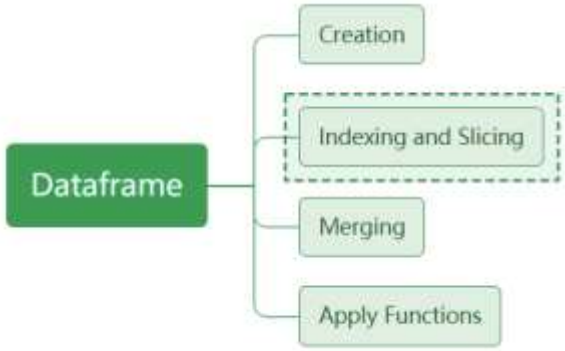
1. An integer
2. A list of integers
3. A slice
4. A boolean array

```
dates = pd.date_range('1/1/2000', periods=8)
pd.set_option('display.max_rows', 8) # Display 8 rows
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  1.143784 -0.036026 -0.110553 -0.792692
2000-01-02 -0.217390 -0.929814 -1.264581 -0.201286
2000-01-03 -0.196320 -2.344816 -2.353854  0.729177
2000-01-04 -1.837181 -0.879914 -2.348380  0.009436
2000-01-05 -0.277465 -0.641988 -0.230079  1.507065
2000-01-06  1.089284 -0.566820  0.925142  0.785233
2000-01-07 -1.129270 -0.386371 -0.222108  0.931982
2000-01-08  0.962160  0.452436 -0.172418  0.507342
--------------------------------------------------

A   -1.837181
B   -0.879914
C   -2.348380
D    0.009436
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

- Allowed inputs:

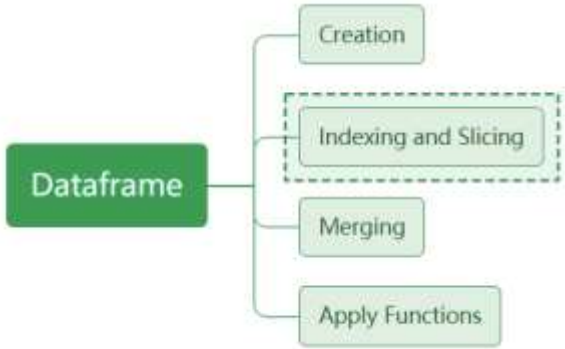
1. An integer
2. A list of integers
3. A slice
4. A boolean array

```
dates = pd.date_range('1/1/2000', periods=8)
pd.set_option('display.max_rows', 8) # Display 8 rows
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  0.541112  0.152200 -1.024768 -0.788707
2000-01-02 -0.210647  0.256112 -0.611746 -0.318302
2000-01-03  0.916152 -0.011732 -1.425540  0.500240
2000-01-04  0.935738 -0.452242 -1.611749 -1.050956
2000-01-05  0.250851 -0.588311  1.379296 -1.600118
2000-01-06  0.290758  0.234161 -1.079233  0.350385
2000-01-07 -1.178539  0.692972 -0.249572  1.422384
2000-01-08 -1.320499 -0.449876 -0.205323  2.287316
--------------------------------------------------

                   B         C
2000-01-01  0.152200 -1.024768
2000-01-03 -0.011732 -1.425540
--------------------------------------------------
```

Dataframe



- Selecting by position .iloc (index based)

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

- Allowed inputs:

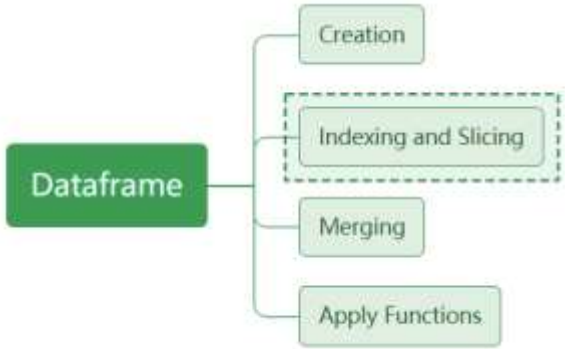
1. An integer
2. A list of integers
3. A slice
4. A boolean array

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

```
                   A         B         C         D
2000-01-01  1.328460 -1.122282  0.594011 -2.127828
2000-01-02 -1.981488  0.942143  0.672081 -0.570387
2000-01-03 -0.449819 -0.429354 -0.484726  0.039944
2000-01-04  0.588429  0.342350 -0.237064 -1.281482
2000-01-05  0.974922 -1.772634  0.471782 -0.546573
2000-01-06 -0.330551 -1.329987  0.666019  0.915355
2000-01-07  1.376066 -0.740158  0.734518  1.923113
2000-01-08 -0.801561 -0.362508 -0.061230  0.112099
--------------------------------------------------

                   C         D
2000-01-01  0.594011 -2.127828
2000-01-02  0.672081 -0.570387
--------------------------------------------------
```

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

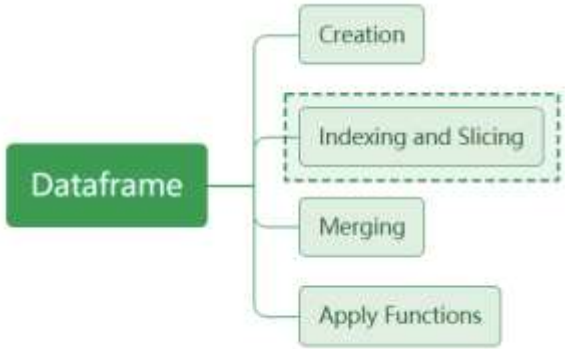
Selecting the dates with positive values in column B

