Python for Data Analytics

Pandas III

Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022

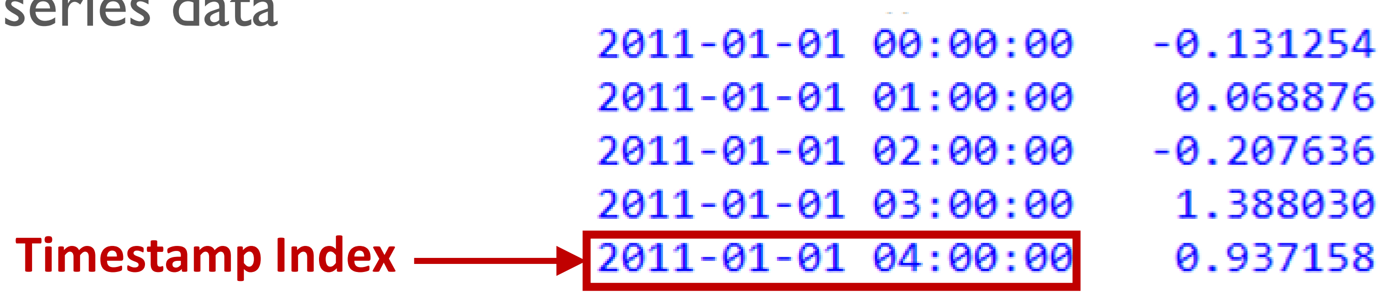
Outline

- Why Pandas?
- Pandas Series
- Pandas DataFrame
  - Creating DataFrame
  - Manipulating Columns
  - Manipulating Rows
  - Arithmetic operations
  - Group Aggregation
  - Hierarchical Indexing
  - Combining and Merging
  - Time Series Data

Time Series Data

Time Series Data

- How to analyze time series data?
  - Sample time series data



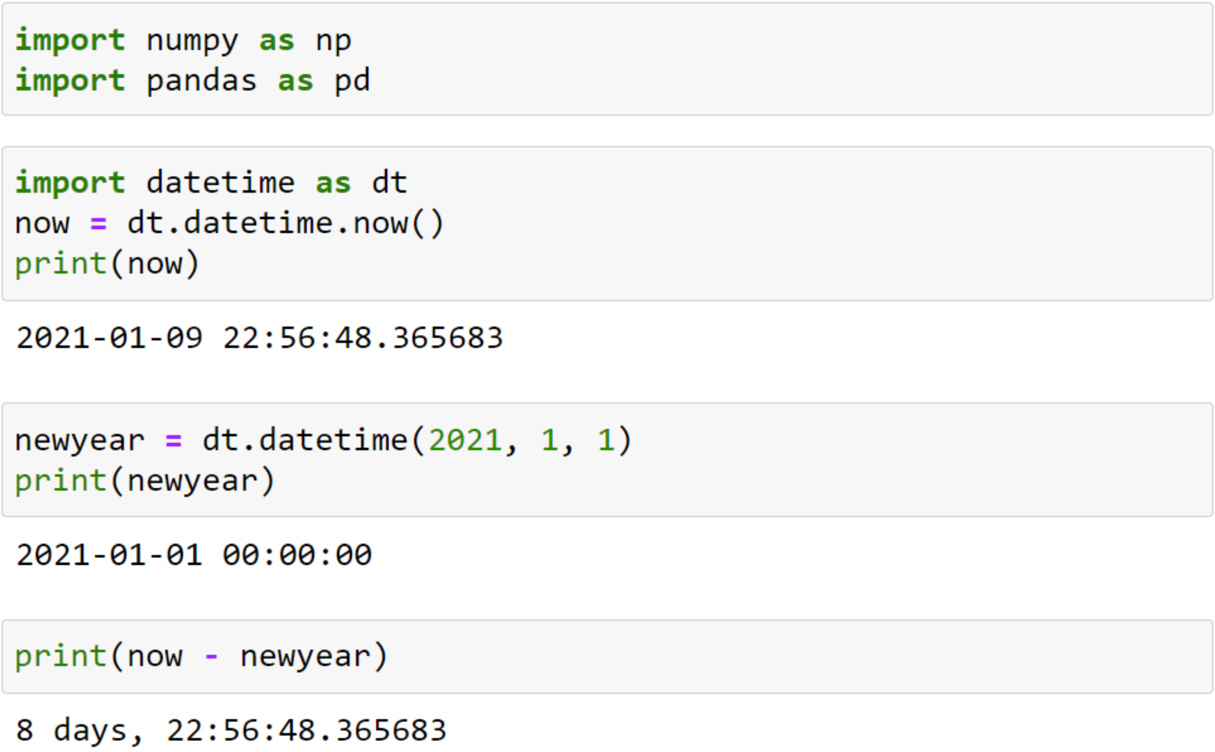
```
2011-01-01 00:00:00   -0.131254
2011-01-01 01:00:00    0.068876
2011-01-01 02:00:00   -0.207636
2011-01-01 03:00:00    1.388030
2011-01-01 04:00:00    0.937158
```

Timestamp Index

- Time series data is the data with the timestamp index
  - How to parse time series information from various sources and formats?
  - How to generate sequences of fixed-frequency dates and time spans
  - How to manipulate and convert date times with timezone information?
  - How to group data by time?
  - …

Python datetime Module

- `datetime.datetime` class: a combination of date and time
  - year, month, day, hour, minute, second, microsecond, tzinfo
- `datetime.now()`: return the current local datetime



```python
import numpy as np
import pandas as pd
```

```python
import datetime as dt
now = dt.datetime.now()
print(now)
```

```
2021-01-09 22:56:48.365683
```

```python
newyear = dt.datetime(2021, 1, 1)
print(newyear)
```

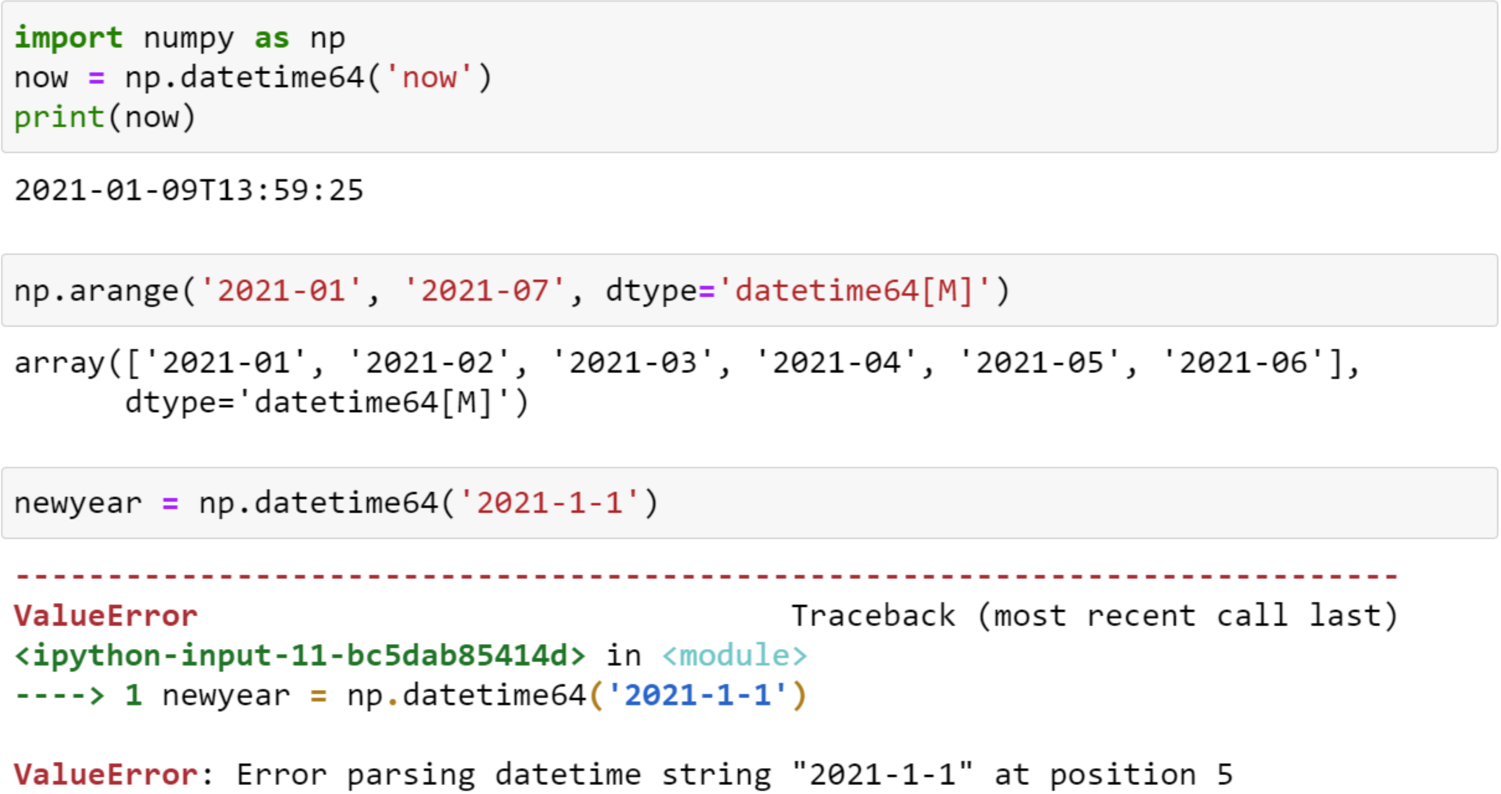
```
2021-01-01 00:00:00
```

```python
print(now - newyear)
```

```
8 days, 22:56:48.365683
```

NumPy datetime64 Type

- NumPy supports datetime functionality with the data type called '`datetime64`'
  - No timezone support



```
import numpy as np
now = np.datetime64('now')
print(now)
```

```
2021-01-09T13:59:25
```

```
np.arange('2021-01', '2021-07', dtype='datetime64[M]')
```

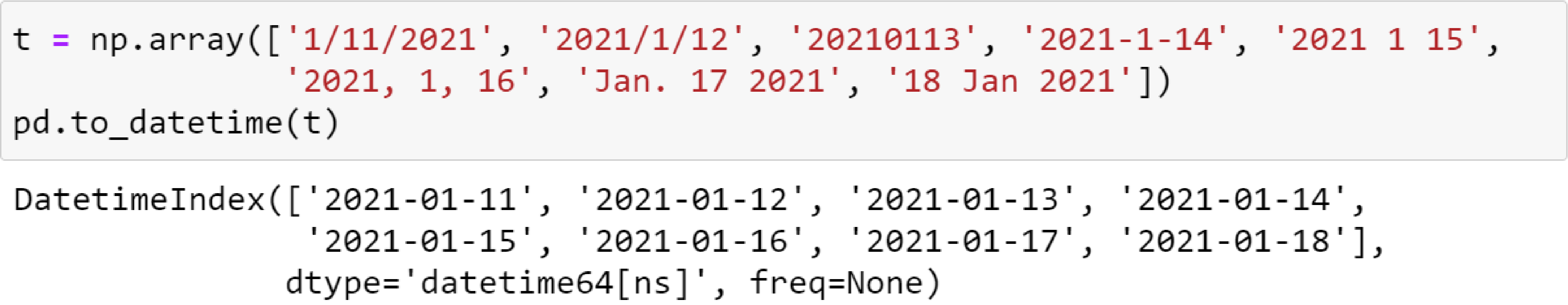
```
array(['2021-01', '2021-02', '2021-03', '2021-04', '2021-05', '2021-06'],
      dtype='datetime64[M]')
```

```
newyear = np.datetime64('2021-1-1')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-bc5dab85414d> in <module>
----> 1 newyear = np.datetime64('2021-1-1')

ValueError: Error parsing datetime string "2021-1-1" at position 5
```

Converting to Datetime

- Pandas supports extensive capabilities and features for working with time series data based on NumPy `datetime64`.
- *pd*.`to_datetime(`*arg*, …`)`
  - Convert argument to datetime.
  - Return type can be a DatetimeIndex, Series, or Timestamp
  - *arg*: integer, float, string, datetime, list, tuple, 1-D array, Series



```
t = np.array(['1/11/2021', '2021/1/12', '20210113', '2021-1-14', '2021 1 15',
              '2021, 1, 16', 'Jan. 17 2021', '18 Jan 2021'])
pd.to_datetime(t)
```

```
DatetimeIndex(['2021-01-11', '2021-01-12', '2021-01-13', '2021-01-14',
               '2021-01-15', '2021-01-16', '2021-01-17', '2021-01-18'],
              dtype='datetime64[ns]', freq=None)
```

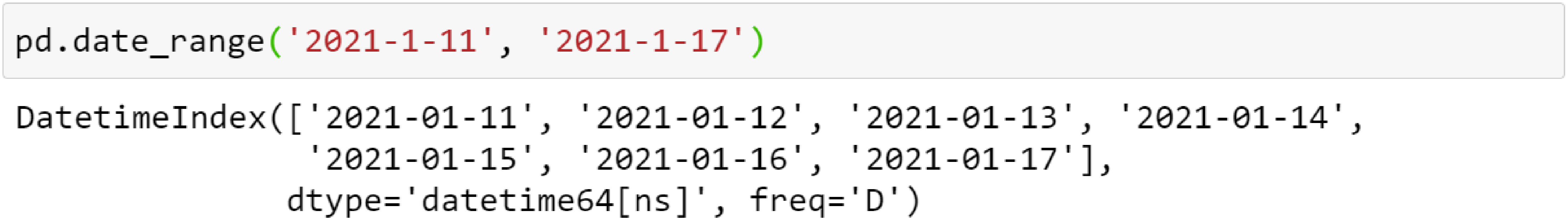
Generating DatetimeIndex

- *pd*.date_range(*start=None, end=None, periods=None, freq=None, …*)
  - Return a fixed frequency DatetimeIndex
  - *start*: left bound for generating dates
  - *end*: right bound for generating dates
  - *periods*: the number of datetime to generate
  - *freq*: the time interval between consecutive datetime values (default: 'D')

Freq string	Description	Freq string	Description
`'D'`	One absolute day	`'M'`	Calendar month end
`'H'`	One hour	`'MS'`	Calendar month begin
`'T' or 'min'`	One minute	`'BM'`	Business month end
`'S'`	One second	`'BMS'`	Business month begin
`'B'`	Business day (weekday)	`'WOM-2THU'`	Second Thursday of the month
`'W'`	One week	`'1h30min'`	One and half hour

date_range() Examples (1)

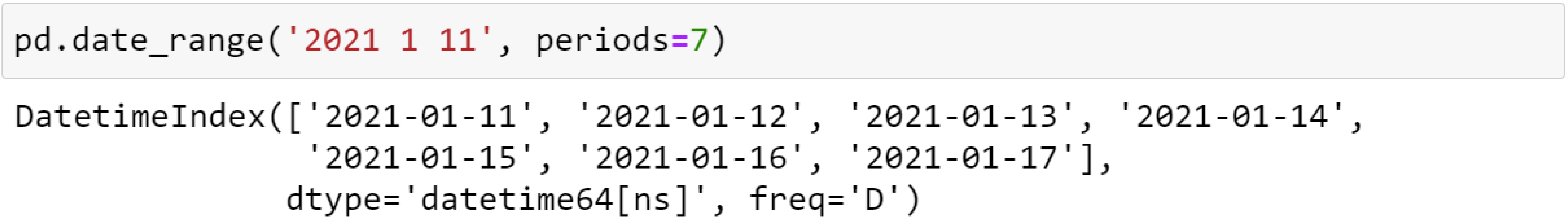
- Default: everyday



```
pd.date_range('2021-1-11', '2021-1-17')
```

```
DatetimeIndex(['2021-01-11', '2021-01-12', '2021-01-13', '2021-01-14',
               '2021-01-15', '2021-01-16', '2021-01-17'],
              dtype='datetime64[ns]', freq='D')
```

