# COMP7035

## Python for Data Analytics and Artificial Intelligence

## Pandas-2

Renjie Wan, Wei Xue

11/7/2022

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

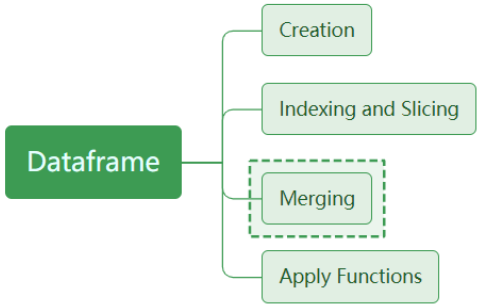
# What we will learn?

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

# Pandas Overview

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

# Dataframe

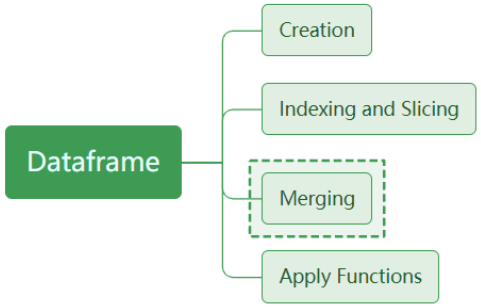



- Merging Dataframes
  - Pandas provides powerful operations for combining dataframes
  - Mainly four operations for merging, indicated by “how”

```
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

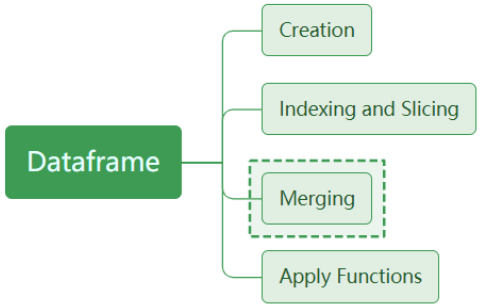
| Merge method | SQL Join Name | Description |
|---|---|---|
| left | LEFT OUTER JOIN | Use keys from left frame only |
| right | RIGHT OUTER JOIN | Use keys from right frame only |
| outer | FULL OUTER JOIN | Use union of keys from both frames |
| inner | INNER JOIN | Use intersection of keys from both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

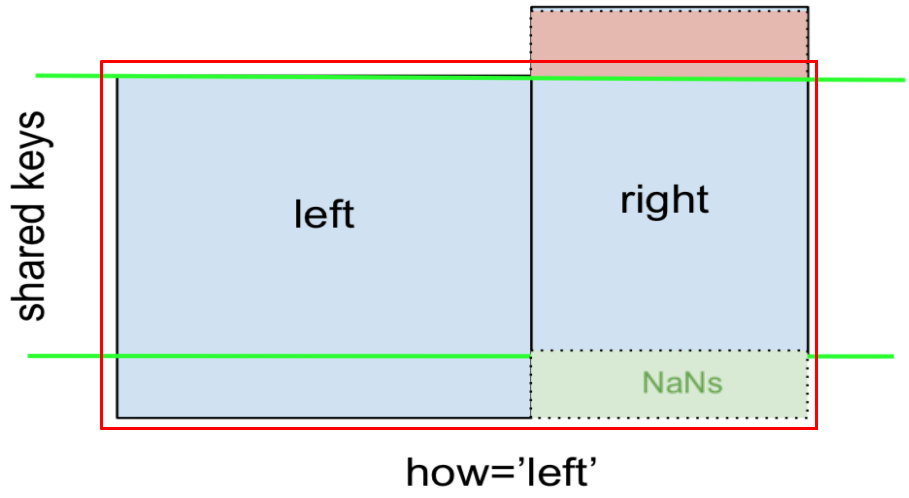
Let us first create two dataframes with different indexes and columns

# Dataframe




- Merging Dataframes

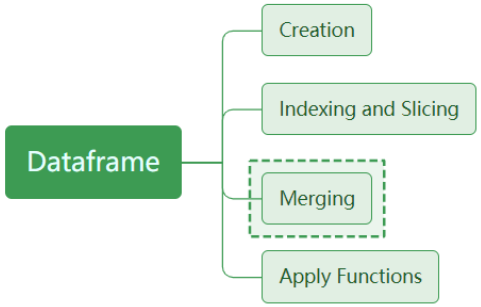
| Merge method | SQL Join Name | Description |
|---|---|---|
| left | LEFT OUTER JOIN | Use keys from left frame only |
| right | RIGHT OUTER JOIN | Use keys from right frame only |
| outer | FULL OUTER JOIN | Use union of keys from both frames |
| inner | INNER JOIN | Use intersection of keys from both 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 merge: ",merged, sep=end_string)
```