```
dates = pd.date_range('1/1/2000', periods=8)
pd.set_option('display.max_rows', 8) # Display 8 rows
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 -0.030275  0.389217 -0.129020 -0.219700
2000-01-02  0.752066 -0.835376  0.045589 -0.290969
2000-01-03  2.186973  0.223427  0.475397  1.046907
2000-01-04  0.061963 -0.557540 -0.332795 -0.778923
2000-01-05  0.668725 -0.544839  1.127679  0.653052
2000-01-06  0.538854 -0.240451 -0.035167 -0.557230
2000-01-07 -0.569799 -0.814577  0.562515 -1.239621
2000-01-08  0.205420 -0.428879 -0.292322  0.841025
--------------------------------------------------
[ True False  True False False False False False]
--------------------------------------------------
                   A         B         C         D
2000-01-01 -0.030275  0.389217 -0.129020 -0.219700
2000-01-03  2.186973  0.223427  0.475397  1.046907
--------------------------------------------------
```

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

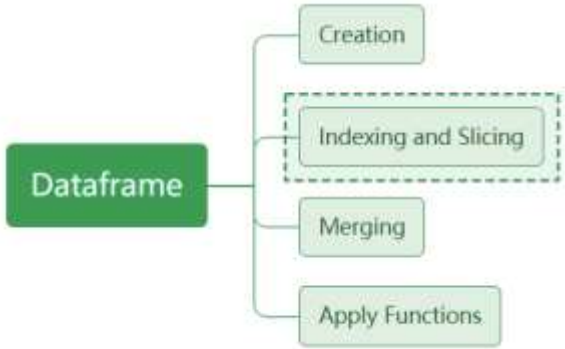
Recap: How to achieve this by using .loc?

```
dates = pd.date_range('1/1/2000', periods=8)
pd.set_option('display.max_rows', 8) # Display 8 rows
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 -0.030275  0.389217 -0.129020 -0.219700
2000-01-02  0.752066 -0.835376  0.045589 -0.290969
2000-01-03  2.186973  0.223427  0.475397  1.046907
2000-01-04  0.061963 -0.557540 -0.332795 -0.778923
2000-01-05  0.668725 -0.544839  1.127679  0.653052
2000-01-06  0.538854 -0.240451 -0.035167 -0.557230
2000-01-07 -0.569799 -0.814577  0.562515 -1.239621
2000-01-08  0.205420 -0.428879 -0.292322  0.841025
--------------------------------------------------
[ True False  True False False False False False]
--------------------------------------------------
                   A         B         C         D
2000-01-01 -0.030275  0.389217 -0.129020 -0.219700
2000-01-03  2.186973  0.223427  0.475397  1.046907
--------------------------------------------------
```

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

Recap: How to achieve this by using .loc?

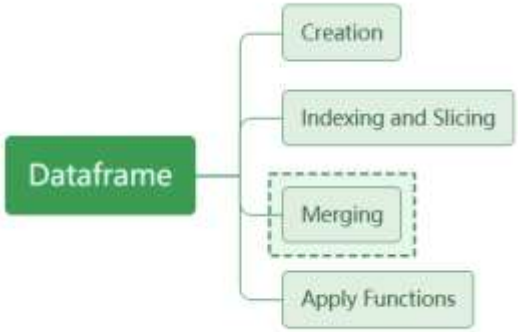
```
dates = pd.date_range('1/1/2000', periods=8)
pd.set_option('display.max_rows', 8) # Display 8 rows
df = pd.DataFrame(np.random.randn(8, 4), index = dates, columns = ['A', 'B', 'C', 'D'])
end_string = '\n' + '-'*50 + '\n'
print(df, end=end_string)

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

```
                   A         B         C         D
2000-01-01  0.531646 -0.794828  0.241826  0.000428
2000-01-02  2.064898  0.234741 -1.554342  0.069370
2000-01-03  0.338027 -0.275559 -0.655810  0.532822
2000-01-04  1.917023 -1.678259  1.607887  0.077468
2000-01-05 -0.196746 -0.106098  0.933458 -0.030213
2000-01-06 -0.473338 -1.659783  0.338613 -0.913054
2000-01-07  0.444095  1.349062 -0.231458 -1.872698
2000-01-08  1.061930  0.126789 -2.098040  1.263365
--------------------------------------------------

                   A         B         C         D
2000-01-02  2.064898  0.234741 -1.554342  0.069370
2000-01-07  0.444095  1.349062 -0.231458 -1.872698
2000-01-08  1.061930  0.126789 -2.098040  1.263365
--------------------------------------------------
```

Dataframe

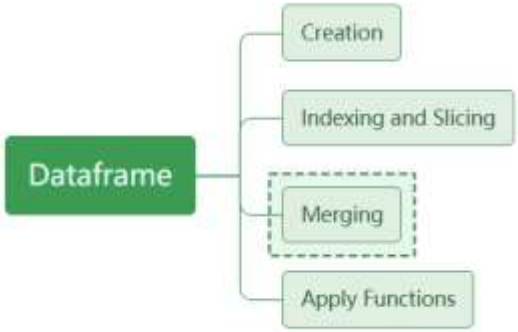


- Merging Dataframes
  - Pandas provides powerful operations for combining dataframes
  - Mainly four operations for merging, indicated by “how”

```
pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
         left_index=False, right_index=False, sort=True)
```

Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames

Dataframe



- Merging Dataframes

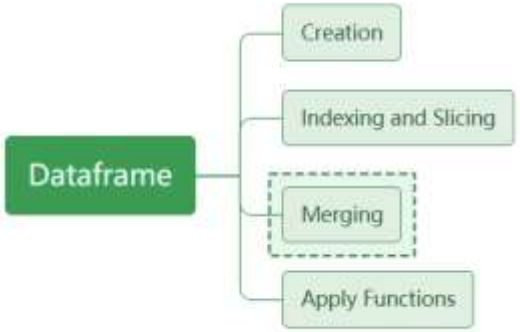
Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames

```
left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [4, 2]})
right = pd.DataFrame({'key': ['bar', 'zoo'], 'rval': [4, 5]})
print("left: ", left, "right: ", right, sep=end_string)
```

```
left:
------------------------------------------------
   key  lval
0  foo     4
1  bar     2
------------------------------------------------
right:
------------------------------------------------
   key  rval
0  bar     4
1  zoo     5
```

