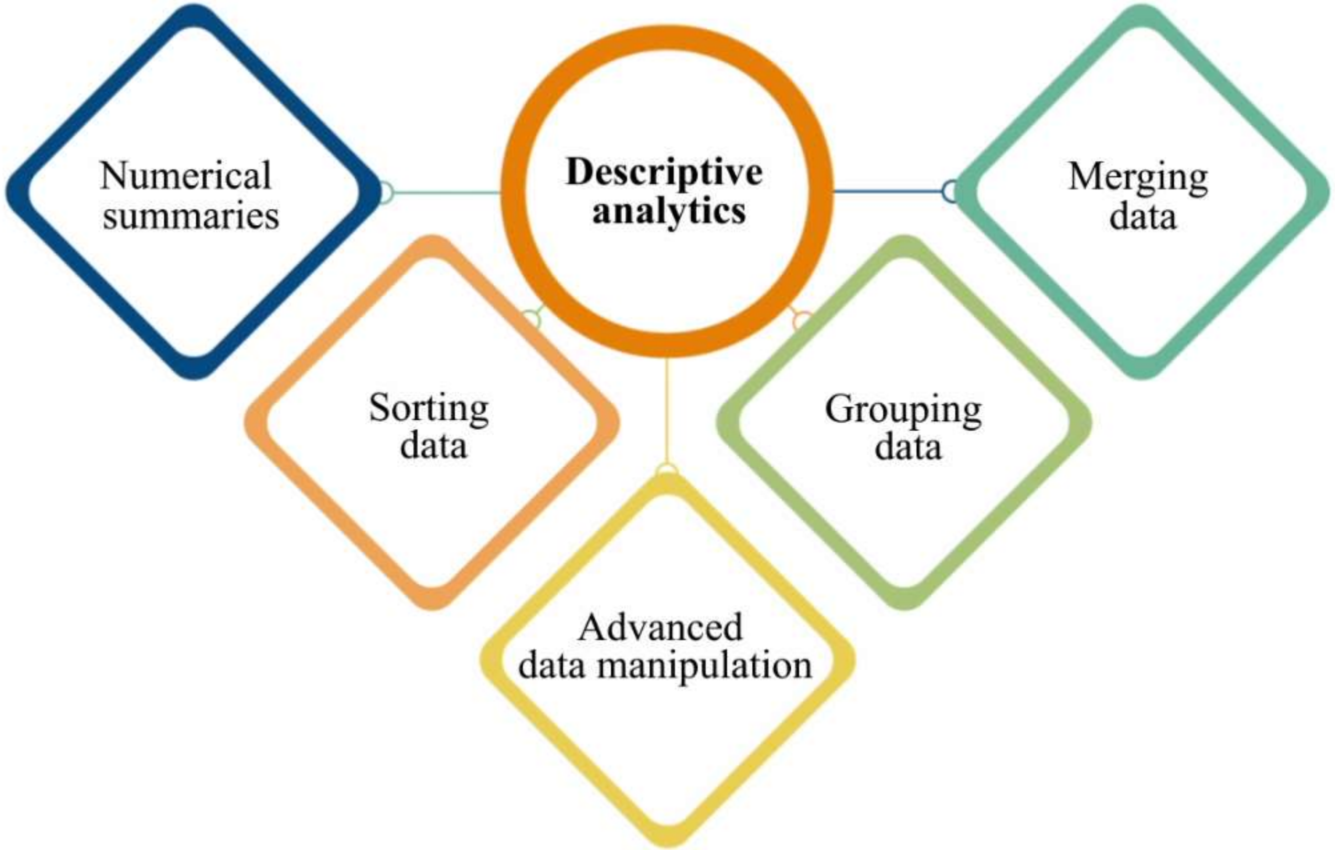
NUS National University of Singapore | NUS Business School

Python for Analytics: Week 7

Descriptive Analytics: Numerical Summary

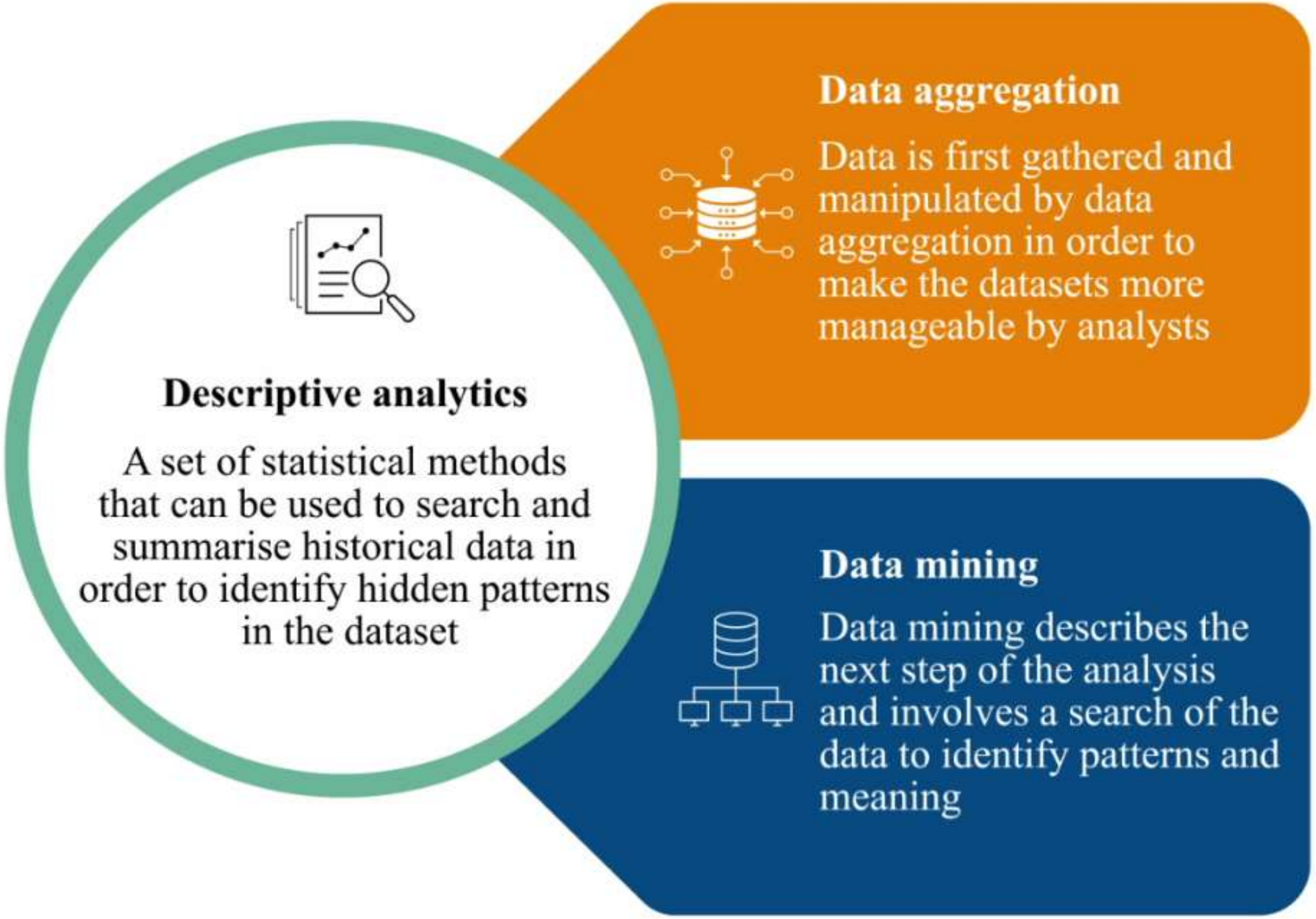
Descriptive Analytics with Numerical Summary: Overview

Module Overview



Introduction to Descriptive Analysis

Major Techniques in Descriptive Analysis



Reading Data in Python

In [38]:
```python
data = pd.read_csv('wage.csv')  # Read data from a file "wage.csv"
data.head(6)                    # Return the first 6 rows
```

Out[38]:

	wage	educ	exper	female	married
0	3.10	11.0	2.0	1.0	0.0
1	3.24	12.0	22.0	1.0	1.0
2	3.00	11.0	2.0	0.0	0.0
3	6.00	8.0	44.0	0.0	1.0
4	5.30	12.0	7.0	0.0	1.0
5	8.75	16.0	9.0	0.0	1.0

	wage	educ	exper	female	married
0	3.10	11.0	2.0	1.0	0.0
1	3.24	12.0	22.0	1.0	1.0
2	3.00	11.0	2.0	0.0	0.0
3	6.00	8.0	44.0	0.0	1.0
4	5.30	12.0	7.0	0.0	1.0
...	...	...	...	...	...
521	15.00	16.0	14.0	1.0	1.0
522	2.27	10.0	2.0	1.0	0.0
523	4.67	15.0	13.0	0.0	1.0
524	11.56	16.0	5.0	0.0	1.0
525	3.50	14.0	5.0	1.0	0.0