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

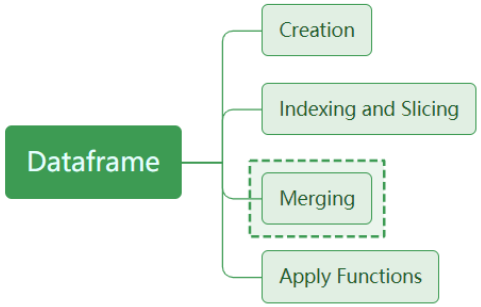
Check the left join of the dataframes

# DataFrame-02

- Exercise

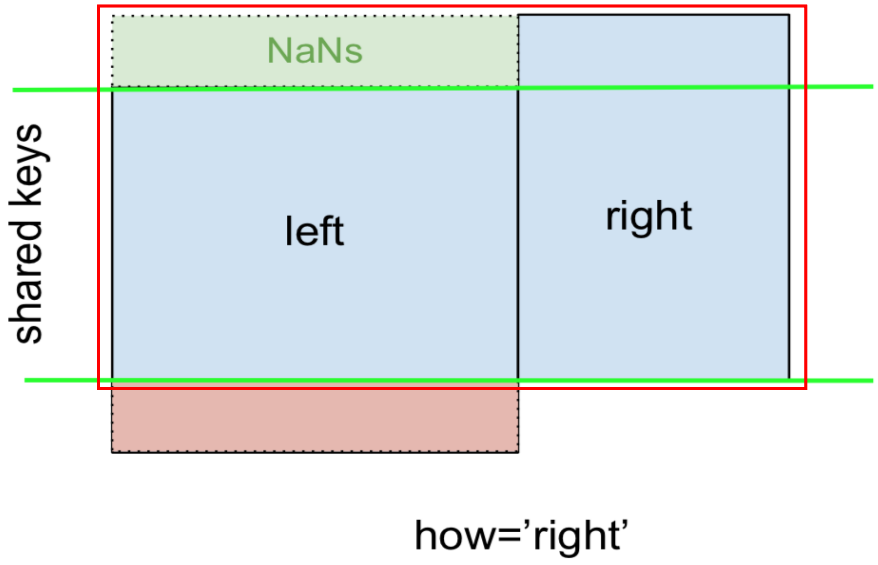
1. Write codes to create two DataFrames, with the columns as “[key, lval1, lval2]” and “[key, rval1, rval2]”, and the values are “[a,b,c]”, and “[b,c,d]” respectively. Generate random numbers with normal distribution to for the “lval*” and “rval*” elements.
2. Compute the left outer join of the two DataFrames, check out the results
3. Change the name “key” of the left DataFrame to “key_left”, re-run step 2 and see what happens

# Dataframe




- Merging Dataframes

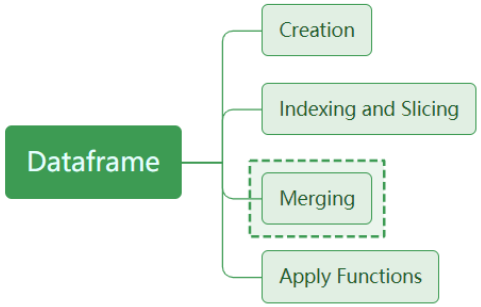
| Merge method | SQL Join Name | Description |
|---|---|---|
| left | LEFT OUTER JOIN | Use keys from left frame only |
| right | RIGHT OUTER JOIN | Use keys from right frame only |
| outer | FULL OUTER JOIN | Use union of keys from both frames |
| inner | INNER JOIN | Use intersection of keys from both 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-02

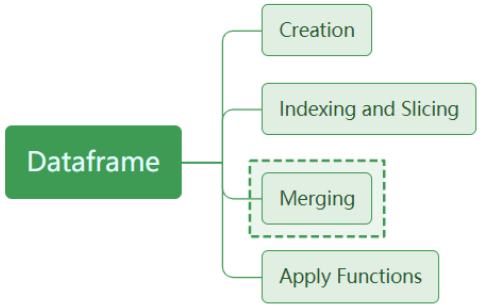
- Exercise

1. Write codes to create two DataFrames, with the columns as “[key, lval1, lval2]” and “[key, rval1, rval2]”, and the key values are “[a,b,c]”, and “[b,c,d]” respectively. Generate random numbers with normal distribution to for the “lval*” and “rval*” elements.
2. Compute the left outer join of the two DataFrames, check out the results
3. Change the name “key” of the left DataFrame to “key_left”, re-run step 2 and see what happens