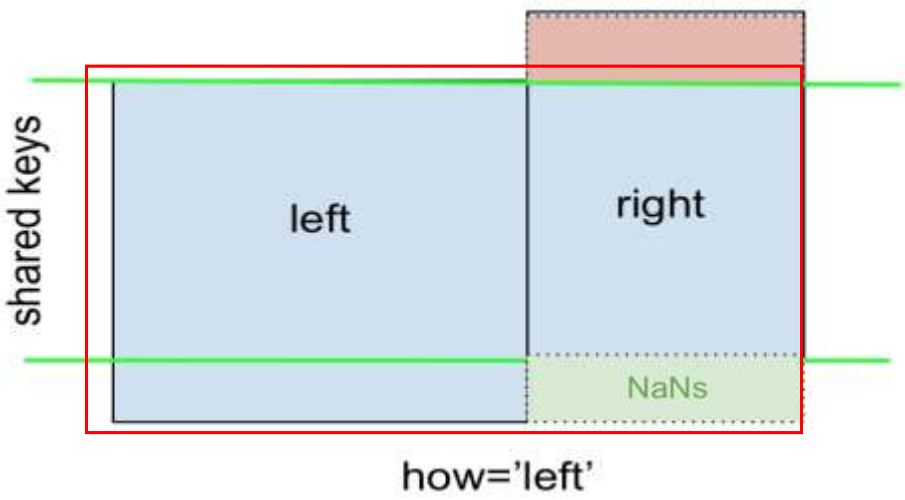
Let us first create two dataframes with different indexes and columns

Dataframe



- Merging Dataframes

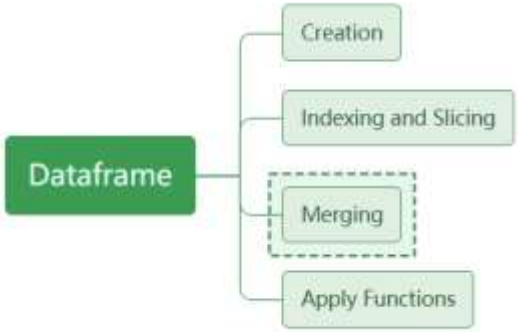
Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames



- Blue indicates rows that are present in the merge result
- Red indicates rows that are excluded from the result (i.e., removed)
- Green indicates missing values that are replaced with NaNs

Follows the **left** dataframe to determine
the resulting keys, fill NaNs to the **right** dataframe

Dataframe



- Merging Dataframes

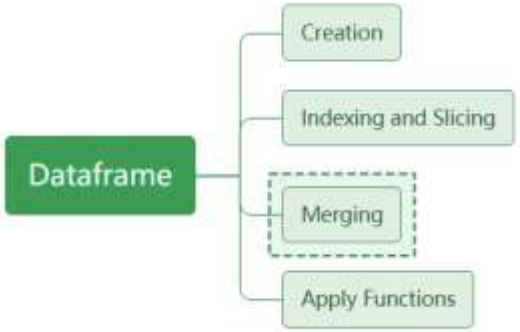
Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames

```
left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [4, 2]})
right = pd.DataFrame({'key': ['bar', 'zoo'], 'rval': [4, 5]})
merged = pd.merge(left, right, how="left")
print("left: ", left, "right: ", right, "left join: ", merged, sep=end_string)
```

```
left:
--------------------------------------------------
   key  lval
0  foo     4
1  bar     2
--------------------------------------------------
right:
--------------------------------------------------
   key  rval
0  bar     4
1  zoo     5
--------------------------------------------------
left join:
--------------------------------------------------
   key  lval  rval
0  foo     4   NaN
1  bar     2   4.0
```

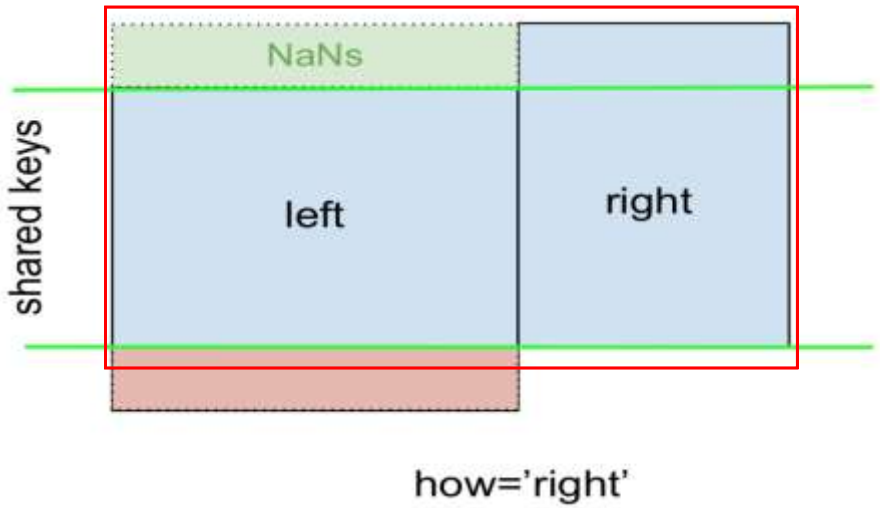
Check the left join of the dataframes

Dataframe



- Merging Dataframes

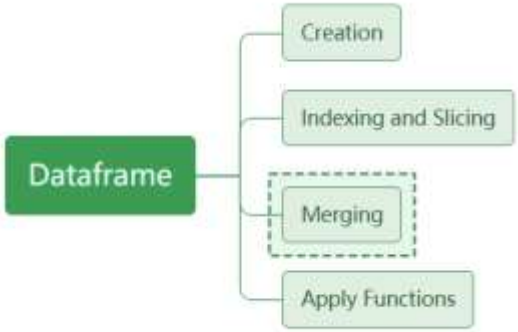
Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames



- Blue indicates rows that are present in the merge result
- Red indicates rows that are excluded from the result (i.e., removed)
- Green indicates missing values that are replaced with NaNs

Follows the **right** dataframe to determine the resulting keys, fill NaNs to the **left** dataframe