- 7 days since 2021-1-11

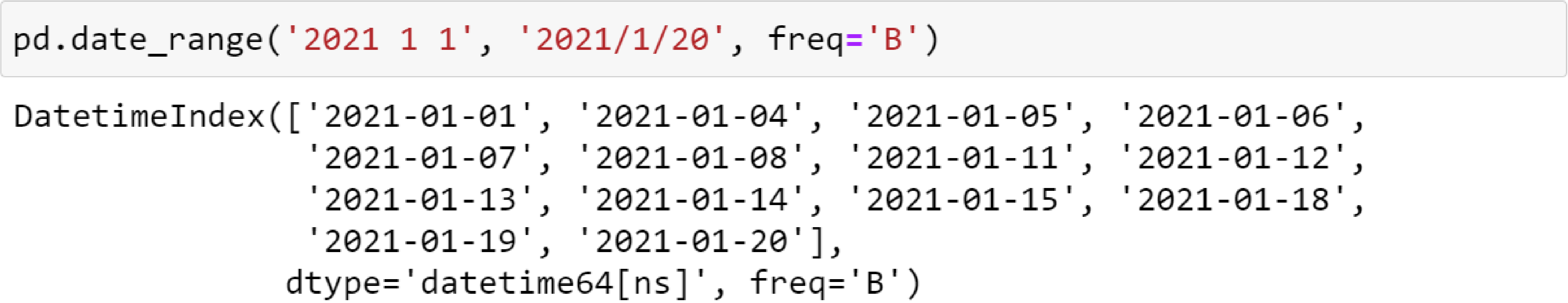


```
pd.date_range('2021 1 11', periods=7)
```

```
DatetimeIndex(['2021-01-11', '2021-01-12', '2021-01-13', '2021-01-14',
               '2021-01-15', '2021-01-16', '2021-01-17'],
              dtype='datetime64[ns]', freq='D')
```

date_range() Examples (2)

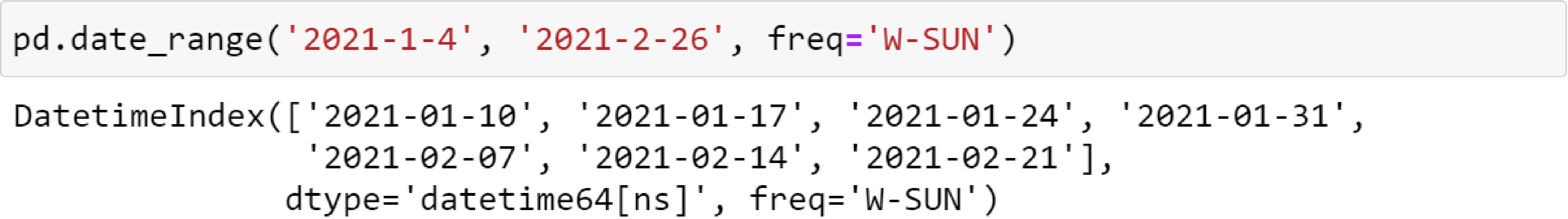
- Just weekdays



```
pd.date_range('2021 1 1', '2021/1/20', freq='B')
```

```
DatetimeIndex(['2021-01-01', '2021-01-04', '2021-01-05', '2021-01-06',
               '2021-01-07', '2021-01-08', '2021-01-11', '2021-01-12',
               '2021-01-13', '2021-01-14', '2021-01-15', '2021-01-18',
               '2021-01-19', '2021-01-20'],
              dtype='datetime64[ns]', freq='B')
```

- Every Sunday

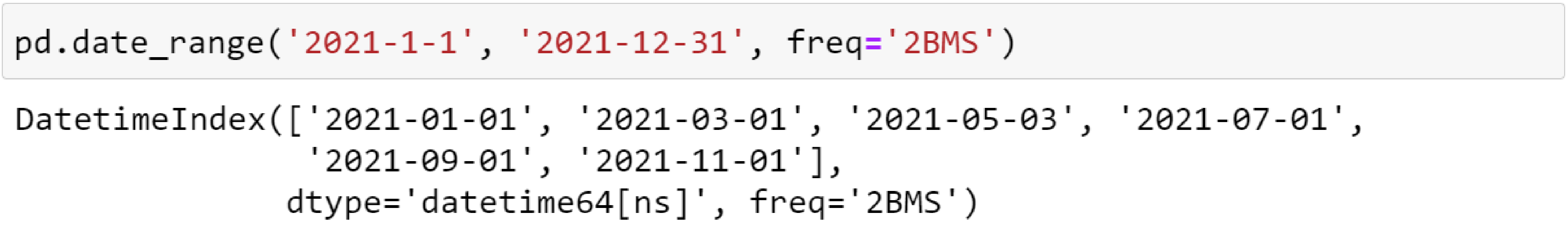


```
pd.date_range('2021-1-4', '2021-2-26', freq='W-SUN')
```

```
DatetimeIndex(['2021-01-10', '2021-01-17', '2021-01-24', '2021-01-31',
               '2021-02-07', '2021-02-14', '2021-02-21'],
              dtype='datetime64[ns]', freq='W-SUN')
```

date_range() Examples (3)

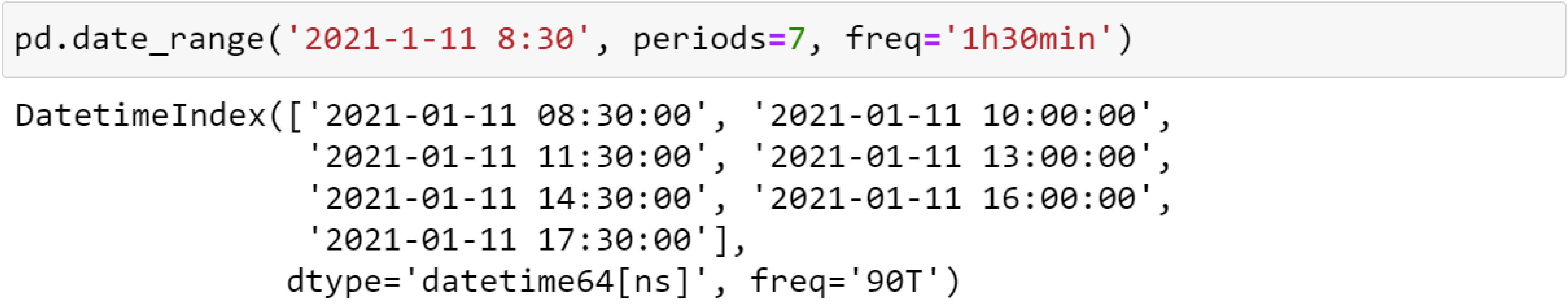
- First business day every two months



```
pd.date_range('2021-1-1', '2021-12-31', freq='2BMS')
```

```
DatetimeIndex(['2021-01-01', '2021-03-01', '2021-05-03', '2021-07-01',
               '2021-09-01', '2021-11-01'],
              dtype='datetime64[ns]', freq='2BMS')
```

- Every one and half hour

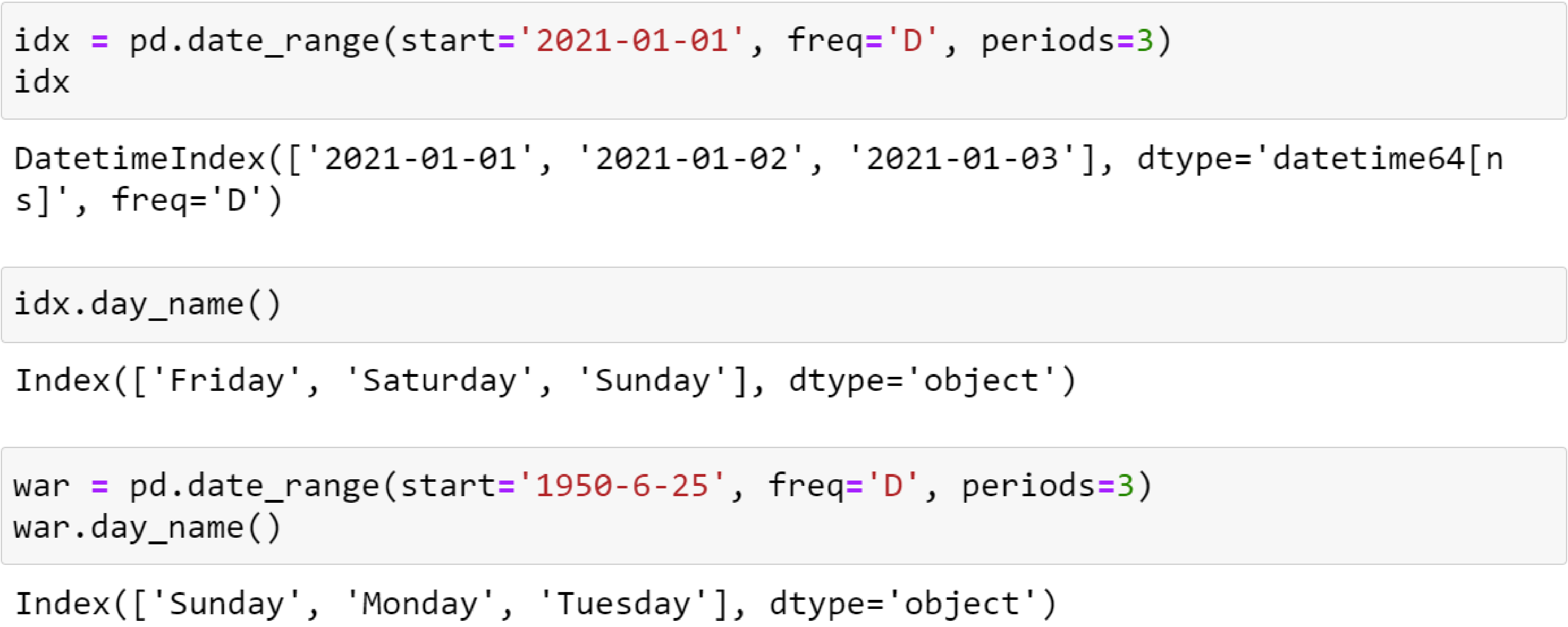


```
pd.date_range('2021-1-11 8:30', periods=7, freq='1h30min')
```

```
DatetimeIndex(['2021-01-11 08:30:00', '2021-01-11 10:00:00',
               '2021-01-11 11:30:00', '2021-01-11 13:00:00',
               '2021-01-11 14:30:00', '2021-01-11 16:00:00',
               '2021-01-11 17:30:00'],
              dtype='datetime64[ns]', freq='90T')
```

Finding the Day of the Week

- *pd*.DatetimeIndex.day_name(**args, …*)
  - Return the day names of the DatetimeIndex



```
idx = pd.date_range(start='2021-01-01', freq='D', periods=3)
idx
```

```
DatetimeIndex(['2021-01-01', '2021-01-02', '2021-01-03'], dtype='datetime64[n
s]', freq='D')
```

```
idx.day_name()
```

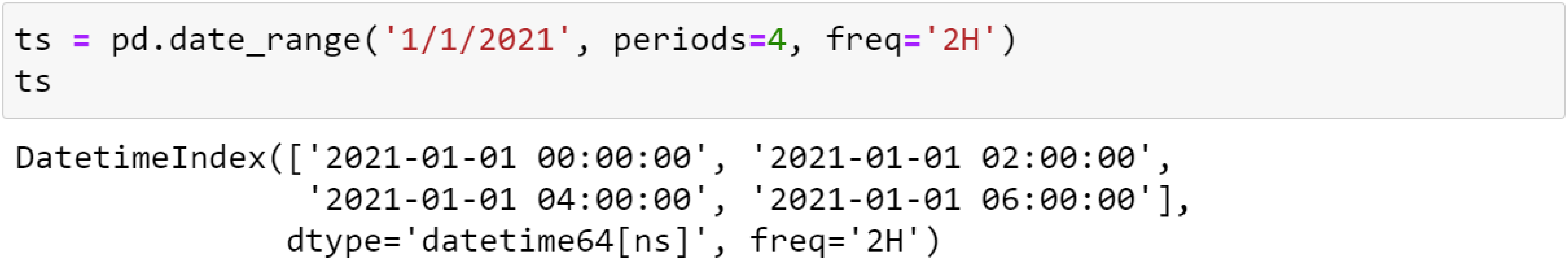
```
Index(['Friday', 'Saturday', 'Sunday'], dtype='object')
```

```
war = pd.date_range(start='1950-6-25', freq='D', periods=3)
war.day_name()
```

```
Index(['Sunday', 'Monday', 'Tuesday'], dtype='object')
```

Creating Time Series Data

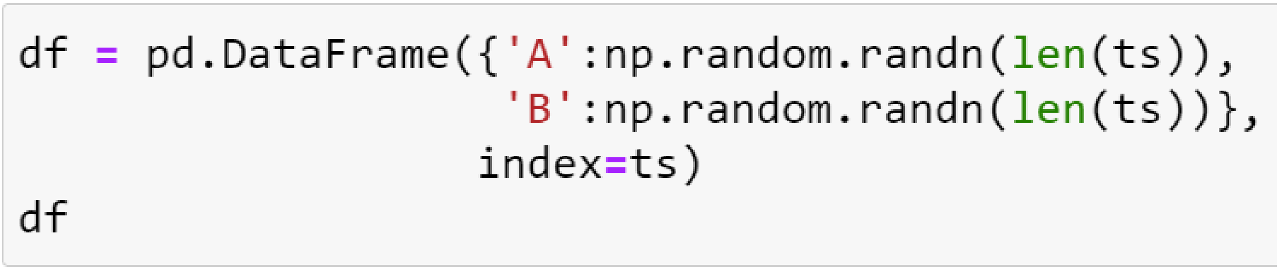
- Create a range of `DatetimeIndex` object



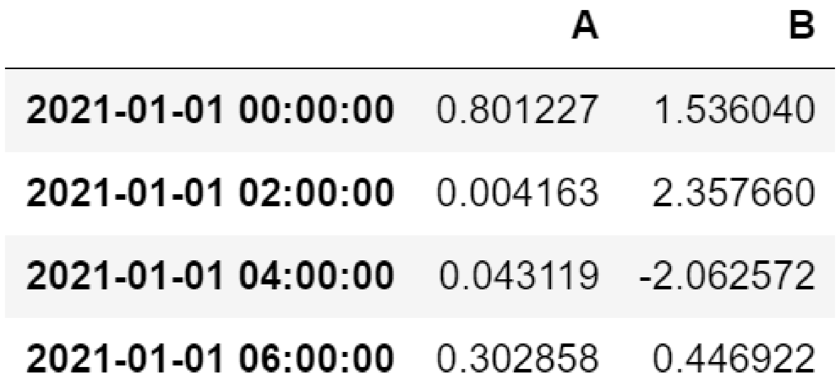
```
ts = pd.date_range('1/1/2021', periods=4, freq='2H')
ts
```

```
DatetimeIndex(['2021-01-01 00:00:00', '2021-01-01 02:00:00',
               '2021-01-01 04:00:00', '2021-01-01 06:00:00'],
              dtype='datetime64[ns]', freq='2H')
```