---

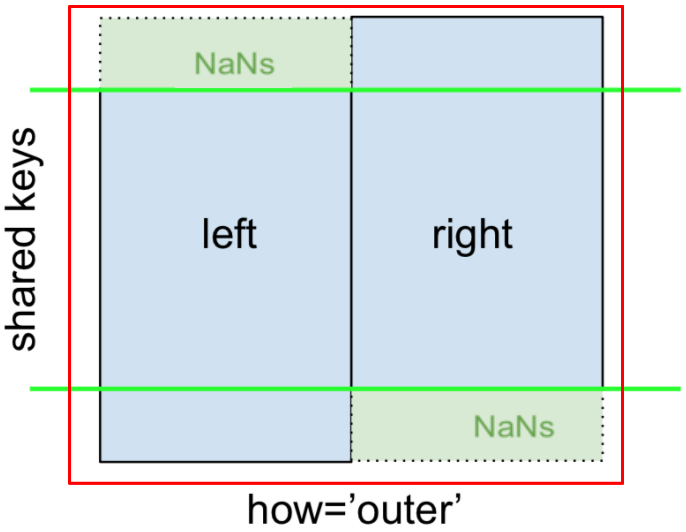
4. Compute the right outer join of the two DataFrames in step 2, check out the results

# Dataframe




- Merging Dataframes

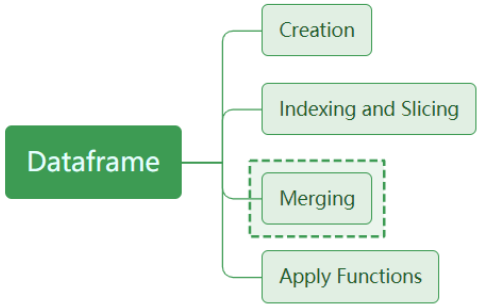
| Merge method | SQL Join Name | Description |
|---|---|---|
| left | LEFT OUTER JOIN | Use keys from left frame only |
| right | RIGHT OUTER JOIN | Use keys from right frame only |
| outer | FULL OUTER JOIN | Use union of keys from both frames |
| inner | INNER JOIN | Use intersection of keys from both 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-02

- Exercise

1. Write codes to create two DataFrames, with the columns as “[key, lval1, lval2]” and “[key, rval1, rval2]”, and the values are “[a,b,c]”, and “[b,c,d]” respectively. Generate random numbers with normal distribution to for the “lval*” and “rval*” elements.
2. Compute the left outer join of the two DataFrames, check out the results
3. Change the name “key” of the left DataFrame to “key_left”, re-run step 2 and see what happens

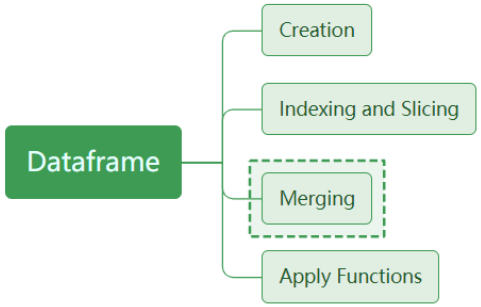
---

4. Compute the right outer join of the two DataFrames in step 2, check out the results

---

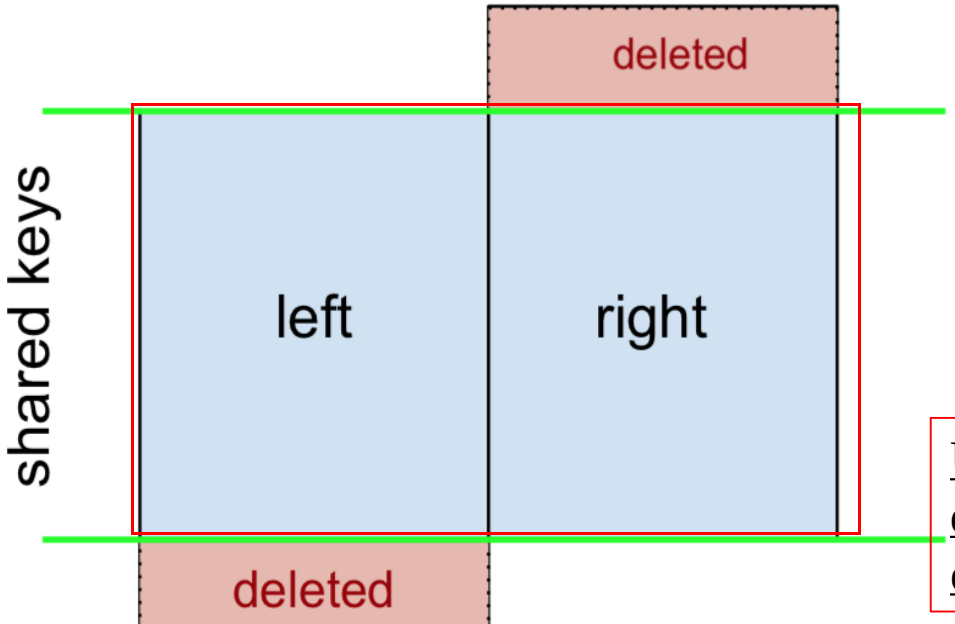
5. Compute the full outer join of the two DataFrames in step 2, check out the results