Dataframe



- Merging Dataframes

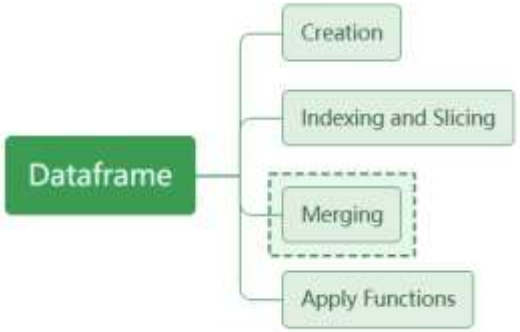
Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames

```
left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [4, 2]})
right = pd.DataFrame({'key': ['bar', 'zoo'], 'rval': [4, 5]})
merged = pd.merge(left, right, how="right")
print("left: ", left, "right: ", right, "right join: ", merged, sep=end_string)
```

```
left:
------------------------------------------------------------
   key  lval
0  foo     4
1  bar     2
------------------------------------------------------------
right:
------------------------------------------------------------
   key  rval
0  bar     4
1  zoo     5
------------------------------------------------------------
right join:
------------------------------------------------------------
   key  lval  rval
0  bar   2.0     4
1  zoo   NaN     5
```

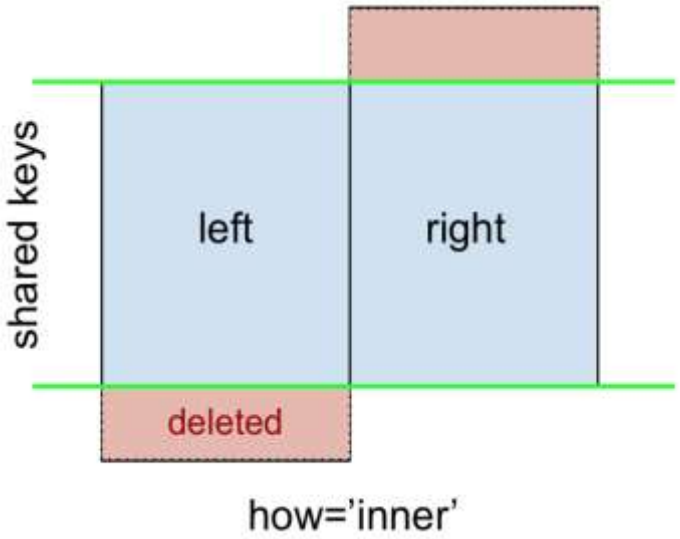
Check the right join of the dataframes

Dataframe



- Merging Dataframes

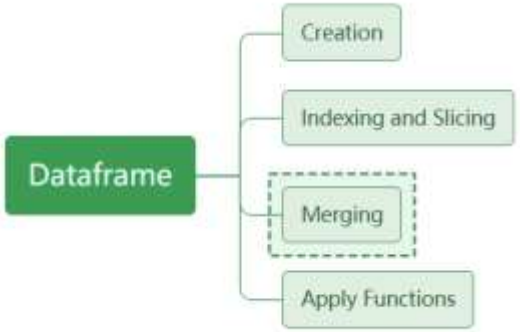
Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames



- Blue indicates rows that are present in the merge result
- Red indicates rows that are excluded from the result (i.e., removed)
- Green indicates missing values that are replaced with NaNs

Uses the **union** of **left & right** dataframe to determine the resulting keys, fill NaNs to the **missing elements**

Dataframe



- Merging Dataframes

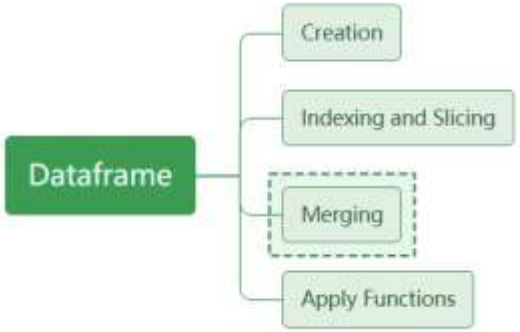
Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames

```
left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [4, 2]})
right = pd.DataFrame({'key': ['bar', 'zoo'], 'rval': [4, 5]})
merged = pd.merge(left, right, how="outer")
print("left: ", left, "right: ", right, "outer join: ", merged, sep=end_string)
```

```
left:
------------------------------------------------------------
   key  lval
0  foo     4
1  bar     2
------------------------------------------------------------
right:
------------------------------------------------------------
   key  rval
0  bar     4
1  zoo     5
------------------------------------------------------------
outer join:
------------------------------------------------------------
   key  lval  rval
0  foo   4.0   NaN
1  bar   2.0   4.0
2  zoo   NaN   5.0
```

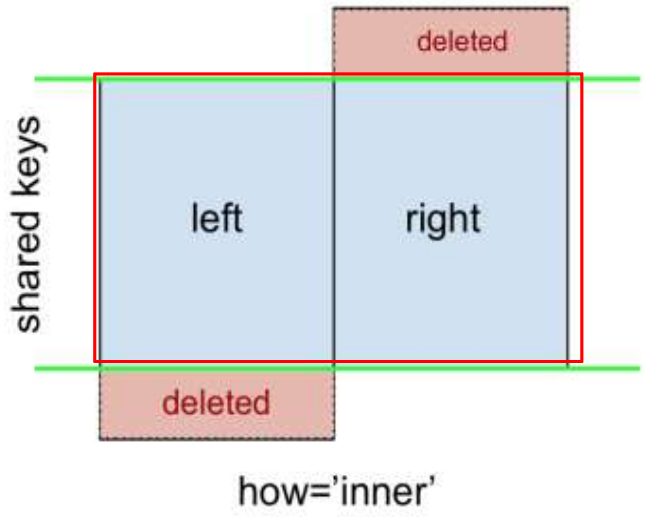
Check the outer join of the dataframes

Dataframe



- Merging Dataframes

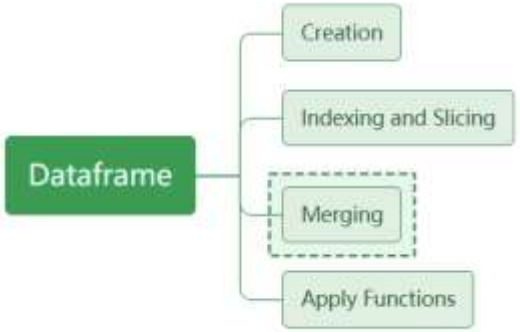
Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames



- Blue indicates rows that are present in the merge result
- Red indicates rows that are excluded from the result (i.e., removed)
- Green indicates missing values that are replaced with NaNs

Uses the **intersection** of **left & right** dataframe to determine the resulting keys, **deleting** the **other elements**

Dataframe



- Merging Dataframes

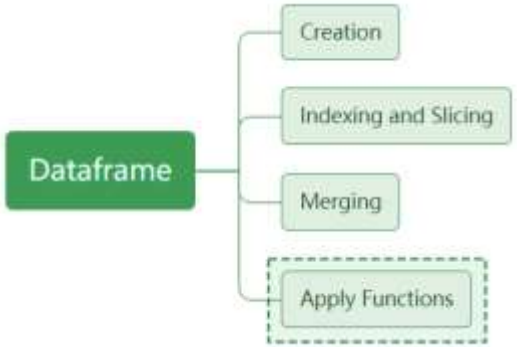
Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames

```
left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [4, 2]})
right = pd.DataFrame({'key': ['bar', 'zoo'], 'rval': [4, 5]})
merged = pd.merge(left, right, how="inner")
print("left: ", left, "right: ", right, "inner join: ", merged, sep=end_string)
```

```
left:
------------------------------------------------
   key  lval
0  foo     4
1  bar     2
------------------------------------------------
right:
------------------------------------------------
   key  rval
0  bar     4
1  zoo     5
------------------------------------------------
inner join:
------------------------------------------------
   key  lval  rval
0  bar     2     4
```

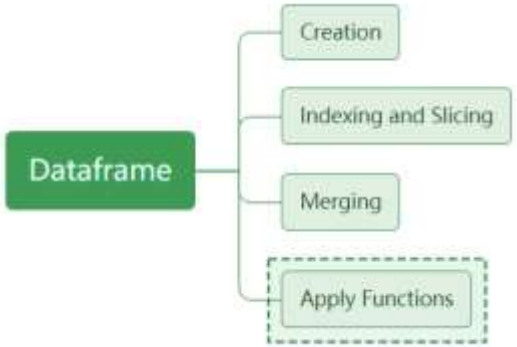
Check the inner join of the dataframes

Dataframe



- Apply functions
  - Pandas provides the interface to apply specfic functions on the dataframe
    - row-wise / column-wise **df.apply(func, axis = 0)**
    - element-wise **df.applymap(func)**

Dataframe



- Apply functions
  - row-wise / column-wise **df.apply(func, axis = 0)**
  - element-wise **df.applymap(func)**

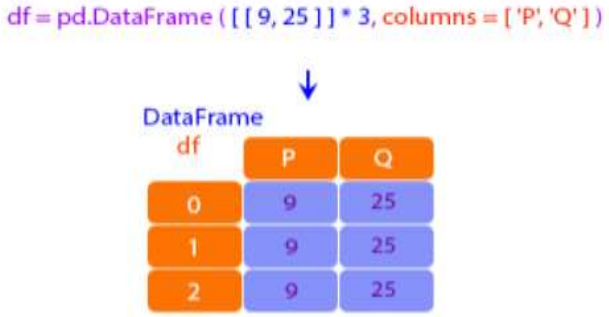
```
In [1]: import numpy as np
        import pandas as pd

In [2]: df = pd.DataFrame([[9, 25]] * 3, columns=['P', 'Q'])
        df
```

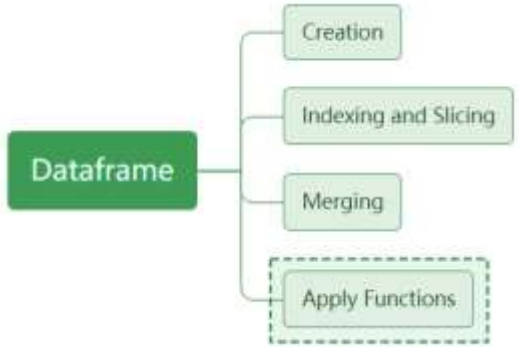
Out[2]:

	P	Q
0	9	25
1	9	25
2	9	25

Let us first create a dataframe



Dataframe



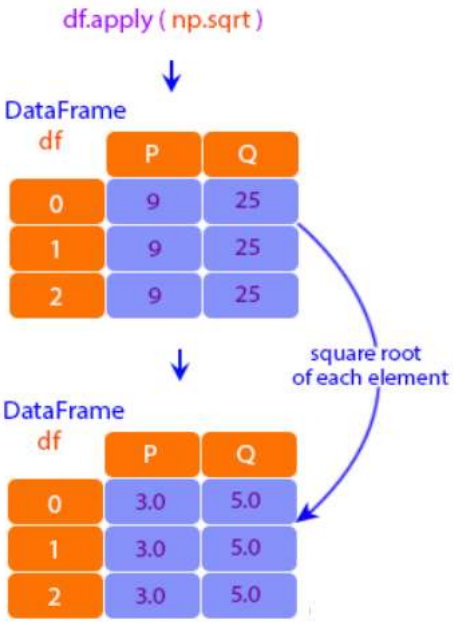
- Apply functions
  - row-wise / column-wise **df.apply(func, axis = 0)**
  - element-wise **df.applymap(func)**

Using a numpy universal function (in this case the same as np.sqrt(df)):

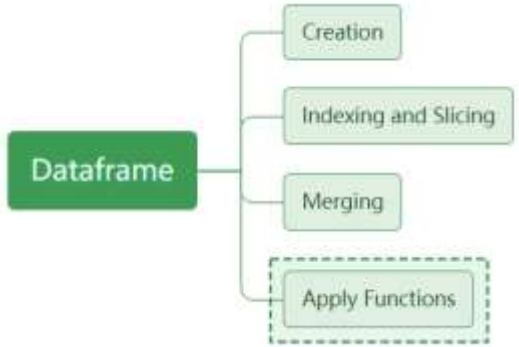
```
In  [3]: df.apply(np.sqrt)
```

Out[3]:

	P	Q
0	3.0	5.0
1	3.0	5.0
2	3.0	5.0



Dataframe

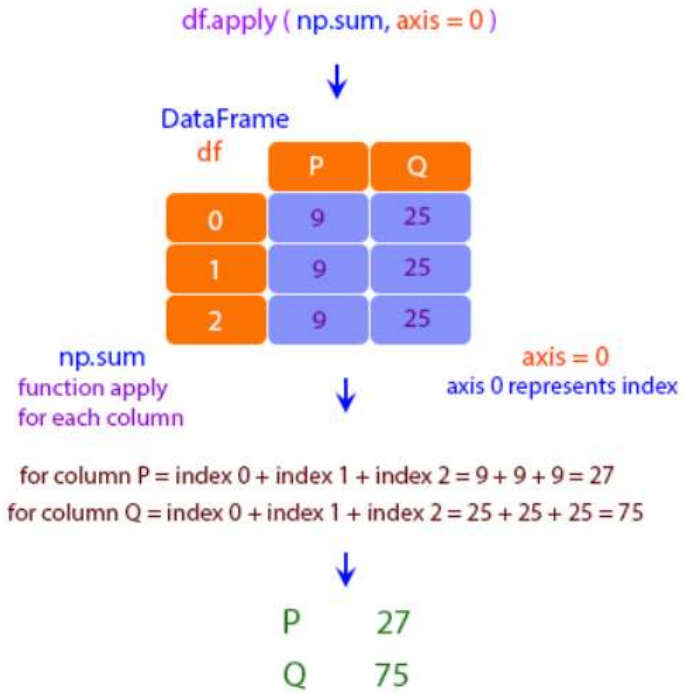


- Apply functions
  - row-wise / column-wise **df.apply(func, axis = 0)**
  - element-wise **df.applymap(func)**

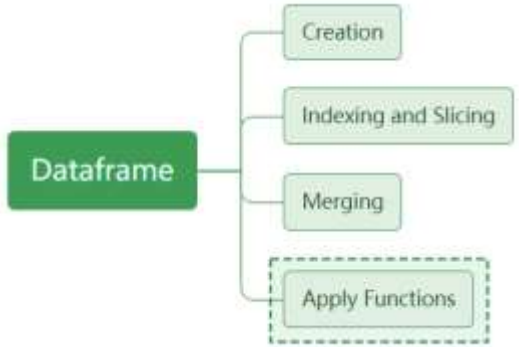
Using a reducing function on either axis

```
In  [4]: df.apply(np.sum, axis=0)

Out[4]: P    27
        Q    75
        dtype: int64
```



Dataframe

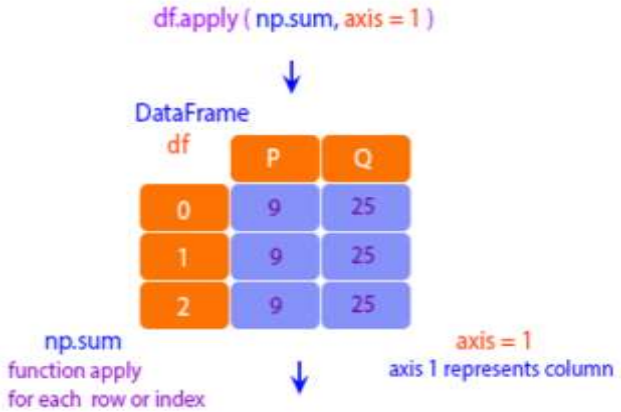


- Apply functions
  - row-wise / column-wise **df.apply(func, axis = 0)**
  - element-wise **df.applymap(func)**

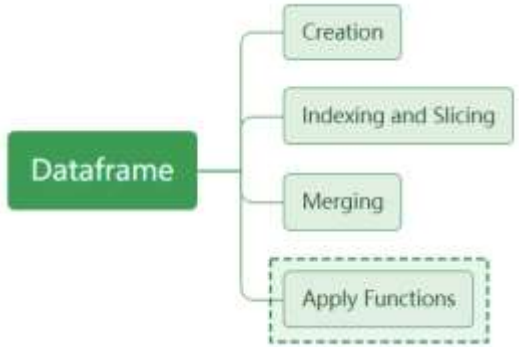
Using a reducing function on either axis

```
In  [5]: df.apply(np.sum, axis=1)

Out[5]: 0    34
        1    34
        2    34
        dtype: int64
```



Dataframe



- Apply functions
  - row-wise / column-wise **df.apply(func, axis = 0)**
  - element-wise **df.applymap(func)**

lambda functions

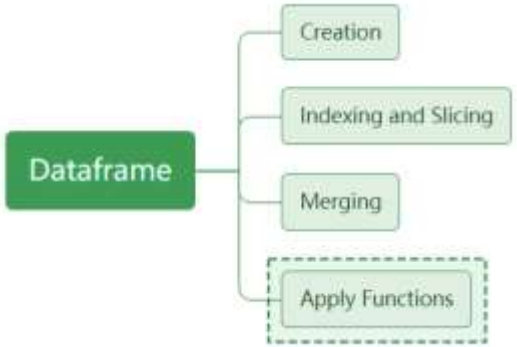
lambda functions allow you to specify a function without giving it a separate declaration.

```
lambda  x:  (x  -  x.mean())/x.std()
```

is equivalent to the function

```
def  normalize(x):
        return  (x  -  x.mean())/x.std()
```