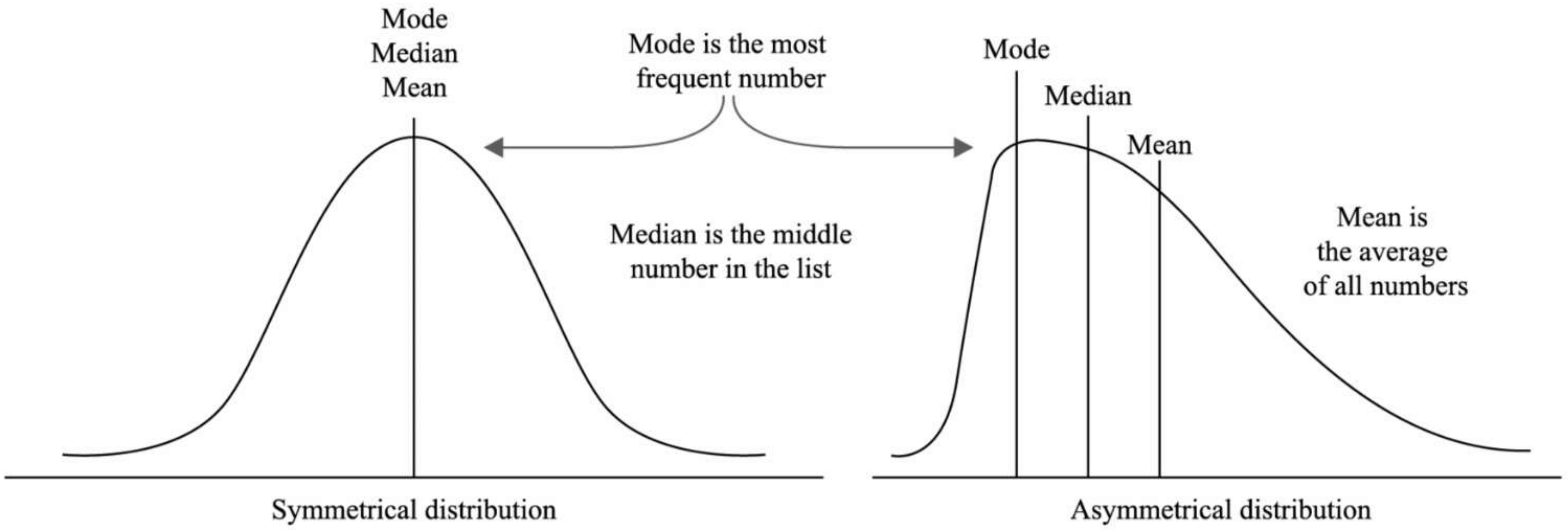
526 rows × 5 columns

Measures of Centre

Aspects of Centre Measures

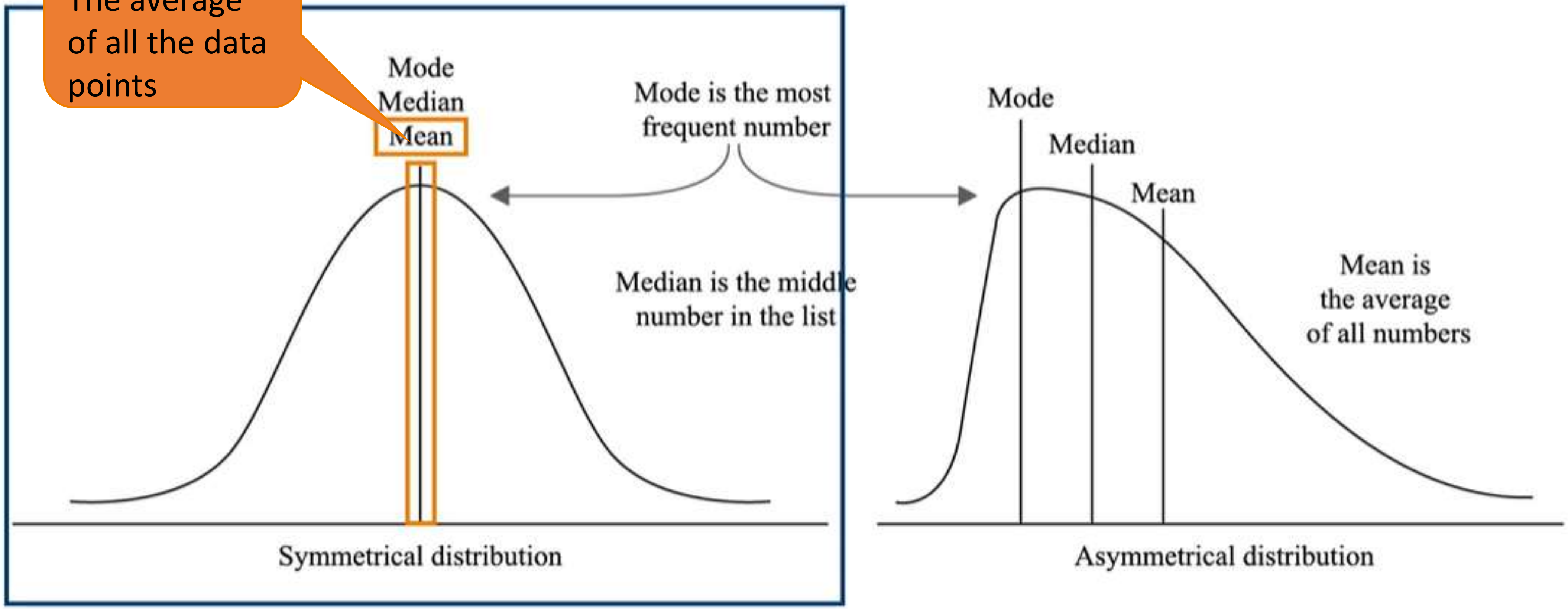
Key centre majors are:

- Mean
- Median
- Mode

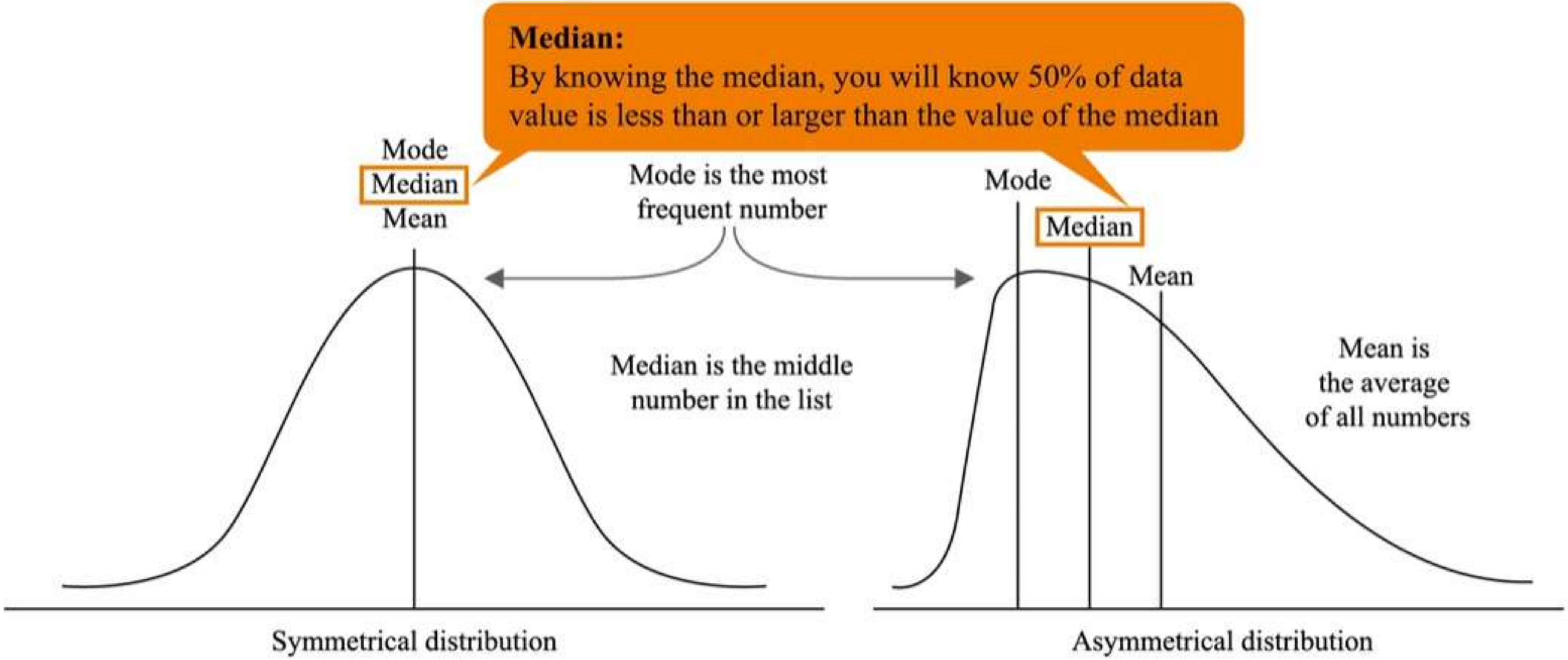


Aspects of Centre Measures

Mean:
The average of all the data points



Aspects of Centre Measures



Aspects of Centre Measures

Right-skewed distribution:

Most of the time, the mean value is larger than the median

Left-skewed distribution:

Most of the time, the median value is greater than the mean

Using the Mean Method: Pandas Series

Mean method calculates the summary statistics

In [39]:
```
print(data.mean( ))
print(type(data.mean( ) ) )
```

```
Wage        5.896103
Educ       12.562738
Exper      17.017110
Female      0.479087
Married     0.608365
dtype: float64
```

```
<class 'pandas.core.series.Series'>
```

	Wage	Educ	Exper	Female	Married
0	3.10	11.0	2.0	1.0	0.0
1	3.24	12.0	22.0	1.0	1.0
2	3.00	11.0	2.0	0.0	0.0
3	6.00	8.0	44.0	0.0	1.0
...	...	...	...	...	...
522	2.27	10.0	2.0	1.0	0.0
523	4.67	15.0	13.0	0.0	1.0
524	11.56	16.0	5.0	0.0	1.0
525	3.50	14.0	5.0	1.0	0.0

data.mean()	5.896	12.563	17.017	0.479	0.608

Using the Mean Method: Pandas Series

In [11]:

```
print (wage_data.mean( ) )
print (type(wage_data.mean( ) ) )
```

Wage	5.896103
Educ	12.562738
Exper	17.017110
Female	0.479087
Married	0.608365

dtype: float64

<class 'pandas.core.series.Series'>

$$\hat{p} = \frac{1}{N} \sum_{i=1}^{N} X_i$$

Total number of cases

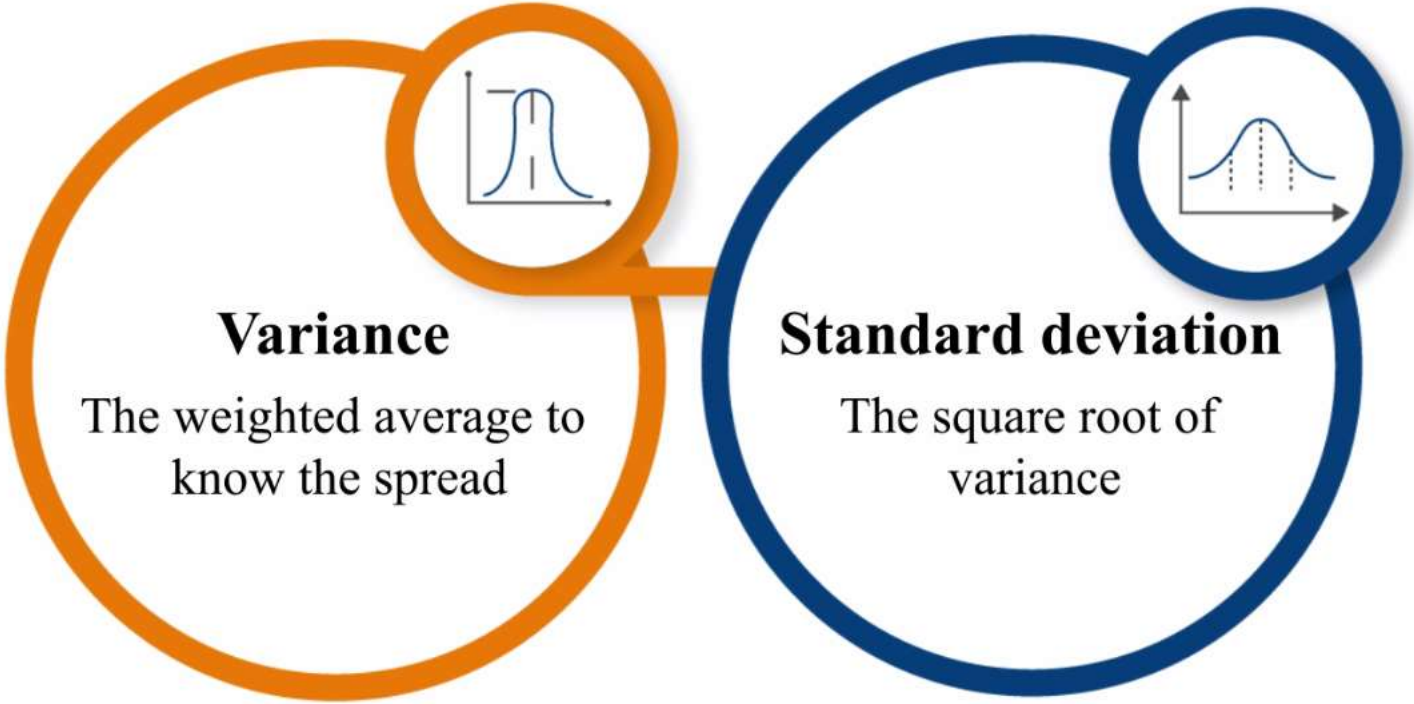
Number of females

The dummy variable is:

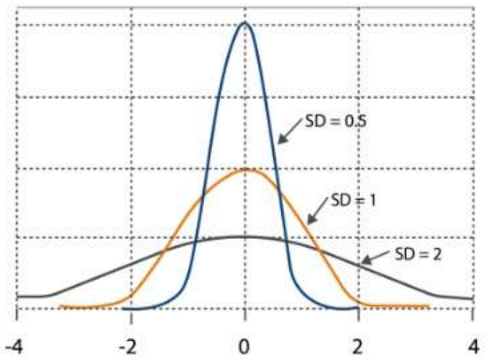
$$X_i = \begin{cases} 1, \text{ if female} \\ 0, \text{ if male} \end{cases}$$

Measures of Variance and Extreme Points

Measures of Variation



Standard deviation



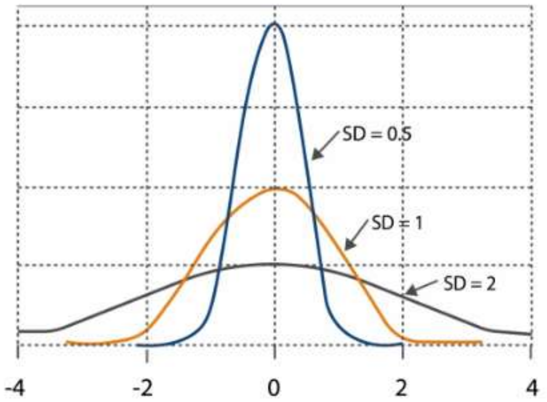

```
In [41]:  data.std()          # Sample Standard deviation of each column

Out[41]:  wage        3.693086
          educ        2.769022
          exper      13.572160
          female      0.500038
          married     0.488580
          dtype: float64
```

Variance

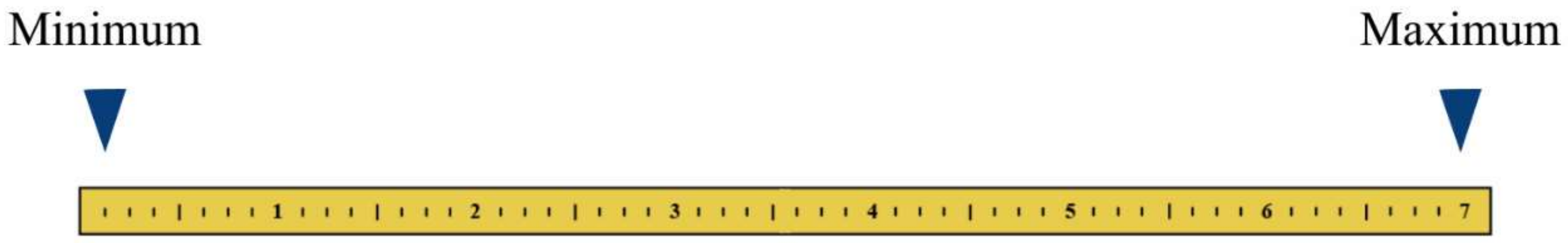
Variance:

$$\hat{\sigma}^2 = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \bar{X})^2$$




```
In [42]: data.var()        # Sample Variance of each column
Out[42]: wage         13.638884
         educ          7.667485
         exper       184.203516
         female        0.250038
         married       0.238711
         dtype: float64
```

Extreme points




```
In [43]:  1  data.max()            # Maximum value of each column
Out[43]: wage        24.98
         educ        18.00
         exper       51.00
         female       1.00
         married      1.00
         dtype: float64
```

Describe Method

Describe Method

In [45]:
```
wage_summary = data.describe()   # Obtain the key descriptive measures
wage_summary                     # Display these measures as a table
```

