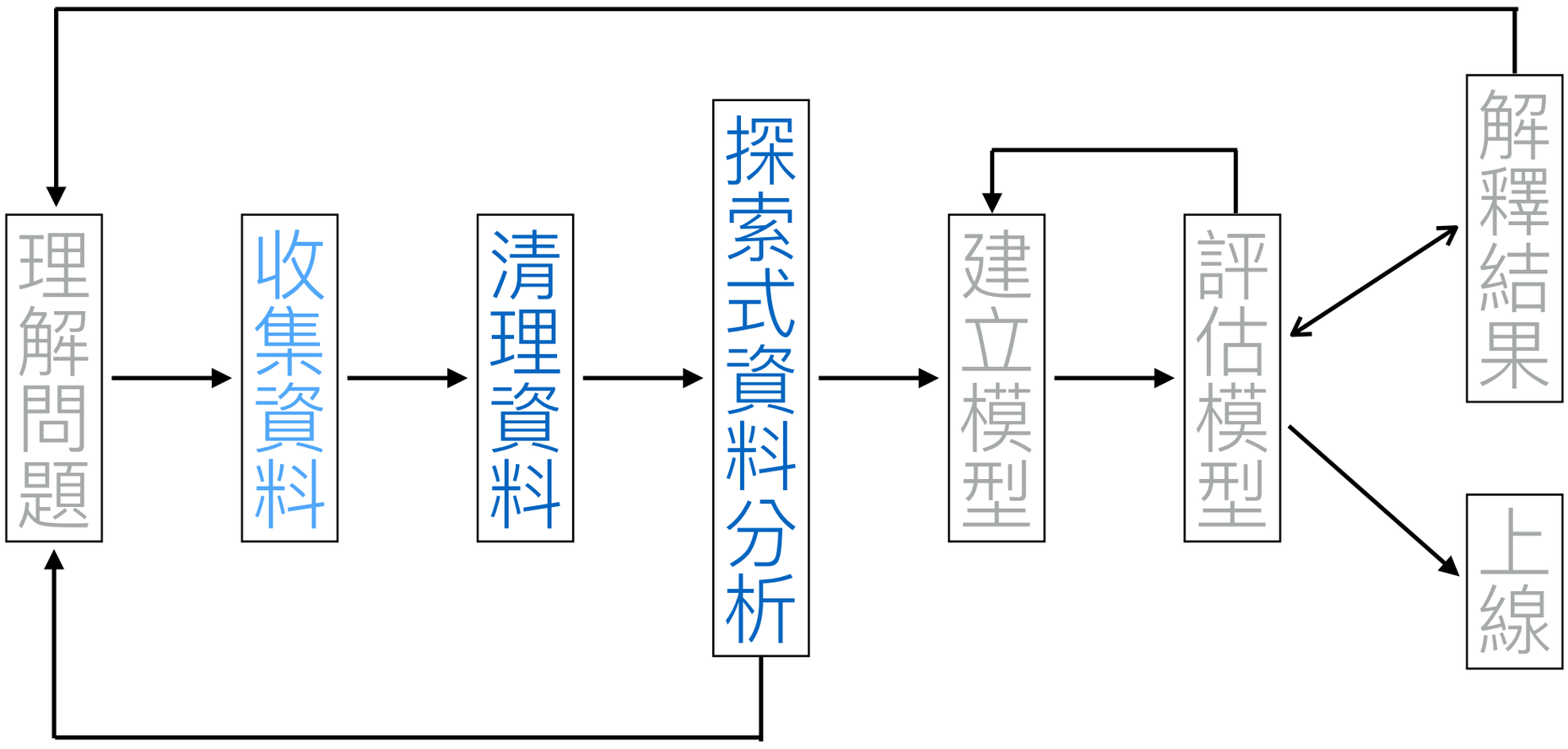
# Pandas

## Part II

# Introduction to Pandas



# Agenda

- DataFrames and Series
  - indexing and selection
  - arithmetics
  - cleansing
  - the apply() function
  - grouping and aggregating

# Pandas

- Open–source library of data analysis tools
    - implemented in Cython (convert python–like codes to C; almost as fast as native C codes)
    - database–like structures, mimics those available in R

# Basic data structures

- DataFrame: table of rows with labeled columns
  - like a spreadsheet in Excel or an R data frame
  - integrated with numpy
- Series: one-dimensional labeled array

# pandas DataFrames

- Fundamental unit of pandas
- 2–dimensional structure that mimics a spreadsheet
  - columns of (potentially) different types
  - labeled rows
- Can be created from many different objects (e.g., dict, 2–dimensional ndarray, etc.)

