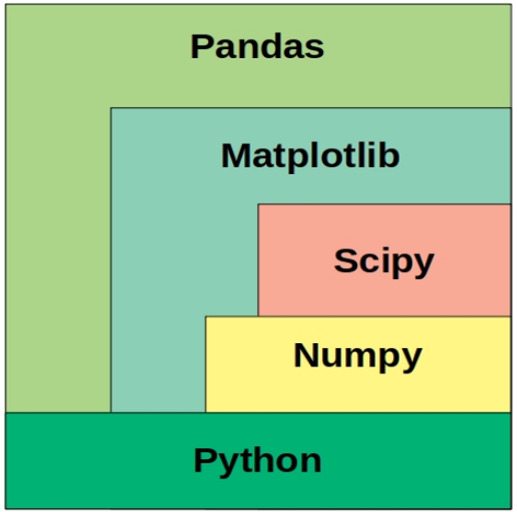
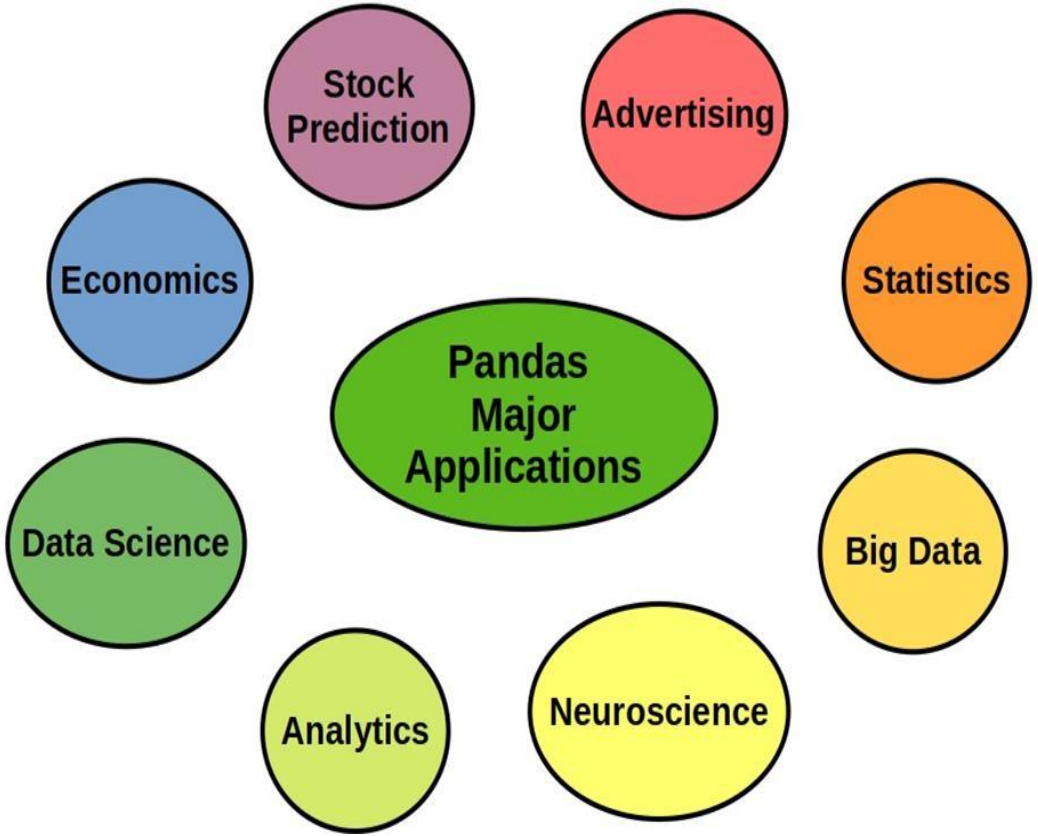
Chapter 6 - Part 3
Data Analysis with pandas

Content

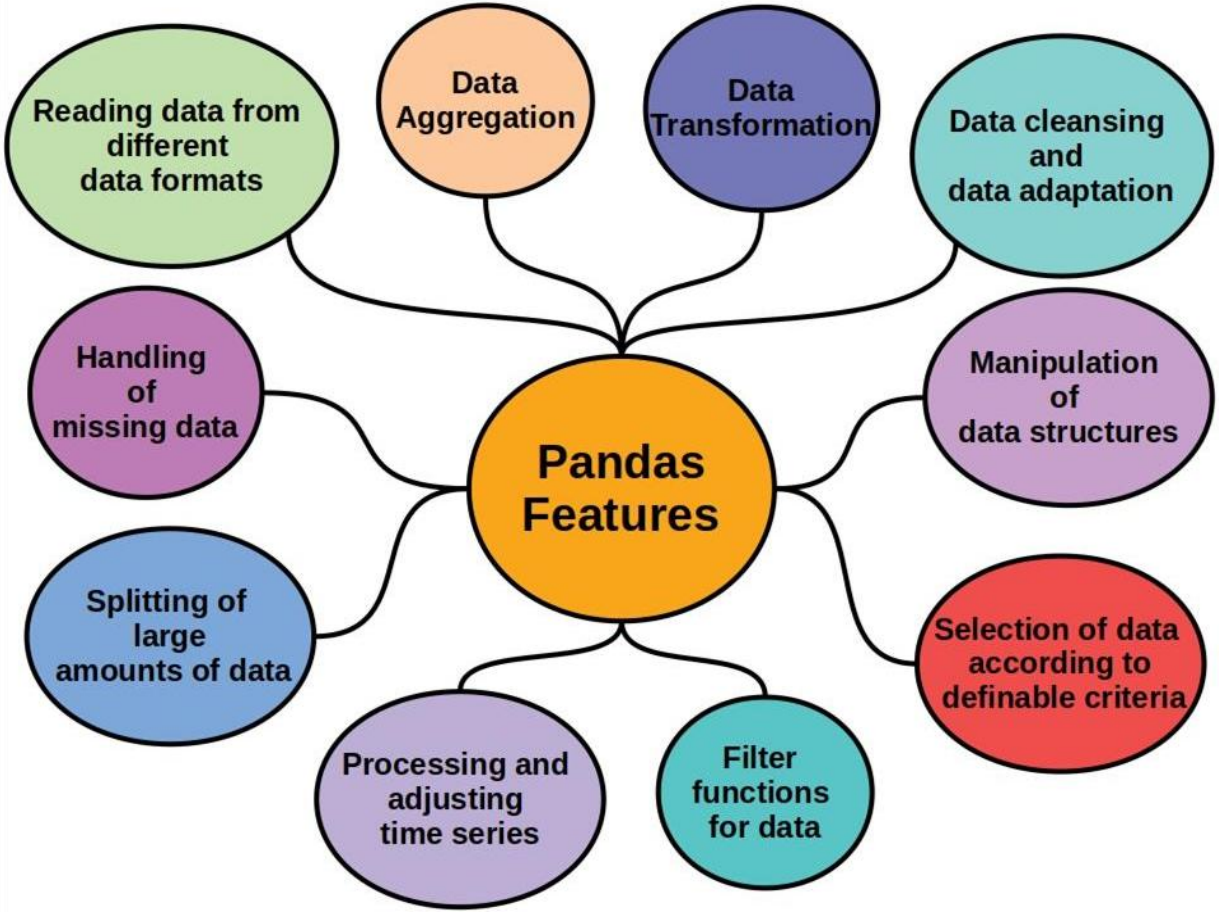
- 1. Introduction
- 2. Pandas Features
- 3. Pandas Getting Started With
- 4. Pandas Data Structure
- 5. Series in Pandas
- 6. Pandas DataFrame
- 7. Working with time series data

- In 2008, pandas development began at AQR Capital Management.
- By the end of 2009 it had been open sourced, and is actively supported today by a community.
- Pandas is an essential tool to data analysis and manipulation.

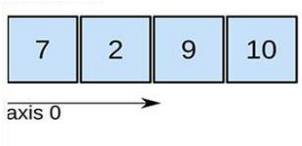
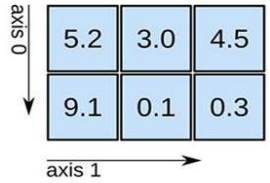
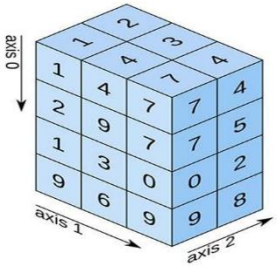



Stock Prediction
Advertising
Economics
Statistics
Pandas Major Applications
Data Science
Big Data
Analytics
Neuroscience

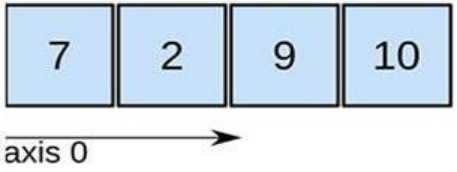
2. Pandas Features

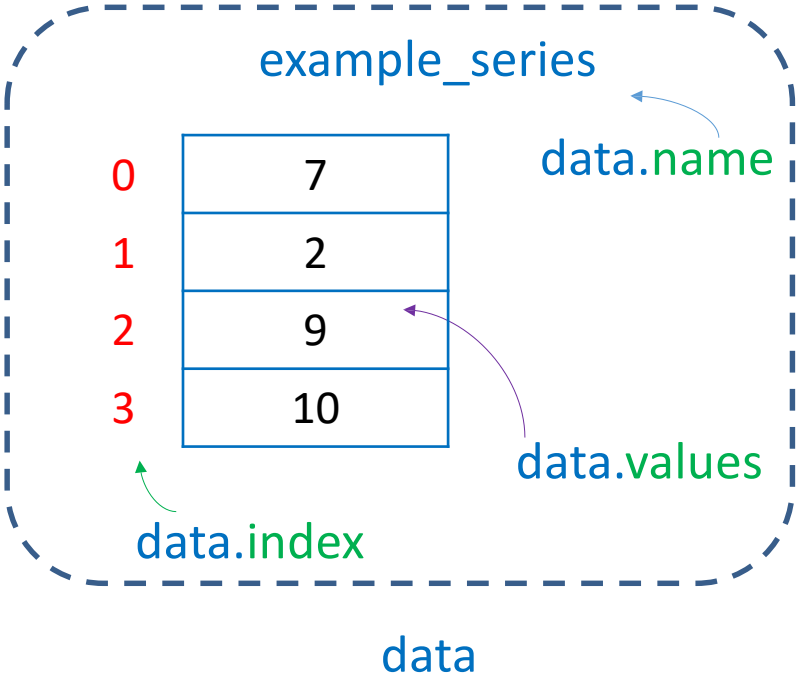


- Installing Pandas: `pip install pandas`
- Import Pandas: `import pandas`
- Alias of Pandas: `import pandas as pd`
- Check Pandas version: `pd.__version__`