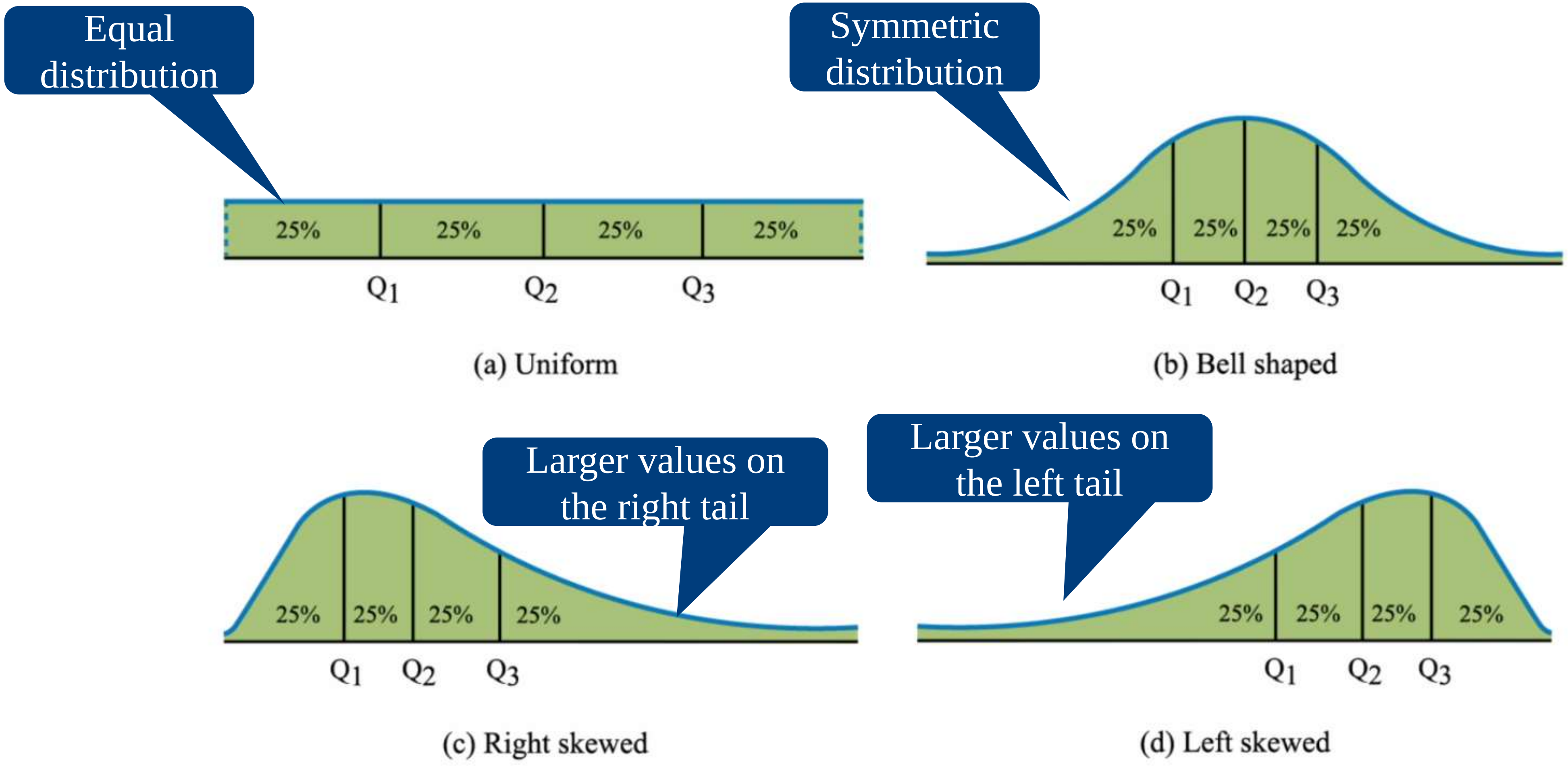
Out[45]:

	wage	educ	exper	female	married
count	526.000000	526.000000	526.00000	526.000000	526.000000
mean	5.896103	12.562738	17.01711	0.479087	0.608365
std	3.693086	2.769022	13.57216	0.500038	0.488580
min	0.530000	0.000000	1.00000	0.000000	0.000000
25%	3.330000	12.000000	5.00000	0.000000	0.000000
50%	4.650000	12.000000	13.50000	0.000000	1.000000
75%	6.880000	14.000000	26.00000	1.000000	1.000000
max	24.980000	18.000000	51.00000	1.000000	1.000000

Q1
Q2
Q3

Quartiles: values that divide a dataset into quarters

Describe Method



Python Demo: Describe Method

Centres, Variations and Extreme Points

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

In [2]:
```
data_dict = {'wage': [3.10, 3.24, 3.00, 6.00, 5.30, 8.75],
             'educ': [11.0, 12.0, 11.0, 8.0, 12.0, 16.0],
             'exper': [2.0, 22.0, 2.0, 44.0, 7.0, 9.0],
             'female': [1.0, 1.0, 0.0, 0.0, 0.0, 0.0],
             'married': [0.0, 1.0, 0.0, 1.0, 1.0, 1.0]}

data = pd.DataFrame(data_dict)     # DataFrame constructor
data                               # Display the DataFrame
```

Out[2]:

	wage	educ	exper	female	married
0	3.10	11.0	2.0	1.0	0.0
1	3.24	12.0	22.0	1.0	1.0
2	3.00	11.0	2.0	0.0	0.0
3	6.00	8.0	44.0	0.0	1.0
4	5.30	12.0	7.0	0.0	1.0
5	8.75	16.0	9.0	0.0	1.0

In [3]:
```
data.mean()             # Mean value of each column
```

Out[3]:
```
wage        4.898333
educ       11.666667
exper      14.333333
female      0.333333
married     0.666667
dtype: float64
```

In [4]:
```
type(data.mean())  # Show the data type of the results
```

Out[4]:
```
pandas.core.series.Series
```

Centres, Variations and Extreme Points

Use indexing to access computed mean values

```
In [4]: type(data.mean())  # Show the data type of the results
Out[4]: pandas.core.series.Series
```

```
In [5]: data.median()         # Median value of each column
Out[5]: wage        4.27
        educ       11.50
        exper       8.00
        female      0.00
        married     1.00
        dtype: float64
```

```
In [6]: type(data.median())  # Show the data type of the results
Out[6]: pandas.core.series.Series
```

Please note that the mean value of each column is stored in a `pandas.Series` , where the index labels are variable names, rather than a sequence of integers.

Also notice that in the case of the 0-1 categorical variable "female", the mean value of 0.333 is the proportion of observations in the dataset labeled as "female". The same concept can be applied to the variable "married" as well.

Similarly, the measures of standard deviations and variances can be calculated by the corresponding methods.

Centres, Variations and Extreme Points

```
In [7]: data.std()        # Sample Standard deviation of each column

Out[7]: wage        2.271479
        educ        2.581989
        exper      16.280868
        female      0.516398
        married     0.516398
        dtype: float64
```

```
In [8]: data.var()        # Sample Variance of each column

Out[8]: wage         5.159617
        educ         6.666667
        exper      265.066667
        female       0.266667
        married      0.266667
        dtype: float64
```

The maximum and minimum points in the dataset can also be found.

```
In [9]: data.max()        # Maximum value of each column

Out[9]: wage        8.75
        educ       16.00
        exper      44.00
        female      1.00
        married     1.00
        dtype: float64
```

describe() Method

Method `describe`

For `pandas.DataFrame` and `pandas.Series` , the method `describe` is a convenient tool to summarize some key measures altogether.

In [11]:
```
wage_summary = data.describe()  # Obtain the key descriptive measures
wage_summary                    # Display these measures as a table
```

Out[11]:

	wage	educ	exper	female	married
count	6.000000	6.000000	6.000000	6.000000	6.000000
mean	4.898333	11.666667	14.333333	0.333333	0.666667
std	2.271479	2.581989	16.280868	0.516398	0.516398
min	3.000000	8.000000	2.000000	0.000000	0.000000
25%	3.135000	11.000000	3.250000	0.000000	0.250000
50%	4.270000	11.500000	8.000000	0.000000	1.000000
75%	5.825000	12.000000	18.750000	0.750000	1.000000
max	8.750000	16.000000	44.000000	1.000000	1.000000

The variable `wage_summary` is a `pandas.DataFrame` table where the index labels are the names of the descriptive measures.

Note that rows `25%` , `50%` , and `75%` represent the first (Q1), second(Q2), and third quartiles(Q3), respectively. The value Q3 - Q1 is called the interquartile range (IQR).