- Use the `DatetimeIndex` object as Pandas Series or DataFrame index



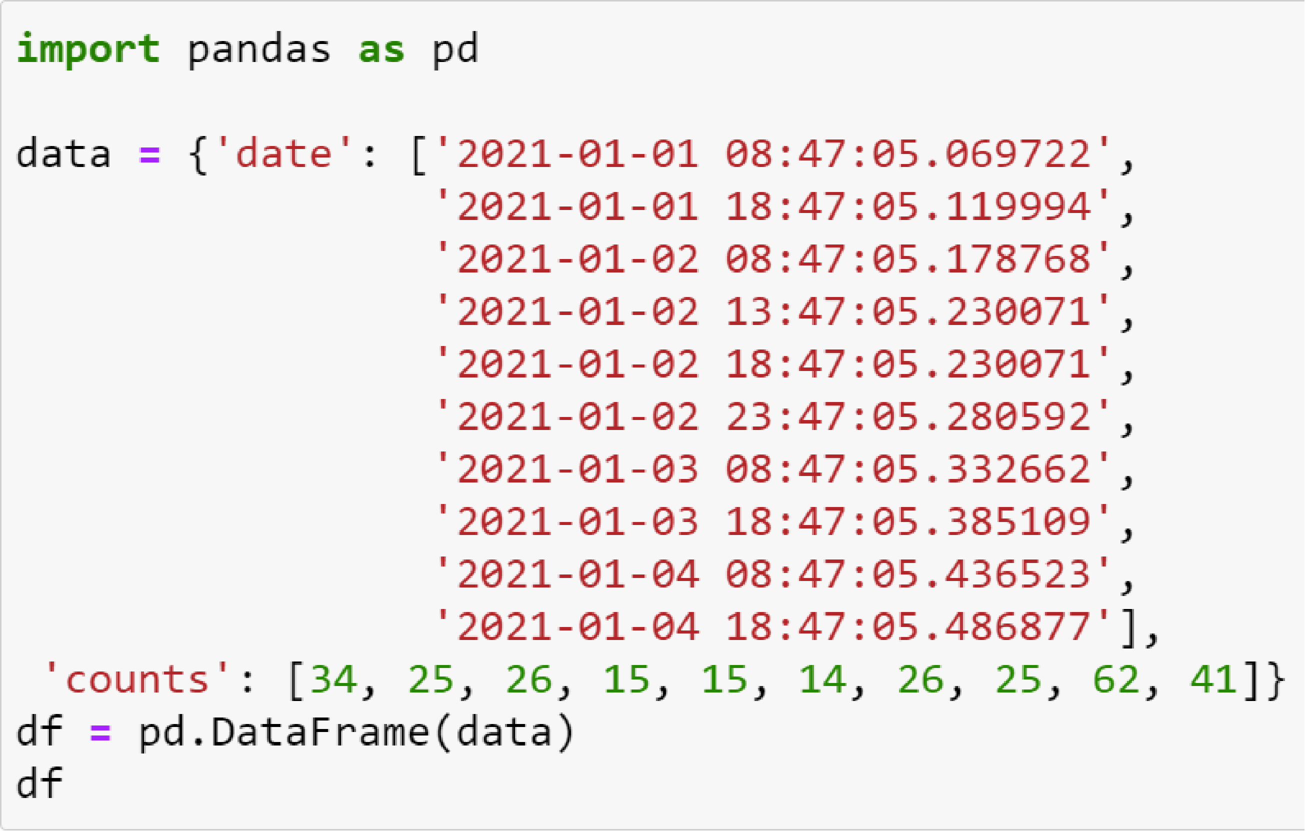
```
df = pd.DataFrame({'A':np.random.randn(len(ts)),
                   'B':np.random.randn(len(ts))},
                  index=ts)
df
```



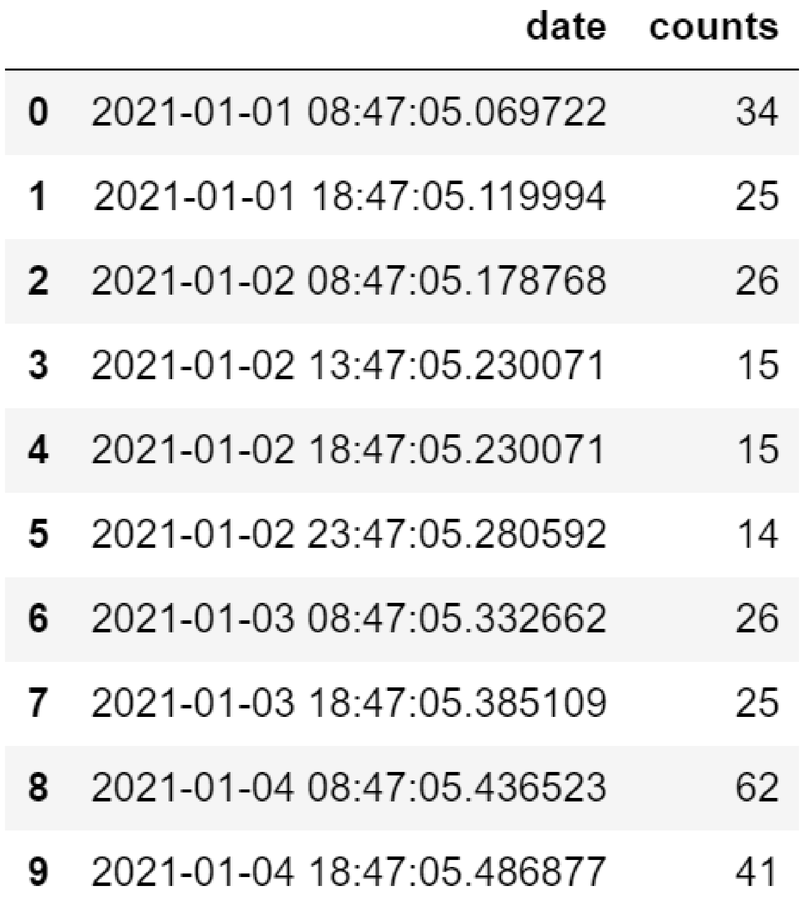
	A	B
2021-01-01 00:00:00	0.801227	1.536040
2021-01-01 02:00:00	0.004163	2.357660
2021-01-01 04:00:00	0.043119	-2.062572
2021-01-01 06:00:00	0.302858	0.446922

Example: Pandas Time Series Data (1)

- Create a dataframe: input dataset = <*timestamp, access count*>



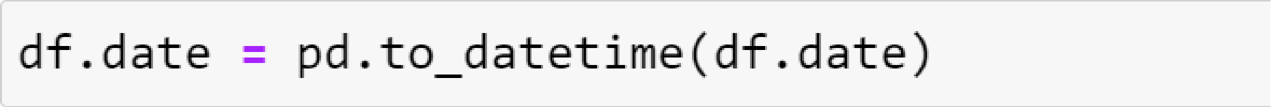
```
import pandas as pd

data = {'date': ['2021-01-01 08:47:05.069722',
                 '2021-01-01 18:47:05.119994',
                 '2021-01-02 08:47:05.178768',
                 '2021-01-02 13:47:05.230071',
                 '2021-01-02 18:47:05.230071',
                 '2021-01-02 23:47:05.280592',
                 '2021-01-03 08:47:05.332662',
                 '2021-01-03 18:47:05.385109',
                 '2021-01-04 08:47:05.436523',
                 '2021-01-04 18:47:05.486877'],
 'counts': [34, 25, 26, 15, 15, 14, 26, 25, 62, 41]}
df = pd.DataFrame(data)
df
```



	date	counts
0	2021-01-01 08:47:05.069722	34
1	2021-01-01 18:47:05.119994	25
2	2021-01-02 08:47:05.178768	26
3	2021-01-02 13:47:05.230071	15
4	2021-01-02 18:47:05.230071	15
5	2021-01-02 23:47:05.280592	14
6	2021-01-03 08:47:05.332662	26
7	2021-01-03 18:47:05.385109	25
8	2021-01-04 08:47:05.436523	62
9	2021-01-04 18:47:05.486877	41

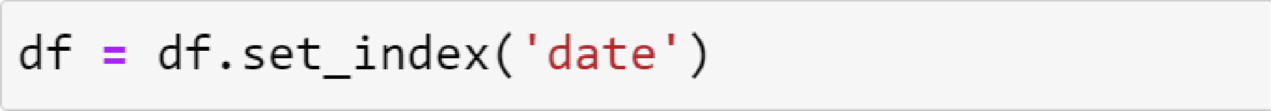
Example: Pandas Time Series Data (2)

- Convert `df['date']` from string to datetime

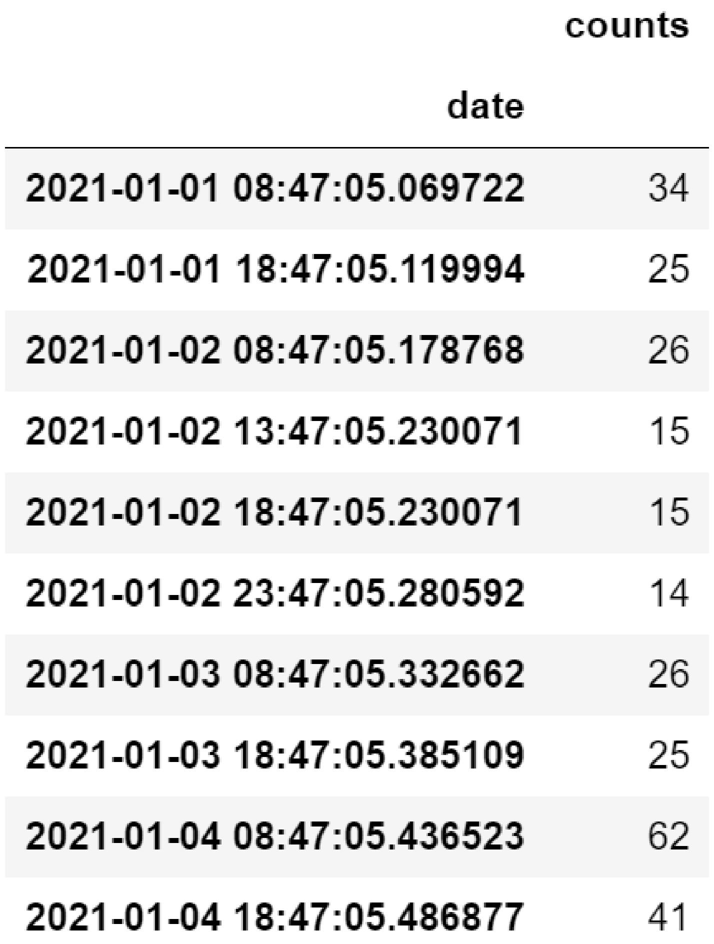


```
df.date = pd.to_datetime(df.date)
```

- Set `df['date']` as the index



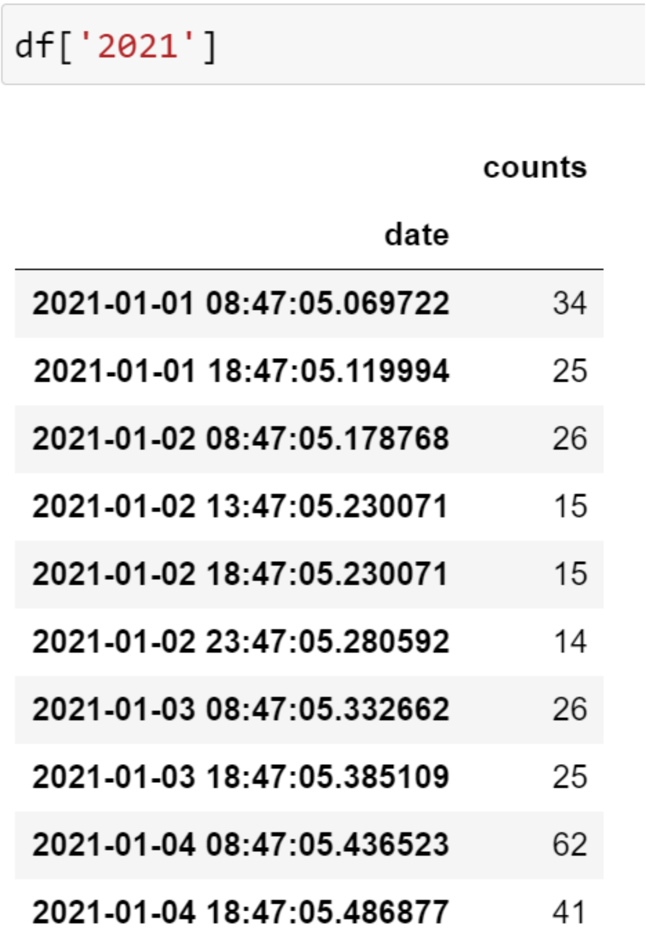
```
df = df.set_index('date')
```



	counts
date	
2021-01-01 08:47:05.069722	34
2021-01-01 18:47:05.119994	25
2021-01-02 08:47:05.178768	26
2021-01-02 13:47:05.230071	15
2021-01-02 18:47:05.230071	15
2021-01-02 23:47:05.280592	14
2021-01-03 08:47:05.332662	26
2021-01-03 18:47:05.385109	25
2021-01-04 08:47:05.436523	62
2021-01-04 18:47:05.486877	41

Accessing Time Series Data (1)

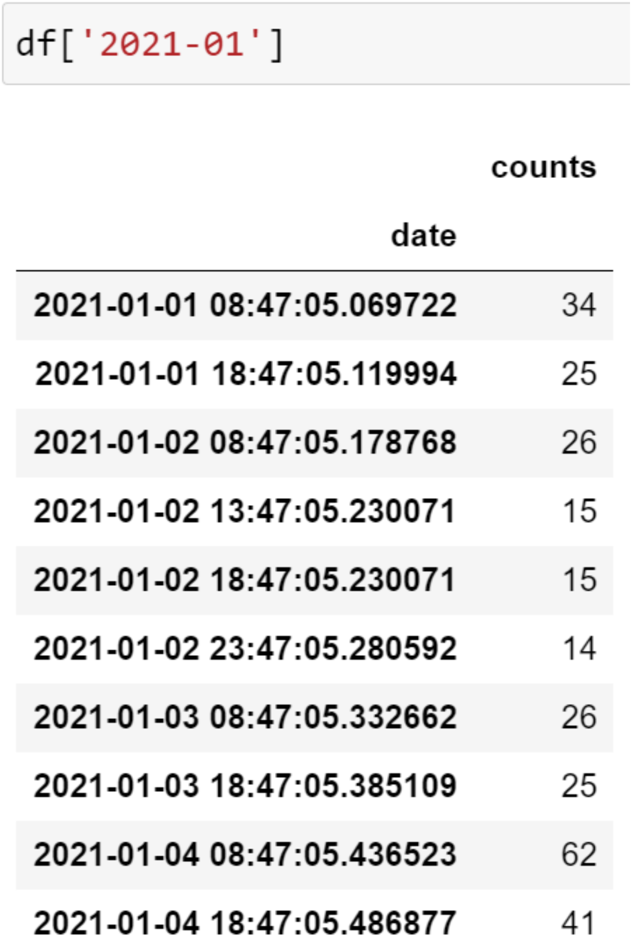
- View data in 2021



```
df['2021']
```

date	counts
2021-01-01 08:47:05.069722	34
2021-01-01 18:47:05.119994	25
2021-01-02 08:47:05.178768	26
2021-01-02 13:47:05.230071	15
2021-01-02 18:47:05.230071	15
2021-01-02 23:47:05.280592	14
2021-01-03 08:47:05.332662	26
2021-01-03 18:47:05.385109	25
2021-01-04 08:47:05.436523	62
2021-01-04 18:47:05.486877	41

