01-Time-Resampling

October 19, 2022

---

1 Time Resampling

Let's learn how to sample time series data! This will be useful later on in the course!

[1]:
```
import pandas as pd
%matplotlib inline
```

1.1 Import the data

For this exercise we'll look at Starbucks stock data from 2015 to 2018 which includes daily closing prices and trading volumes.

[2]:
```
df = pd.read_csv('../Data/starbucks.csv', index_col='Date', parse_dates=True)
```

Note: the above code is a faster way of doing the following:

[3]:
```
df.head()
```

[3]:
```
              Close    Volume
Date
2015-01-02  38.0061   6906098
2015-01-05  37.2781  11623796
2015-01-06  36.9748   7664340
2015-01-07  37.8848   9732554
2015-01-08  38.4961  13170548
```

1.2 resample()

A common operation with time series data is resampling based on the time series index. Let's see how to use the resample() method. [reference]

[4]:
```
# Our index
df.index
```

[4]:
```
DatetimeIndex(['2015-01-02', '2015-01-05', '2015-01-06', '2015-01-07',
               '2015-01-08', '2015-01-09', '2015-01-12', '2015-01-13',
               '2015-01-14', '2015-01-15',
               …
               '2018-12-17', '2018-12-18', '2018-12-19', '2018-12-20',
               '2018-12-21', '2018-12-24', '2018-12-26', '2018-12-27',
               '2018-12-28', '2018-12-31'],
              dtype='datetime64[ns]', name='Date', length=1006, freq=None)
```

When calling `.resample()` you first need to pass in a **rule** parameter, then you need to call some sort of aggregation function.

The **rule** parameter describes the frequency with which to apply the aggregation function (daily, monthly, yearly, etc.) It is passed in using an "offset alias" - refer to the table below. [reference]

The aggregation function is needed because, due to resampling, we need some sort of mathematical rule to join the rows (mean, sum, count, etc.)

`<caption style="text-align: center"><strong>TIME SERIES OFFSET ALIASES</strong></caption>`

ALIAS	DESCRIPTION
B	business day frequency
C	custom business day frequency (experimental)
D	calendar day frequency
W	weekly frequency
M	month end frequency
SM	semi-month end frequency (15th and end of month)
BM	business month end frequency
CBM	custom business month end frequency

MS

month start frequency

SMS

semi-month start frequency (1st and 15th)

BMS

business month start frequency

CBMS

custom business month start frequency

Q

quarter end frequency

intentionally left blank

ALIAS

DESCRIPTION

BQ

business quarter endfrequency

QS

quarter start frequency

BQS

business quarter start frequency

A

year end frequency

BA

business year end frequency

AS

year start frequency

BAS

business year start frequency

BH

business hour frequency

H

hourly frequency

T, min

minutely frequency

S

secondly frequency

L, ms

milliseconds

U, us

microseconds

N

nanoseconds

[5]:
```
# Yearly Means
df.resample(rule='A').mean()
```

[5]:
```
                Close        Volume
Date
2015-12-31  50.078100  8.649190e+06
2016-12-31  53.891732  9.300633e+06
2017-12-31  55.457310  9.296078e+06
2018-12-31  56.870005  1.122883e+07
```

Resampling rule 'A' takes all of the data points in a given year, applies the aggregation function (in this case we calculate the mean), and reports the result as the last day of that year.

1.2.1 Custom Resampling Functions

We're not limited to pandas built-in summary functions (min/max/mean etc.). We can define our own function:

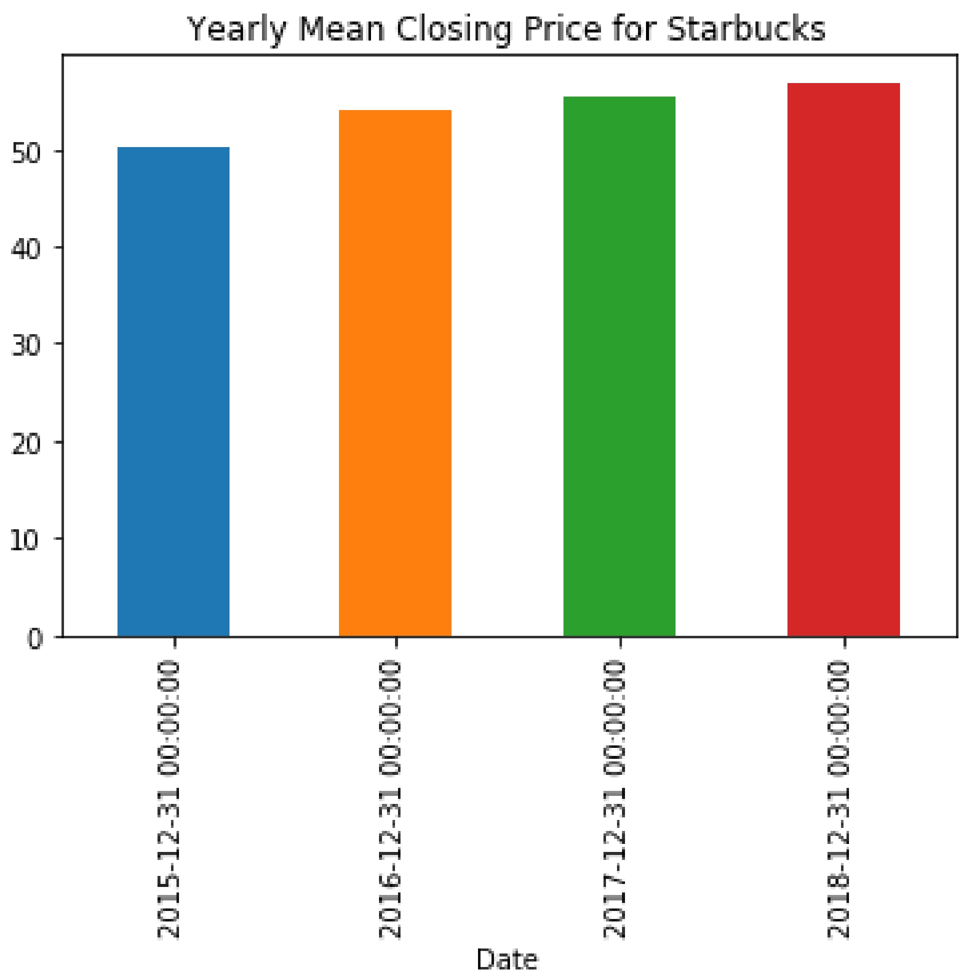
[6]:
```
def first_day(entry):
    """
    Returns the first instance of the period, regardless of sampling rate.
    """
    if len(entry):  # handles the case of missing data
        return entry[0]
```

[7]:
```
df.resample(rule='A').apply(first_day)
```

[7]:
```
              Close    Volume
Date
2015-12-31  38.0061   6906098
2016-12-31  55.0780  13521544
2017-12-31  53.1100   7809307
2018-12-31  56.3243   7215978
```

1.2.2 Plotting

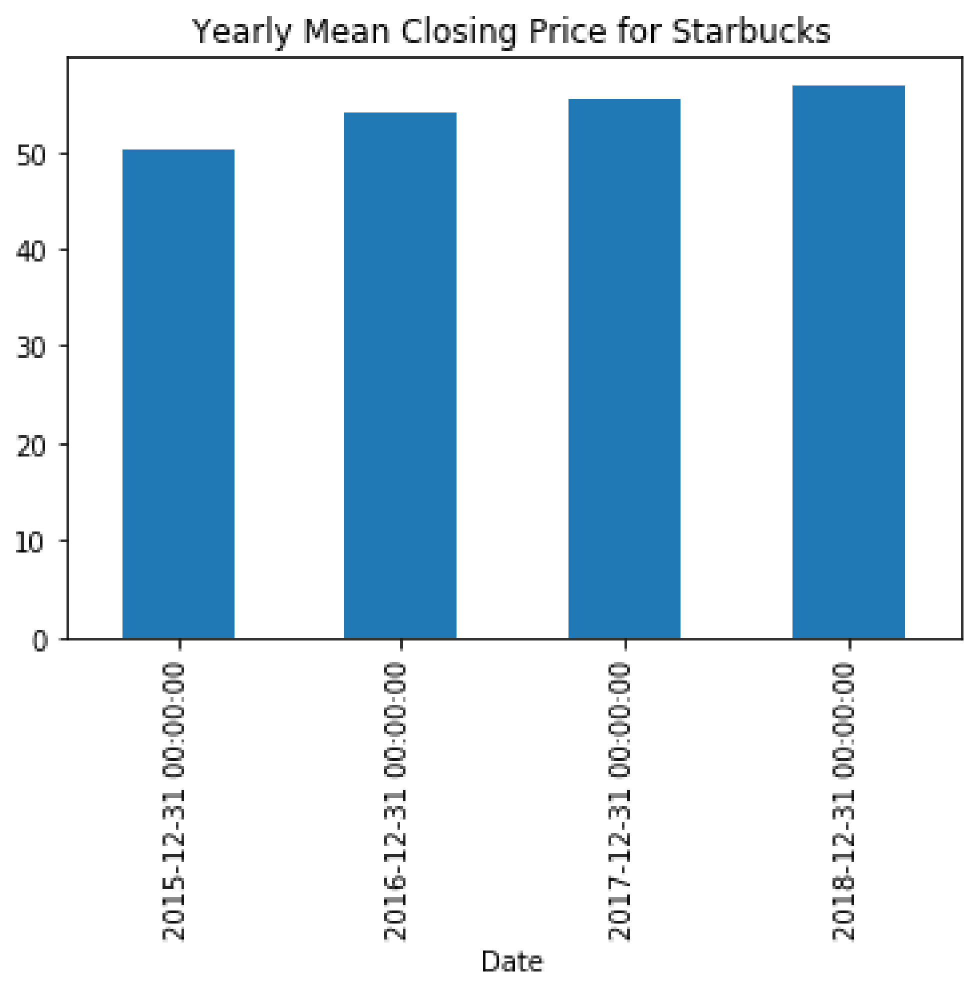
[8]:
```
df['Close'].resample('A').mean().plot.bar(title='Yearly Mean Closing Price for␣
↪Starbucks');
```



