Pandas

Part III

Recap

- DataFrame and Series
    - adding/deleting columns
    - indexing and selection
    - removing NaNs
    - arithmetics
    - split–apply by groupby() and apply()

Recap: pandas DataFrames

variables

observations

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

Agenda

- Combining DataFrames
    - concat, merge and join
- Time series

Combining DataFrames

Concatenating DataFrames

pandas concat function concatenates DataFrames into a single DataFrame.

```
df1 = pd.DataFrame({'A':np.random.randn(4),
                    'B':np.random.randn(4),
                    'C':np.random.randn(4)},
                   index=[0,1,2,3])
df2 = pd.DataFrame({'A':np.random.randn(4),
                    'B':np.random.randn(4),
                    'C':np.random.randn(4)},
                   index=[4,5,6,7])
pd.concat([df1,df2])
```

Indices are concatenates as well.

	A	B	C
0	1.038178	0.080400	0.467206
1	0.197325	-1.138109	-1.666838
2	2.107052	-0.401569	-0.918375
3	-0.834861	0.156551	0.157192
4	0.649614	-0.400683	-0.462590
5	0.389805	0.403617	-1.873245
6	1.235875	0.637920	-0.438777
7	-0.598117	-0.784065	-1.304863

Different variables

```
del df2['C']
pd.concat([df1,df2])
```

	A	B	C
0	-0.557067	-0.076055	2.812915
1	1.108947	-0.597991	-0.511211
2	0.338162	-0.105120	-1.264078
3	0.082425	-1.787306	0.156730
4	1.113247	0.959691	NaN
5	0.414716	0.259693	NaN
6	1.056374	-0.277165	NaN
7	-0.348092	-1.388347	NaN

Missing values get NaN.

Handling variables by join

```
pd.concat([df1,df2], join='inner')
```

	A	B
0	-0.557067	-0.076055
1	1.108947	-0.597991
2	0.338162	-0.105120
3	0.082425	-1.787306
4	1.113247	0.959691
5	0.414716	0.259693
6	1.056374	-0.277165
7	-0.348092	-1.388347

Intersection of variables.

```
pd.concat([df1,df2], join='outer')
```

	A	B	C
0	-0.557067	-0.076055	2.812915
1	1.108947	-0.597991	-0.511211
2	0.338162	-0.105120	-1.264078
3	0.082425	-1.787306	0.156730
4	1.113247	0.959691	NaN
5	0.414716	0.259693	NaN
6	1.056374	-0.277165	NaN
7	-0.348092	-1.388347	NaN

Union of variables; default behavior.

Repeated indices

```
df2.set_index(pd.Series([3,4,5,6]), inplace=True)
pd.concat([df1,df2])
```

	A	B	C
0	-0.557067	-0.076055	2.812915
1	1.108947	-0.597991	-0.511211
2	0.338162	-0.105120	-1.264078
3	0.082425	-1.787306	0.156730
3	1.113247	0.959691	NaN
4	0.414716	0.259693	NaN
5	1.056374	-0.277165	NaN
6	-0.348092	-1.388347	NaN

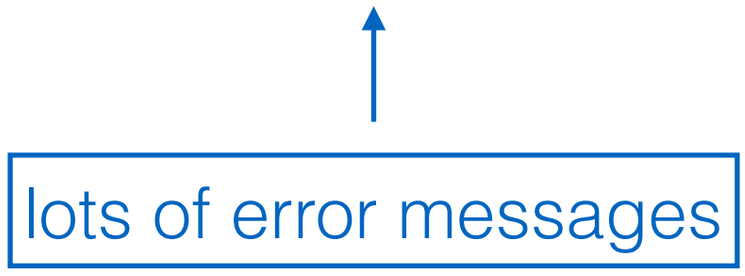
Repeated indices remain repeated in the resulting DataFrame.

Repeated observations

- Longitudinal data
- Repeated measurements (to reduce measurement error)
- etc.

Check for repeated indices

```
pd.concat([df1,df2], verify_integrity=True)
```



Ignore repeated indices

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

	A	B	C
0	-0.557067	-0.076055	2.812915
1	1.108947	-0.597991	-0.511211
2	0.338162	-0.105120	-1.264078
3	0.082425	-1.787306	0.156730
4	1.113247	0.959691	NaN
5	0.414716	0.259693	NaN
6	1.056374	-0.277165	NaN
7	-0.348092	-1.388347	NaN

Merging DataFrames

```
學生名單 = pd.DataFrame({'學號':['1001','1002','1003','1004'],
                         '姓名':['大雄','靜香','胖虎','小夫']})
Calculus = pd.DataFrame({'學號':['1001','1002','1003','1004'],
                         '微積分':['15','99','60','92']})
Linear_Algebra = pd.DataFrame({'學號':['1002','1003','1004'],
                         '線性代數':['90','38','70']})
```

```
pd.merge(學生名單, Calculus, on='學號')
```