- View data in January 2021

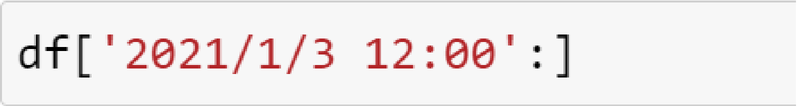


```
df['2021-01']
```

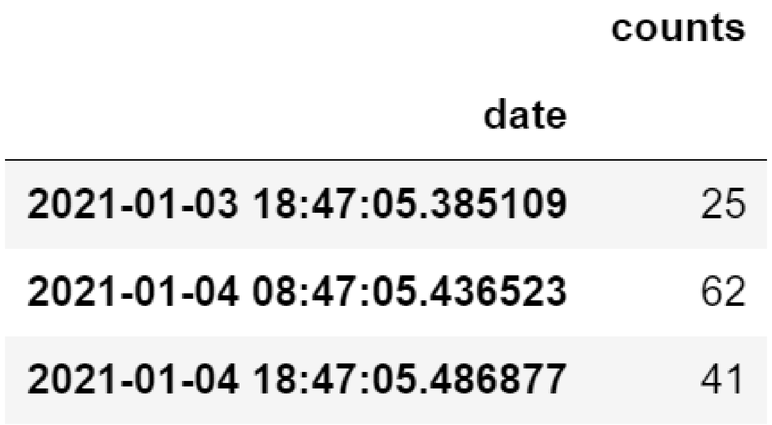
date	counts
2021-01-01 08:47:05.069722	34
2021-01-01 18:47:05.119994	25
2021-01-02 08:47:05.178768	26
2021-01-02 13:47:05.230071	15
2021-01-02 18:47:05.230071	15
2021-01-02 23:47:05.280592	14
2021-01-03 08:47:05.332662	26
2021-01-03 18:47:05.385109	25
2021-01-04 08:47:05.436523	62
2021-01-04 18:47:05.486877	41

Accessing Time Series Data (2)

- Observations after 12:00, Jan. 3, 2021

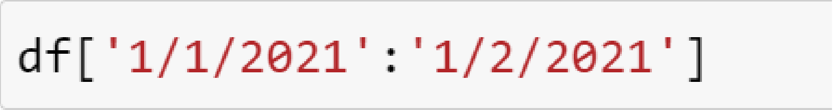


```
df['2021/1/3 12:00':]
```

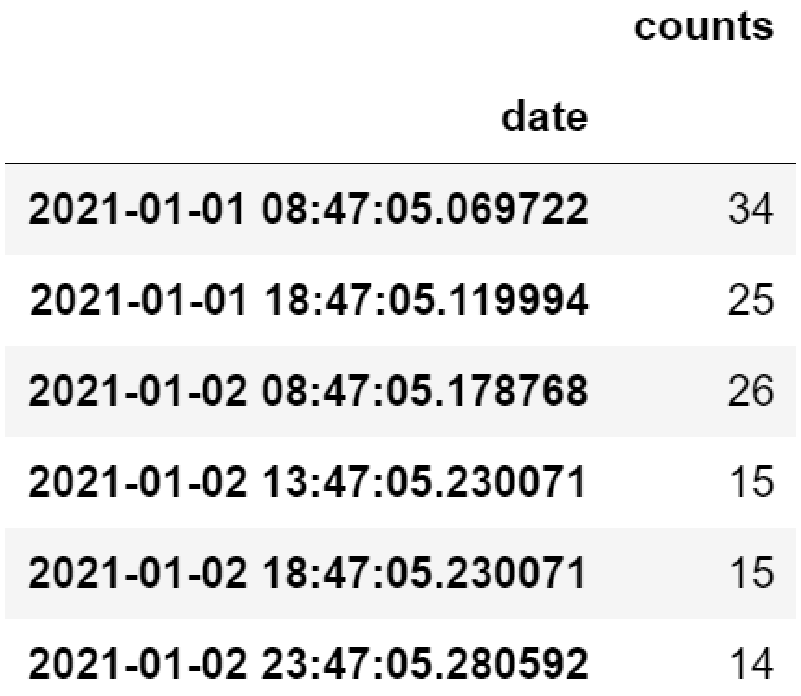


date	counts
2021-01-03 18:47:05.385109	25
2021-01-04 08:47:05.436523	62
2021-01-04 18:47:05.486877	41

- Observations between Jan. 1 - 2



```
df['1/1/2021':'1/2/2021']
```

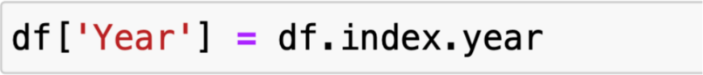


date	counts
2021-01-01 08:47:05.069722	34
2021-01-01 18:47:05.119994	25
2021-01-02 08:47:05.178768	26
2021-01-02 13:47:05.230071	15
2021-01-02 18:47:05.230071	15
2021-01-02 23:47:05.280592	14

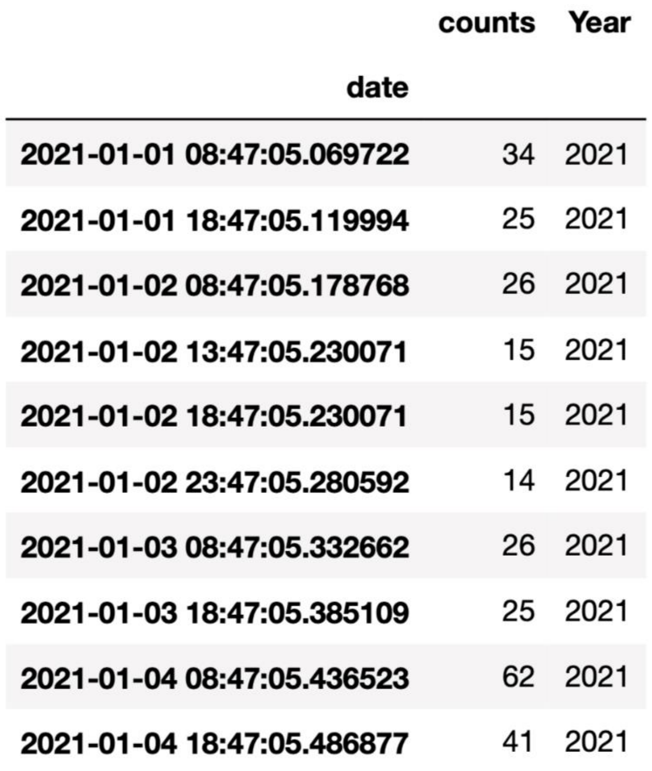
These are the "views"!

Accessing Time Series Data (3)

- Extracting years



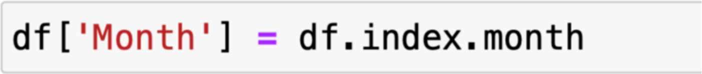
```
df['Year'] = df.index.year
```



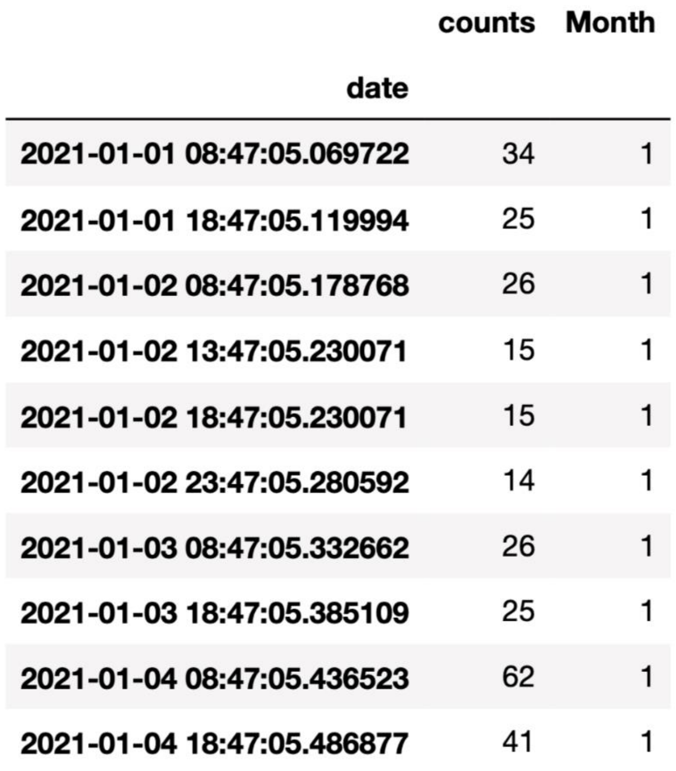
date	counts	Year
2021-01-01 08:47:05.069722	34	2021
2021-01-01 18:47:05.119994	25	2021
2021-01-02 08:47:05.178768	26	2021
2021-01-02 13:47:05.230071	15	2021
2021-01-02 18:47:05.230071	15	2021
2021-01-02 23:47:05.280592	14	2021
2021-01-03 08:47:05.332662	26	2021
2021-01-03 18:47:05.385109	25	2021
2021-01-04 08:47:05.436523	62	2021
2021-01-04 18:47:05.486877	41	2021

```
.year
.month
.day
.hour
.minute
.second
.microsecond
…
```

- Extracting months



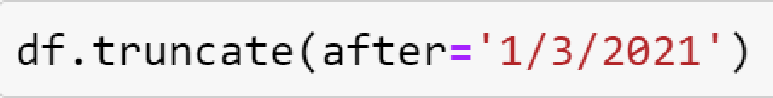
```
df['Month'] = df.index.month
```



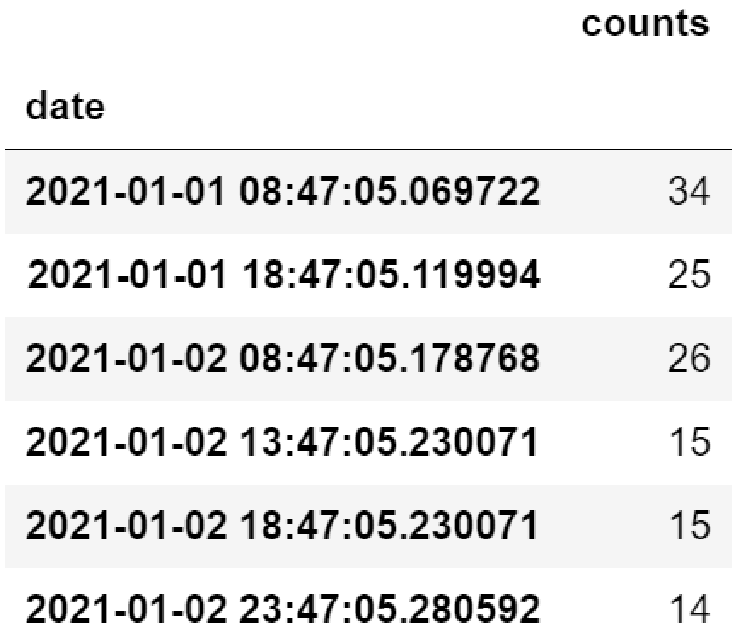
date	counts	Month
2021-01-01 08:47:05.069722	34	1
2021-01-01 18:47:05.119994	25	1
2021-01-02 08:47:05.178768	26	1
2021-01-02 13:47:05.230071	15	1
2021-01-02 18:47:05.230071	15	1
2021-01-02 23:47:05.280592	14	1
2021-01-03 08:47:05.332662	26	1
2021-01-03 18:47:05.385109	25	1
2021-01-04 08:47:05.436523	62	1
2021-01-04 18:47:05.486877	41	1

Accessing Time Series Data (4)

- Truncate observations after Jan. 3, 2021

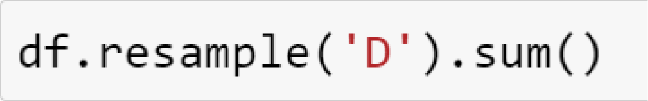


```
df.truncate(after='1/3/2021')
```

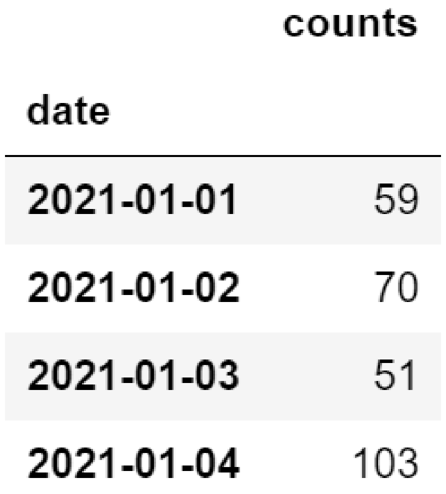


	counts
date	
2021-01-01 08:47:05.069722	34
2021-01-01 18:47:05.119994	25
2021-01-02 08:47:05.178768	26
2021-01-02 13:47:05.230071	15
2021-01-02 18:47:05.230071	15
2021-01-02 23:47:05.280592	14

- Total counts per day



```
df.resample('D').sum()
```



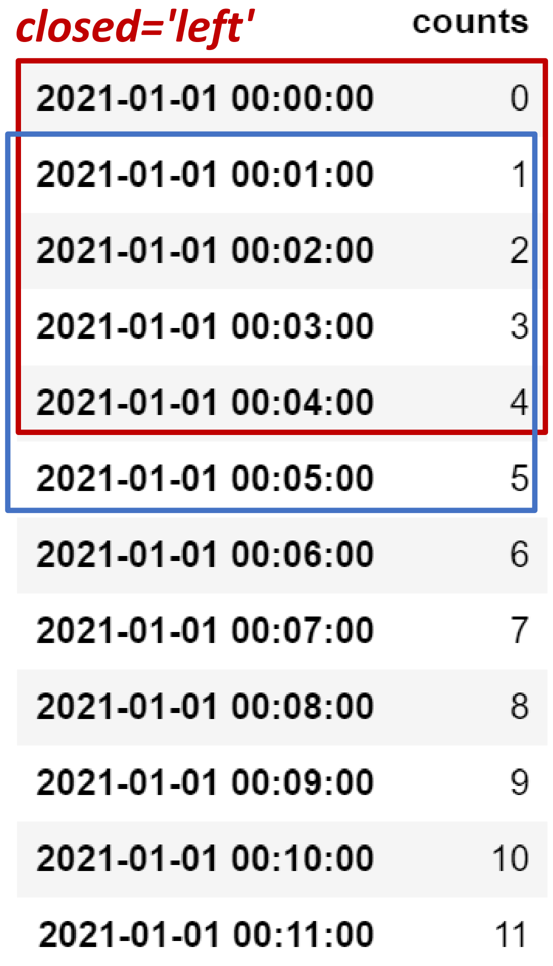
	counts
date	
2021-01-01	59
2021-01-02	70
2021-01-03	51
2021-01-04	103

Resampling

- The process of converting a time series from one frequency to another
  - Downsampling (similar to groupby operation): aggregating higher frequency data to lower frequency
  - Upsampling: converting lower frequency to higher frequency

- *df*.`resample(`*rule, axis=0, closed=None, label=None, …*`)`
  - *rule*: the offset string or object representing target conversion
  - *axis*: axis to use for up- or down-sampling
  - *closed*: which side of bin interval is closed (default: 'right' for 'M', 'A', 'Q', 'BM', 'BA', 'BQ', and 'W', and 'left' for others)
  - *label*: which bin edge label to label bucket with (same default value as *closed*)

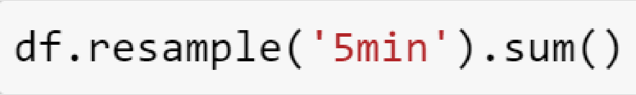
Downsampling



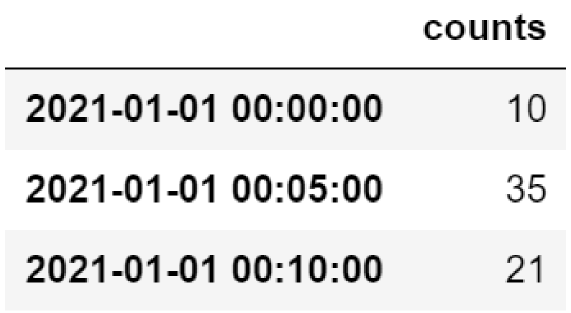
closed='left'

	counts
2021-01-01 00:00:00	0
2021-01-01 00:01:00	1
2021-01-01 00:02:00	2
2021-01-01 00:03:00	3
2021-01-01 00:04:00	4
2021-01-01 00:05:00	5
2021-01-01 00:06:00	6
2021-01-01 00:07:00	7
2021-01-01 00:08:00	8
2021-01-01 00:09:00	9
2021-01-01 00:10:00	10
2021-01-01 00:11:00	11