Sorting Data

Sorting Data

Sort numerical column in a DataFrame –

df.sort_values(by = 'col1', ascending = True):

Two input arguments

- **by:** Sort data by a specific column
- **ascending:** Sort data in ascending order or descending order by providing **true** value

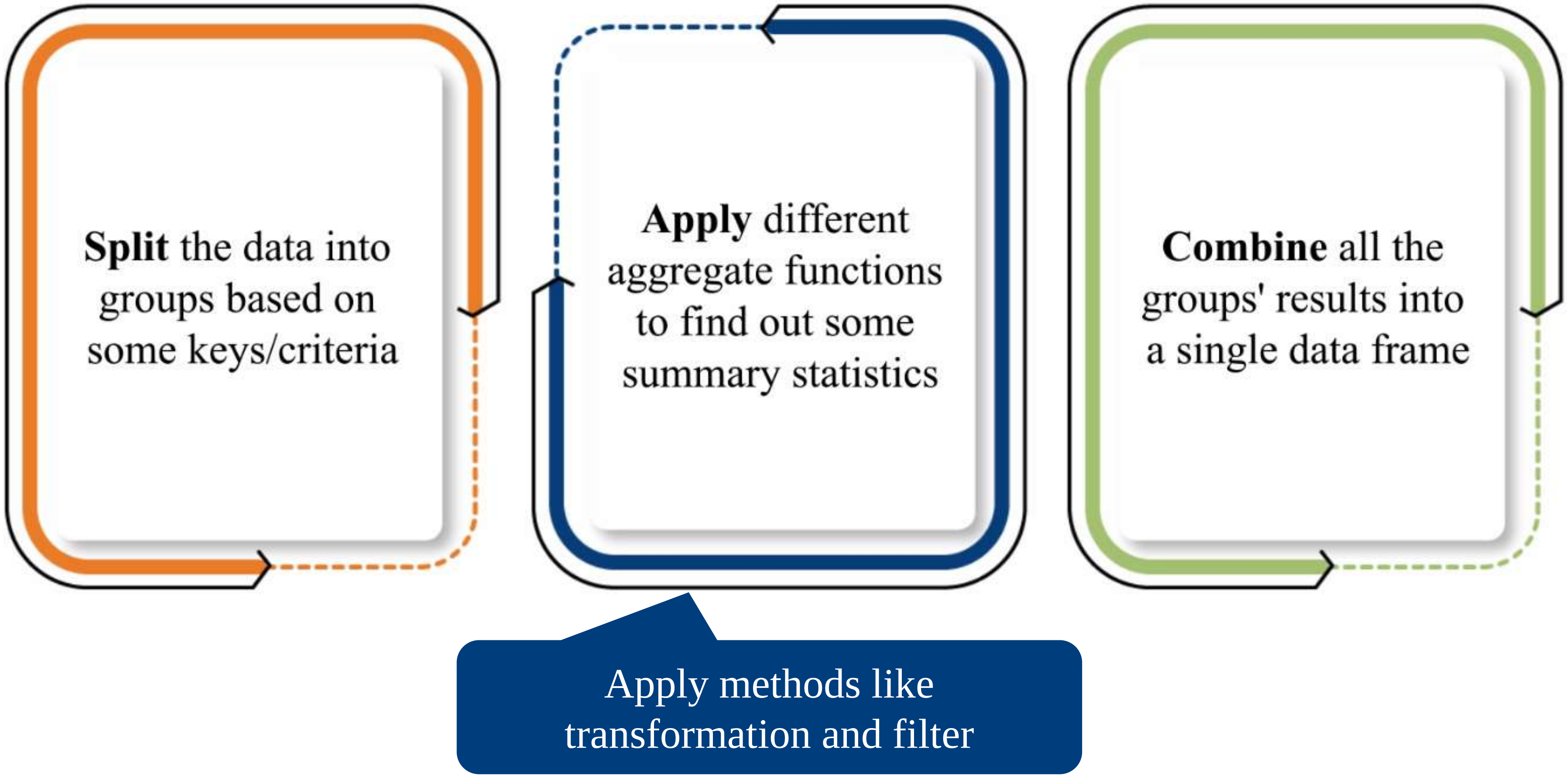
Sort multiple columns at the same time –

df.sort_values(by = ['col1', 'col2'], ascending = [True, False]):

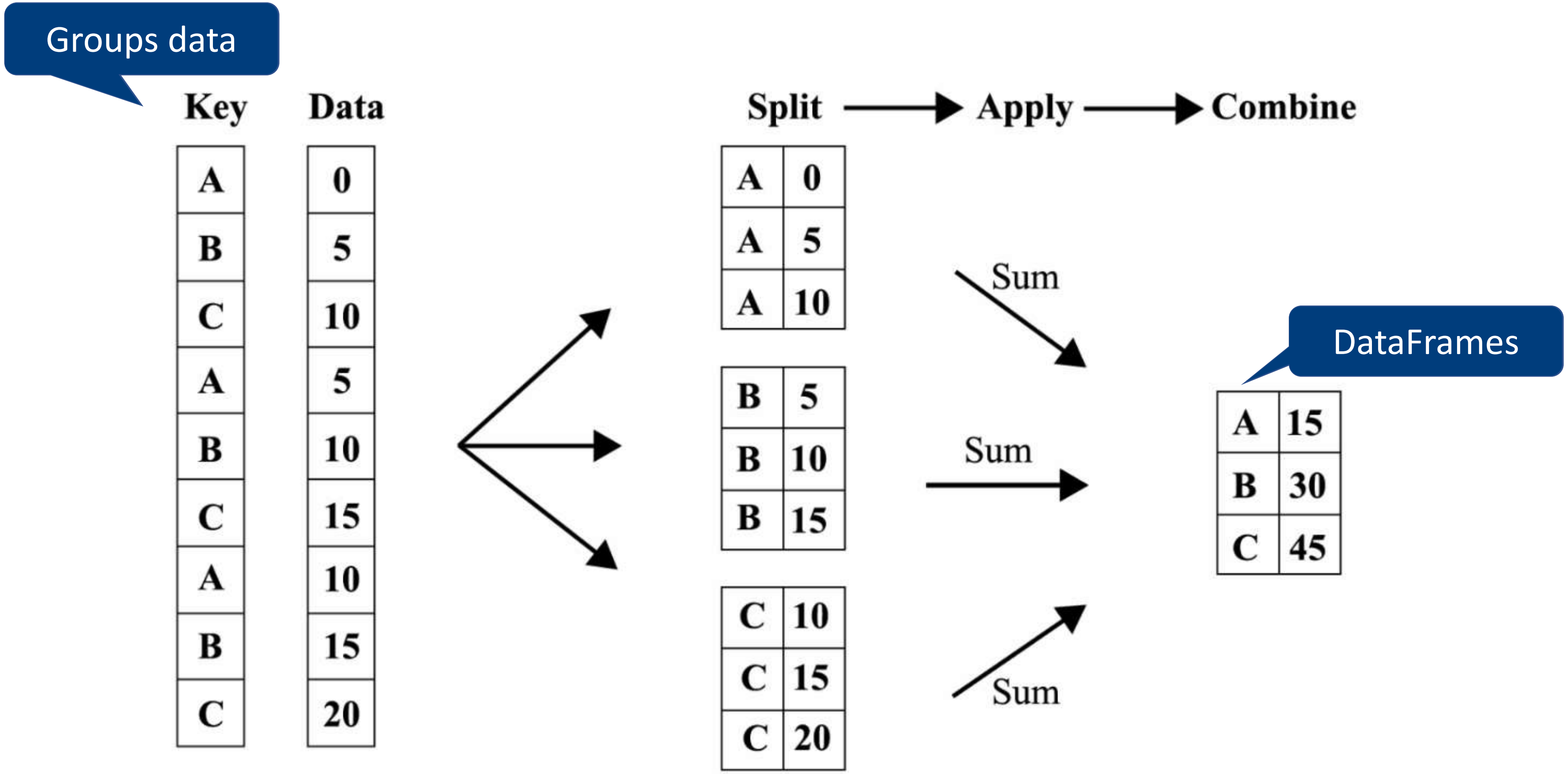
- Provide a list of the columns in **by** argument
- Specify a Boolean list