	學號	姓名	微積分
0	1001	大雄	15
1	1002	靜香	99
2	1003	胖虎	60
3	1004	小夫	92

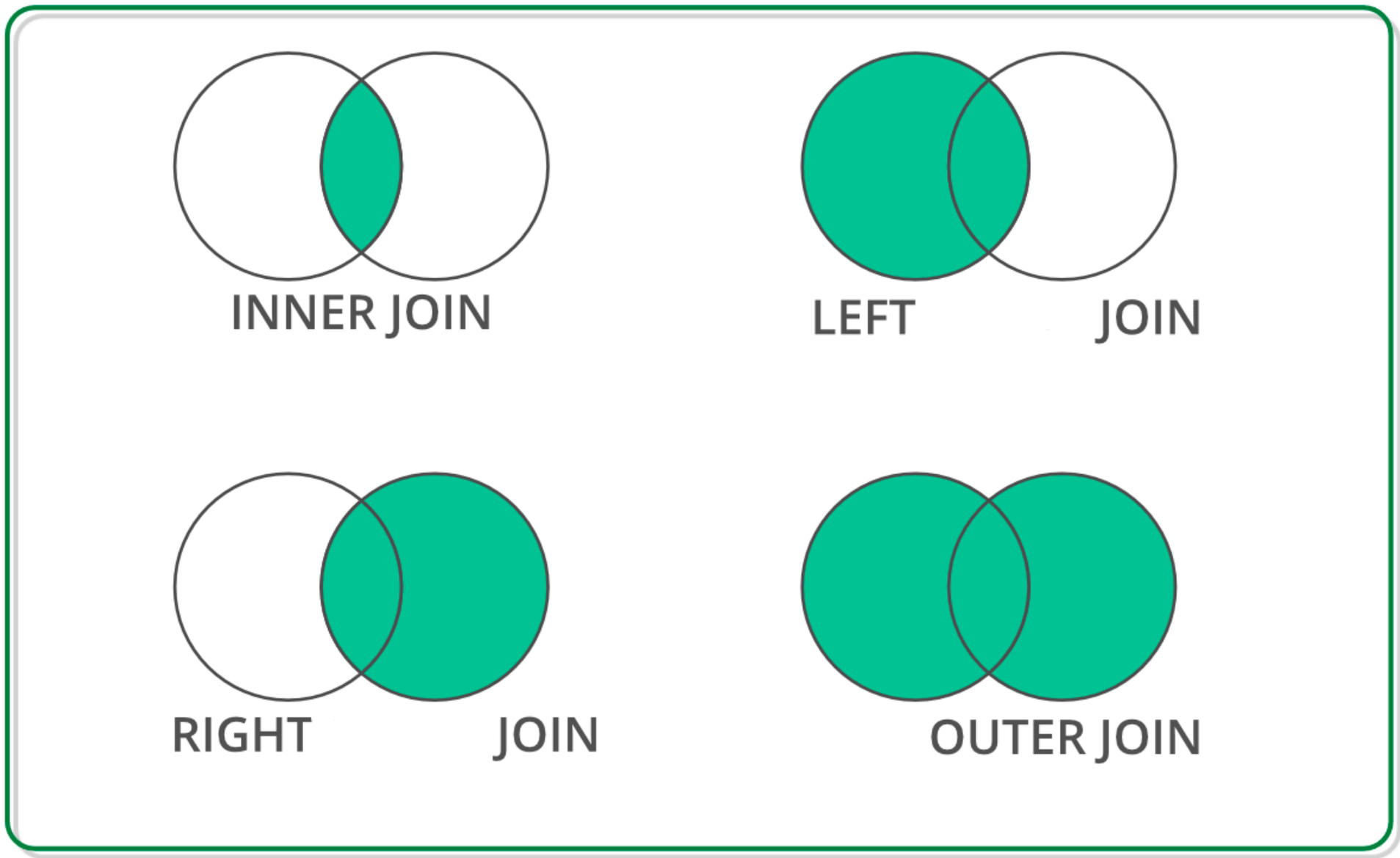
根據哪個變數合併

Merging DataFrames

```
pd.merge(pd.merge(學生名單, Calculus), Linear_Algebra)
```

	學號	姓名	微積分	線性代數
0	1002	靜香	99	90
1	1003	胖虎	60	38
2	1004	小夫	92	70

Different types of joins



https://medium.com/swlh/merging-dataframes-with-pandas-pd-merge-7764c7e2d46d

Outer joint

```
pd.merge(pd.merge(學生名單, Calculus), Linear_Algebra, how='outer')
```

	學號	姓名	微積分	線性代數
0	1001	大雄	15	NaN
1	1002	靜香	99	90
2	1003	胖虎	60	38
3	1004	小夫	92	70

Inner join

```
pd.merge(pd.merge(學生名單, Calculus), Linear_Algebra, how='inner')
```

	學號	姓名	微積分	線性代數
0	1002	靜香	99	90
1	1003	胖虎	60	38
2	1004	小夫	92	70

Left join

```
pd.merge(pd.merge(學生名單, Calculus), Linear_Algebra, how='left')
```