# pandas DataFrames


| index | device_id | SiteName | PM25 | timestamp |
|---|---|---|---|---|
| 0 | 08BEAC0A0756 | 雲林縣縣立元長國小 | 15 | 2020-09-12 00:00:00 |
| 1 | 08BEAC09FFF4 | 臺中市立六寶國小 | 23 | 2020-09-12 00:00:00 |
| 2 | 08BEAC0A0404 | 臺中市立樂業國小 | 24 | 2020-09-12 00:00:00 |
| 3 | 74DA38F20DC4 | 高雄市市立後勁國小 | 9 | 2020-09-12 00:00:00 |
| 4 | 74DA38F20E78 | 苗栗縣縣立景山國小 | 24 | 2020-09-12 00:00:00 |
| 5 | 74DA38F7C622 | 市立福德國小 | 38 | 2020-09-12 00:00:00 |

```
df.columns
```

```
Index(['device_id', 'SiteName', 'PM25', 'timestamp'], dtype='object')
```

```
df.index
```

```
RangeIndex(start=0, stop=406349, step=1)
```

# Creating DataFrames

```
d = {'Undergrad': ['台大農經系','台大法律系','台大法律系','台大法律系'],
     'PhD': ['Cornell',None,'Harvard','LSE']}
Presidents = pd.DataFrame(d, index=['李登輝','陳水扁','馬英九','蔡英文'])
Presidents
```

| | Undergrad | PhD |
|---|---|---|
| **李登輝** | 台大農經系 | Cornell |
| **陳水扁** | 台大法律系 | None |
| **馬英九** | 台大法律系 | Harvard |
| **蔡英文** | 台大法律系 | LSE |

# Adding columns

```
Presidents['Years'] = [8,8,8,5]
Presidents
```

← Like a dictionary, we can create new key-value pairs

| | Undergrad | PhD | Years |
|---|---|---|---|
| 李登輝 | 台大農經系 | Cornell | 8 |
| 陳水扁 | 台大法律系 | None | 8 |
| 馬英九 | 台大法律系 | Harvard | 8 |
| 蔡英文 | 台大法律系 | LSE | 5 |

# Adding columns

```
Presidents['連任'] = True
Presidents
```

← scalars are broadcast across the rows

| | Undergrad | PhD | Years | 連任 |
|---|---|---|---|---|
| **李登輝** | 台大農經系 | Cornell | 8 | True |
| **陳水扁** | 台大法律系 | None | 8 | True |
| **馬英九** | 台大法律系 | Harvard | 8 | True |
| **蔡英文** | 台大法律系 | LSE | 5 | True |

# Deleting columns

```
del Presidents['連任']
Presidents
```

← deleting columns is identical to deleting keys from a dictionary

| | Undergrad | PhD | Years |
|---|---|---|---|
| 李登輝 | 台大農經系 | Cornell | 8 |
| 陳水扁 | 台大法律系 | None | 8 |
| 馬英九 | 台大法律系 | Harvard | 8 |
| 蔡英文 | 台大法律系 | LSE | 5 |

# Indexing and selection

```
Presidents['PhD']
```

← DataFrame acts like a dictionary whose keys are column names

```
李登輝      Cornell
陳水扁         None
馬英九      Harvard
蔡英文          LSE
Name: PhD, dtype: object
```

# Indexing and selection

```
Presidents.loc['馬英九']
```

```
Undergrad       台大法律系
PhD             Harvard
Years                 8
Name: 馬英九, dtype: object
```

```
Presidents.iloc[1]
```

```
Undergrad     台大法律系
PhD            None
Years             8
Name: 陳水扁, dtype: object
```

df.loc select rows by their labels;
df.iloc selelct rows by their orders (starting from 0)

# Indexing and selection

```
Presidents[0:2]
```

← select rows by their numerical indices (again, starting from 0)

| | Undergrad | PhD | Years |
|---|---|---|---|
| 李登輝 | 台大農經系 | Cornell | 8 |
| 陳水扁 | 台大法律系 | None | 8 |

```
Presidents[['PhD','Years']]
```

← select columns with list of column names

| | PhD | Years |
|---|---|---|
| 李登輝 | Cornell | 8 |
| 陳水扁 | None | 8 |
| 馬英九 | Harvard | 8 |
| 蔡英文 | LSE | 5 |

# Indexing and selection

```
Presidents[Presidents['Years']==8]
```

← select rows by boolean expressions

| | Undergrad | PhD | Years |
|---|---|---|---|
| **李登輝** | 台大農經系 | Cornell | 8 |
| **陳水扁** | 台大法律系 | None | 8 |
| **馬英九** | 台大法律系 | Harvard | 8 |