# Dataframe




- Merging Dataframes

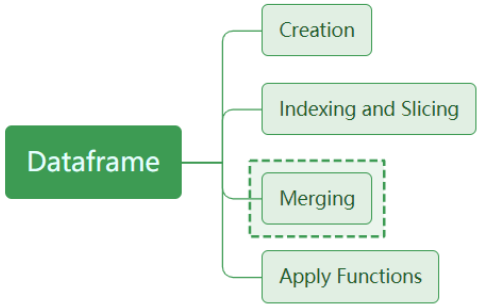
| Merge method | SQL Join Name | Description |
|---|---|---|
| left | LEFT OUTER JOIN | Use keys from left frame only |
| right | RIGHT OUTER JOIN | Use keys from right frame only |
| outer | FULL OUTER JOIN | Use union of keys from both frames |
| inner | INNER JOIN | Use intersection of keys from both 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-02

- Exercise

1. Write codes to create two DataFrames, with the columns as “[key, lval1, lval2]” and “[key, rval1, rval2]”, and the values are “[a,b,c]”, and “[b,c,d]” respectively. Generate random numbers with normal distribution to for the “lval*” and “rval*” elements.
2. Compute the left outer join of the two DataFrames, check out the results
3. Change the name “key” of the left DataFrame to “key_left”, re-run step 2 and see what happens

---

4. Compute the right outer join of the two DataFrames in step 2, check out the results

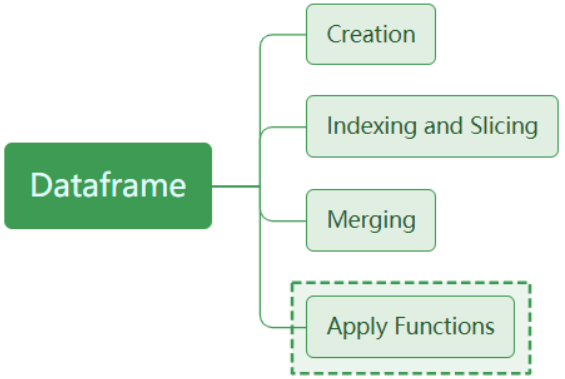
---

5. Compute the full outer join of the two DataFrames in step 2, check out the results

---

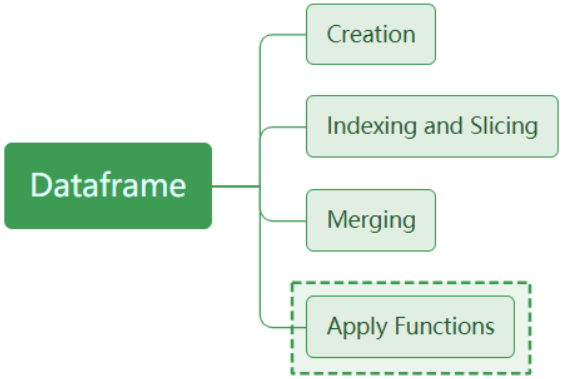
6. Compute the inner join of the two DataFrames in step 2, check out the results