	學號	姓名	微積分	線性代數
0	1001	大雄	15	NaN
1	1002	靜香	99	90
2	1003	胖虎	60	38
3	1004	小夫	92	70

Right join

```
pd.merge(pd.merge(學生名單, Calculus), Linear_Algebra, how='right')
```

	學號	姓名	微積分	線性代數
0	1002	靜香	99	90
1	1003	胖虎	60	38
2	1004	小夫	92	70

Working with datetimes

Example: airbox data

```
import pandas as pd
filenm = 'iis_airbox_20200912.csv'
df = pd.read_csv(filenm)
df.dtypes
```

```
device_id     object
SiteName      object
PM25           int64
timestamp     object
dtype: object
```

```
type(df['timestamp'].iloc[0])
```

```
str
```

The datetime module

- Classes for manipulating dates and times
  - datetime objects
  - time zone
  - time delta
  - **strings ⟷ datetimes**

Strings ⟷ datetimes

```
df['timestamp'].iloc[0]
```

```
'2020-09-12 00:00:00'
```

```
from datetime import datetime
dt = datetime.strptime(df['timestamp'].iloc[0], '%Y-%m-%d %H:%M:%S')
dt
```

```
datetime.datetime(2020, 9, 12, 0, 0)
```

```
str = datetime.strftime(dt, '%b %d, %Y.')
str
```

← Format codes

```
'Sep 12, 2020.'
```

Extract year, month, day, etc.

```
dt.year
```

```
2020
```

```
dt.month
```

```
9
```

More attributes here

```
dt.day
```

```
12
```

Timezone

```
dt.tzinfo
```

```
from pytz import timezone
tw = timezone('Asia/Taipei')
dt = tw.localize(dt)
dt.tzinfo
```

```
<DstTzInfo 'Asia/Taipei' CST+8:00:00 STD>
```

```
NY = timezone('America/New_York')
dt_NY = dt.astimezone(NY)
datetime.strftime(dt_NY, '%Y-%m-%d %H:%M:%S')
```

```
'2020-09-11 12:00:00'
```

Difference between datetimes

```
dt2 = datetime.strptime(df['timestamp'].iloc[300000], '%Y-%m-%d %H:%M:%S')
dt2 = tw.localize(dt2)
datetime.strftime(dt2, '%Y-%m-%d %H:%M:%S')
```

```
'2020-09-12 17:52:55'
```

```
dt2 - dt
```

```
datetime.timedelta(seconds=64375)
```

```
from datetime import timedelta
delta = timedelta(days=-1, hours=8)
dt + delta
```

```
datetime.datetime(2020, 9, 11, 8, 0, tzinfo=<DstTzInfo 'Asia/Taipei' CST+8:00:00
STD>)
```

datetime in Pandas

```
pd.to_datetime(df['timestamp'], format='%Y-%m-%d %H:%M:%S')
```

```
0         2020-09-12 00:00:00
1         2020-09-12 00:00:00
2         2020-09-12 00:00:00
3         2020-09-12 00:00:00
4         2020-09-12 00:00:00
                  ...
406344    2020-09-12 23:59:59
406345    2020-09-13 00:00:00
406346    2020-09-13 00:00:00
406347    2020-09-13 00:00:00
406348    2020-09-13 00:00:00
Name: timestamp, Length: 406349, dtype: datetime64[ns]
```

把series的所有成員轉成datetime型別

```
df['datetime'] = pd.to_datetime(df['timestamp'], format='%Y-%m-%d %H:%M:%S')
df['datetime'].iloc[0]
```

```
Timestamp('2020-09-12 00:00:00')
```

```
df['datetime'].dt.day
```

```
0          12
1          12
2          12
3          12
4          12
           ..
406344     12
406345     13
406346     13
406347     13
406348     13
Name: datetime, Length: 406349, dtype: int64
```

利用dt存取器存取series中datetime物件的屬性

```
df['datetime'].dt.tz_localize('Asia/Taipei')
```

```
0         2020-09-12 00:00:00+08:00
1         2020-09-12 00:00:00+08:00
2         2020-09-12 00:00:00+08:00
3         2020-09-12 00:00:00+08:00
4         2020-09-12 00:00:00+08:00
                     ...
406344    2020-09-12 23:59:59+08:00
406345    2020-09-13 00:00:00+08:00
406346    2020-09-13 00:00:00+08:00
406347    2020-09-13 00:00:00+08:00
406348    2020-09-13 00:00:00+08:00
Name: datetime, Length: 406349, dtype: datetime64[ns, Asia/Taipei]
```

pandas將pytz中設定timezone的功能包裝成tz_localize

Readings

1. Merge, join and concatenation
2. Time series
3. Pandas IO tools

Homework

1. 下載2020/10/1–5的airbox資料與其測站資料

2. 留下位於高雄市且在2018年前（含2018）設站之站點的資料

3. 為每個觀測值加入測站所在經緯度

4. 將所有滿足1–3的資料合併成一個DataFrame，並將其輸出為json及Feather檔（Readings 3）。比較其存檔速度及檔案大小。