Advanced Data Manipulation Using Pandas: Introduction

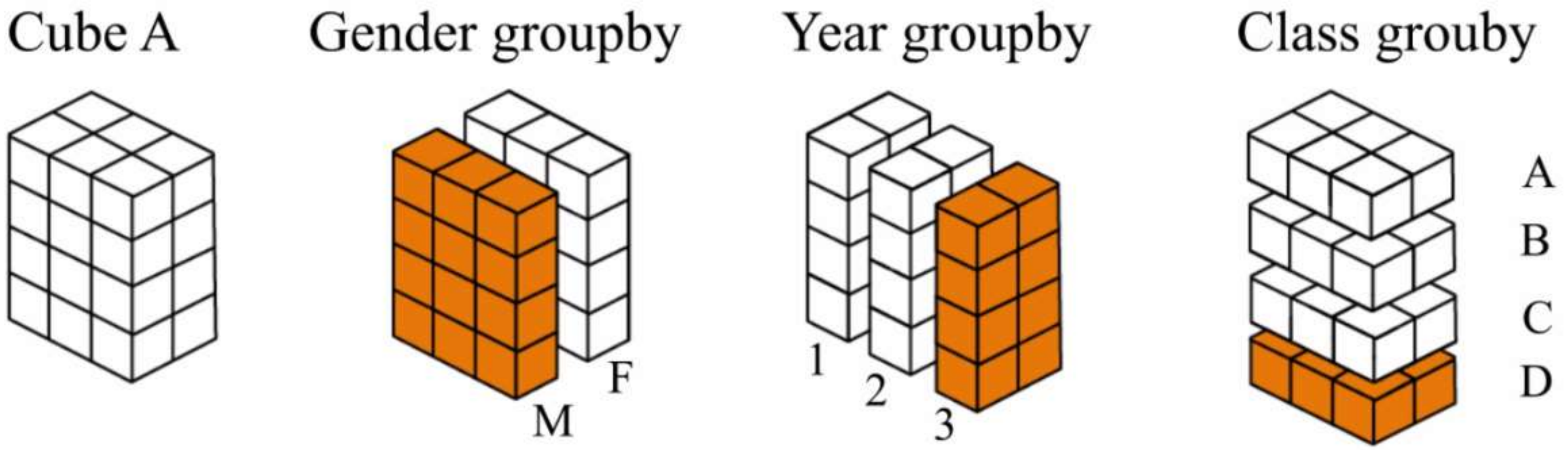
Group-Wise Operators



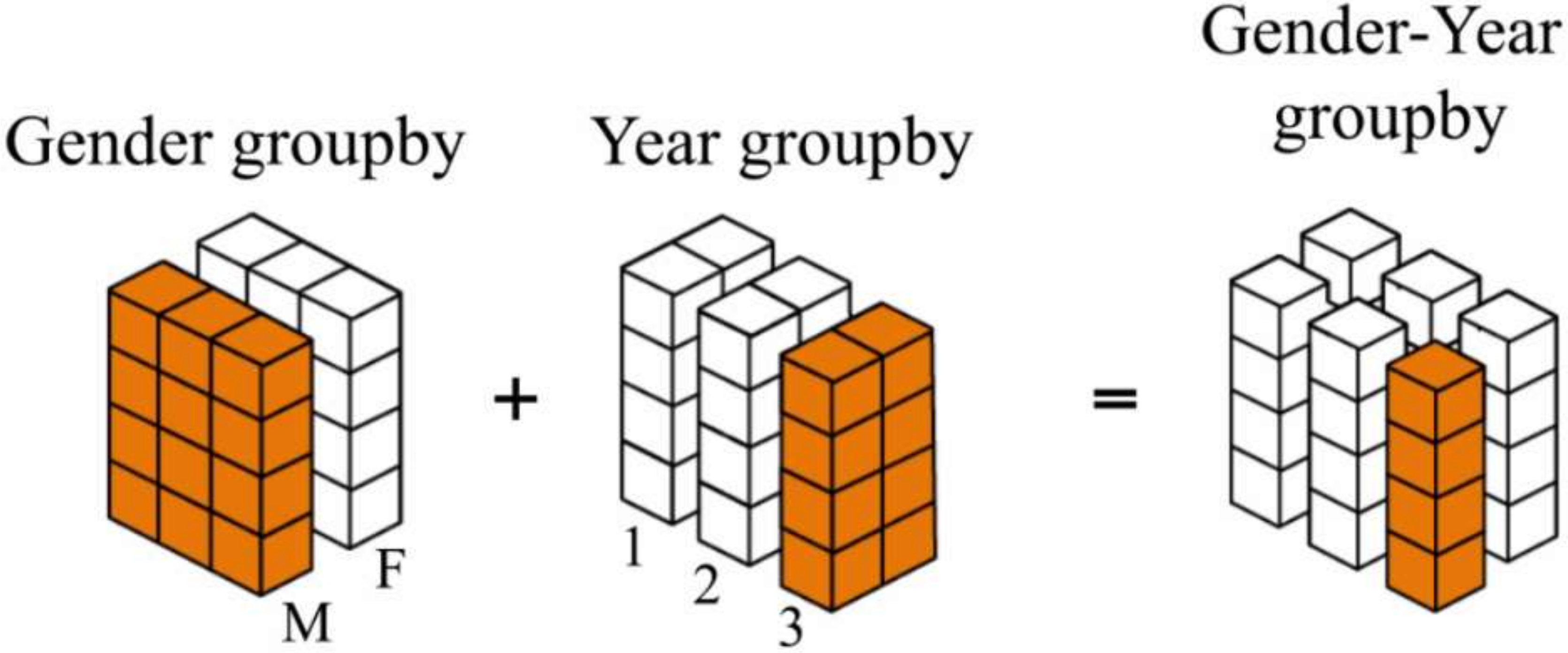
Group-Wise Operators



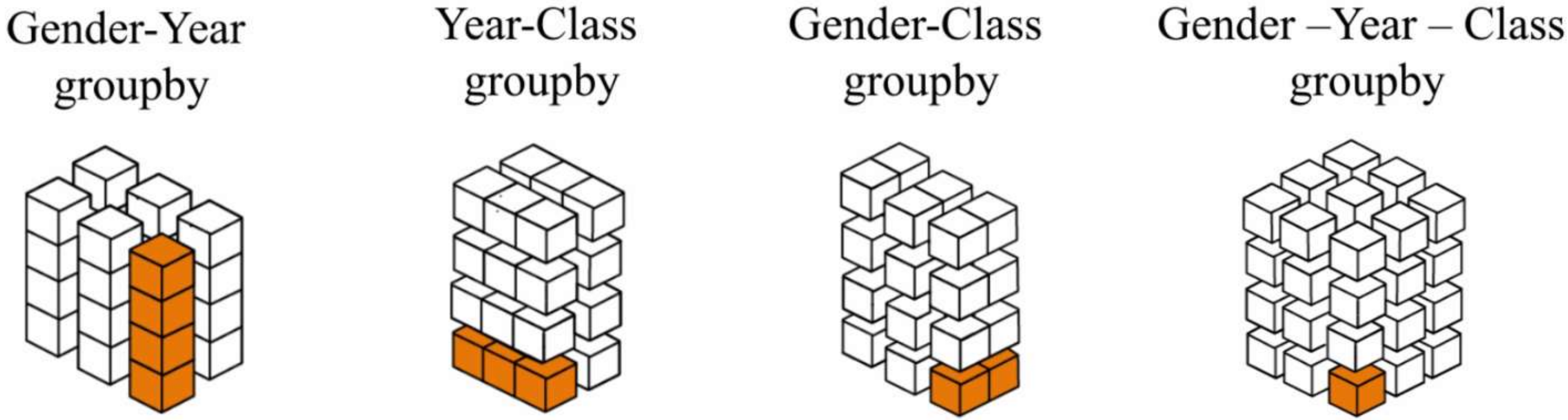
Split by Groupby()



Split by Groupby()



Split by Groupby()



Groupby() Attributes

Downsize method

- Output the number of rows in each group
- ***size()***

Assess each groupby object

df_grouped['Score']:

Assess multiple columns for each group

df_grouped[['Class', 'Score']]:

Click to add text

Assess all groups

df_grouped.groups:

Assess specific group

df_grouped.get_group(('M', '2')):

Python Demo: Data Sorting

Sort Data

For numerical data, sorting in ascending order is important to help understand percentile values of the data. In Pandas, we have a useful method to sort values in a DataFrame or Series. The Pandas' method is `sort_values`

```
In [3]: data.sort_values(by = 'wage', ascending = False)
Out[3]:
```

	wage	educ	exper	female	married
5	8.75	16.0	9.0	0.0	1.0
3	6.00	8.0	44.0	0.0	1.0
4	5.30	12.0	7.0	0.0	1.0
1	3.24	12.0	22.0	1.0	1.0
0	3.10	11.0	2.0	1.0	0.0
2	3.00	11.0	2.0	0.0	0.0

```
In [4]: data
Out[4]:
```

	wage	educ	exper	female	married
0	3.10	11.0	2.0	1.0	0.0
1	3.24	12.0	22.0	1.0	1.0
2	3.00	11.0	2.0	0.0	0.0
3	6.00	8.0	44.0	0.0	1.0
4	5.30	12.0	7.0	0.0	1.0
5	8.75	16.0	9.0	0.0	1.0

Sort Data

```
In [6]: data.sort_values(by = 'wage', ascending = False, inplace = True)
        data
```

Out[6]:

	wage	educ	exper	female	married
5	8.75	16.0	9.0	0.0	1.0
3	6.00	8.0	44.0	0.0	1.0
4	5.30	12.0	7.0	0.0	1.0
1	3.24	12.0	22.0	1.0	1.0
0	3.10	11.0	2.0	1.0	0.0
2	3.00	11.0	2.0	0.0	0.0

```
In [7]: data.sort_values(by = ['married','wage'], ascending = [False,True])
```

Out[7]:

	wage	educ	exper	female	married
1	3.24	12.0	22.0	1.0	1.0
4	5.30	12.0	7.0	0.0	1.0
3	6.00	8.0	44.0	0.0	1.0
5	8.75	16.0	9.0	0.0	1.0
2	3.00	11.0	2.0	0.0	0.0
0	3.10	11.0	2.0	1.0	0.0