Data Structure	Visual	Dimension	Description
Series		1	• 1 – Dimension • Size Immutable • Value of Data Mutable
Data Frame		2	• 2 – Dimension • Size Immutable • Heterogeneous Typed Columns
Panel		3	• 3 – Dimension • Size Mutable

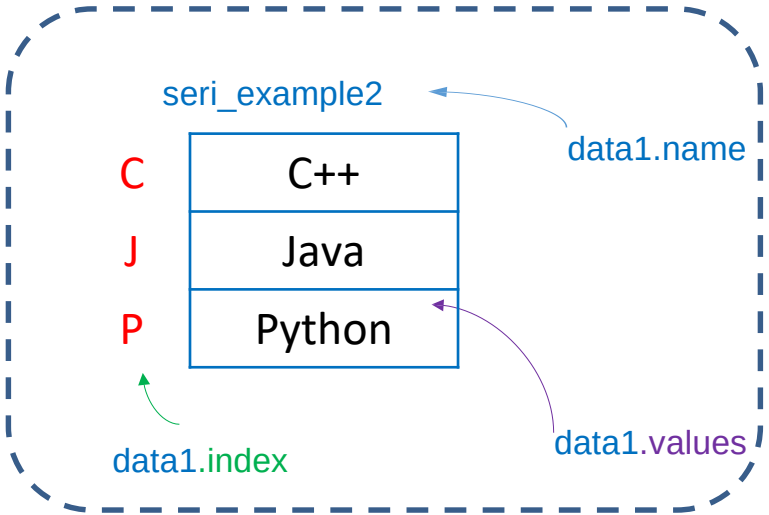
- Data Structure:

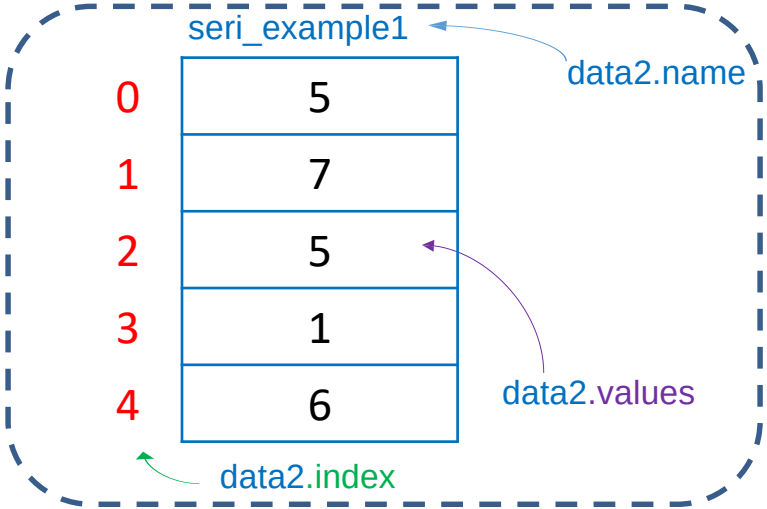




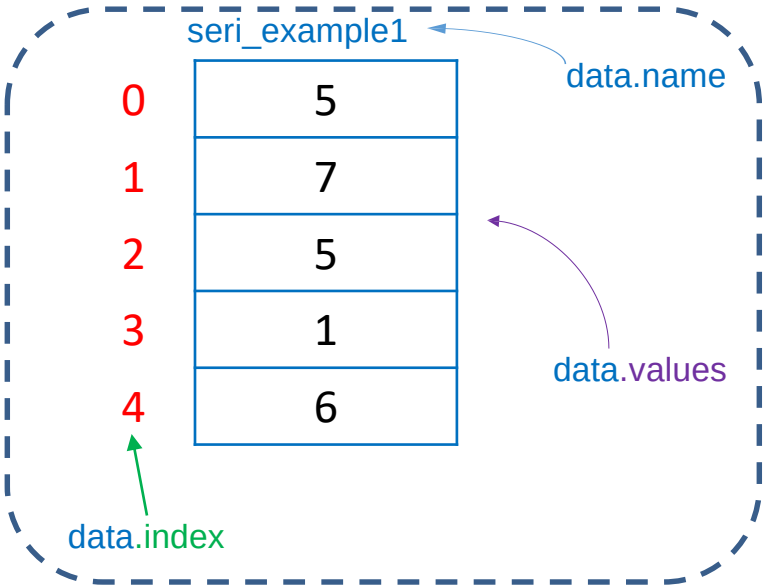
- Create a series:

```
import pandas as pd
data1 = pd.Series([5,7,5,1,6], name = "seri_example1")
data2 = pd.Series(["C++","Java","Python"], index = list("CJP"),
                                                name = "seri_example2")
print(data1)
print(data2)
```





- Get rows:



data

index values

data[0] ➔ 5

data[1] ➔ 7

data[2] ➔ 5

data[3] ➔ 1

data[4] ➔ 6

data[2:3] ➔

2	5

data.loc[2:3] ➔

2	5
3	1

data[data>5] ➔
Or: data[data.values>5]

1	7
4	6

data[data.between(3, 6)] ➔

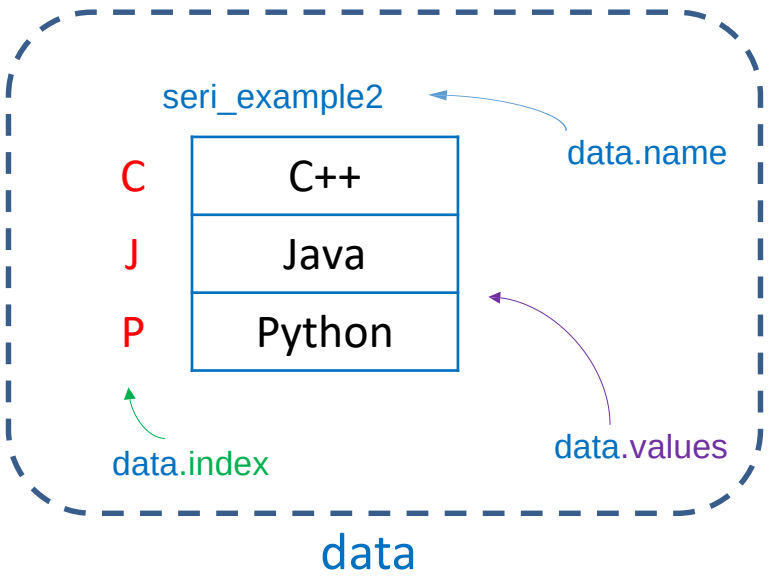
0	5
2	5
4	6

- Get rows:
- Example 1

```
import pandas as pd
data = pd.Series([5,7,5,1,6], name = "seri_example1")

print(data[0])
print(data[1])
print(data[2])
print(data[3])
print(data[4])
print(data[2:3])
print(data.loc[2:3])
print(data[data.values>5])
print(data[data.between(3,6)])
```

- Get rows:



data['C':'J'] ➔

C	C++

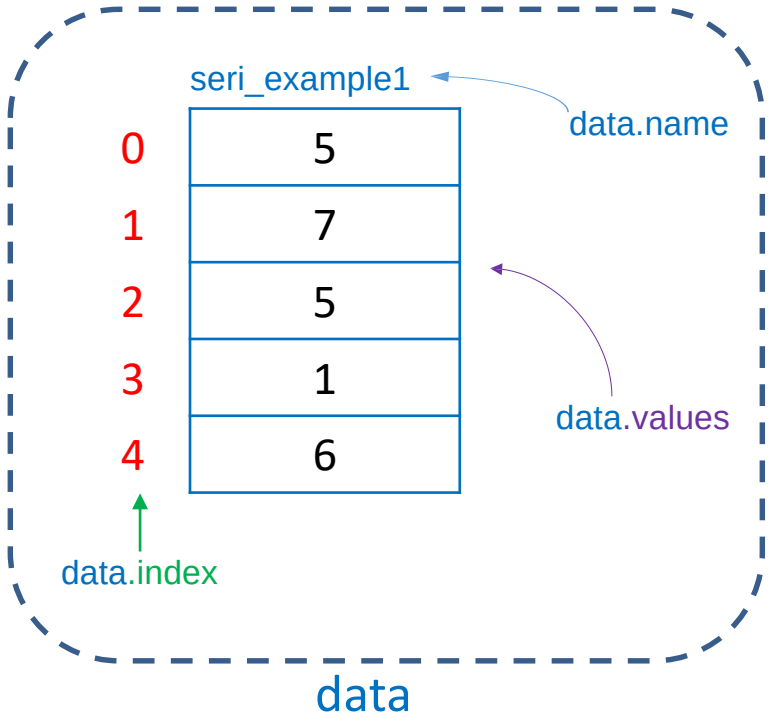
data[['C', 'J']] ➔

C	C++
J	Java

data[0:2] ➔

C	C++
J	Java

- Drop a row:



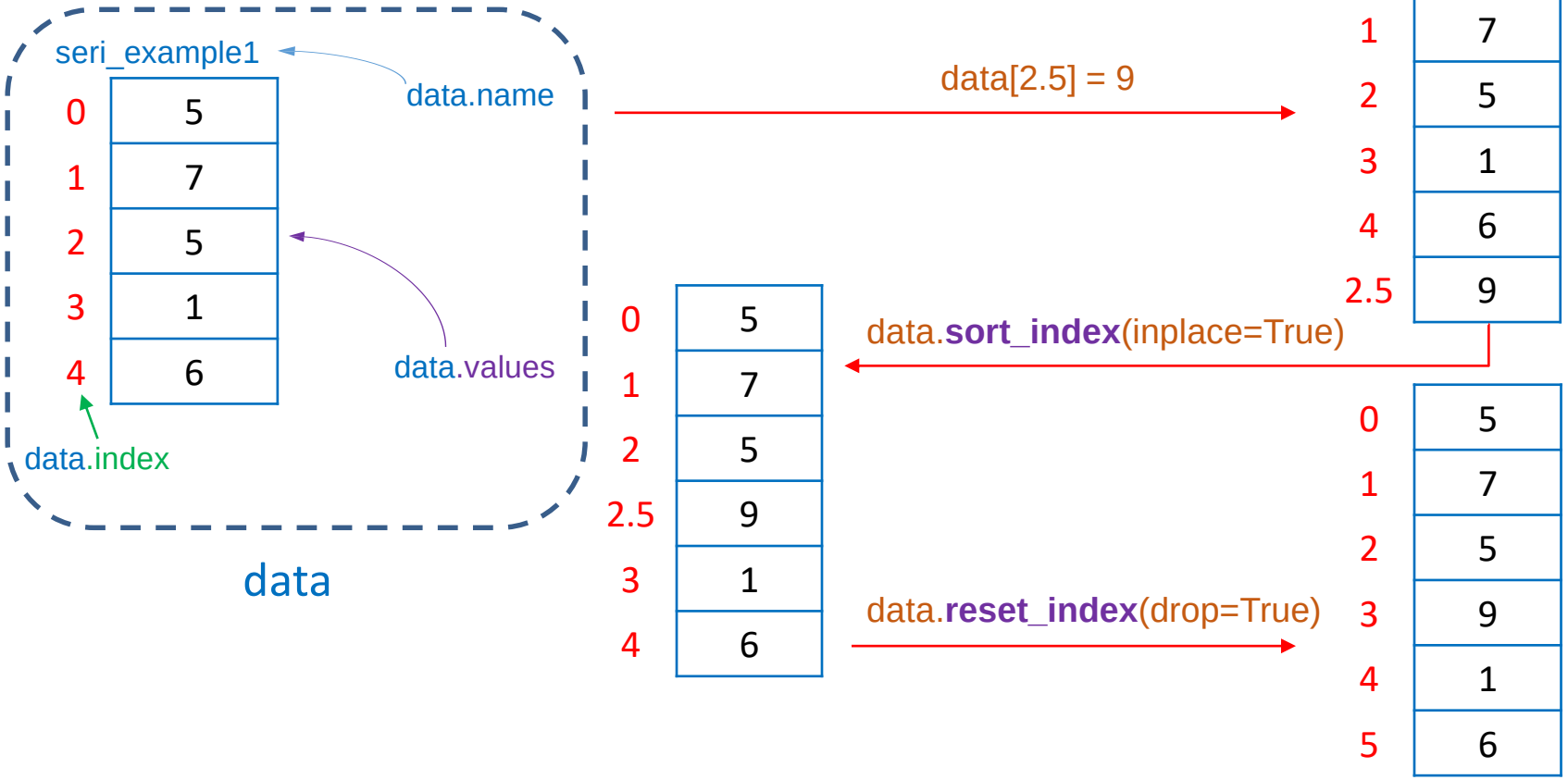
data.drop(2) ➔

5
7
1
6

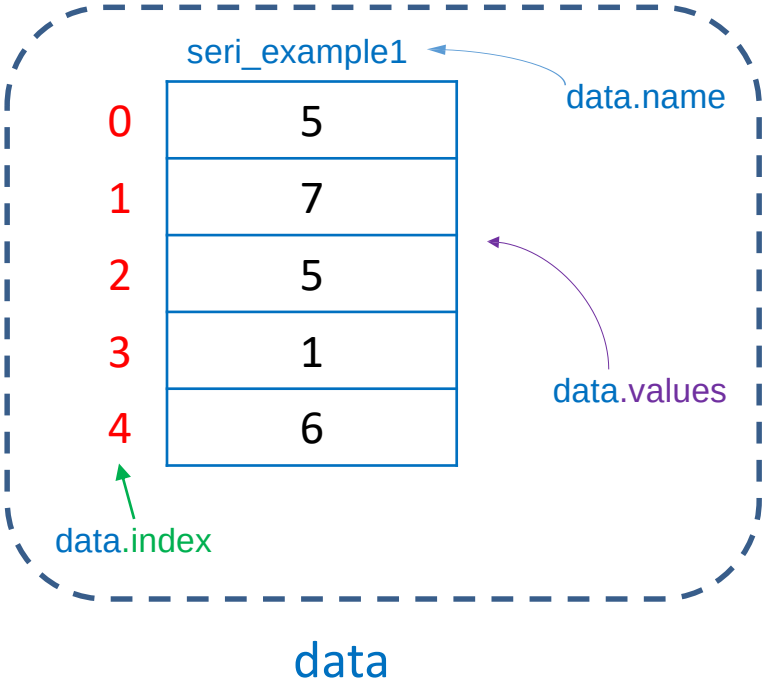
data.drop([2, 4]) ➔

5
7
1

- Insert and Sort a row:



- Calculations and statistics Methods:



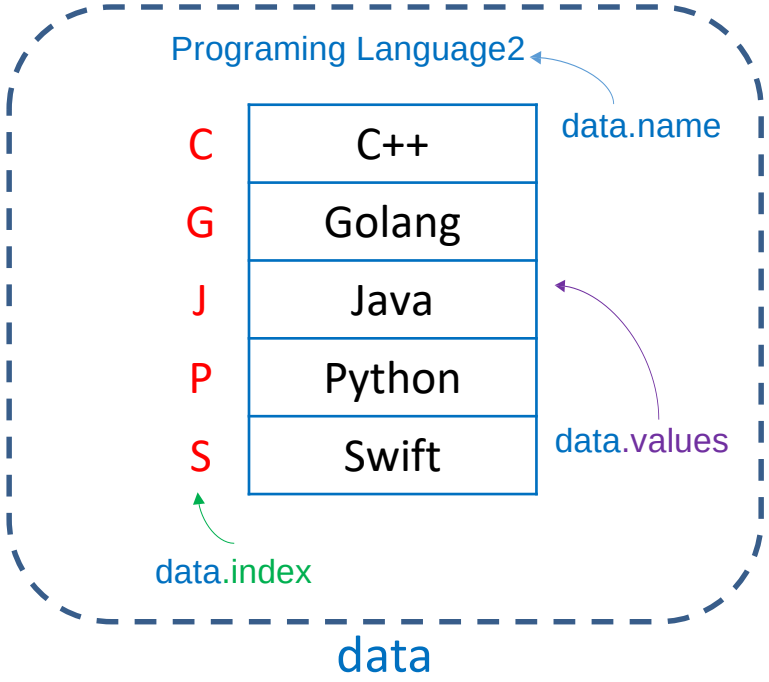
data.**min()** ➔ 1 data.**sum()** ➔ 24

data.**max()** ➔ 7 data.**std()** ➔ 2.28

data.**mean()** ➔ 4.8 data.**var()** ➔ 5.2

data.**idxmax()** ➔ 1 data.**argmax()** ➔ 1

- Calculations and statistics Methods:



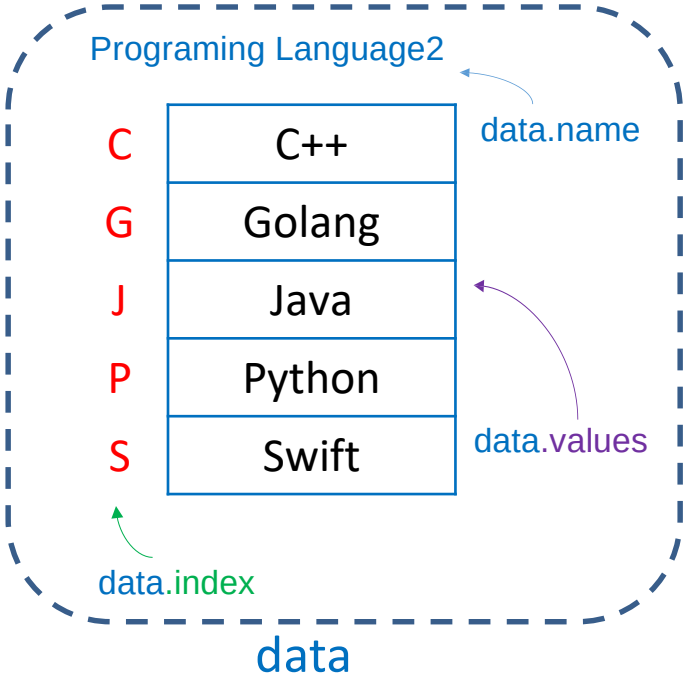
data.str.count('Java')

C	0
G	0
J	1
P	0
S	0

data.str.count('a')

C	0
G	1
J	2
P	0
S	0

- String Methods:



data.str.upper()

C	C++
G	GOLANG
J	JAVA
P	PYTHON
S	SWIFT

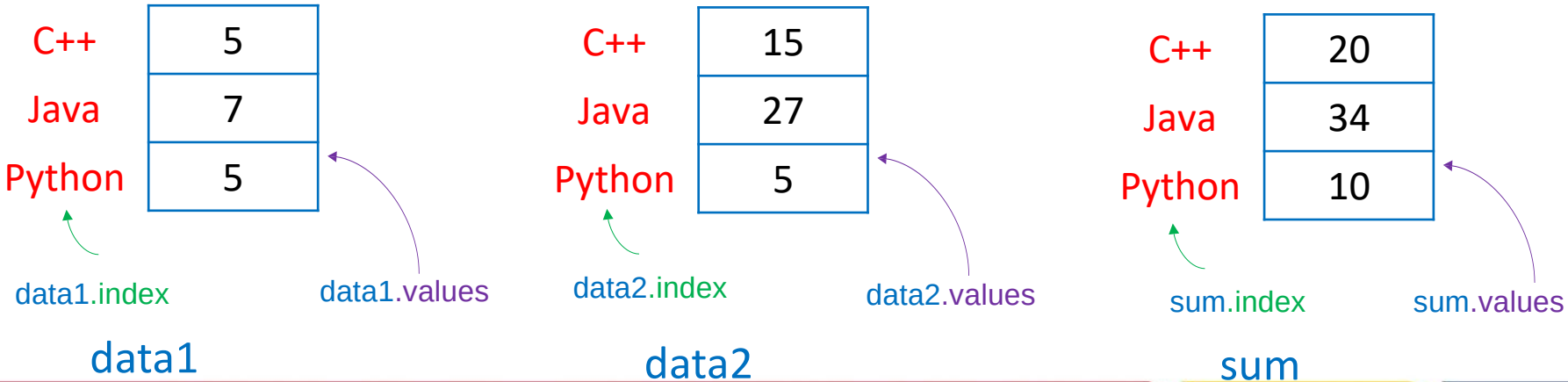
data.replace('Java', 'C#')