# Indexing and selection

```
type(Presidents.loc['馬英九'])
```

← these expressions return Series

```
pandas.core.series.Series
```

```
type(Presidents['PhD'])
```

```
pandas.core.series.Series
```

```
type(Presidents.iloc[1])
```

```
pandas.core.series.Series
```

# Indexing and selection

```
type(Presidents[0:2])
```

```
pandas.core.frame.DataFrame
```

```
type(Presidents[Presidents['Years']==8])
```

```
pandas.core.frame.DataFrame
```

these expressions return DataFrames

# Removing NaNs

| | A | B | C | D |
|---|---|---|---|---|
| 0 | -0.249461 | -0.602798 | -2.075586 | NaN |
| 1 | -0.624237 | 0.742127 | 2.117243 | NaN |
| 2 | 0.232589 | 0.241982 | -0.679220 | NaN |
| 3 | 0.956435 | -0.765218 | -0.167972 | NaN |
| 4 | 0.188545 | -1.952676 | NaN | NaN |

dropna removes rows or columns that contains NaNs

axis argument controls whether we act on rows, columns, etc.

how='any' will remove all rows/columns that contain even one NaN. how='all' removes rows/columns that have all entries NaN

```
df.dropna(axis=1, how='any')
```

| | A | B |
|---|---|---|
| 0 | -0.249461 | -0.602798 |
| 1 | -0.624237 | 0.742127 |
| 2 | 0.232589 | 0.241982 |
| 3 | 0.956435 | -0.765218 |
| 4 | 0.188545 | -1.952676 |

```
df2 = df.dropna(axis=1, how='all')
df2
```

| | A | B | C |
|---|---|---|---|
| 0 | -0.249461 | -0.602798 | -2.075586 |
| 1 | -0.624237 | 0.742127 | 2.117243 |
| 2 | 0.232589 | 0.241982 | -0.679220 |
| 3 | 0.956435 | -0.765218 | -0.167972 |
| 4 | 0.188545 | -1.952676 | NaN |

# Row- and column-wise functions: apply()

```
df.apply(np.nanmean)
```

```
A    0.100774
B   -0.467316
C   -0.201384
D         NaN
dtype: float64
```

← DataFrame.apply() takes a function and applies it to each column/row of the DataFrame

```
df.apply(np.nanmean, axis=1)
```

```
0   -0.975948
1    0.745044
2   -0.068216
3    0.007748
4   -0.882066
dtype: float64
```

← axis argument is 0 by default (column-wise). Change it to 1 for row-wise application

# Aggregating data

```
df2.agg([np.median, np.mean, np.max])
```

| | A | B | C |
|---|---|---|---|
| **median** | 0.188545 | -0.602798 | -0.423596 |
| **mean** | 0.100774 | -0.467316 | -0.201384 |
| **amax** | 0.956435 | 0.742127 | 2.117243 |

```
df2.agg({'A':'mean', 'B':'median', 'C':'max'})
```

```
A    0.100774
B   -0.602798
C    2.117243
dtype: float64
```

agg can take a dictionary whose keys are column names and values are functions

# Grouping and aggregating data

```
df2['id'] = [1, 1, 2, 2, 3]
df2
```

| | A | B | C | id |
|---|---|---|---|---|
| **0** | -0.249461 | -0.602798 | -2.075586 | 1 |
| **1** | -0.624237 | 0.742127 | 2.117243 | 1 |
| **2** | 0.232589 | 0.241982 | -0.679220 | 2 |
| **3** | 0.956435 | -0.765218 | -0.167972 | 2 |
| **4** | 0.188545 | -1.952676 | NaN | 3 |

```
gr = df2.groupby('id')
gr.size()
```

```
id
1    2
2    2
3    1
dtype: int64
```

split df2 into groups according to 'id'

```
gr.agg(np.mean)
```

| id | A | B | C |
|---|---|---|---|
| **1** | -0.436849 | 0.069665 | 0.020828 |
| **2** | 0.594512 | -0.261618 | -0.423596 |
| **3** | 0.188545 | -1.952676 | NaN |

aggregate each group

# Readings

- 10 minutes to pandas
- Pandas data structures
- Basic operations to DataFrame and Series
- Group by: split–apply–combine
- Chapters 5–7 of Python for Data Analysis, 2nd Edition

# Homework

1. 下載9/20日的airbox資料

2. 利用Pandas找出所有台中市測站資料

3. 計算每個測站遺失資料(NaNs)數

4. 計算每個測站的日平均PM 2.5濃度