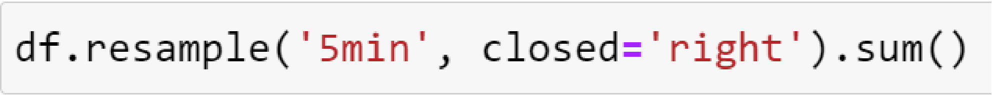
closed='right'



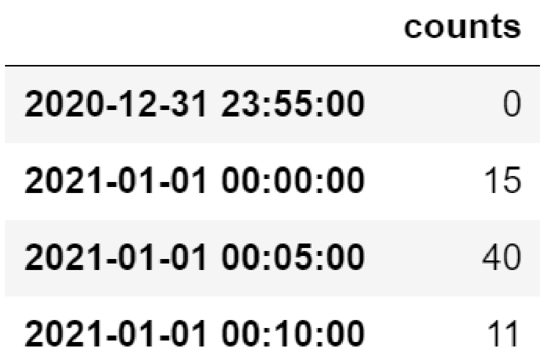
```
df.resample('5min').sum()
```



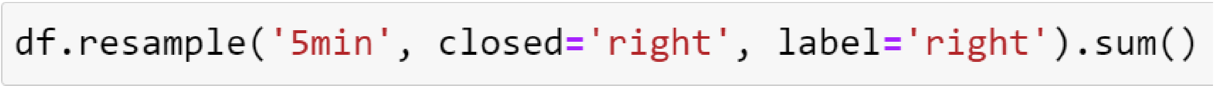
	counts
2021-01-01 00:00:00	10
2021-01-01 00:05:00	35
2021-01-01 00:10:00	21



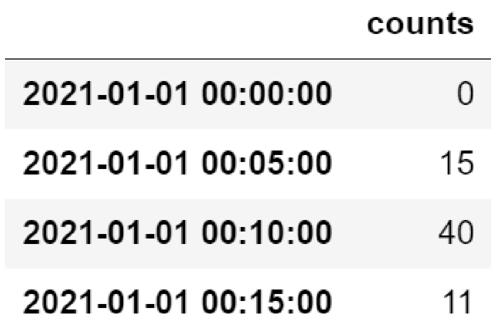
```
df.resample('5min', closed='right').sum()
```



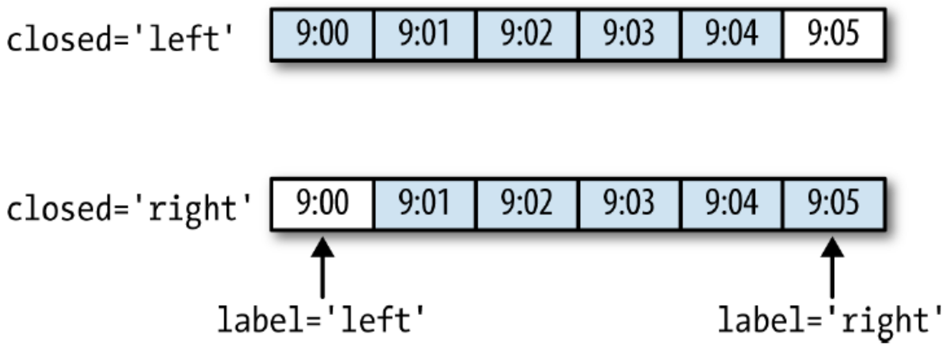
	counts
2020-12-31 23:55:00	0
2021-01-01 00:00:00	15
2021-01-01 00:05:00	40
2021-01-01 00:10:00	11



```
df.resample('5min', closed='right', label='right').sum()
```

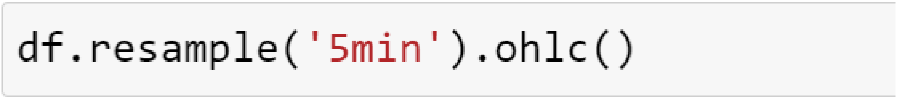


	counts
2021-01-01 00:00:00	0
2021-01-01 00:05:00	15
2021-01-01 00:10:00	40
2021-01-01 00:15:00	11

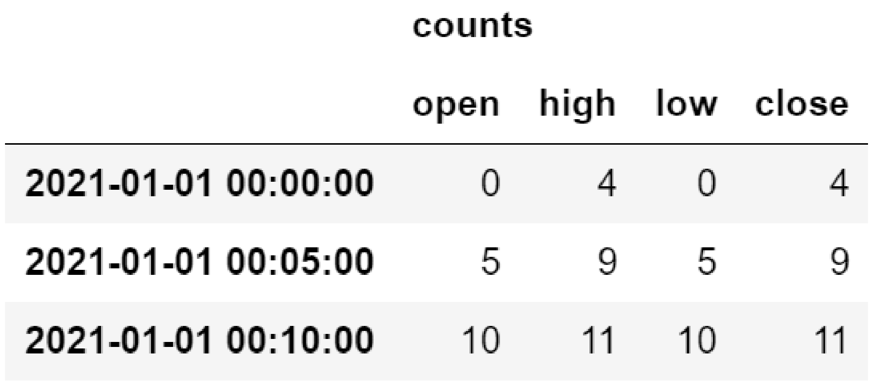


OHLC Resampling

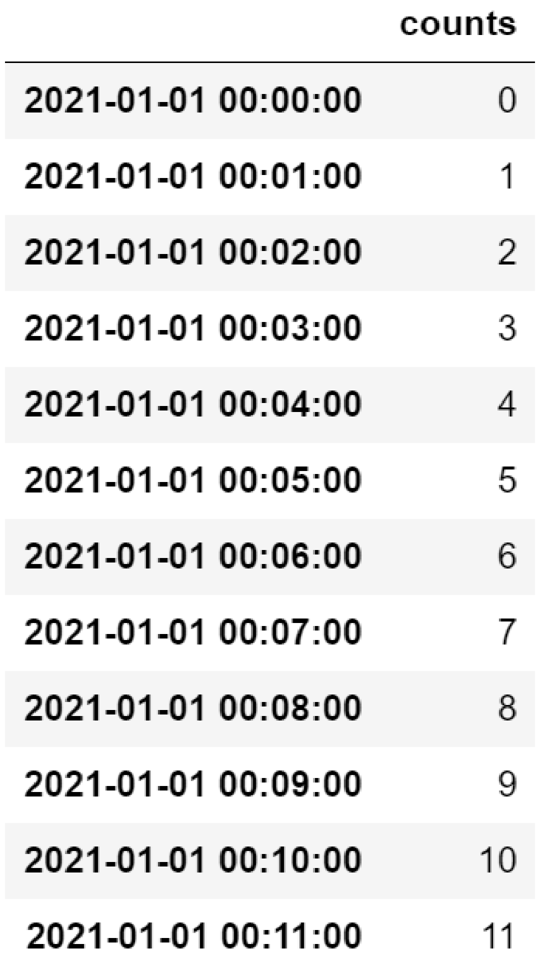
- Open-High-Low-Close resampling
  - Used in finance



```python
df.resample('5min').ohlc()
```

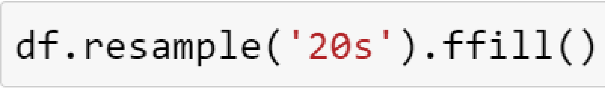


	counts			
	open	high	low	close
2021-01-01 00:00:00	0	4	0	4
2021-01-01 00:05:00	5	9	5	9
2021-01-01 00:10:00	10	11	10	11

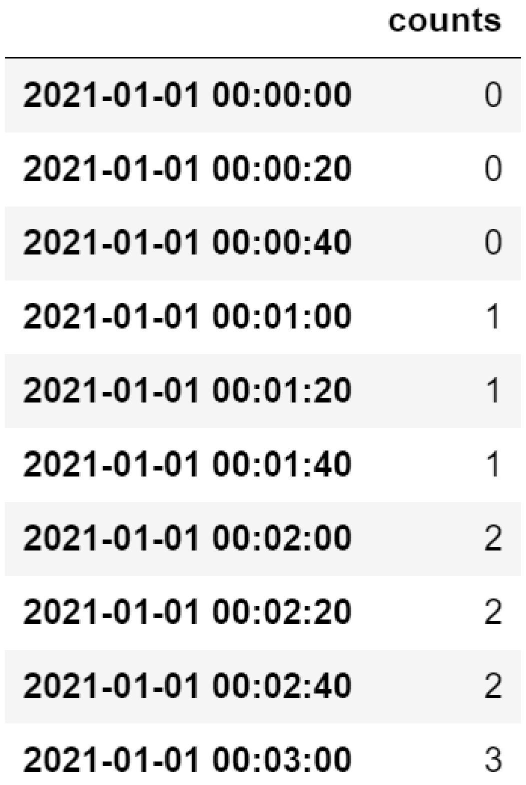


	counts
2021-01-01 00:00:00	0
2021-01-01 00:01:00	1
2021-01-01 00:02:00	2
2021-01-01 00:03:00	3
2021-01-01 00:04:00	4
2021-01-01 00:05:00	5
2021-01-01 00:06:00	6
2021-01-01 00:07:00	7
2021-01-01 00:08:00	8
2021-01-01 00:09:00	9
2021-01-01 00:10:00	10
2021-01-01 00:11:00	11

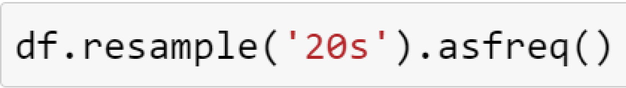
Upsampling



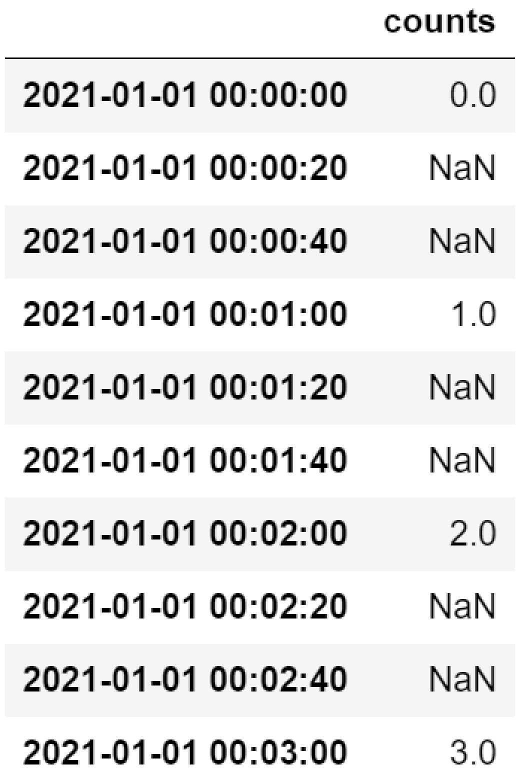
```
df.resample('20s').ffill()
```



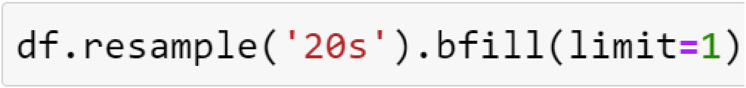
	counts
2021-01-01 00:00:00	0
2021-01-01 00:00:20	0
2021-01-01 00:00:40	0
2021-01-01 00:01:00	1
2021-01-01 00:01:20	1
2021-01-01 00:01:40	1
2021-01-01 00:02:00	2
2021-01-01 00:02:20	2
2021-01-01 00:02:40	2
2021-01-01 00:03:00	3



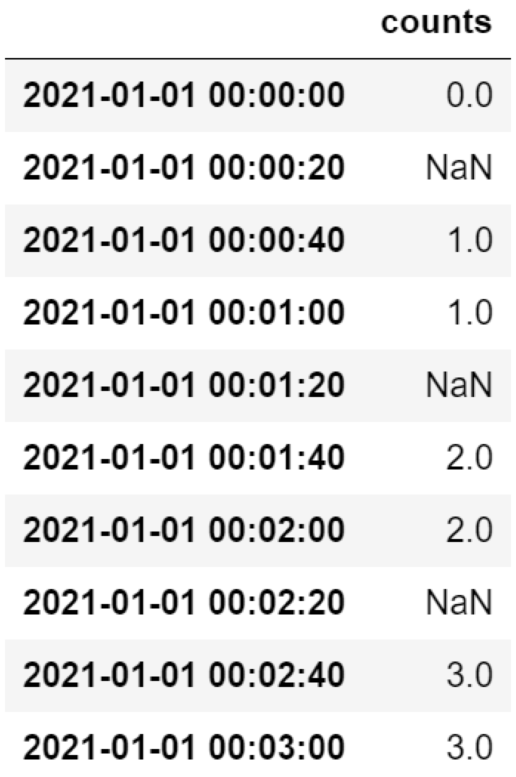
```
df.resample('20s').asfreq()
```



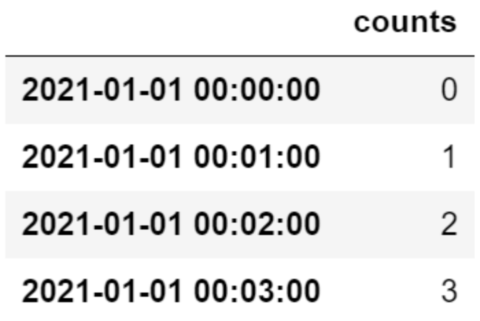
	counts
2021-01-01 00:00:00	0.0
2021-01-01 00:00:20	NaN
2021-01-01 00:00:40	NaN
2021-01-01 00:01:00	1.0
2021-01-01 00:01:20	NaN
2021-01-01 00:01:40	NaN
2021-01-01 00:02:00	2.0
2021-01-01 00:02:20	NaN
2021-01-01 00:02:40	NaN
2021-01-01 00:03:00	3.0



```
df.resample('20s').bfill(limit=1)
```

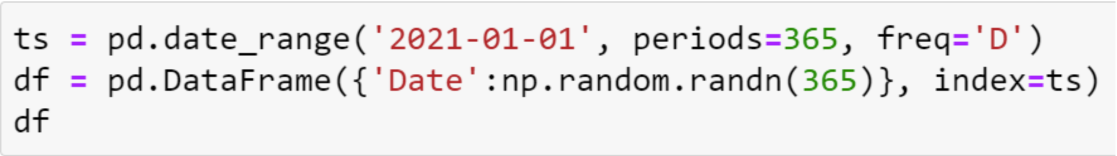


	counts
2021-01-01 00:00:00	0.0
2021-01-01 00:00:20	NaN
2021-01-01 00:00:40	1.0
2021-01-01 00:01:00	1.0
2021-01-01 00:01:20	NaN
2021-01-01 00:01:40	2.0
2021-01-01 00:02:00	2.0
2021-01-01 00:02:20	NaN
2021-01-01 00:02:40	3.0
2021-01-01 00:03:00	3.0

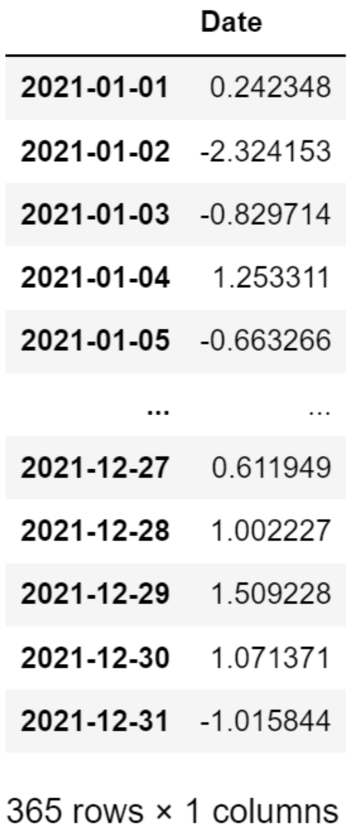


	counts
2021-01-01 00:00:00	0
2021-01-01 00:01:00	1
2021-01-01 00:02:00	2
2021-01-01 00:03:00	3