# 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)**

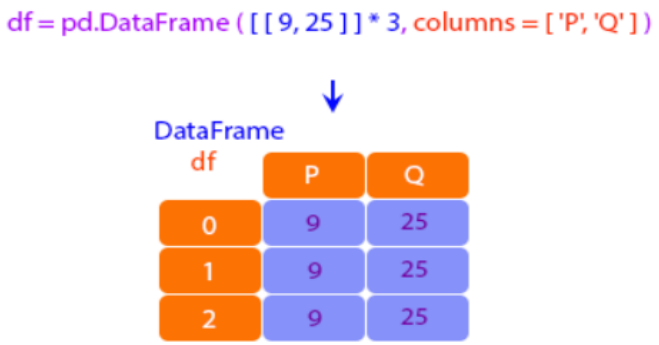
```
In [1]: import numpy as np
        import pandas as pd

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

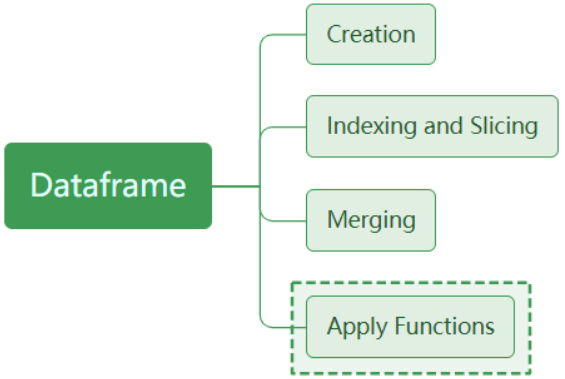
Out[2]:

| | P | Q |
|---|---|---|
| 0 | 9 | 25 |
| 1 | 9 | 25 |
| 2 | 9 | 25 |

Let us first create a dataframe



# Dataframe



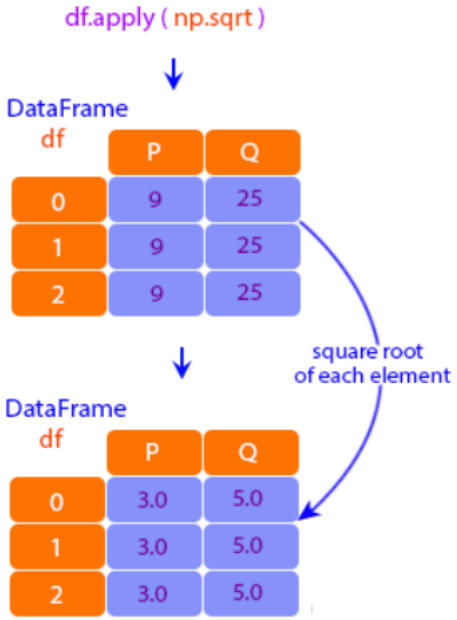

- Apply functions
  - row-wise / column-wise **df.apply(func, axis = 0)**
  - element-wise **df.applymap(func)**

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

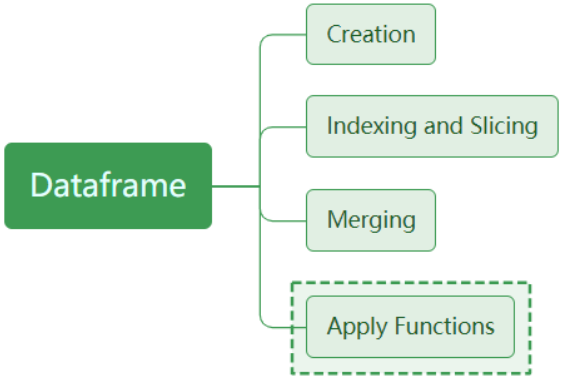
```
In [3]: df.apply(np.sqrt)
```

Out[3]:

| | P | Q |
|---|---|---|
| **0** | 3.0 | 5.0 |
| **1** | 3.0 | 5.0 |
| **2** | 3.0 | 5.0 |



# Dataframe

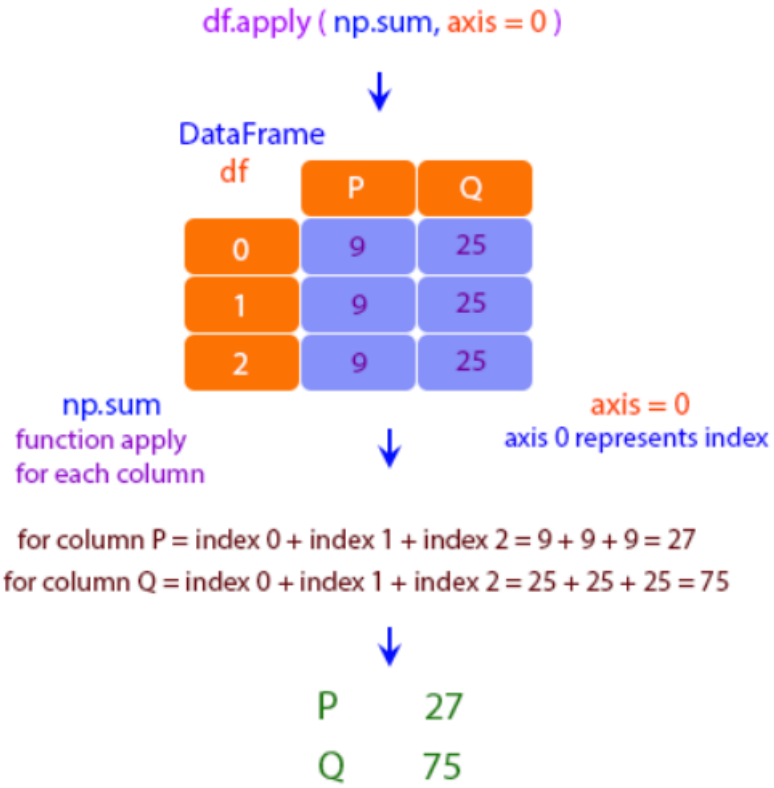



- Apply functions
  - row-wise / column-wise **df.apply(func, axis = 0)**
  - element-wise **df.applymap(func)**

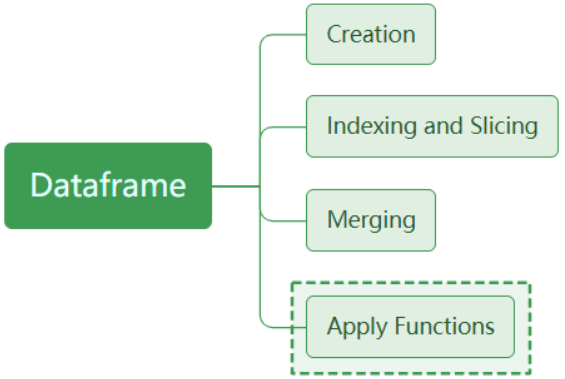
Using a reducing function on either axis

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

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



# Dataframe

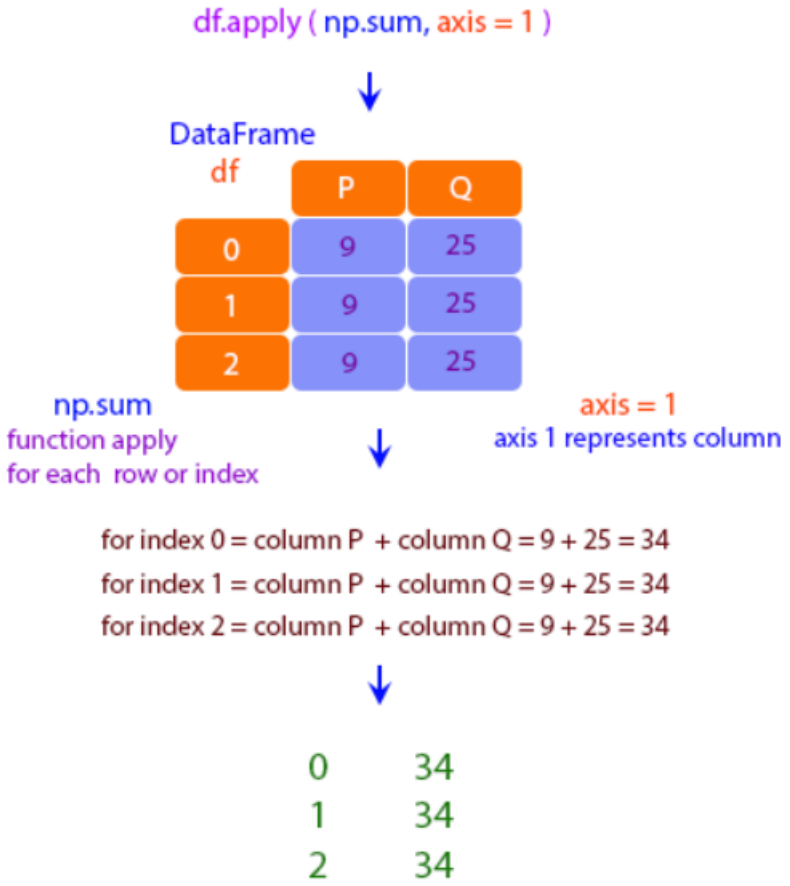



- Apply functions
  - row-wise / column-wise **df.apply(func, axis = 0)**