C	C++
G	Golang
J	C#
P	Python
S	Swift

- Addition between two series:
- Example 2

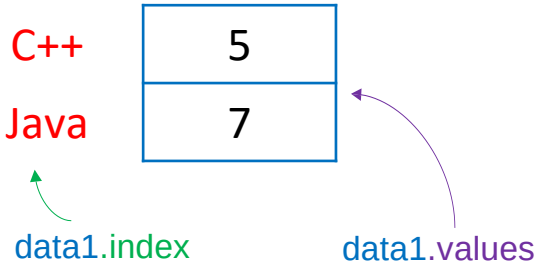
```
import pandas as pd
data1 = pd.Series([5,7,5], index=list(["C++","Java", "python"]),
                                                    name = "data1")
data2 = pd.Series([15,27,5], index=list(["C++","Java", "python"]),
                                                    name = "data2")

sum = data1+data2
```

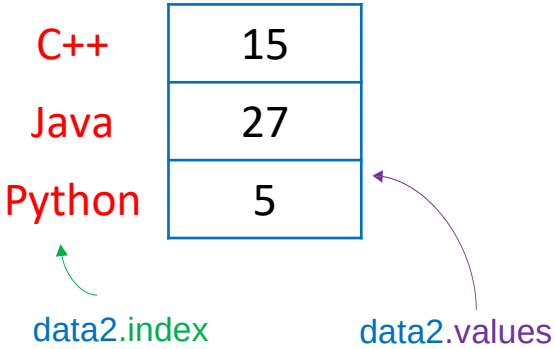


- Addition between two series:
- Example 3

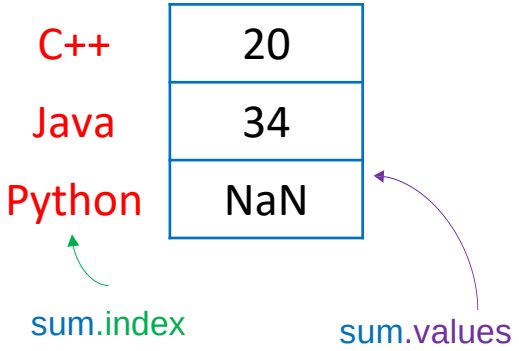
```
import pandas as pd
data1 = pd.Series([5,7], index=list(["C++","Java"]), name = "data1")
data2 = pd.Series([15,27,5], index=list(["C++","Java", "python"]),
                                                    name = "data2")
sum = data1+data2
```



data1

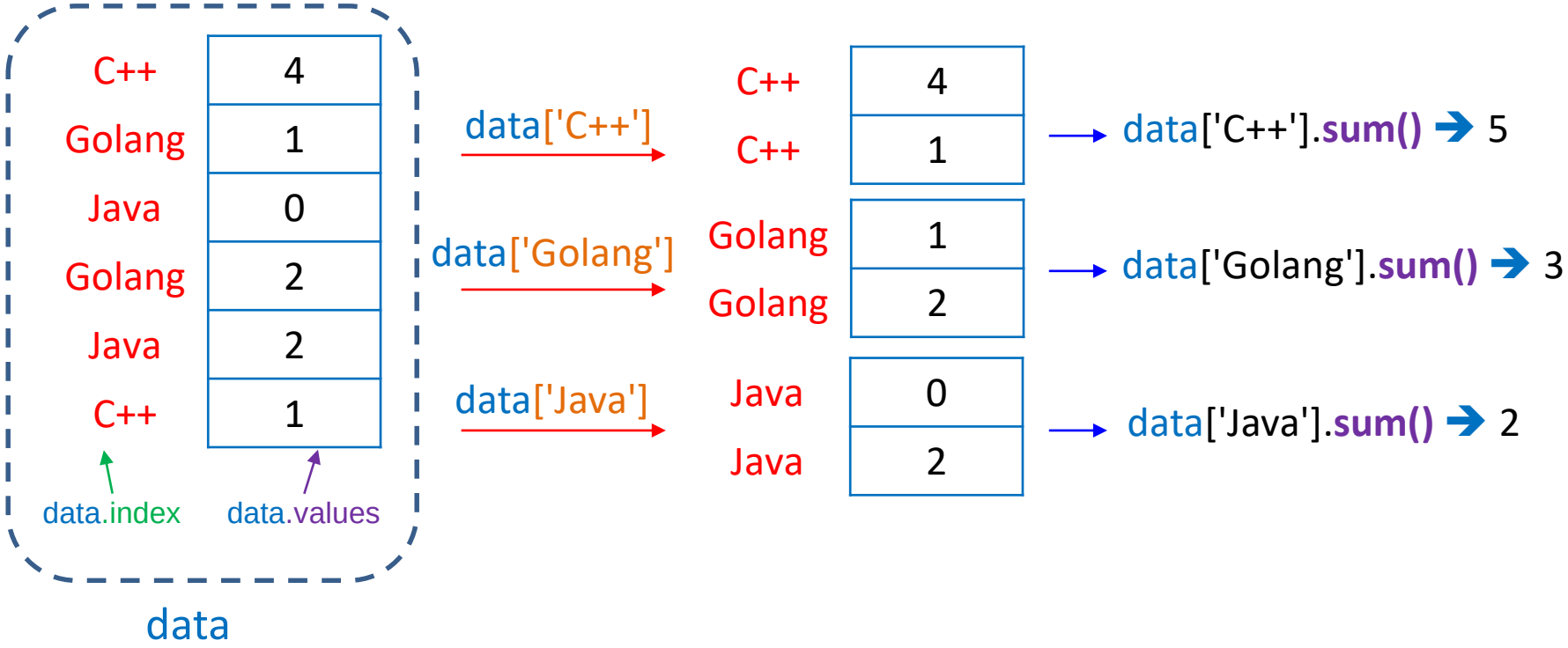


data2

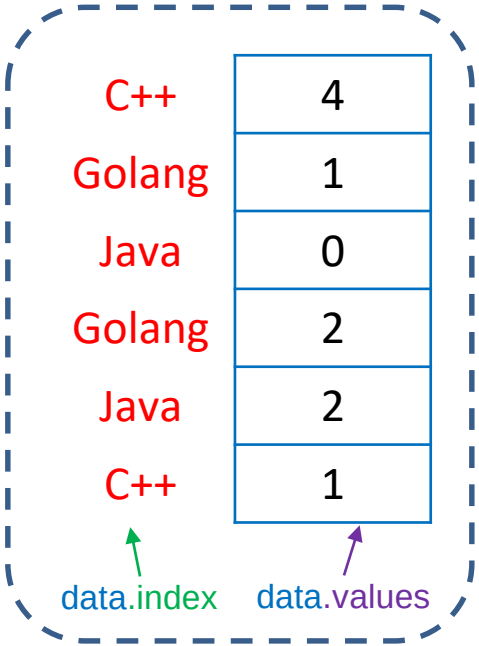


sum

- Group values:



- Group values:
  - Pandas groupby is used for grouping the data according to the categories and applying a function to the categories



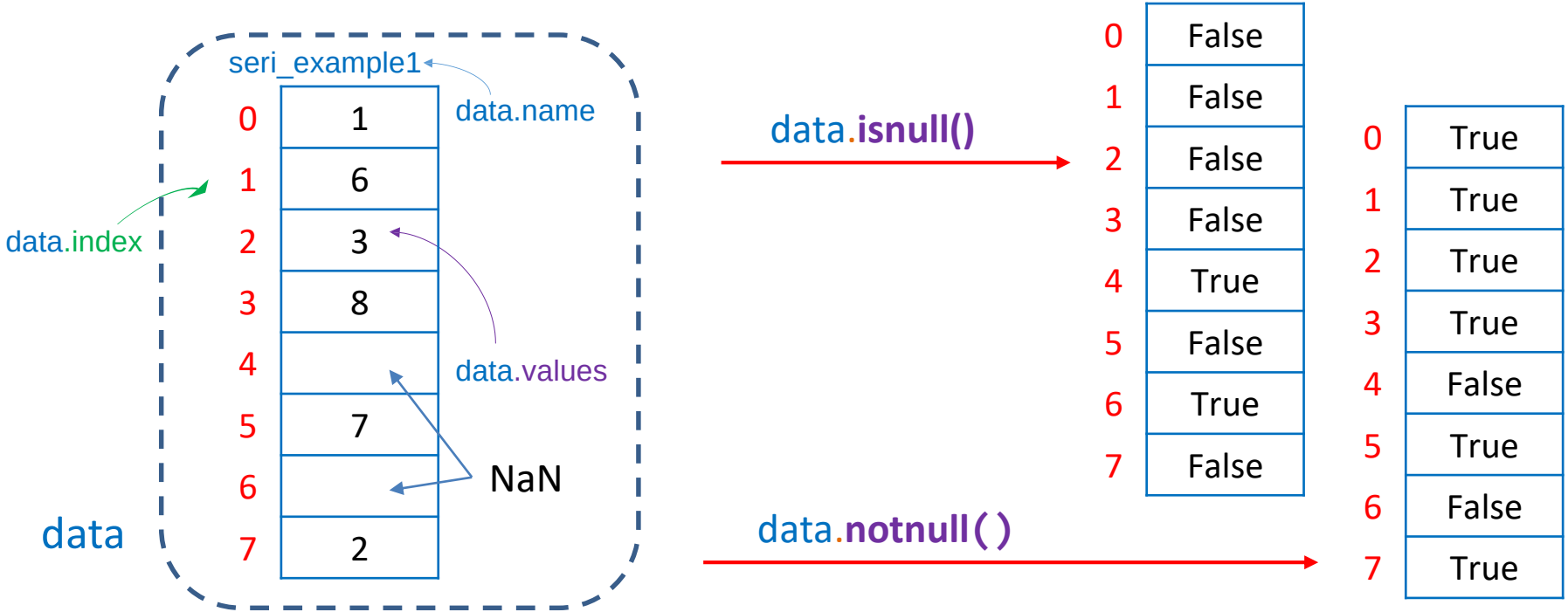
data.groupby(level=0).sum()