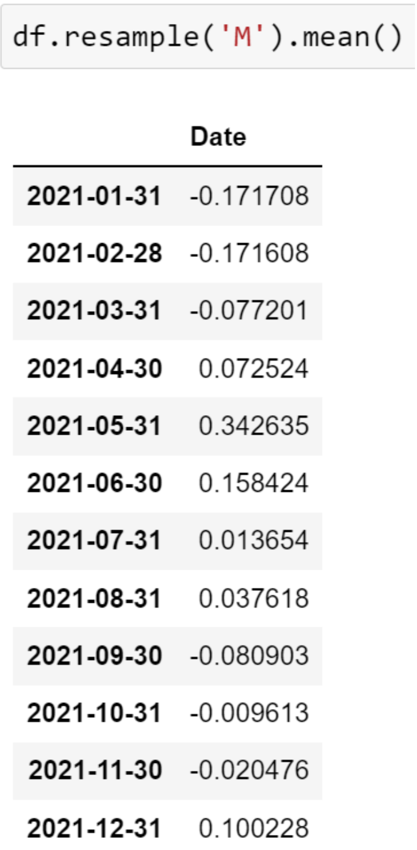
Downsampling with Periods



```
ts = pd.date_range('2021-01-01', periods=365, freq='D')
df = pd.DataFrame({'Date':np.random.randn(365)}, index=ts)
df
```

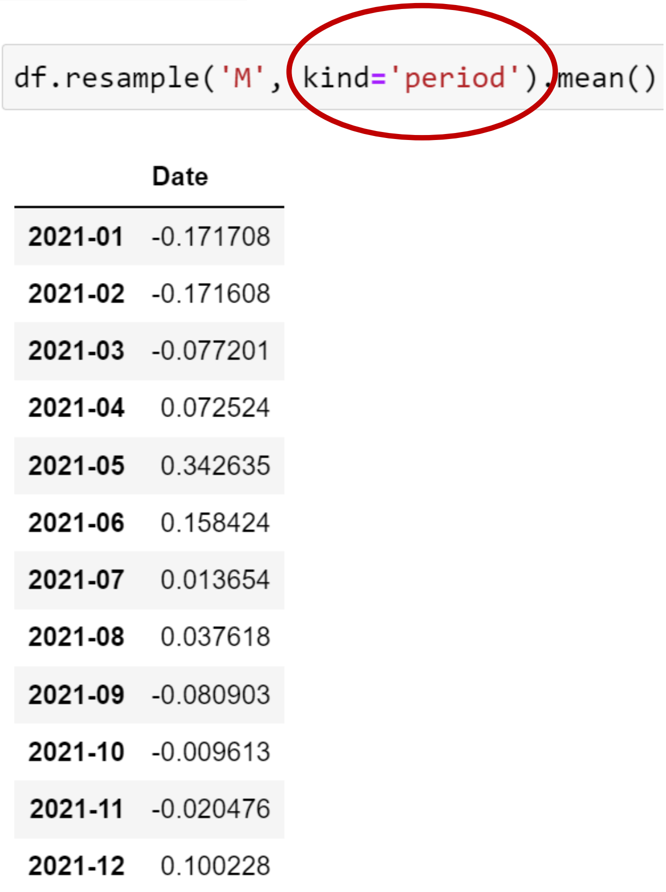


```
                Date
2021-01-01   0.242348
2021-01-02  -2.324153
2021-01-03  -0.829714
2021-01-04   1.253311
2021-01-05  -0.663266
       ...        ...
2021-12-27   0.611949
2021-12-28   1.002227
2021-12-29   1.509228
2021-12-30   1.071371
2021-12-31  -1.015844

365 rows × 1 columns
```



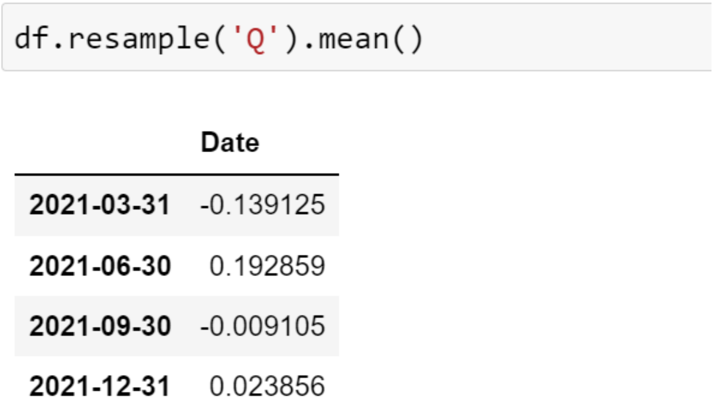
```
df.resample('M').mean()
```

```
                Date
2021-01-31  -0.171708
2021-02-28  -0.171608
2021-03-31  -0.077201
2021-04-30   0.072524
2021-05-31   0.342635
2021-06-30   0.158424
2021-07-31   0.013654
2021-08-31   0.037618
2021-09-30  -0.080903
2021-10-31  -0.009613
2021-11-30  -0.020476
2021-12-31   0.100228
```



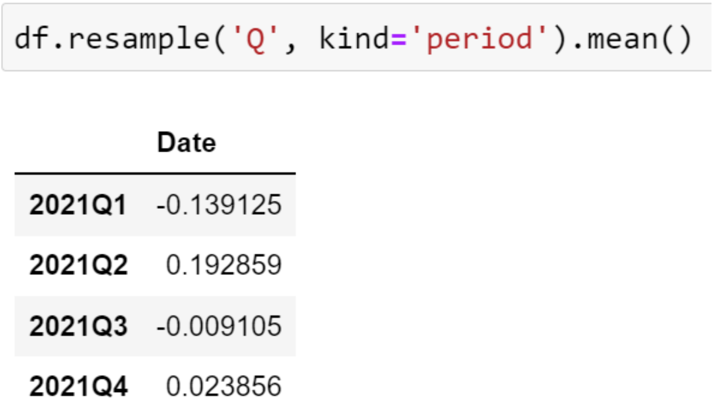
```
df.resample('M', kind='period').mean()
```

```
             Date
2021-01  -0.171708
2021-02  -0.171608
2021-03  -0.077201
2021-04   0.072524
2021-05   0.342635
2021-06   0.158424
2021-07   0.013654
2021-08   0.037618
2021-09  -0.080903
2021-10  -0.009613
2021-11  -0.020476
2021-12   0.100228
```



```
df.resample('Q').mean()
```

```
                Date
2021-03-31  -0.139125
2021-06-30   0.192859
2021-09-30  -0.009105
2021-12-31   0.023856
```



```
df.resample('Q', kind='period').mean()
```

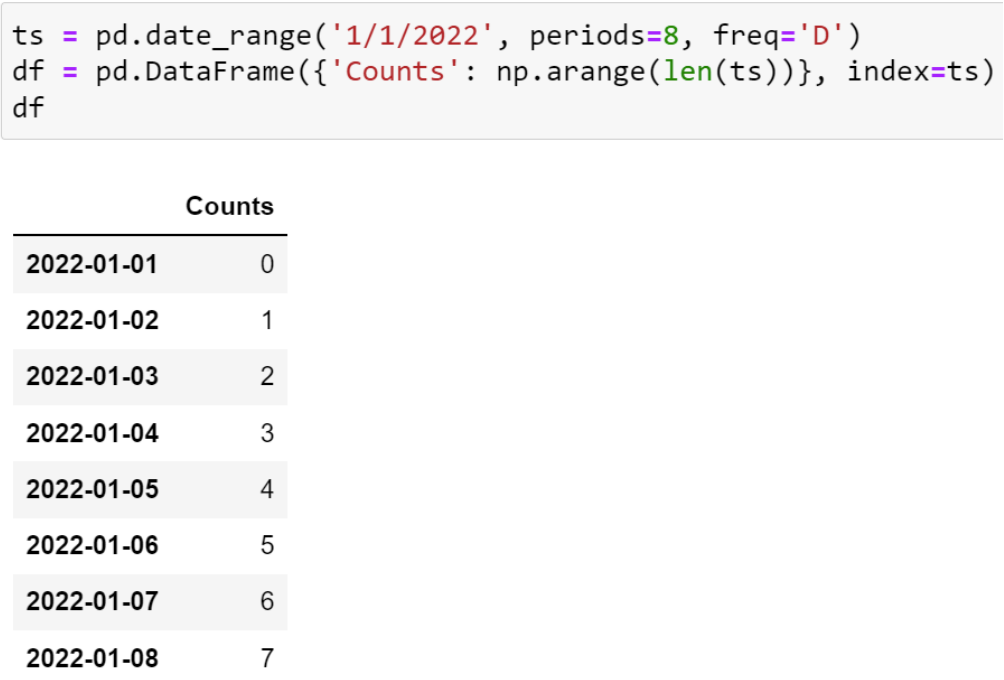
```
            Date
2021Q1  -0.139125
2021Q2   0.192859
2021Q3  -0.009105
2021Q4   0.023856
```

Aggregation Functions

function	Description
`.asfreq()`	Return the values at the new freq, essentially a reindex
`.fillna()`	Fill missing values introduced by upsampling ('ffill', 'bfill', 'nearest')
`.ffill() / .bfill()`	Forward (or backward) fill the values
`.first() / .last()`	Compute first (or last) of group values
`.min() / .max()`	Compute min (or max) of group values
`.mean() / .median()`	Compute mean (or median) of groups, excluding missing values
`.sum()`	Compute sum of group values
`.std() / .var()`	Compute standard deviation (or variance) of groups, excluding missing values
`.ohlc()`	Compute open, high, low and close values of a group, excluding missing values
`.count()`	Compute count of group, excluding missing values
`.nunique()`	Return number of unique elements in the group
`.quantile()`	Return value at the given quantile
`.interpolate()`	Interpolate values according to different methods

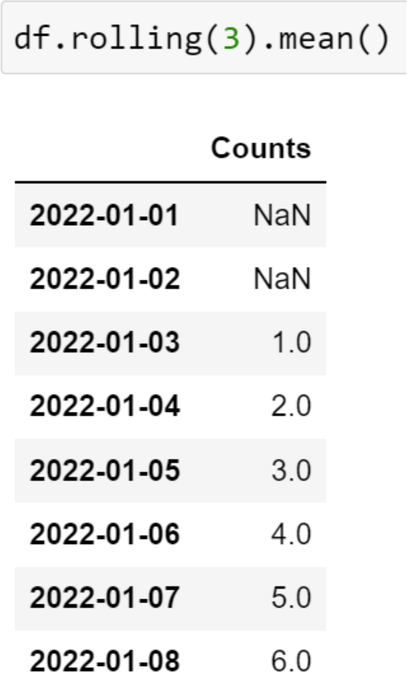
Moving Window Functions: rolling()

- *df*.`rolling(`*window, min_periods=None, …*`)`
  - Provide rolling window calculations
  - *window*: size of the moving window
  - *min_periods*: minimum number of observations in window required to have a value



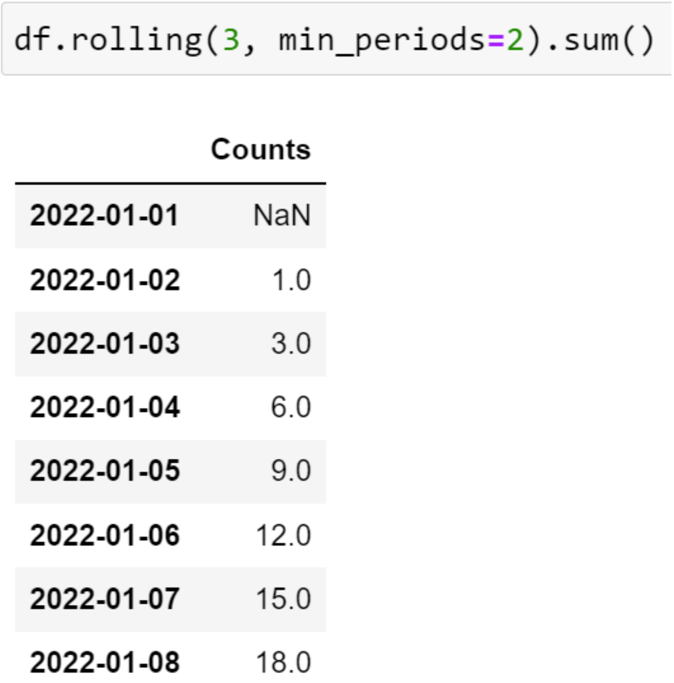
```
ts = pd.date_range('1/1/2022', periods=8, freq='D')
df = pd.DataFrame({'Counts': np.arange(len(ts))}, index=ts)
df
```

	Counts
2022-01-01	0
2022-01-02	1
2022-01-03	2
2022-01-04	3
2022-01-05	4
2022-01-06	5
2022-01-07	6
2022-01-08	7



```
df.rolling(3).mean()
```

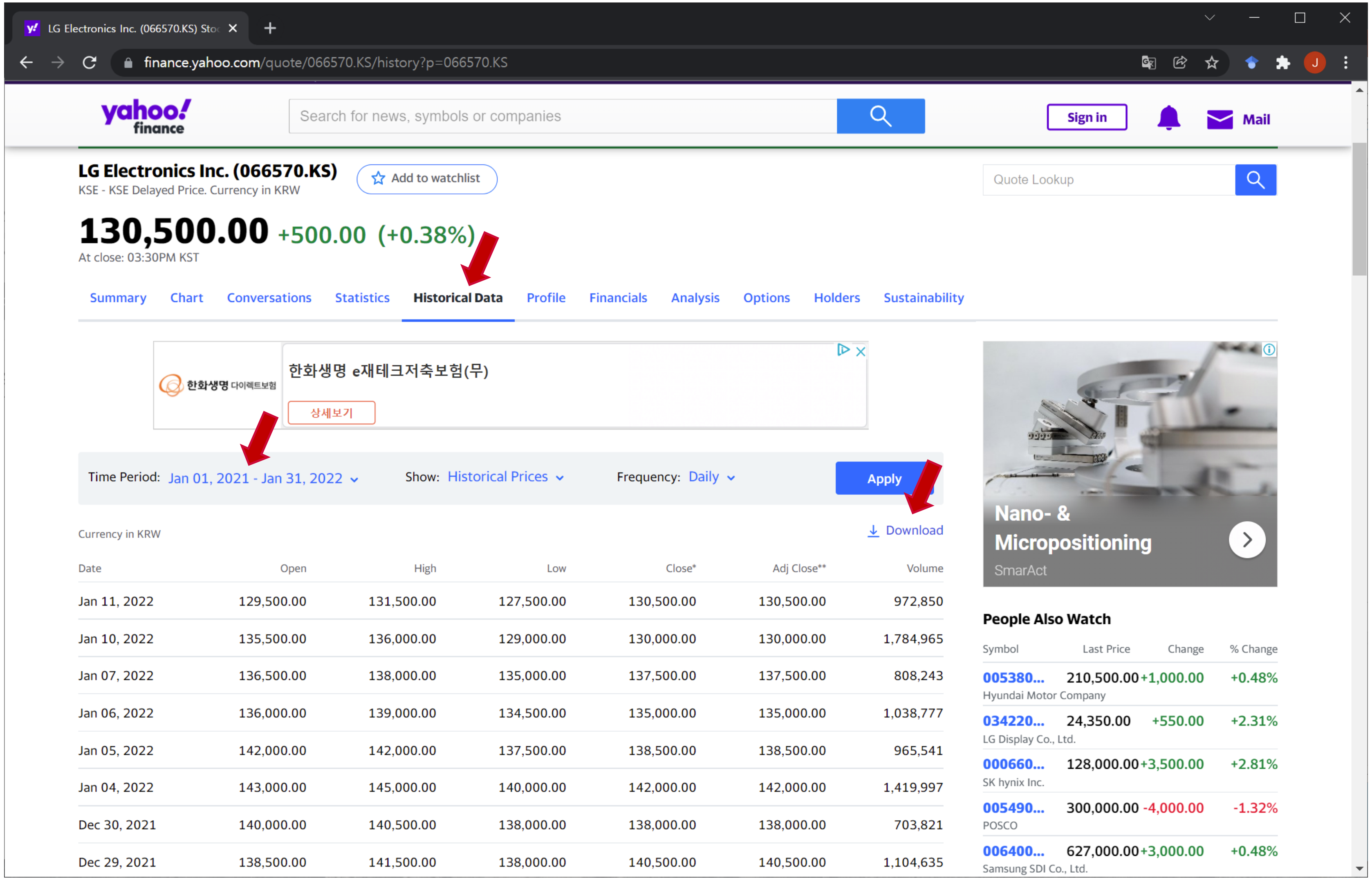
	Counts
2022-01-01	NaN
2022-01-02	NaN
2022-01-03	1.0
2022-01-04	2.0
2022-01-05	3.0
2022-01-06	4.0
2022-01-07	5.0
2022-01-08	6.0



```
df.rolling(3, min_periods=2).sum()
```