  - element-wise **df.applymap(func)**

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-02

- Exercise

1. Write codes to create two DataFrames df_left, df_right, with the columns as "[key, lval1, lval2]" and "[key, rval1, rval2]", and the values are "[a,b,c]", and "[b,c,d]" respectively. Generate random numbers with normal distribution to for the "lval*" and "rval*" elements.
2. Compute the left outer join of df_left and df_right, check out the results
3. Change the name "key" of df_left to "key_left", re-run step 2 and see what happens

---

4. Compute the right outer join of df_left and df_right in step 2, check out the results

---

5. Compute the full outer join of df_left and df_right in step 2, check out the results

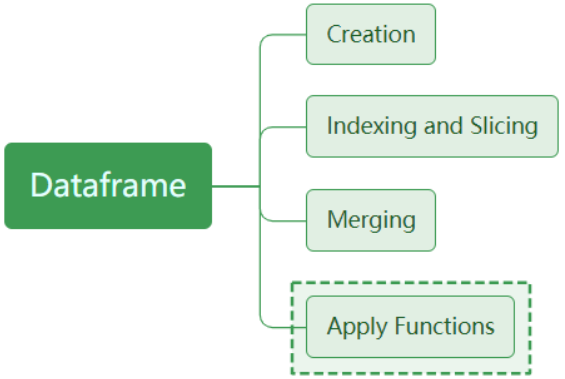
---

6. Compute the inner join of df_left and df_right in step 2, check out the results

---

7. Get the floating value columns of df_left (lval1,lval2), get the square root of the absolute values using apply
8. Try using numpy to directly calculate the above values on df_left

# Dataframe




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

lambda functions

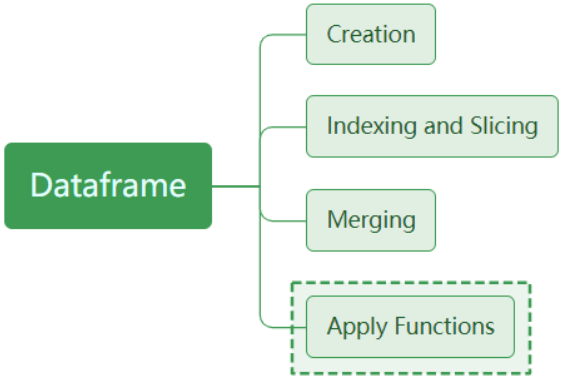
lambda functions allow you to specify a function without giving it a separate declaration.