C++	5
Golang	3
Java	2

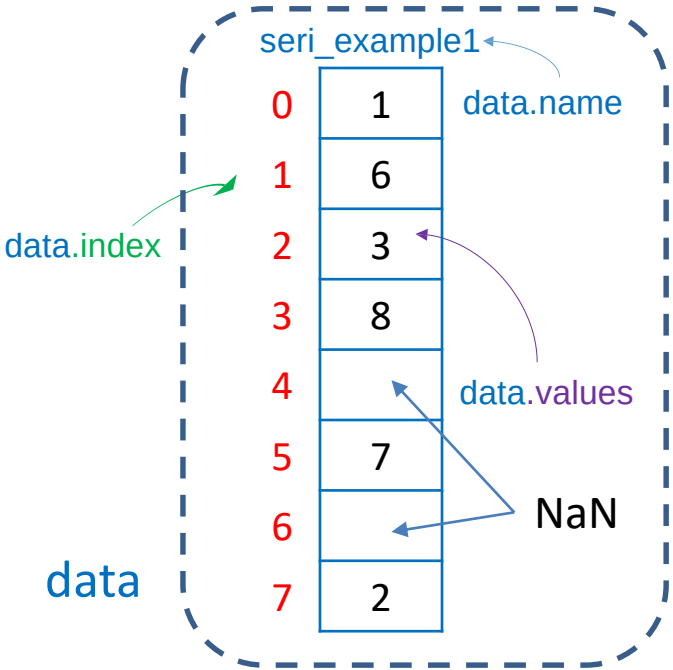
data.groupby(data>3).sum()

FALSE	6
TRUE	4

- Missing values:
  - Missing Data can occur when no information is provided for one or more items or for a whole unit ➔ Missing Data represented for NaN (Not Available Number)



- Missing values:



data.**dropna()**

0	1
1	6
2	3
3	8
5	7
7	2

data.**fillna**(100)

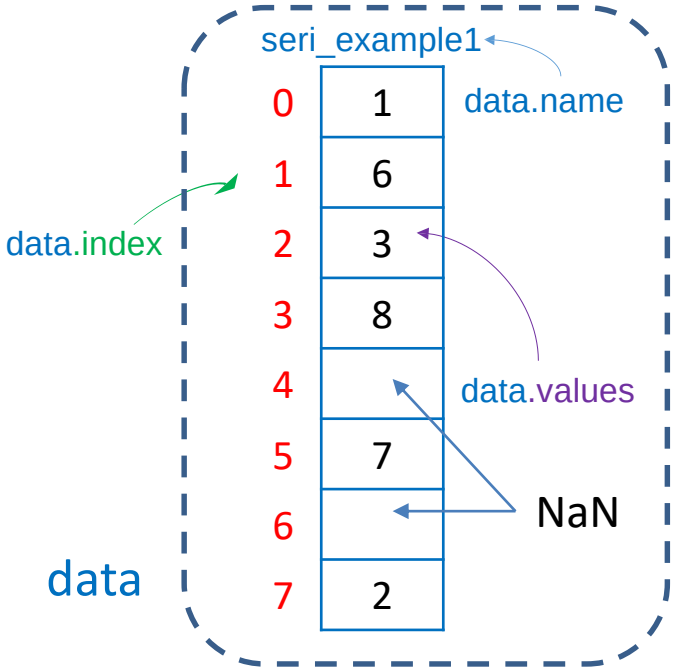
0	1
1	6
2	3
3	8
4	100
5	7
6	100
7	2

data.**interpolate()**

0	1
1	6
2	3
3	8
4	7.5
5	7
6	4.5
7	2

Exercise: How to use a replace() method

- Missing values:



data.**min()** ➔ 1

data.**max()** ➔ 8

data.**sum()** ➔ 27

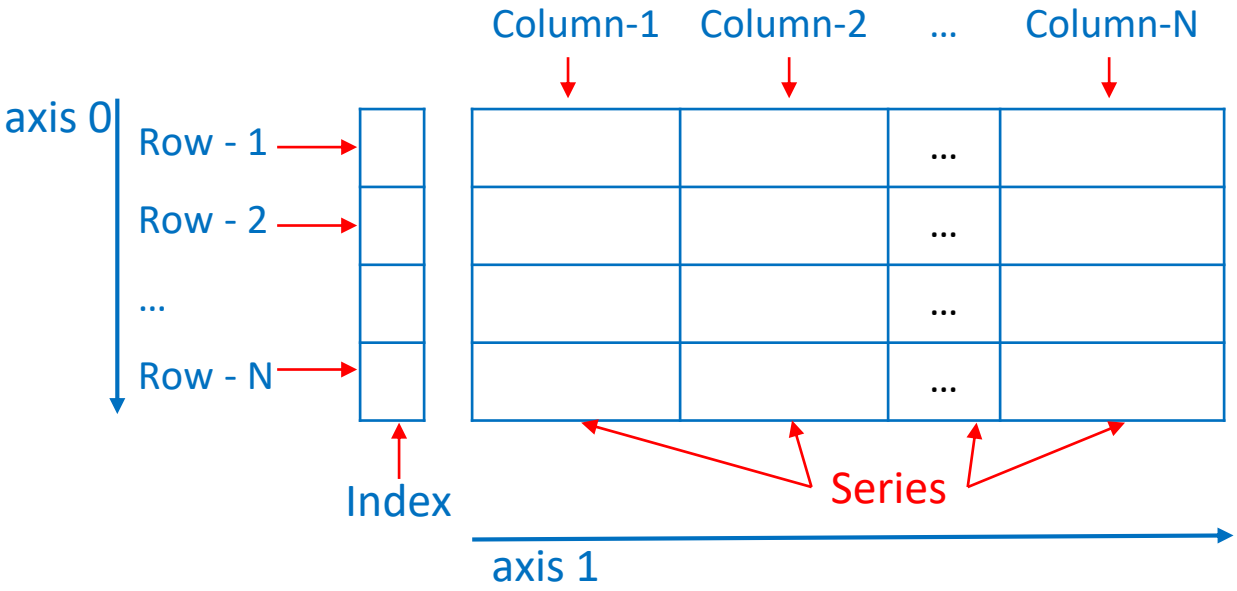
data.**std()** ➔ 2.88

data.**mean()** ➔ 4.5

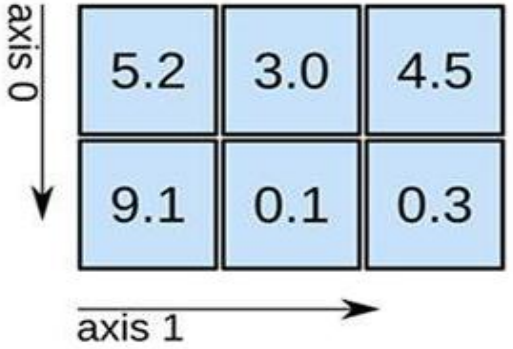
data.**idxmax()** ➔ 3

data.**argmax()** ➔ 3

- Data Structure: Dataframe is a 2 dimensional data structure with mutable size and potentially heterogeneous tabular data.



Example



- Create a dataframe:
- Example 4

```
import pandas as pd
data = pd.DataFrame([[1,"Le V Hung",1980],
                     [2,"Le V Phu",1982],
                     [3,"Tran V Dung",1970]],
                    columns=["STT","Ho Ten","Nam sinh"])
print(data)
```

	STT	Ho Ten	Nam sinh
0	1	Le V Hung	1980
1	2	Le V Phu	1982
2	3	Tran V Dung	1970

- Create a dataframe by loading *.csv file:
- Example 5 **"example.csv"**

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

```
import pandas as pd
data = pd.read_csv("example.csv")
print(data)
data_T = data.T
print(data_T)
```

data.columns

Data_T.columns

Data_T.index

data.index

	0	1	2	3	4	5	6	7
TV	44	17	150	200	180	10	57	100
Radio	300	500	150	200	180	110	157	10
Newspaper	69	70	150	200	180	100	720	400
Sales	44	112	342	334	180	670	20	30

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

- Sorting:
- Example 6

"example.csv"

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

```
import pandas as pd
data = pd.read_csv("example.csv")
print(data.sort_values("Sales"))
print(data.sort_values(["Newspaper","Sales"]))
```

	TV	Radio	Newspaper	Sales
6	57	157	720	20
7	100	10	400	30
0	44	300	69	44
1	17	500	70	112
4	180	180	180	180
3	200	200	200	334
2	150	150	150	342
5	10	110	100	670

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
5	10	110	100	670
2	150	150	150	342
4	180	180	180	180
3	200	200	200	334
7	100	10	400	30
6	57	157	720	20

- Add a column:
- Example 7

"example.csv"

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

```
import pandas as pd
data = pd.read_csv("example.csv")

data["ID"] = list(range(2,10))
print(data)

data = data.set_index("ID")
print(data)
```

	TV	Radio	Newspaper	Sales	ID
0	44	300	69	44	2
1	17	500	70	112	3
2	150	150	150	342	4
3	200	200	200	334	5
4	180	180	180	180	6
5	10	110	100	670	7
6	57	157	720	20	8
7	100	10	400	30	9

ID	TV	Radio	Newspaper	Sales
2	44	300	69	44
3	17	500	70	112
4	150	150	150	342
5	200	200	200	334
6	180	180	180	180
7	10	110	100	670
8	57	157	720	20
9	100	10	400	30

- Delete a column:
- Example 8 **"example.csv"**

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

data.**drop**(label, axis = 1)

```
import pandas as pd
data = pd.read_csv("example.csv")
print(data)
print(data.drop("Radio", axis=1))
print(data.drop(["Radio","Sales"], axis=1))
```

	TV	Newspaper	Sales
0	44	69	44
1	17	70	112
2	150	150	342
3	200	200	334
4	180	180	180
5	10	100	670
6	57	720	20
7	100	400	30

	TV	Newspaper
0	44	69
1	17	70
2	150	150
3	200	200
4	180	180
5	10	100
6	57	720
7	100	400

- Delete a row:
- Example 9

"example.csv"

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

data.**drop**(Index, axis = 0)

```
import pandas as pd
data = pd.read_csv("example.csv")


print(data.drop(1, axis=0))
print(data.drop([1,2,3], axis=0))
```

	TV	Radio	Newspaper	Sales
0	44	300	69	44
2	150	150	150	342
3	200	200	200	334
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

	TV	Radio	Newspaper	Sales
0	44	300	69	44
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