	Counts
2022-01-01	NaN
2022-01-02	1.0
2022-01-03	3.0
2022-01-04	6.0
2022-01-05	9.0
2022-01-06	12.0
2022-01-07	15.0
2022-01-08	18.0

Example: Stock Data (1)



LG Electronics Inc. (066570.KS) Sto
finance.yahoo.com/quote/066570.KS/history?p=066570.KS
yahoo! finance
Search for news, symbols or companies
Sign in
Mail
Quote Lookup

LG Electronics Inc. (066570.KS) ☆ Add to watchlist

KSE - KSE Delayed Price. Currency in KRW

130,500.00 +500.00 (+0.38%)

At close: 03:30PM KST

Time Period: Jan 01, 2021 - Jan 31, 2022 Show: Historical Prices Frequency: Daily Apply

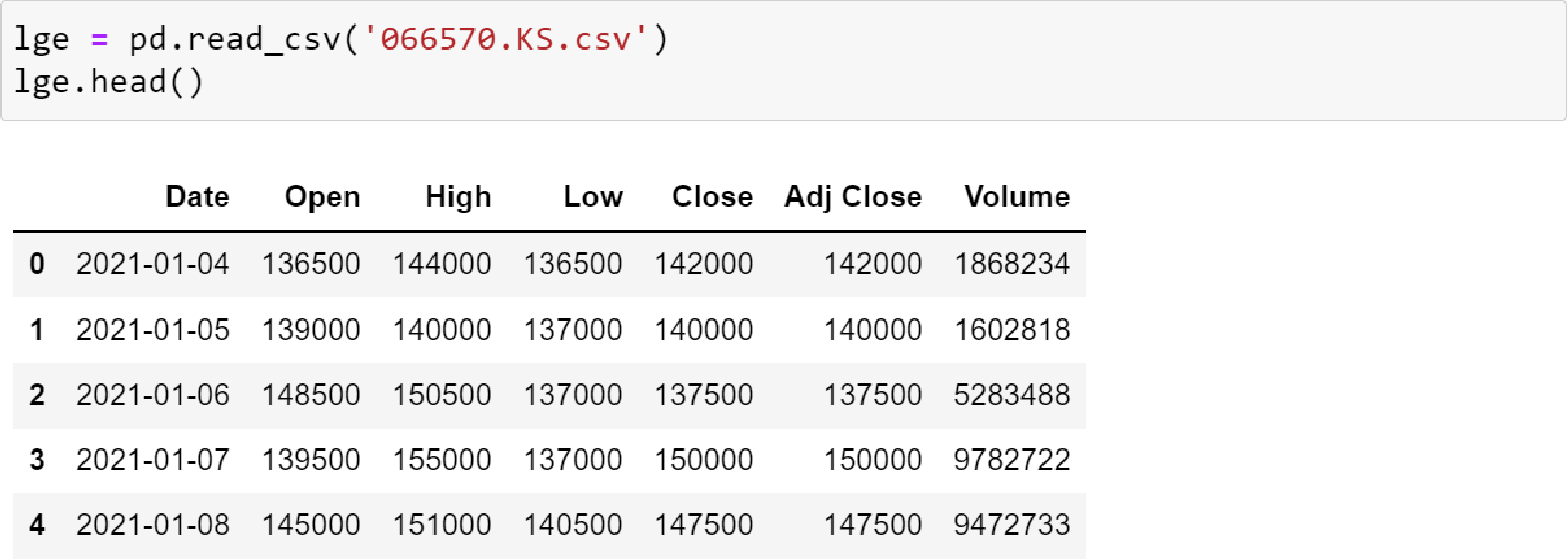
Currency in KRW Download

Date	Open	High	Low	Close*	Adj Close**	Volume
Jan 11, 2022	129,500.00	131,500.00	127,500.00	130,500.00	130,500.00	972,850
Jan 10, 2022	135,500.00	136,000.00	129,000.00	130,000.00	130,000.00	1,784,965
Jan 07, 2022	136,500.00	138,000.00	135,000.00	137,500.00	137,500.00	808,243
Jan 06, 2022	136,000.00	139,000.00	134,500.00	135,000.00	135,000.00	1,038,777
Jan 05, 2022	142,000.00	142,000.00	137,500.00	138,500.00	138,500.00	965,541
Jan 04, 2022	143,000.00	145,000.00	140,000.00	142,000.00	142,000.00	1,419,997
Dec 30, 2021	140,000.00	140,500.00	138,000.00	138,000.00	138,000.00	703,821
Dec 29, 2021	138,500.00	141,500.00	138,000.00	140,500.00	140,500.00	1,104,635

People Also Watch

Symbol	Last Price	Change	% Change
005380... Hyundai Motor Company	210,500.00	+1,000.00	+0.48%
034220... LG Display Co., Ltd.	24,350.00	+550.00	+2.31%
000660... SK hynix Inc.	128,000.00	+3,500.00	+2.81%
005490... POSCO	300,000.00	-4,000.00	-1.32%
006400... Samsung SDI Co., Ltd.	627,000.00	+3,000.00	+0.48%

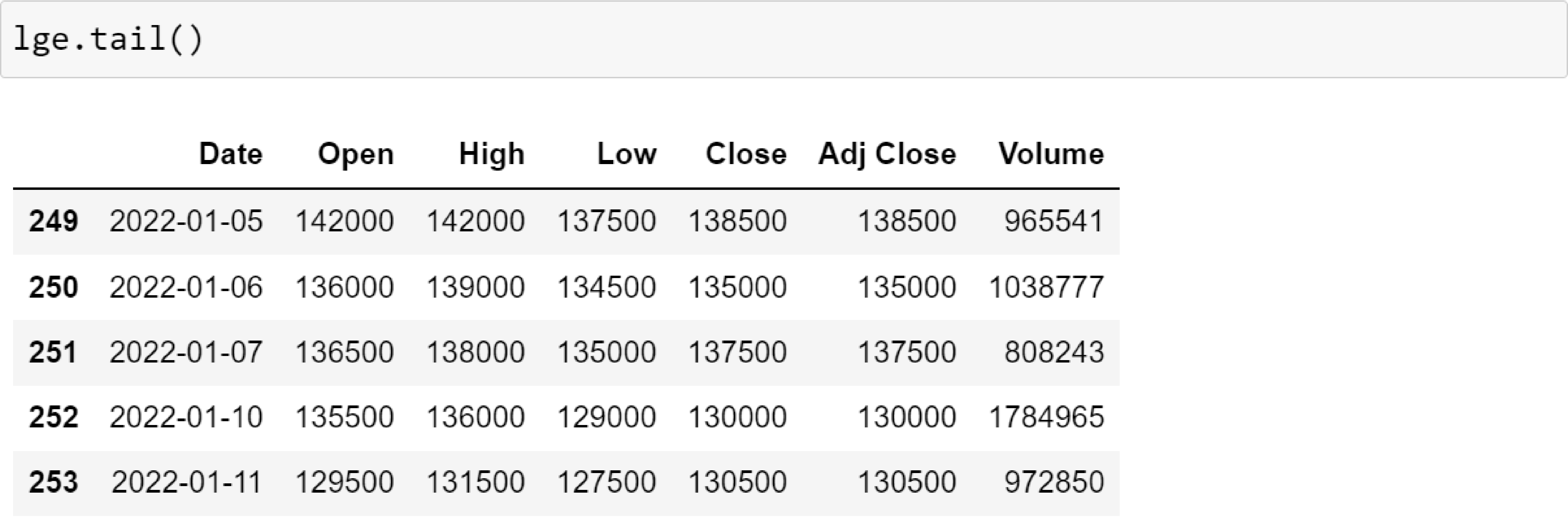
Example: Stock Data (2)



```
lge = pd.read_csv('066570.KS.csv')
lge.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2021-01-04	136500	144000	136500	142000	142000	1868234
1	2021-01-05	139000	140000	137000	140000	140000	1602818
2	2021-01-06	148500	150500	137000	137500	137500	5283488
3	2021-01-07	139500	155000	137000	150000	150000	9782722
4	2021-01-08	145000	151000	140500	147500	147500	9472733

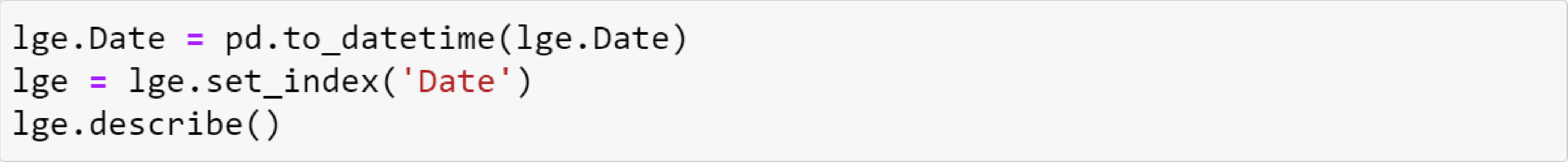
Example: Stock Data (3)



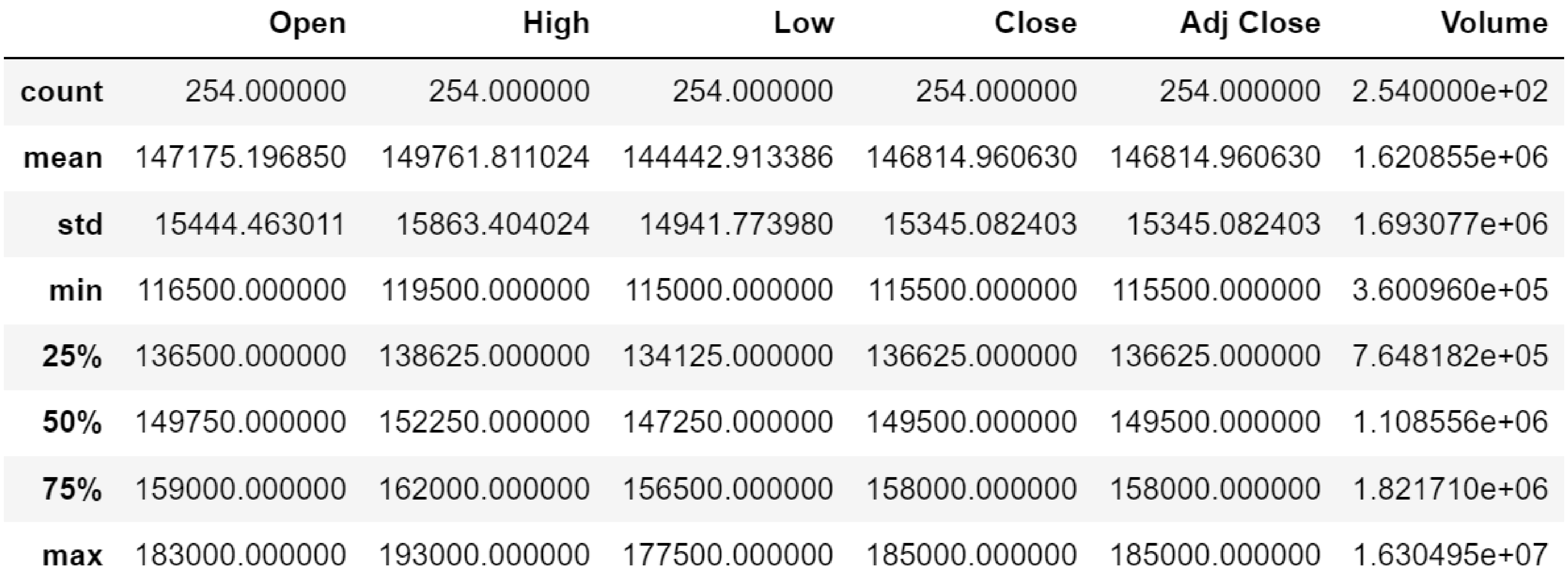
```
lge.tail()
```

	Date	Open	High	Low	Close	Adj Close	Volume
249	2022-01-05	142000	142000	137500	138500	138500	965541
250	2022-01-06	136000	139000	134500	135000	135000	1038777
251	2022-01-07	136500	138000	135000	137500	137500	808243
252	2022-01-10	135500	136000	129000	130000	130000	1784965
253	2022-01-11	129500	131500	127500	130500	130500	972850

Example: Stock Data (4)

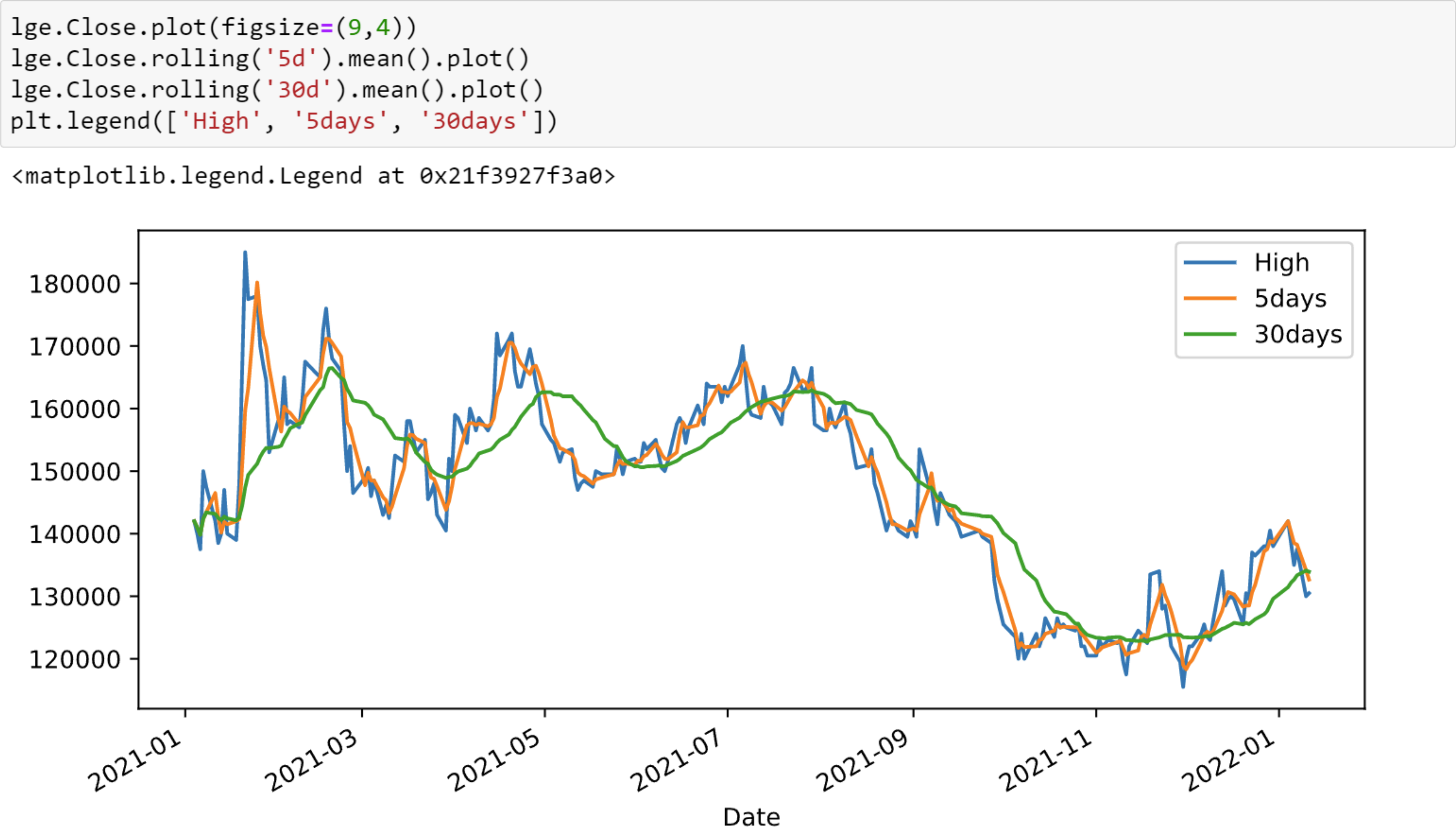