Sort Data

```
In [8]: data
```

Out[8]:

	wage	educ	exper	female	married
5	8.75	16.0	9.0	0.0	1.0
3	6.00	8.0	44.0	0.0	1.0
4	5.30	12.0	7.0	0.0	1.0
1	3.24	12.0	22.0	1.0	1.0
0	3.10	11.0	2.0	1.0	0.0
2	3.00	11.0	2.0	0.0	0.0

```
In [9]: data.sort_values(by = ['married','wage'], ascending = [False,True], inplace = True)
```

```
In [10]: data
```

Out[10]:

	wage	educ	exper	female	married
1	3.24	12.0	22.0	1.0	1.0
4	5.30	12.0	7.0	0.0	1.0
3	6.00	8.0	44.0	0.0	1.0
5	8.75	16.0	9.0	0.0	1.0
2	3.00	11.0	2.0	0.0	0.0
0	3.10	11.0	2.0	1.0	0.0

Group-wise Operations: Aggregation, Transformation and Filtration

Group-wise Operations

Aggregation

- The commonly used aggregation operation
  - size()
  - mean()
  - sum()
  - std()

Aggregation

- Input: a chunk of the DataFrame
- Output: some summary statistics

Lambda Function

Special function in Pandas

Can be viewed as a single-use function or anonymous function

Contradicts tradition function
- Does not have a function name
- Can have multiple input arguments
- Only has one-line expression as the return value

Example: a = lambda x, y: x+y

- Key word: **'lambda'** to start a Lambda Function
- Input arguments: **x** and **y** (Lambda Function can take any number of arguments)
- One-line function body and return value: **x+y**
- Type: **a(3,5)**

Group-wise Operators

Transformation

- Modify a column in each group's DataFrame by applying transform method
- Input: A chunk of the table
- Output: The modified table (must be of the same shape as the input)

Example: Group-wise normalisation

- zscore = lambda x: (x - x.mean()) / x.std()
- data.groupby([‘Class’])[[‘Score’]].transform(zscore)

Group-wise Operators

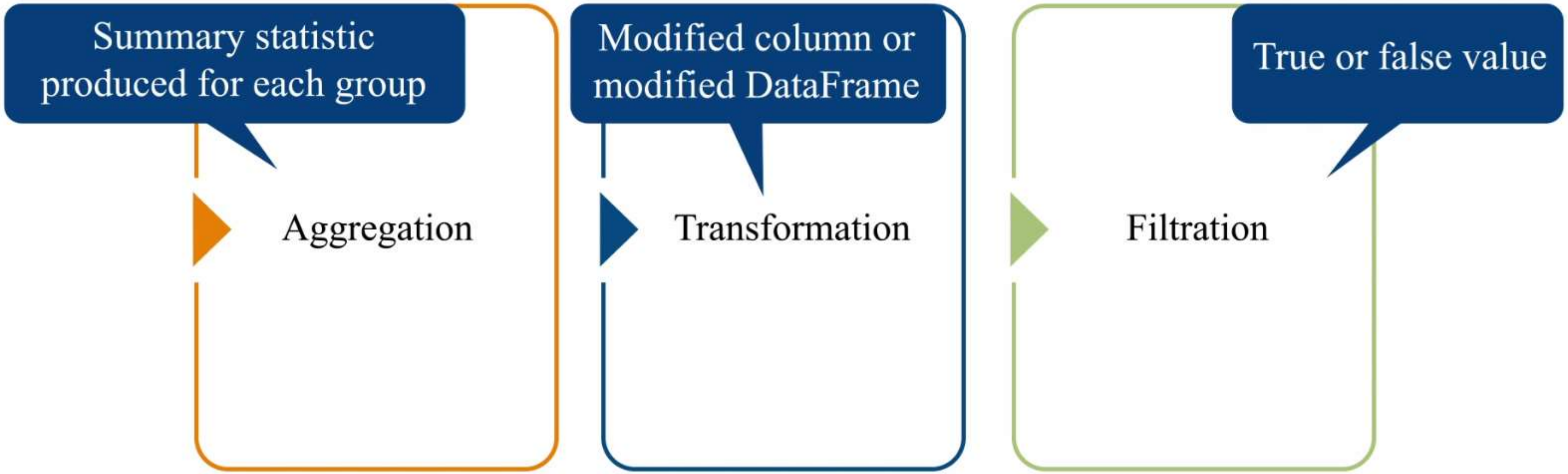
Filtration

- Filter groups by filter(func)
- Input: A chunk of the table
- Output: True/False

Example: Select those classes with a class average score > 80

- high_class_avg = lambda x: x ['Score'].mean() > 80
- data.groupby('Class').filter (high_class_avg)

Group-wise Operators



apply(func)

- Input: A chunk of the table
- Output: An arbitrary list/dict/DataFrame

Python Demo: Group-Wise Operations

Group-Wise Operations

Background: In the following, we will use Singapore's 4-Digits data set to exemplify the power of group-wise operations using Pandas. The 4-Digits (abbreviation: 4-D) is a lottery in Singapore. Individuals play by choosing any number from `0000` to `9999` . Then, twenty-three winning numbers are drawn each time. If one of the numbers matches the one that the player has bought, a prize is won. A draw is conducted to select these winning numbers. 4-Digits is a fixed-odds game. There are five prize categories: **1st Prize**, **2nd Prize**, **3rd Prize**, **Starter Prizes** and **Consolation Prizes**. We would like to know prize-specific summary of the 4-D lottery.

Load data

The following data file "4D_results_long.csv" has been generated. First, please load data into Python.

```
In [11]: data = pd.read_csv('4D_results_long.csv')
```

```
In [ ]: data.head(10)
```

```
In [ ]: data.shape
```

Column Summary

value_counts()

```
In [ ]: # count the instances of each prize type
        data['prize_type'].value_counts()
```

```
In [ ]: # show the Top 10 most frequent winning number in the dataset
        data['number'].value_counts().head(10)
```

describe()

```
In [16]: # summary statistics of all the winning numbers
         data['number'].describe()

Out[16]: count    10810.000000
         mean      4957.073636
         std       2905.037198
         min          0.000000
         25%       2400.250000
         50%       4964.000000
         75%       7479.750000
         max       9999.000000
         Name: number, dtype: float64
```

```
In [ ]: # summary of a categorical variable
        data['weekday'].describe()
```

GroupBy

In [18]:
```
data_by_prizetype = data.groupby(['prize_type'])
```

In [ ]:
```
data_by_prizetype
```

In [20]:
```
data_by_prizetype.size()
```

Out[20]:
```
prize_type
1st             470
2nd             470
3rd             470
consolation    4700
starter        4700
dtype: int64
```

In [ ]:

In [ ]:
```
#DataFrame column selection in GroupBy
data_by_prizetype['number']
```

In [ ]:
```
data_by_prizetype[['number']]
```

In [ ]:

In [ ]:
```
data_by_prizetype.groups
```

In [ ]:
```
# Selecting a group
data_by_prizetype.get_group('2nd').head(10)
```

GroupBy

```
In [23]: data_by_prizetype.groups
```

```
In [ ]: # Selecting a group
        data_by_prizetype.get_group('2nd').head(10)
```

```
In [ ]:
```

```
In [ ]: # Iterating through groups
        for name, group in data_by_prizetype:
            print(name)
            print(group.shape)
            print(type(group))
```

Aggregation

Mean of winning numbers for each prizetype

In [ ]:
```
# as series
data_by_prizetype['number'].mean()
```

In [ ]:
```
# as dataframe with the group variable as index
data_by_prizetype[['number']].mean()
```

In [ ]:
```
# as dataframe without index
data_by_prizetype[['number']].mean().reset_index()
```

In [ ]:
```
# avoid index when grouping
data.groupby('prize_type', as_index=False)[['number']].mean()
```

Count of prize_type for each draw_no

In [ ]:
```
data.groupby('draw_no')['prize_type'].value_counts()
```

Aggregation by agg()

 # Applying multiple functions at once

```
In [ ]: import numpy as np
        # Min/Median/Max of number for each weekday-prize_type combination
        # group by 'weekday' and 'prize_type' and extract 'number' column
        weekday_prize_gpby = data.groupby(['weekday', 'prize_type'])
```

```
In [ ]: weekday_prize_gpby
```

```
In [ ]: weekday_prize_gpby[['number']].agg([np.min, np.median, np.max])
```

Applying different functions to different columns

```
In [ ]: weekday_prize_gpby.agg({'number': np.mean, 'date':np.max})
```

agg with a customized function

```
In [36]: year_gpby = data.groupby('year')
```

```
In [ ]: year_gpby['number'].agg(lambda x: sum(x > 9900))
```

```
In [ ]: # count how many times that the prize number > 9900 for each year
        year_gpby[['number']].agg(lambda x: sum(x > 9900))
```

Aggregation by agg()

Transformation

Normalize number by year

```
In [ ]: year_gpby['number'].transform(lambda x: (x - x.mean())/x.std())
```