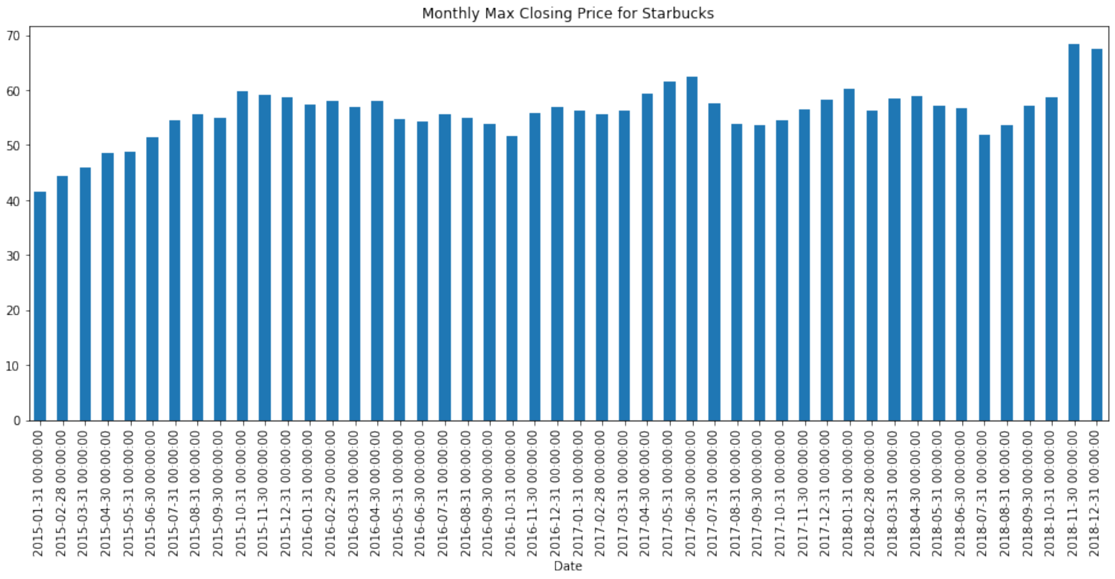
Pandas treats each sample as its own trace, and by default assigns different colors to each one. If you want, you can pass a color argument to assign your own color collection, or to set a uniform color. For example, color='#1f77b4' sets a uniform "steel blue" color.

Also, the above code can be broken into two lines for improved readability.

[9]:
```
title = 'Yearly Mean Closing Price for Starbucks'
df['Close'].resample('A').mean().plot.bar(title=title,color=['#1f77b4']);
```



[10]:
```
title = 'Monthly Max Closing Price for Starbucks'
df['Close'].resample('M').max().plot.bar(figsize=(16,6),␣
 ↪title=title,color='#1f77b4');
```



That is it! Up next we'll learn about time shifts!