```python
lge.Date = pd.to_datetime(lge.Date)
lge = lge.set_index('Date')
lge.describe()
```



	Open	High	Low	Close	Adj Close	Volume
count	254.000000	254.000000	254.000000	254.000000	254.000000	2.540000e+02
mean	147175.196850	149761.811024	144442.913386	146814.960630	146814.960630	1.620855e+06
std	15444.463011	15863.404024	14941.773980	15345.082403	15345.082403	1.693077e+06
min	116500.000000	119500.000000	115000.000000	115500.000000	115500.000000	3.600960e+05
25%	136500.000000	138625.000000	134125.000000	136625.000000	136625.000000	7.648182e+05
50%	149750.000000	152250.000000	147250.000000	149500.000000	149500.000000	1.108556e+06
75%	159000.000000	162000.000000	156500.000000	158000.000000	158000.000000	1.821710e+06
max	183000.000000	193000.000000	177500.000000	185000.000000	185000.000000	1.630495e+07

Example: Stock Data (5)



```
lge.Close.plot(figsize=(9,4))
lge.Close.rolling('5d').mean().plot()
lge.Close.rolling('30d').mean().plot()
plt.legend(['High', '5days', '30days'])
```

```
<matplotlib.legend.Legend at 0x21f3927f3a0>
```

Moving Window Functions: ewm()

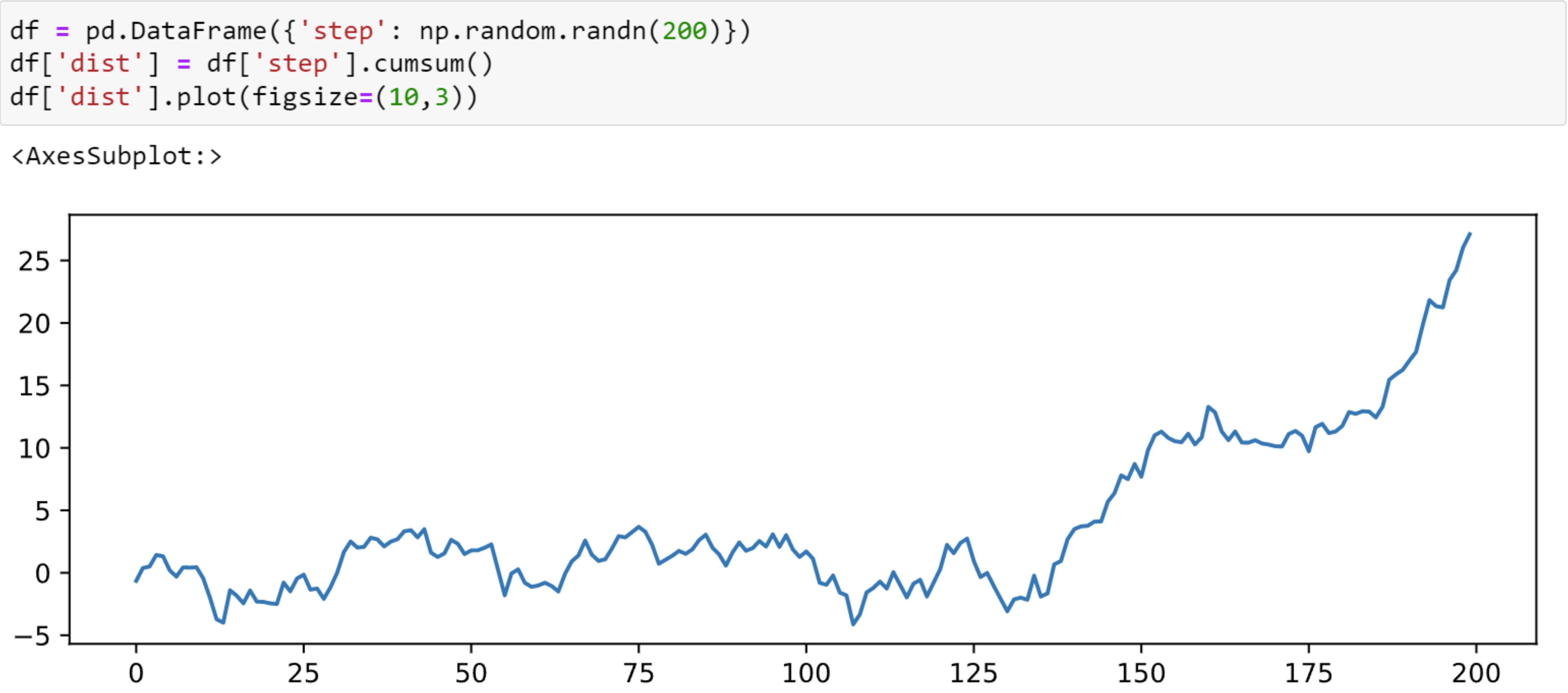
- *df*.ewm(*alpha=None, min_periods=0, adjust=True, …*)
  - Provide exponential weighted (EW) functions
  - *alpha*: specify smoothing factor α directly, $0 < \alpha \leq 1$
  - When *adjust*=True (default), the weighted moving average is calculated with $w_i = (1-\alpha)^i$:

$$\boldsymbol{y_t = \frac{\sum_{i=0}^{t} w_i x_{t-1}}{\sum_{i=0}^{t} w_i} = \frac{x_t + (1-\alpha)x_{t-1} + (1-\alpha)^2 x_{t-2} + \cdots + (1-\alpha)^t x_0}{1 + (1-\alpha) + (1-\alpha)^2 + \cdots + (1-\alpha)^t}}$$

  - When *adjust*=False, the weighted moving average is calculated recursively:

$$\boldsymbol{y_0 = x_0}$$
$$\boldsymbol{y_t = (1-\alpha)y_{t-1} + \alpha x_t}$$

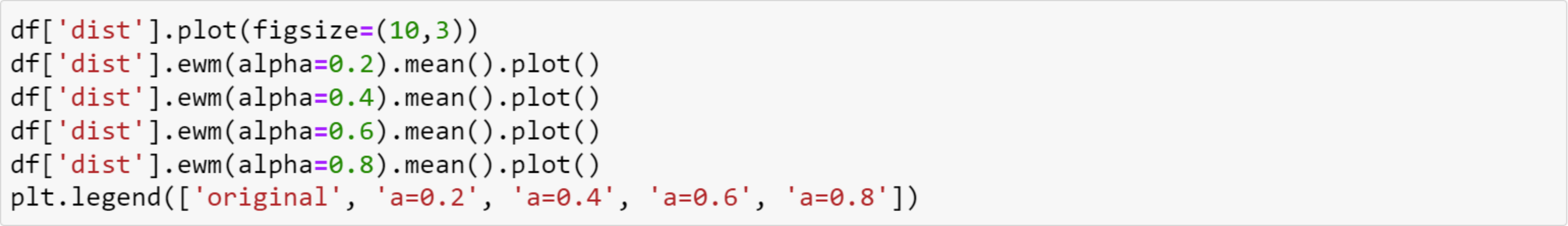
emw(): Example (1)



```
df = pd.DataFrame({'step': np.random.randn(200)})
df['dist'] = df['step'].cumsum()
df['dist'].plot(figsize=(10,3))
```

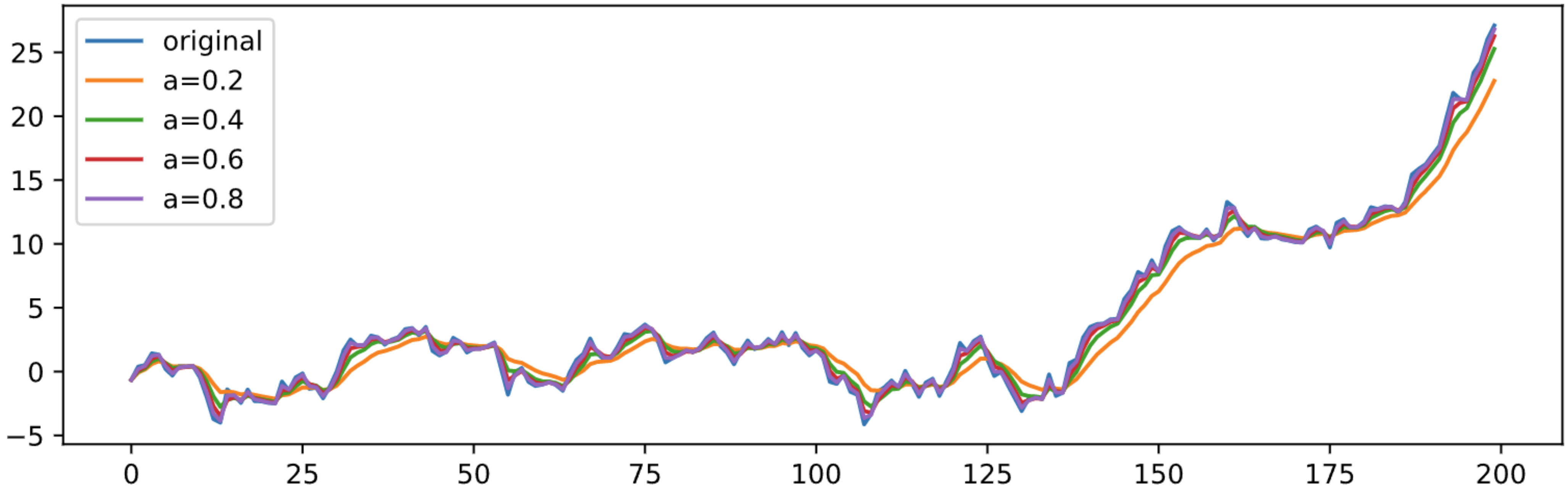
```
<AxesSubplot:>
```

emw(): Example (2)

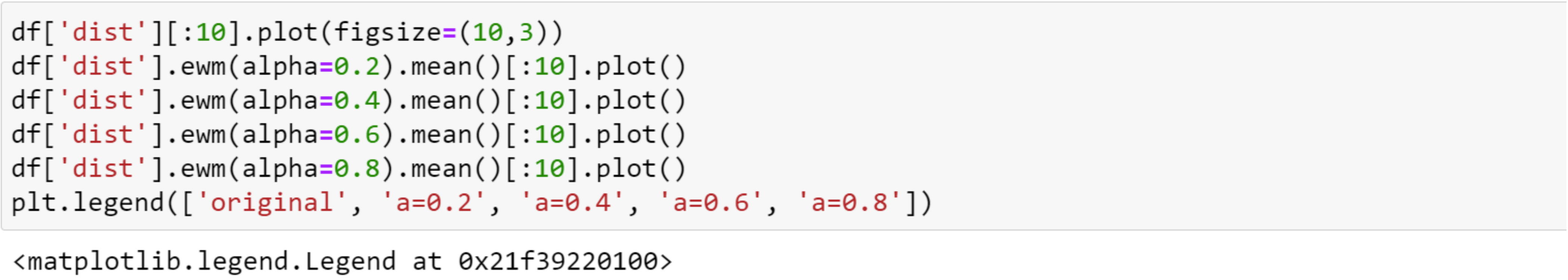


```
df['dist'].plot(figsize=(10,3))
df['dist'].ewm(alpha=0.2).mean().plot()
df['dist'].ewm(alpha=0.4).mean().plot()
df['dist'].ewm(alpha=0.6).mean().plot()
df['dist'].ewm(alpha=0.8).mean().plot()
plt.legend(['original', 'a=0.2', 'a=0.4', 'a=0.6', 'a=0.8'])
```

```
<matplotlib.legend.Legend at 0x21f391691c0>
```

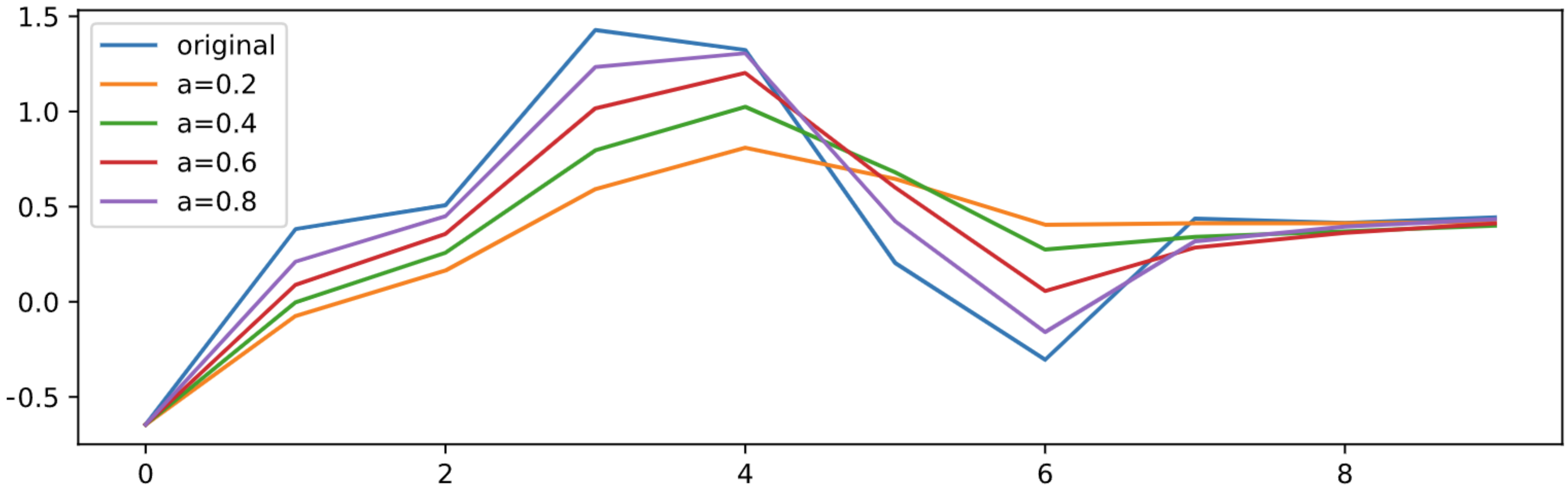


emw(): Example (3)



```
df['dist'][:10].plot(figsize=(10,3))
df['dist'].ewm(alpha=0.2).mean()[:10].plot()
df['dist'].ewm(alpha=0.4).mean()[:10].plot()
df['dist'].ewm(alpha=0.6).mean()[:10].plot()
df['dist'].ewm(alpha=0.8).mean()[:10].plot()
plt.legend(['original', 'a=0.2', 'a=0.4', 'a=0.6', 'a=0.8'])
```

```
<matplotlib.legend.Legend at 0x21f39220100>
```


original
a=0.2
a=0.4
a=0.6
a=0.8