- Get values: `"example.csv"`
- Example 10

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

```
0    44
1    17
2   150
3   200
4   180
5    10
6    57
7   100
```

```
    TV  Sales
0   44     44
1   17    112
2  150    342
3  200    334
4  180    180
5   10    670
6   57     20
7  100     30
```

```
    TV  Radio  Newspaper  Sales
1   17    500        70    112
```

```
    TV  Radio  Newspaper  Sales
4  180    180       180    180
5   10    110       100    670
6   57    157       720     20
7  100     10       400     30
```

```
    TV  Radio  Newspaper  Sales
1   17    500        70    112
3  200    200       200    334
```

```
import pandas as pd
data = pd.read_csv("example.csv")
print(data["TV"])
print(data[["TV","Sales"]])
print(data[1:2])
print(data[4:])
print(data[1:4:2])
```

step

- loc and iloc: **"example.csv"**
- Example 11

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

- loc for indexing by labels
- iloc for indexing by positional index

```
import pandas as pd
data = pd.read_csv("example.csv")
data.set_index([["A", "B", "C", "D", "E",
      "F", "H", "I", "J"]],inplace = true)
```

	TV	Radio	Newspaper	Sales
A	44	300	69	44
B	17	500	70	112
C	150	150	150	342
D	200	200	200	334
E	180	180	180	180
F	10	110	100	670
G	57	157	720	20
H	100	10	400	30

data.loc["C", "Radio"]

data.iloc[2, 1]

data.loc[“A", :]

data.loc[[“B", "E", :]

data.loc["F":"H", "Radio":"Sales"]

data.index

data.column

- Select values:
- Example 12

"example.csv"

TV	Radio	Newspaper	Sales
44	300	69	44
17	500	70	112
150	150	150	342
200	200	200	334
180	180	180	180
10	110	100	670
57	157	720	20
100	10	400	30

data.loc[data.Newspaper < 100]

TV	Radio	Newspaper	Sales
44	300	69	44
17	500	70	112

data.loc[data.Sales > 200]

TV	Radio	Newspaper	Sales
150	150	150	342
200	200	200	334
10	110	100	670

data.loc[(data.Sales > 200) & (data.Radio > 150)]

TV	Radio	Newspaper	Sales
200	200	200	334

- Convert categories to numbers:
- Example 13

```
import pandas as pd
data = pd.read_csv("example.csv")
data["Name"] = pd.Categorical(data["Name"]).codes
```

Name	Unit price	Quantity
A	10	5
A	12	6
A	15	7
A	14	8
B	10	9
B	12	2
B	14	1
B	15	3

Name	Unit price	Quantity
0	10	5
0	12	6
0	15	7
0	14	8
1	10	9
1	12	2
1	14	1
1	15	3

- Missing data:
  - Missing Data in Dataframe represented for NaN(Not Available Number)
  - Using the `isnull()`, `notnull()`, `fillna()`, `replace()`, `interpolate()`, `dropna()` methods to Handling of missing data then same Series Pandas

TV	Radio	Newspaper	Sales
44	300	69	44
17	500	70	112
150		150	342
200	200	200	334
180	180		180
10	110	100	670
57	157	720	
100	10	400	30

NaN

- Data Filter:
  - Using **`filter()`** method

dataframe.**filter**(items, like, regex, axis)

- Item: Takes list of axis labels that need to filter
- Like: Takes axis string label that need to filter
- Regex: regular expression
- Axis: {0 or 'index', 1 or 'columns', None}, default None.

"example.csv"

TV	Radio	Newspaper	Sales
44	300	69	44
17	500	70	112
150	150	150	342
200	200	200	334
180	180	180	180

data.**filter**(items=["TV", "Sales"],axis=1)

TV	Sales
44	44
17	112
150	342
200	334
180	180

data.**filter**(items=[0, 1],axis=0)

TV	Radio	Newspaper	Sales
44	300	69	44
17	500	70	112

Exercise: How to use **filter()** for Series Pandas

- Concatenate data ➔ Apply to add row

"example1.csv"

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334

df1.index

"example1.csv"

	TV	Radio	Newspaper	Sales
0	180	180	180	180
1	10	110	100	670
2	57	157	720	20
3	100	10	400	30

df2.index

df3 = pd.concat([df1, df2])

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334
0	180	180	180	180
1	10	110	100	670
2	57	157	720	20
3	100	10	400	30

df4 = df3.reset_index(drop=True)

	TV	Radio	Newspaper	Sales
0	44	300	69	44
1	17	500	70	112
2	150	150	150	342
3	200	200	200	334
4	180	180	180	180
5	10	110	100	670
6	57	157	720	20
7	100	10	400	30

df5 = pd.concat([df1, df2], ignore_index=True)

- Groupby:
  - Using **`groupby()`** for splitting the dataframe over some criteria into data subsets.

	Name	Unit price	Quantity
0	A	10	5
1	A	12	6
2	A	15	7
3	B	14	8
4	B	10	9
5	B	12	2
6	C	14	1
7	C	15	3

data.**groupby**(**"Name"**).groups

{'A': [0, 1, 2], 'B': [3, 4, 5], 'C': [6, 7]}

data.**groupby**(**"Name", "Unit price"**).groups

{('A', 5): [0], ('A', 6): [1], ('A', 7): [2], ('B', 2): [5], ('B', 8): [3], ('B', 9): [4], ('C', 1): [6], ('C', 3): [7]}

- Groupby:
- Example 14

```
import pandas as pd
data = pd.read_csv("donhang.csv")
print(data.describe())
print(data.groupby("Name").sum())
data["Amount"]= data["Unit price"]*data["Quantity"]
print(data)
```

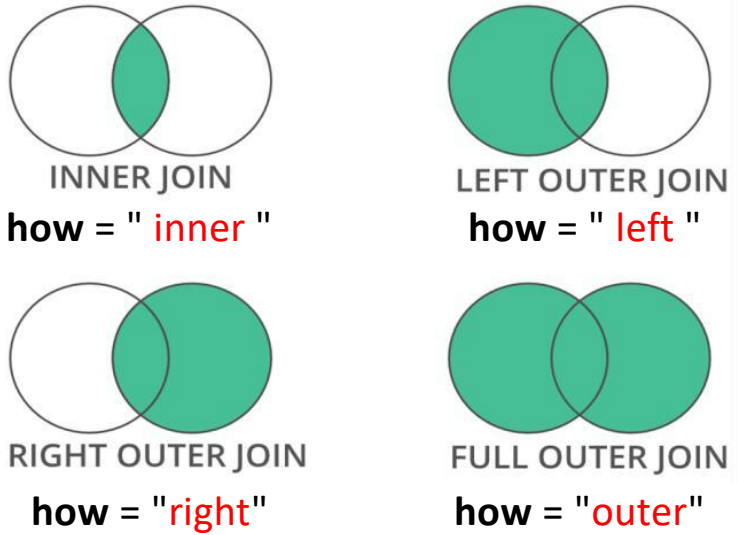
Name	Unit price	Quantity
A1	28	9
A2	30	10
A3	24	8
A4	20	14

	Name	Unit price	Quantity
0	A4	10	5
1	A3	12	6
2	A2	15	7
3	A1	14	8
4	A4	10	9
5	A3	12	2
6	A1	14	1
7	A2	15	3

	Unit price	Quantity
count	8.000000	8.000000
mean	12.750000	5.125000
std	2.052873	2.900123
min	10.000000	1.000000
25%	11.500000	2.750000
50%	13.000000	5.500000
75%	14.250000	7.250000
max	15.000000	9.000000

	Name	Unit price	Quantity	Amount
0	A4	10	5	50
1	A3	12	6	72
2	A2	15	7	105
3	A1	14	8	112
4	A4	10	9	90
5	A3	12	2	24
6	A1	14	1	14
7	A2	15	3	45

- Data Merging/Joining:
  - Using **`merge()`** method to combine data on common columns or indices.
  - Using **`join()`** method to combine the columns of two differently-indexed DataFrames into a single result DataFrame based on a:



- with one unique key combination
  on =[key]
- using multiple join keys
  on =[key1,key2,...]

- Data Merging/Joining:

df

	Name	Age
I1	An	23
I2	Binh	12
I3	Hoa	30

df1

	address	Class
I1	DaNang	CCA
I2	QN	CCB
I3	Hue	CCD
I4	HoaVang	CCE

df2 = df.**join**(df1)

df2

	Name	Age	address	Class
I1	An	23	DaNang	CCA
I2	Binh	12	QN	CCB
I3	Hoa	30	Hue	CCD

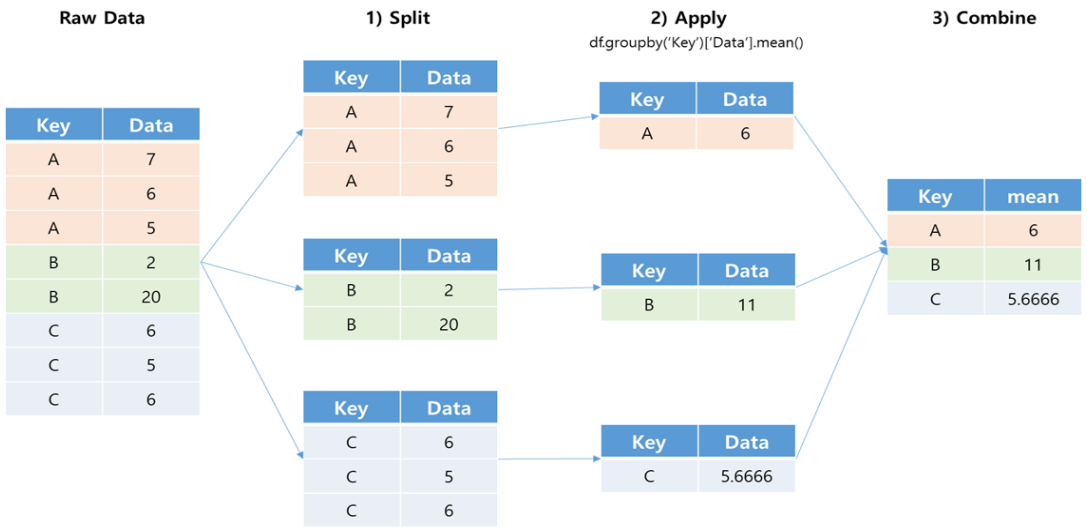
df2 = df.**join**(df1, how = “outer”)

df2

	Name	Age	address	Class
I1	An	23	DaNang	CCA
I2	Binh	12	QN	CCB
I3	Hoa	30	Hue	CCD
I4			HoaVang	CCE

NaN

- Grouping the data according to the categories and apply a function to the categories.
- often involves 3 operations:
  - Splitting the Data Object
  - Applying a function
  - Combining the results



- Summary Statistical Methods:

`sum()`	Compute sum of column values	`first()`	Compute first of group values
`min()`	Compute min of column values	`last()`	Compute last of group values
`max()`	Compute max of column values	`count()`	Compute count of column values
`mean()`	Compute mean of column	`std()`	Standard deviation of column
`size()`	Compute column sizes	`var()`	Compute variance of column
`describe()`	Generates descriptive statistics	`sem()`	Standard error of the mean of column

- Transformation methods:
  - Returns a self-produced dataframe with transformed values after applying the function specified in its parameter.
    - `apply()`
    - `applymap()`
    - `melt()`
    - `transform()`

- **apply()** method:

```
data.apply(sum, axis=0)
```

P	51
Q	26

```
data.apply(sum, axis=1)
```

0	15
1	18
2	22
3	22

```
data.apply(lambda x:[10,20], axis=1)
```

0	[10,20]
1	[10,20]
2	[10,20]
3	[10,20]

	P	Q
0	10	5
1	12	6
2	15	7
3	14	8

```
data.apply(lambda x:[10,20], axis=1,result_type="expand")
```

	0	1
0	10	20
1	10	20
2	10	20
3	10	20

```
data.apply(lambda x:[10,20], axis=1,result_type="broadcast")
```

	P	Q
0	10	20
1	10	20
2	10	20
3	10	20

- **`applymap()`** method:
  - Used to apply a function to a Dataframe elementwise.