```
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  -0.065156  0.180012 -0.623430 -0.315099
2  -0.391124 -1.333620 -0.530239  1.315869
4   1.760117 -1.542066  0.081087  0.807097
6   0.405075  0.341628 -0.051328  0.491693
8  -0.464656  0.043696 -0.880729 -1.857030
10 -0.916261 -0.458325  1.278605  0.698612
----------------------------------------------------
df2:
----------------------------------------------------
           a         b         c         d
0  -0.126913  0.796858 -0.648445 -0.445105
2  -0.472174 -1.083467 -0.528169  0.991601
4   1.806389 -1.342410  0.260827  0.543427
6   0.371148  0.997626  0.089929  0.265591
8  -0.550058  0.627517 -0.980523 -1.803379
10 -1.028391  0.003877  1.806381  0.447864
----------------------------------------------------
df3:
----------------------------------------------------
           a         b         c         d
0   0.409201  1.121914 -1.213724 -0.317392
2  -0.140333 -0.986298 -0.265199  1.391830
4   1.064400 -1.304798 -0.140244  0.380642
6   0.451139  0.186861 -1.449936  0.811937
8   0.403382  1.034290 -0.113000 -1.324673
10 -1.052682 -0.600857  1.112896  0.540643
```

# DataFrame-02

- Exercise

1. Write codes to create two DataFrames df_left, df_right, with the columns as "[key, lval1, lval2]" and "[key, rval1, rval2]", and the values are "[a,b,c]", and "[b,c,d]" respectively. Generate random numbers with normal distribution to for the "lval*" and "rval*" elements.
2. Compute the left outer join of df_left and df_right, check out the results
3. Change the name "key" of df_left to "key_left", re-run step 2 and see what happens

---

4. Compute the right outer join of df_left and df_right in step 2, check out the results

---

5. Compute the full outer join of df_left and df_right in step 2, check out the results

---

6. Compute the inner join of df_left and df_right in step 2, check out the results

---

7. Get the floating value columns of df_left (lval1,lval2), get the square root of the absolute values using apply
8. Try using numpy to directly calculate the above operations on df_left
9. Write the applymap functions to accomplish step 7

# Pandas Overview

- 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 |
| **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