Dataframe



- Apply functions
  - row-wise / column-wise **df.apply(func, axis = 0)**
  - element-wise **df.applymap(func)**

lambda functions

lambda functions allow you to specify a function without giving it a separate declaration.

```
df1 = pd.DataFrame(np.random.randn(6, 4), index=list(range(0, 12, 2)), columns=list('abcd'))
df2 = df1.apply(lambda x: (x - x.mean()) / x.std(), axis = 0)
df3 = df1.apply(lambda x: (x - x.mean()) / x.std(), axis = 1)
print("df1: ", df1, "df2: ", df2, "df3: ", df3, sep=end_string)
```

```
df1:
----------
          a         b         c         d
0  0.857782 -0.042206 -1.716331  0.274145
2 -1.449939  0.114080  0.593545  0.751275
4  0.786573  0.211597  0.251546  0.304242
6 -0.199261 -1.385909 -1.008863 -0.257183
8  1.064075 -0.127475  0.352189  0.260657
10 0.055163  0.374692  1.531703  0.892779
----------
df2:
----------
          a         b         c         d
0  0.714394  0.158043 -1.466973 -0.236620
2 -1.738731  0.404228  0.506585  0.929200
4  0.638699  0.557839  0.214382 -0.163083
6 -0.409251 -1.958586 -0.862512 -1.534870
8  0.933685  0.023726  0.300371 -0.269579
10-0.138796  0.814749  1.308148  1.274952
----------
df3:
----------
          a         b         c         d
0  0.918381  0.103610 -1.411997  0.390007
2 -1.444489  0.111248  0.588174  0.745068
4  1.484902 -0.659833 -0.510815 -0.314255
6  0.884176 -1.158898 -0.509730  0.784452
8  1.362275 -1.036405 -0.070804 -0.255066
10-1.020105 -0.525053  1.267528  0.277630
```

Dataframe **describe**

```python
import pandas as pd

# Sample DataFrame
df = pd.DataFrame({
    'A': [1, 2, 3, 4, 5],
    'B': [10, 20, 30, 40, 50],
    'C': ['foo', 'bar', 'foo', 'bar', 'foo']
})

# Generate descriptive statistics
summary = df.describe()
print(summary)
```

```
              A          B
count  5.000000   5.000000
mean   3.000000  30.000000
std    1.581139  15.811388
min    1.000000  10.000000
25%    2.000000  20.000000
50%    3.000000  30.000000
75%    4.000000  40.000000
max    5.000000  50.000000
```

df.describe() function in pandas generates descriptive statistics for a DataFrame, providing insights into the central tendency, dispersion, and shape of the dataset's distribution. By default, it analyzes numeric columns and returns statistics such as count, mean, standard deviation, minimum, maximum, and specific percentiles (**25th**, **50th**, and **75th**).

Dataframe replace

DataFrame.replace() method is a versatile function used to replace specified values within a DataFrame. This method allows for various types of replacements, including single values, lists, dictionaries, and even regular expressions.

```
import pandas as pd
import numpy as np

# Create a sample DataFrame
df = pd.DataFrame({
    'A': [1, 2, np.nan, 4],
    'B': [5, np.nan, 7, 8]
})

# Replace NaN with 0
df_replaced = df.replace(np.nan, 0)
print(df_replaced)
```

```
css

   A  B
0  1  5
1  2  0
2  0  7
3  4  8
```

We can use replace to identify the missing values.

Dataframe **groupby()**

groupby() function is a powerful tool for grouping data based on one or more keys, allowing for subsequent aggregation, transformation, or filtration operations on these groups. This method is essential for analyzing and summarizing large datasets.

```
import pandas as pd

# Sample DataFrame
df = pd.DataFrame({
    'Category': ['A', 'B', 'A', 'B', 'A'],
    'Values': [10, 20, 30, 40, 50]
})

# Group by 'Category' and calculate the sum of 'Values' for each group
grouped = df.groupby('Category')['Values'].sum()
print(grouped)
```

```
Category
A    90
B    60
Name: Values, dtype: int64
```

Data structures in Pandas

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

Pandas I/O Functions

- Pandas can load dataframe data from
  - csv/excel files
  - table in a webpage

```
import pandas as pd
iris_data = pd.read_csv('https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/639388c2cbc2120a14dcf466e85730eb8be498bb/iris.csv')
iris_data
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
...	...	...	...	...	...
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

'https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/639388c2cbc2120a14dcf466e85730eb8be498bb/iris.csv'

Pandas I/O Functions

- Pandas can load dataframe data from
  - csv/excel files: read_csv/read_excel
  - table in a webpage

What is the content of the csv file?

```
import pandas as pd
iris_data = pd.read_csv('https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/639388c2cbc2120a14dcf466e85730eb8be498bb/iris.csv')
iris_data
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
...	...	...	...	...	...
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

	A	B	C	D	E
1	sepal_length	sepal_width	petal_length	petal_width	species
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	4.6	3.1	1.5	0.2	setosa
6	5	3.6	1.4	0.2	setosa
7	5.4	3.9	1.7	0.4	setosa
8	4.6	3.4	1.4	0.3	setosa
9	5	3.4	1.5	0.2	setosa
10	4.4	2.9	1.4	0.2	setosa
11	4.9	3.1	1.5	0.1	setosa
12	5.4	3.7	1.5	0.2	setosa
13	4.8	3.4	1.6	0.2	setosa
14	4.8	3	1.4	0.1	setosa
15	4.3	3	1.1	0.1	setosa
16	5.8	4	1.2	0.2	setosa

Pandas I/O Functions

- Pandas can load dataframe data from
  - csv/excel files: read_csv/read_excel
  - table in a webpage

```
import pandas as pd
iris_data = pd.read_excel('./iris.xlsx')
iris_data
```