	P	Q
0	10	5
1	12	6
2	15	7
3	14	8

```
data.applymap(lambda x:len(str(x)))
```

	P	Q
0	2	1
1	2	1
2	2	1
3	2	1

```
data.applymap(lambda x:x**2)
```

	P	Q
0	100	25
1	144	36
2	225	49
3	196	64

- `melt()` method:
  - Used to unpivot a given DataFrame from wide format to long format

```
DataFrame.melt([id_vars], [value_vars],var_name, value_name, [col_level])
```

  - [**id_vars**]: column(s) to use as identifier variables.
  - [**value_vars**]: column(s) to unpivot. If not specified, uses all columns that are not set as id_vars.
  - **var_name**: name to use for the “variable” column. If None it uses **frame.columns.name** or “variable”.
  - **value_name**: name to use for the “value” column.
  - [**col_level**]: if columns are a MultiIndex then use this level to melt.

- `transform()` method:
  - Used to call function(func) on self producing a DataFrame with transformed values and that has the same axis length as self.

DataFrame.transform(func, axis, *args, **kwargs)

  - **Func**: : Function to use for transforming the data
  - **Axis**: : 0 or “index”: apply function to each column; 1 or “columns”: apply function to each row.
  - ***args**: Positional arguments to pass to func.
  - ****kwargs**: Keyword arguments to pass to func.

- Categorical Data:
  - A pandas data type corresponding to categorical variables in statistics, e.g. gender, social class, blood type...
  - Using the standard pandas **Categorical()** constructor to create categorical object

pandas.**Categorical**(values, categories, ordered)

- Example 15

```
import pandas as pd
data = pd.Categorical(["a","b","c","a","b","c"],
                      categories = ["b","a","c"],ordered=True)
print(data)
```

['a', 'b', 'c', 'a', 'b', 'c']
Categories (3, object): ['b' < 'a' < 'c']

7. Working with time series data

- A time series is any data set where the values are measured at different points in time.
  - Uniformly spaced, Eg. hourly weather measurements, daily counts of web site visits, or monthly sales totals.
  - Irregularly spaced. Eg. timestamped data in a computer system's event log, a history of 115 emergency calls
- Pandas provides useful objects in working with time series data:
  - **Timestamp** Object
  - **Period** Object
  - **Timedelta** Object

- **Period** Object:
  - Period object represents an interval in time used to check if a specific event occurs within a certain period such as when monitoring the number of flights taking off or the average stock price during a period.
- Example 16

```
import pandas as pd
# Create time period
p1 = pd.Period('2020-12-25')
# Create time stamp
t1 = pd.Timestamp('2020-12-25 18:12')
# Test Time interval

print(p1)
print(t1)
print(p1.start_time < t1 < p1.end_time)
```

```
2020-12-25
2020-12-25 18:12:00
True
```

1. Explain the meaning of each command line and show the result of the following code:

1.
```
import pandas as pd
data = pd.Series([5,7,1], name = "example")
print(data[[1,2]]
```

2.
```
import pandas as pd
data = pd.Series([5,7,1], name = "example")
print(data[[True, False]]
```

3.
```
import pandas as pd
data = pd.Series([5,7,1], name = "example")
print(data[[True, False, True]]
```

4.
```
import pandas as pd
data = pd.Series([5,7,1], name = "example")
print(data.groupby(data>3).sum())
```

5.
```
import pandas as pd
data = pd.Series([5,7,1,2], name = "example")
print(data.groupby([True, False, True, True]).sum())
```

6.
```
import pandas as pd
data = pd.Series(["Java", "C++", "Python"], name = "PL")
```

6.1.
```
data_sorted  = pd.sort_index(inplace = True)
print(data_sorted)
```

6.2.
```
data_sorted  = pd.sort_index(inplace =False)
print(data_sorted)
```

2. Quiz

```
import pandas as pd
data = pd.read_csv("example")
df.loc[1, "Radio"]
```

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

(a) 39 (b) 45 (c) 41 (d) another

2. Quiz

```
import pandas as pd
data = pd.read_csv("example")
print(df.loc[1, 3]
```

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

(a) 45 (b) 69 (c) 58 (d) another

2. Quiz

```
import pandas as pd
data = pd.read_csv("example")
Data.loc[:, "TV" : "Newspaper" :]
```

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11

(a)

	TV	Radio	Newspaper
0	44	39	45
1	17	45	69
2	151	41	58
3	180	10	58
4	8	48	75
5	57	32	23

(b)

	TV	Radio
0	44	39
1	17	45
2	151	41
3	180	10
4	8	48
5	57	32

(c)

another

(d)

2. Quiz

```
import pandas as pd
data = pd.read_csv("example")
Data.loc[:, "TV": "Newspaper":2]
```

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11

(a)

	TV	Radio	Newspaper
0	44	39	45
1	17	45	69
2	151	41	58
3	180	10	58
4	8	48	75
5	57	32	23

(b)

	TV	Radio
0	44	39
1	17	45
2	151	41
3	180	10
4	8	48
5	57	32

(c)

	TV	Newspaper
0	44	45
1	17	69
2	151	58
3	180	58
4	8	75
5	57	23

another

(d)

2. Quiz

```
import pandas as pd
data = pd.read_csv("example")
Data.loc[2:4, 1:3]
```

	TV	Radio	News	Sales
0	44	39	45	10
1	17	45	69	12
2	15	41	58	16
3	18	10	58	17
4	8	48	75	7
5	57	32	23	11

(a)

	TV	Radio	News	Sales
0	44	39	45	10
1	17	45	69	12
2	15	41	58	16
3	18	10	58	17
4	8	48	75	7
5	57	32	23	11

(b)

	TV	Radio	News	Sales
0	44	39	45	10
1	17	45	69	12
2	15	41	58	16
3	18	10	58	17
4	8	48	75	7
5	57	32	23	11

(c)

	TV	Radio	News	Sales
0	44	39	45	10
1	17	45	69	12
2	15	41	58	16
3	18	10	58	17
4	8	48	75	7
5	57	32	23	11

(d)

1. Sử dụng phương thức **pd.read_csv()** nạp dữ liệu từ một tệp CSV vào dataframe với tên data

```
import pandas as pd

filename = 'tr_eikon_eod_data.csv' #➔ Khai báo đường dẫn và tên file
data = pd.read_csv(filename, #➔ Mở file nạp vào biến data
                   index_col=0, #➔ Chỉ ra cột được xử lý là cột có chỉ số 0
                   parse_dates=True) #➔ Xác định rằng các giá trị của chỉ số (index)
                                        thuộc kiểu dữ liệu datetime
```

2. Thống kê tóm tắt khác nhau của dữ liệu để có được cái nhìn tổng quan về dữ liệu

```
data.info()
```
➔ Đối tượng DataFrame

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2216 entries, 2010-01-01 to 2018-06-29
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   AAPL.O  2138 non-null   float64
 1   MSFT.O  2138 non-null   float64
 2   INTC.O  2138 non-null   float64
 3   AMZN.O  2138 non-null   float64
 4   GS.N    2138 non-null   float64
 5   SPY     2138 non-null   float64
 6   .SPX    2138 non-null   float64
 7   .VIX    2138 non-null   float64
 8   EUR=    2216 non-null   float64
 9   XAU=    2211 non-null   float64
 10  GDX     2138 non-null   float64
 11  GLD     2138 non-null   float64
dtypes: float64(12)
memory usage: 225.1 KB
```

```
data.head()  #➔ 05 hàng đầu tiên
```

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2010-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.4323	1096.35	NaN	NaN
2010-01-04	30.572827	30.950	20.88	133.90	173.08	113.33	1132.99	20.04	1.4411	1120.00	47.71	109.80
2010-01-05	30.625684	30.960	20.87	134.69	176.14	113.63	1136.52	19.35	1.4368	1118.65	48.17	109.70
2010-01-06	30.138541	30.770	20.80	132.25	174.26	113.71	1137.14	19.16	1.4412	1138.50	49.34	111.51
2010-01-07	30.082827	30.452	20.60	130.00	177.67	114.19	1141.69	19.06	1.4318	1131.90	49.10	110.82

`data.tail()#` ➔ 05 hàng cuối cùng.

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2018-06-25	182.17	98.39	50.71	1663.15	221.54	271.00	2717.07	17.33	1.1702	1265.00	22.01	119.89
2018-06-26	184.43	99.08	49.67	1691.09	221.58	271.60	2723.06	15.92	1.1645	1258.64	21.95	119.26
2018-06-27	184.16	97.54	48.76	1660.51	220.18	269.35	2699.63	17.91	1.1552	1251.62	21.81	118.58
2018-06-28	185.50	98.63	49.25	1701.45	223.42	270.89	2716.31	16.85	1.1567	1247.88	21.93	118.22
2018-06-29	185.11	98.61	49.71	1699.80	220.57	271.28	2718.37	16.09	1.1683	1252.25	22.31	118.65

`data.describe().round(2)` #➔ Phương thức **`describe()`** cung cấp các thống kê tiêu chuẩn hữu ích cho từng cột.

	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
count	2138.00	2138.00	2138.00	2138.00	2138.00	2138.00	2138.00	2138.00	2216.00	2211.00	2138.00	2138.00
mean	93.46	44.56	29.36	480.46	170.22	180.32	1802.71	17.03	1.25	1349.01	33.57	130.09
std	40.55	19.53	8.17	372.31	42.48	48.19	483.34	5.88	0.11	188.75	15.17	18.78
min	27.44	23.01	17.66	108.61	87.70	102.20	1022.58	9.14	1.04	1051.36	12.47	100.50
25%	60.29	28.57	22.51	213.60	146.61	133.99	1338.57	13.07	1.13	1221.53	22.14	117.40
50%	90.55	39.66	27.33	322.06	164.43	186.32	1863.08	15.58	1.27	1292.61	25.62	124.00
75%	117.24	54.37	34.71	698.85	192.13	210.99	2108.94	19.07	1.35	1428.24	48.34	139.00
max	193.98	102.49	57.08	1750.08	273.38	286.58	2872.87	48.00	1.48	1898.99	66.63	184.59

`data.mean()#` ➔ Giá trị trung bình của mỗi cột