Filtration

Focus on first prizes, select data only in weeks with 3 draws

```
In [ ]: filt_1st = (data['prize'] == 'first_prize')
        data_1st = data.loc[filt_1st,:].copy()        # Create a new data set with 1st prize data only
```

```
In [ ]: year_wkno_1st_gpby = data_1st.groupby(['year', 'week_no'])
```

```
In [ ]: year_wkno_1st_gpby.filter(lambda x: len(x) > 2)
```

Apply

For each year and each prize_type, find the draw with smallest number and the one with the largest number

```
In [ ]: def draw_min_max(x):
            i = x['number'].idxmin()
            j = x['number'].idxmax()
            return pd.concat([x.loc[i, ['draw_no', 'number']], x.loc[j, ['draw_no', 'number']]])
```

```
In [ ]: data.groupby(['year', 'prize_type']).apply(draw_min_max)
```

Joining and Merging Data: Introduction

Introduction to Joining and Merging Data

- Combine information from different sources
- Form a complete dataset
- Apply the dataset to answer business or analytics questions

Pandas methods to combine datasets

- Concat
- Merge

Concat and Merge

Concat Method

- Concat(): Combine DataFrames row-wise or column-wise by index
- Concat method is versatile
- One-to-one relationship only

Simple Concatenation: Example

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

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

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Column-Wise Stacking: Example

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3
6	NaN	NaN	NaN	NaN	B6	D6	F6
7	NaN	NaN	NaN	NaN	B7	D7	F7

Inner Join: Example

Inner join: Intersection of two set of indices

pd.concat([df1, df4], axis=1, join='inner')

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

	A	B	C	D	B	D	F
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3

Merge Method

- Merge(): Column-wise stacking only (i.e., axis=1)
- One-to-many and many-to-many on top of one-to-one
- Align by multiple column (instead of by an index)
- Reference: http://pandas.pydata.org/pandas-docs/stable/merging.html

Merge Method: Example

left

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

right

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

Result

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

Data Merge with Multiple Keys

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

Outer Join Method: Example

Outer join: **Union** of the keys, missing values filled with NaN

pd.merge(left, right, on=['key1', 'key2'], how='outer')

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN
5	K2	K0	NaN	NaN	C3	D3

Merge Method: Outer Join Left Indices

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2
3	K2	K0	NaN	NaN	C3	D3

Python Demo: Concat and Merge

Data Merge in Pandas

Background: In the following, we will use Google Public Data to illustrate Pandas' `concat` and `merge` methods. Please load in some data files first. We want to create a complete data set that can be used to explore the relationship between **fertility rate** and a country's **GDP per capita**. At the same time, two other demographic factors are also included: **household expenditure** and **population density of the country**. We will not do the detailed analysis here; instead, we just demonstrate how to prepare the required data using data concatenation and merging.

In [ ]:
```
import pandas as pd
data_fer = pd.read_csv("GooglePublicData_fer.csv")
data_gdp = pd.read_csv("GooglePublicData_gdp.csv")
data_exp = pd.read_csv("GooglePublicData_exp.csv")
data_pop = pd.read_csv("GooglePublicData_pop.csv")
data_country = pd.read_csv("GooglePublicData_country.csv")
```

Data Merge in Pandas

Understand the relationship between fertility rate and GDP per capita in 2015

- Plot the following metrics for Year 2015
  - Y: Health / Fertility Rate
  - X: Economic Policy and Debt / GDP per capita (constant 2000 USD)
  - Size: Environment / Population density (people per sq. km of land area)
  - Color: Economic Policy and Debt / Household final consumption expenditure per capita (constant 2000 USD)

1. Extract data for Year 2015

In [ ]:
```
data_fer2015 = data_fer[data_fer['year'] == 2015]
data_fer2015
```

In [ ]:
```
data_gdp2015 = data_gdp[data_gdp['year'] == 2015]
data_gdp2015
```

In [ ]:
```
data_exp2015 = data_exp[data_exp['year'] == 2015]
data_exp2015
```

In [ ]:
```
data_pop2015 = data_pop[data_pop['year'] == 2015]
data_pop2015
```

Data Merge in Pandas

2. Merge different data sets

The four datasets have different numbers of columns and each is for the data in 2015. To combine the four datasets, we can concatenate them with "outer" join.

In [ ]:
```
# By default, pd.concat will do row-wise stacking with outer join
data2015 = pd.concat([data_fer2015, data_gdp2015, data_exp2015, data_pop2015])
data2015
```

In [ ]:
```
data2015_inner = pd.concat([data_fer2015, data_gdp2015, data_exp2015, data_pop2015], join = "inner")
data2015_inner
```

In [ ]:
```
# delete the redundant year column
data2015 = data2015.drop(columns = "year")
data2015
```


In [ ]:
```
# By default, pd.concat will do row-wise stacking with outer join
data2015 = pd.concat([data_fer2015, data_gdp2015, data_exp2015, data_pop2015])
data2015
```

In [ ]:
```
data2015_inner = pd.concat([data_fer2015, data_gdp2015, data_exp2015, data_pop2015], join = "inner")
data2015_inner
```

In [ ]:
```
# delete the redundant year column
data2015 = data2015.drop(columns = "year")
data2015
```

Data Merge in Pandas

The plot function expects each data point on a row, which represents a country's results. Here we just need to do a transpose.

In [56]:
```
data2015 = data2015.T
data2015.head(3)
```

Re-name columns to have meaningful headers

In [57]:
```
data2015.columns = ['FertilityRate', 'GDPperCapita', 'HouseholdExpense', 'PopulationDensity']
data2015.head(3)
```

Data Merge in Pandas

Merge all into one complete dataset

Merge the four imported tables into one such that each subject is one country-year pair, and the four measures are on four separate columns.

country_id	year	fer	gdp	exp	pop

In [ ]:
```
data_fer_long = pd.read_csv("GooglePublicData_fer_long.csv")
data_gdp_long = pd.read_csv("GooglePublicData_gdp_long.csv")
data_exp_long = pd.read_csv("GooglePublicData_exp_long.csv")
data_pop_long = pd.read_csv("GooglePublicData_pop_long.csv")
```

In [ ]:
```
data_fer_long.head()
```

In [ ]:
```
data_gdp_long.head()
```

In [ ]:
```
data_all = pd.merge(left=data_fer_long, right=data_gdp_long, on=['country_id', 'year'])
data_all.head()
```

In [ ]:
```
data_all = pd.merge(left=data_all, right=data_exp_long, on=['country_id', 'year'])
data_all = pd.merge(left=data_all, right=data_pop_long, on=['country_id', 'year'])
data_all.head()
```

Data Merge in Pandas

Merge all into one complete dataset

Merge the four imported tables into one such that each subject is one country-year pair, and the four measures are on four separate columns.

country_id	year	fer	gdp	exp	pop

In [ ]:
```
data_fer_long = pd.read_csv("GooglePublicData_fer_long.csv")
data_gdp_long = pd.read_csv("GooglePublicData_gdp_long.csv")
data_exp_long = pd.read_csv("GooglePublicData_exp_long.csv")
data_pop_long = pd.read_csv("GooglePublicData_pop_long.csv")
```

In [ ]:
```
data_fer_long.head()
```

In [ ]:
```
data_gdp_long.head()
```

In [ ]:
```
data_all = pd.merge(left=data_fer_long, right=data_gdp_long, on=['country_id', 'year'])
data_all.head()
```

In [ ]:
```
data_all = pd.merge(left=data_all, right=data_exp_long, on=['country_id', 'year'])
data_all = pd.merge(left=data_all, right=data_pop_long, on=['country_id', 'year'])
data_all.head()
```

Data Merge in Pandas

Merge country info

In [66]:
```
data_country.head()
```

Out[66]:

	country_id	country_name	income_level	latitude	longitude	region
0	AFG	Afghanistan	LIC	33.939110	67.709953	SAS
1	ALB	Albania	UMC	41.153332	20.168331	ECS
2	DZA	Algeria	UMC	28.033886	1.659626	MEA
3	ASM	American Samoa	UMC	-14.270972	-170.132217	EAS
4	AND	Andorra	NaN	42.546245	1.601554	ECS

In [ ]:
```
data_all = pd.merge(left=data_all, right=data_country[['country_id', 'income_level', 'region']], on='country_id')
data_all.head()
```

Save the complete dataset for future use

In [ ]:
```
data_all.to_csv("GooglePublicData_all.csv", index=False)
```

Descriptive Analytics with Numerical Summary: Summary

Module Summary

Descriptive analytics

- Numerical column
- Centre values – Mean, median or mode value
- Spread of numerical column
- Compute standard deviation
- Maximum and minimum value

Sorting data

- Percentile of numerical column

Advanced data manipulation

- Group-wise operations

Data merge

- Concat – Focus on row indices, column names and keys
- Merge – Specify a special column by using a unique ID

NUS
National University
of Singapore

NUS
BUSINESS
SCHOOL