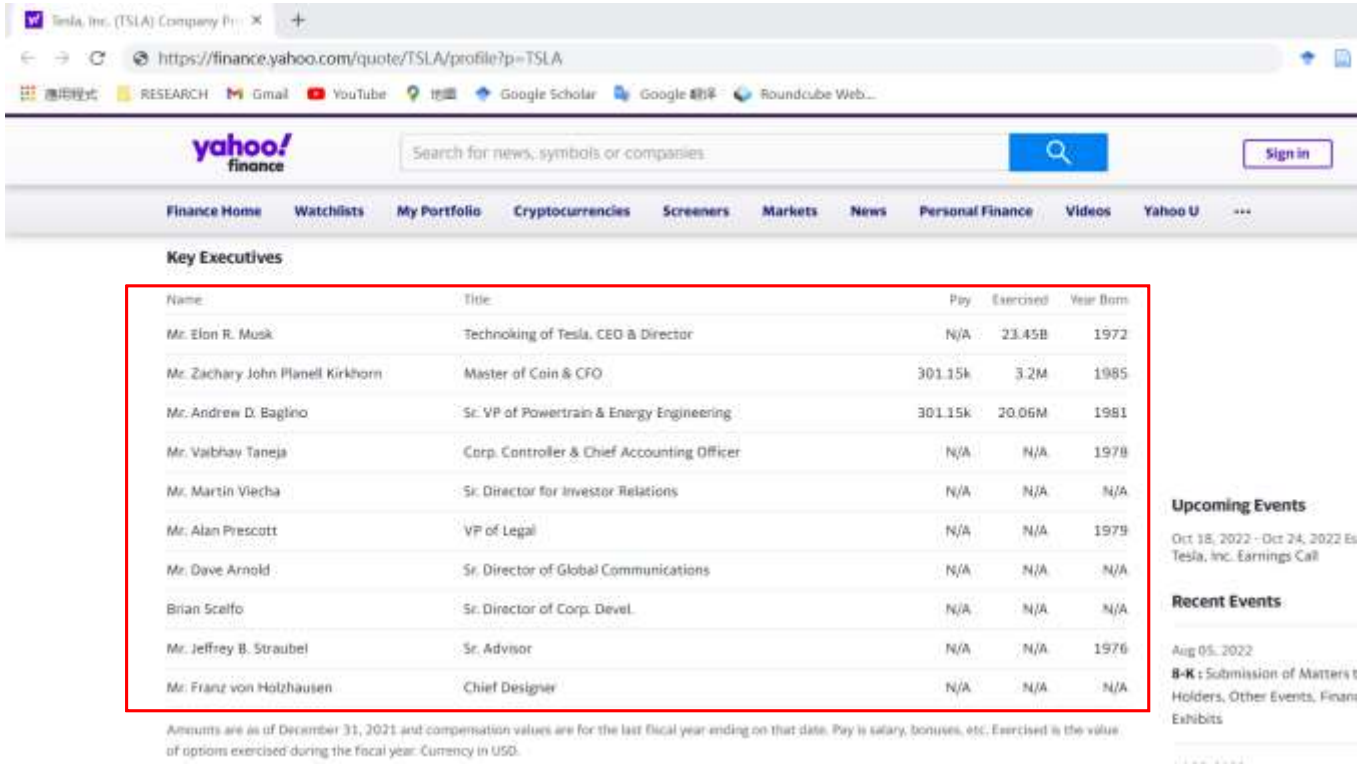
	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

Now let us save the csv file to the xlsx format, and read the data again using pandas

Pandas I/O Functions

- Pandas can load dataframe data from
  - csv/excel files: read_csv/read_excel
  - table in a webpage



Sometimes you see information online like this.

How to import into the pandas dataframe?

Pandas I/O Functions

- Pandas can load dataframe data from
  - csv/excel files: read_csv/read_excel
  - table in a webpage (read_html)

```
import pandas as pd
import requests
url_link = 'https://finance.yahoo.com/quote/TSLA/profile?p=TSLA'
r = requests.get(url_link,headers ={'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'})
data = pd.read_html(r.text)
data
```

```
[                          Name  \
0                Mr. Elon R. Musk
1  Mr. Zachary John Planell Kirkhorn
2             Mr. Andrew D. Baglino
3                Mr. Vaibhav Taneja
4                Mr. Martin Viecha
5                Mr. Alan Prescott
6                 Mr. Dave Arnold
7                    Brian Scelfo
8           Mr. Jeffrey B. Straubel
9          Mr. Franz von Holzhausen

                                Title      Pay Exercised  Year Born
0          Technoking of Tesla, CEO & Director      NaN    23.45B     1972.0
1                        Master of Coin & CFO  301.15k     3.2M     1985.0
2    Sr. VP of Powertrain & Energy Engineering  301.15k    20.06M     1981.0
3  Corp. Controller & Chief Accounting Officer      NaN       NaN     1978.0
4          Sr. Director for Investor Relations      NaN       NaN        NaN
5                                VP of Legal      NaN       NaN     1979.0
6        Sr. Director of Global Communications      NaN       NaN        NaN
7                  Sr. Director of Corp. Devel.     NaN       NaN        NaN
8                                 Sr. Advisor      NaN       NaN     1976.0
9                              Chief Designer      NaN       NaN        NaN ]
```

Pandas I/O Functions

- Pandas can load dataframe data from
  - csv/excel files: read_csv/read_excel
  - table in a webpage (read_html)

```
import pandas as pd
import requests
url_link = 'https://finance.yahoo.com/quote/TSLA/profile?p=TSLA'
r = requests.get(url_link,headers ={'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'})
data = pd.read_html(r.text)
data[0]
```

	Name	Title	Pay	Exercised	Year Born
0	Mr. Elon R. Musk	Technoking of Tesla, CEO & Director	NaN	23.45B	1972.0
1	Mr. Zachary John Planell Kirkhorn	Master of Coin & CFO	301.15k	3.2M	1985.0
2	Mr. Andrew D. Baglino	Sr. VP of Powertrain & Energy Engineering	301.15k	20.06M	1981.0
3	Mr. Vaibhav Taneja	Corp. Controller & Chief Accounting Officer	NaN	NaN	1978.0
4	Mr. Martin Viecha	Sr. Director for Investor Relations	NaN	NaN	NaN
5	Mr. Alan Prescott	VP of Legal	NaN	NaN	1979.0
6	Mr. Dave Arnold	Sr. Director of Global Communications	NaN	NaN	NaN
7	Brian Scelfo	Sr. Director of Corp. Devel.	NaN	NaN	NaN
8	Mr. Jeffrey B. Straubel	Sr. Advisor	NaN	NaN	1976.0
9	Mr. Franz von Holzhausen	Chief Designer	NaN	NaN	NaN

Pandas I/O Functions

- Pandas can load dataframe data from
  - csv/excel files: read_csv/read_excel

```
import pandas as pd
import requests
url_link = 'https://finance.yahoo.com/quote/TSLA/profile?p=TSLA'
r = requests.get(url_link,headers ={'User-Agent':'Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
Safari/537.36'})
data = pd.read_html(r.text)
data[0]
data[0].to_csv('tsla.csv')
data[0].to_csv('tsla2.csv',index=False)
data[0].to_excel('tsla.xlsx')
data[0].to_excel('tsla2.xlsx',index=False)
```