```
AAPL.O      93.455973
MSFT.O      44.561115
INTC.O      29.364192
AMZN.O     480.461251
GS.N       170.216221
SPY        180.323029
.SPX      1802.713106
.VIX        17.027133
EUR=         1.248587
XAU=      1349.014130
GDX         33.566525
GLD        130.086590
dtype: float64
```

```
import numpy as np
data.aggregate([min,  #➔ Giá trị nhỏ nhất của mỗi cột
                np.mean,  #➔ Giá trị trung bình của mỗi cột
                np.std,   #➔ Độ lệch chuẩn của mỗi cột
                np.median,  #➔ Giá trị trung vị của mỗi cột
                max]).round(2)#➔ Giá trị lớn nhất của mỗi cột
```

	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
min	27.44	23.01	17.66	108.61	87.70	102.20	1022.58	9.14	1.04	1051.36	12.47	100.50
mean	93.46	44.56	29.36	480.46	170.22	180.32	1802.71	17.03	1.25	1349.01	33.57	130.09
std	40.55	19.53	8.17	372.31	42.48	48.19	483.34	5.88	0.11	188.75	15.17	18.78
median	90.55	39.66	27.33	322.06	164.43	186.32	1863.08	15.58	1.27	1292.61	25.62	124.00
max	193.98	102.49	57.08	1750.08	273.38	286.58	2872.87	48.00	1.48	1898.99	66.63	184.59

3. Thống kê về sự thay đổi giá trị theo thời gian, sử dụng các phương pháp: chênh lệch tuyệt đối, thay đổi phần trăm và lợi nhuận logarit.

`data.diff().head()` # ➔ Phương thức **`diff()`** trả về sự thay đổi giữa hai giá trị chỉ mục.

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2010-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-01-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0088	23.65	NaN	NaN
2010-01-05	0.052857	0.010	-0.01	0.79	3.06	0.30	3.53	-0.69	-0.0043	-1.35	0.46	-0.10
2010-01-06	-0.487142	-0.190	-0.07	-2.44	-1.88	0.08	0.62	-0.19	0.0044	19.85	1.17	1.81
2010-01-07	-0.055714	-0.318	-0.20	-2.25	3.41	0.48	4.55	-0.10	-0.0094	-6.60	-0.24	-0.69

```
data.diff().mean()
```

```
AAPL.O    0.064737
MSFT.O    0.031246
INTC.O    0.013540
AMZN.O    0.706608
GS.N      0.028224
SPY       0.072103
.SPX      0.732659
.VIX     -0.019583
EUR=     -0.000119
XAU=      0.041887
GDX      -0.015071
GLD      -0.003455
dtype: float64
```

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2010-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-01-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.006	0.022	NaN	NaN
2010-01-05	0.002	0.000	-0.000	0.006	0.018	0.003	0.003	-0.034	-0.003	-0.001	0.010	-0.001
2010-01-06	-0.016	-0.006	-0.003	-0.018	-0.011	0.001	0.001	-0.010	0.003	0.018	0.024	0.016
2010-01-07	-0.002	-0.010	-0.010	-0.017	0.020	0.004	0.004	-0.005	-0.007	-0.006	-0.005	-0.006

```
data.pct_change().round(3).head()
```
➔ Phương thức `pct_change()` trả về phần trăm thay đổi giữa hai giá trị chỉ mục

```
rets = np.log(data / data.shift(1))#➔ Trả về lợi nhuận logarit
rets.head().round(3)    #➔ Hiển thị 05 hang đầu tiên.
```

Date												
2010-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-01-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.006	0.021	NaN	NaN
2010-01-05	0.002	0.000	-0.000	0.006	0.018	0.003	0.003	-0.035	-0.003	-0.001	0.010	-0.001
2010-01-06	-0.016	-0.006	-0.003	-0.018	-0.011	0.001	0.001	-0.010	0.003	0.018	0.024	0.016
2010-01-07	-0.002	-0.010	-0.010	-0.017	0.019	0.004	0.004	-0.005	-0.007	-0.006	-0.005	-0.006

4. Resampling dữ liệu chuỗi thời gian (tài chính) ➔ chuyển đổi dữ liệu theo từng tick sang khoảng thời gian một phút, hoặc dữ liệu hàng ngày sang khoảng thời gian hàng tuần hoặc hàng tháng…

```
data.resample('1w', label='right').last().head()
```

#➔ Dữ liệu EOD (End Of Day – dữ liệu cuối ngày) được lấy mẫu lại theo khoảng thời gian hàng tuần …

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2010-01-03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.4323	1096.35	NaN	NaN
2010-01-10	30.282827	30.66	20.83	133.52	174.31	114.57	1144.98	18.13	1.4412	1136.10	49.84	111.37
2010-01-17	29.418542	30.86	20.80	127.14	165.21	113.64	1136.03	17.91	1.4382	1129.90	47.42	110.86
2010-01-24	28.249972	28.96	19.91	121.43	154.12	109.21	1091.76	27.31	1.4137	1092.60	43.79	107.17
2010-01-31	27.437544	28.18	19.40	125.41	148.72	107.39	1073.87	24.62	1.3862	1081.05	40.72	105.96

Trong tài chính, đặc biệt là khi làm việc với dữ liệu chuỗi thời gian (time series), thì:

- Tick là một đơn vị thay đổi nhỏ nhất của giá hoặc một bản ghi giao dịch/biến động giá xảy ra tại một thời điểm cụ thể trên thị trường.
- Dữ liệu tick (tick data) thường bao gồm: Giá giao dịch (price); Khối lượng (volume); Thời điểm giao dịch (timestamp)

Ví dụ:

- Nếu bạn theo dõi giá cổ phiếu, mỗi lần giá thay đổi hoặc có giao dịch khớp lệnh thì đó được ghi lại thành một tick.
- Dữ liệu tick thường có tần suất rất cao (giao dịch theo mili-giây), sau đó mới được resample thành dữ liệu phút, giờ, ngày... để dễ phân tích.

➔ Tick = một thay đổi/ghi nhận nhỏ nhất của dữ liệu thị trường (giá/khối lượng) theo thời gian thực.

```
data.resample('1m', label='right').last().head()
```

#➔ Dữ liệu EOD (End Of Day – dữ liệu cuối ngày) được lấy mẫu lại theo khoảng thời gian hàng tháng

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2010-01-31	27.437544	28.1800	19.40	125.41	148.72	107.3900	1073.87	24.62	1.3862	1081.05	40.72	105.960
2010-02-28	29.231399	28.6700	20.53	118.40	156.35	110.7400	1104.49	19.50	1.3625	1116.10	43.89	109.430
2010-03-31	33.571395	29.2875	22.29	135.77	170.63	117.0000	1169.43	17.59	1.3510	1112.80	44.41	108.950
2010-04-30	37.298534	30.5350	22.84	137.10	145.20	118.8125	1186.69	22.05	1.3295	1178.25	50.51	115.360
2010-05-31	36.697106	25.8000	21.42	125.46	144.26	109.3690	1089.41	32.07	1.2305	1215.71	49.86	118.881

- 5. Thống kê dạng cuộn (rolling statistics) còn được gọi là các chỉ báo tài chính, thường được những người phân tích biểu đồ và các nhà giao dịch kỹ thuật sử dụng trong tài chính

```
sym = 'AAPL.O'
data = pd.DataFrame(data[sym]).dropna()
data.tail()
```

Date	AAPL.O
2018-06-25	182.17
2018-06-26	184.43
2018-06-27	184.16
2018-06-28	185.50
2018-06-29	185.11

`window = 20 #➔` Xác định cửa sổ; tức là số lượng giá trị chỉ mục cần bao gồm

`data['min'] = data[sym].rolling(window=window).min()#➔` Tính giá trị nhỏ nhất trong cửa sổ trượt

`data['mean'] = data[sym].rolling(window=window).mean()#➔` Tính giá trị trung bình trong cửa sổ trượt.

`data['std'] = data[sym].rolling(window=window).std()#➔` Tính độ lệch chuẩn trong cửa sổ trượt.

`data['median'] = data[sym].rolling(window=window).median()#➔` Tính giá trị trung vị trong cửa sổ trượt

`data['max'] = data[sym].rolling(window=window).max()#➔` Tính giá trị lớn nhất trong cửa sổ trượt.

`data['ewma'] = data[sym].ewm(halflife=0.5, min_periods=window).mean()#➔` Tính trung bình trượt có trọng số theo hàm mũ (exponentially weighted moving average – EWMA), với hệ số suy giảm được xác định theo chu kỳ bán rã (half-life) bằng 0.5.

```
data.dropna().head()
```

Date	AAPL.O	min	mean	std	median	max	ewma
2010-02-01	27.818544	27.437544	29.580892	0.933650	29.821542	30.719969	27.805432
2010-02-02	27.979972	27.437544	29.451249	0.968048	29.711113	30.719969	27.936337
2010-02-03	28.461400	27.437544	29.343035	0.950665	29.685970	30.719969	28.330134
2010-02-04	27.435687	27.435687	29.207892	1.021129	29.547113	30.719969	27.659299
2010-02-05	27.922829	27.435687	29.099892	1.037811	29.419256	30.719969	27.856947

The end of Chapter