```
'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 |
| **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

| | 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

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 |
| **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

| | 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
```

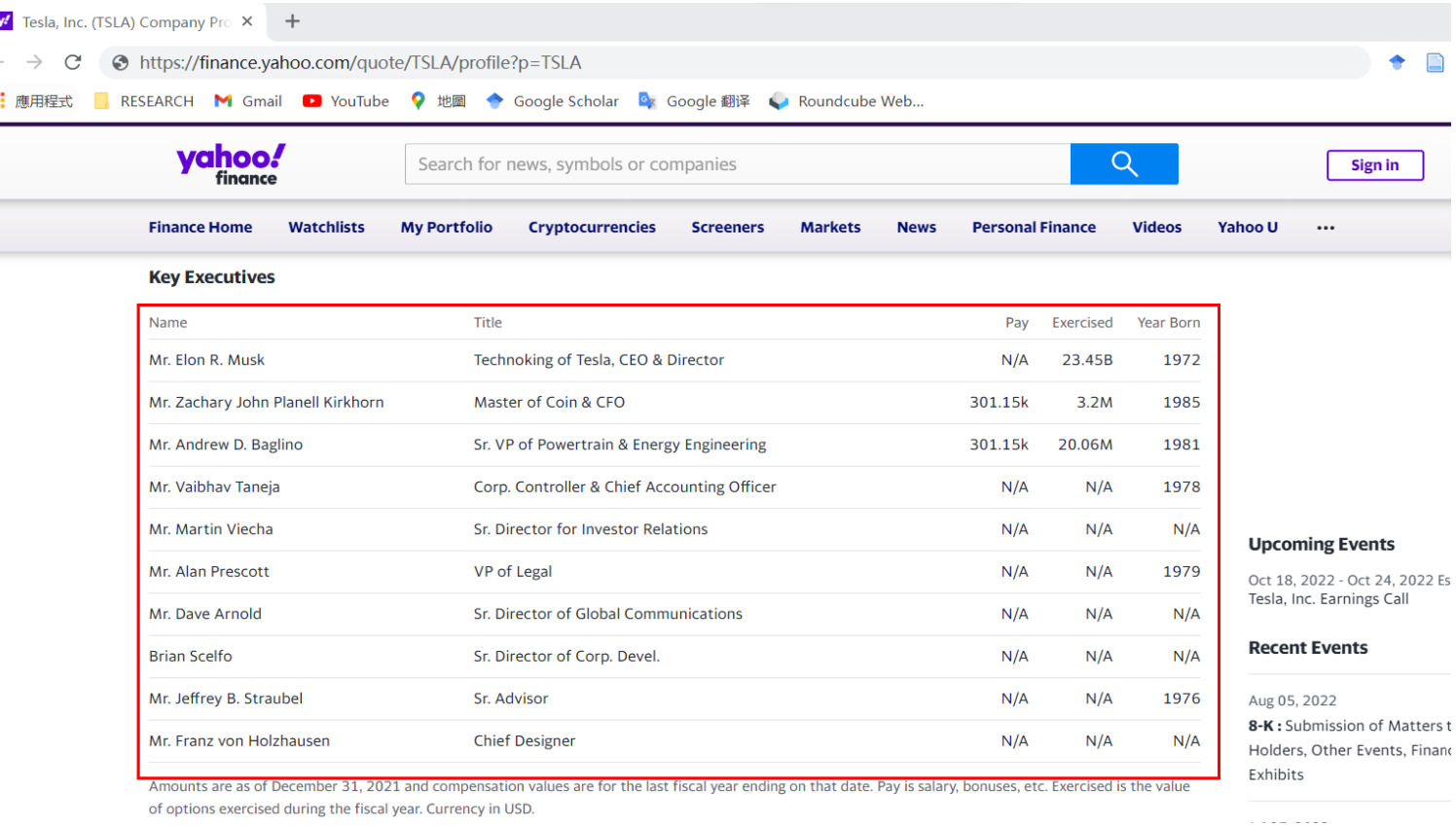
| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 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 write dataframe data into
  - csv/excel files: to_csv/to_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)
```

# DataFrame-02

- Exercise

1. Write codes to create two DataFrames df_left, df_right, with the columns as "[key, lval1, lval2]" and "[key, rval1, rval2]", and the values are "[a,b,c]", and "[b,c,d]" respectively. Generate random numbers with normal distribution to for the "lval*" and "rval*" elements.
2. Compute the left outer join of df_left and df_right, check out the results
3. Change the name "key" of df_left to "key_left", re-run step 2 and see what happens

---

4. Compute the right outer join of df_left and df_right in step 2, check out the results

---

5. Compute the full outer join of df_left and df_right in step 2, check out the results

---

6. Compute the inner join of df_left and df_right in step 2, check out the results

---

7. Get the floating value columns of df_left (lval1,lval2), get the square root of the absolute values using apply
8. Try using numpy to directly calculate the above operations on df_left
9. Write the apply_map functions to accomplish step 7
10. Get the data of "Countries and dependencies by area" from wiki and save to the excel excluding index.

# Recap of Today