04-Visualizing-Time-Series-Data

October 19, 2022

For more information, visit us at www.pieriandata.com

1 Visualizing Time Series Data

Let's go through a few key points of creating nice time series visualizations!

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

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

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

[3]:

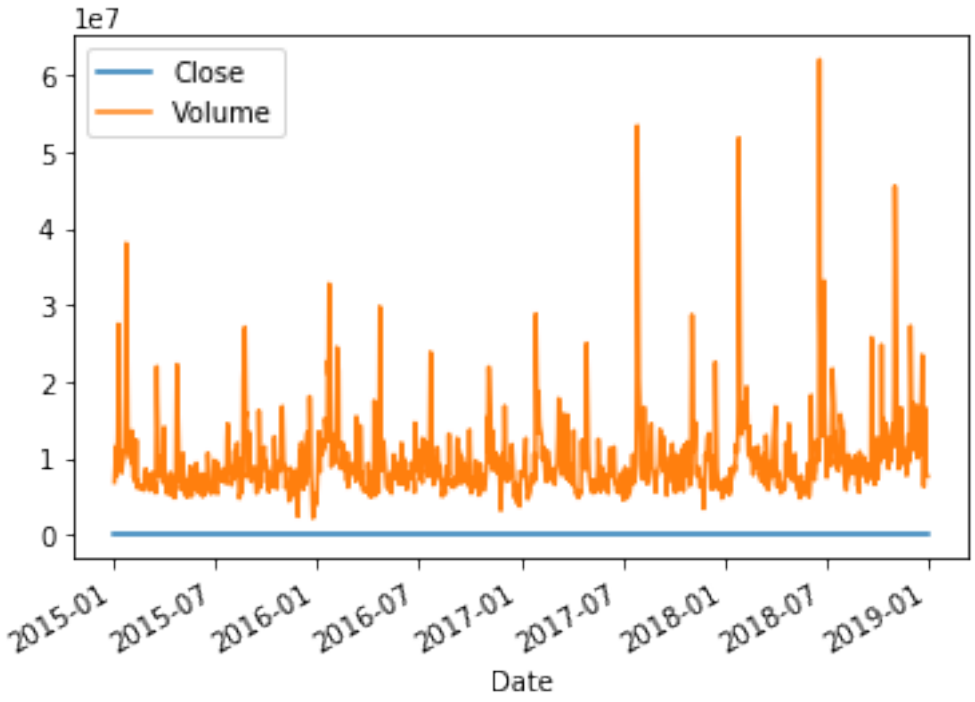
Date	Close	Volume
2015-01-02	38.0061	6906098
2015-01-05	37.2781	11623796
2015-01-06	36.9748	7664340
2015-01-07	37.8848	9732554
2015-01-08	38.4961	13170548

[4]:
```
# To show that dates are already parsed
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)
```

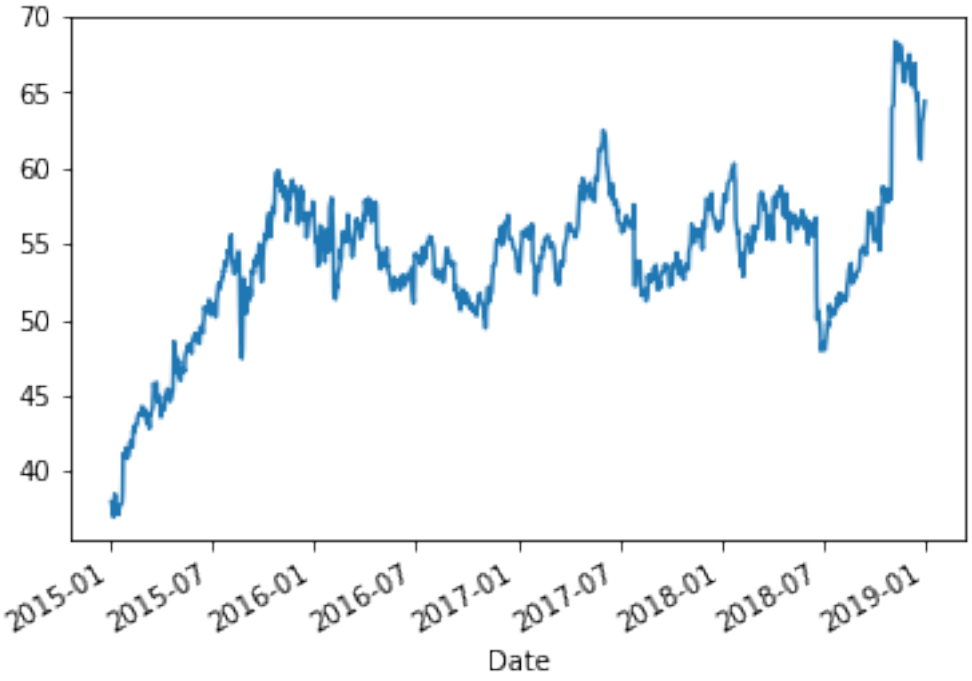
First we'll create a line plot that puts both 'Close' and 'Volume' on the same graph.Remember that we can use df.plot() in place of df.plot.line()

[5]:
```
df.plot();
```

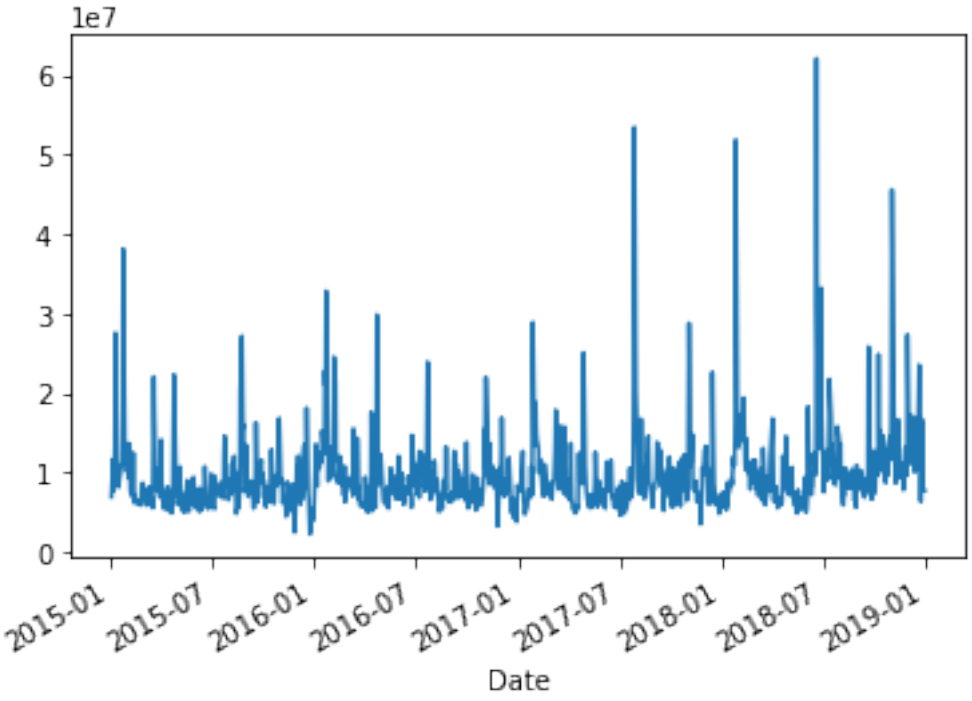


This isn't very helpful due to the difference in y-values, so we'll split them up.

[6]:
```
df['Close'].plot();
```



[7]:
```
df['Volume'].plot();
```

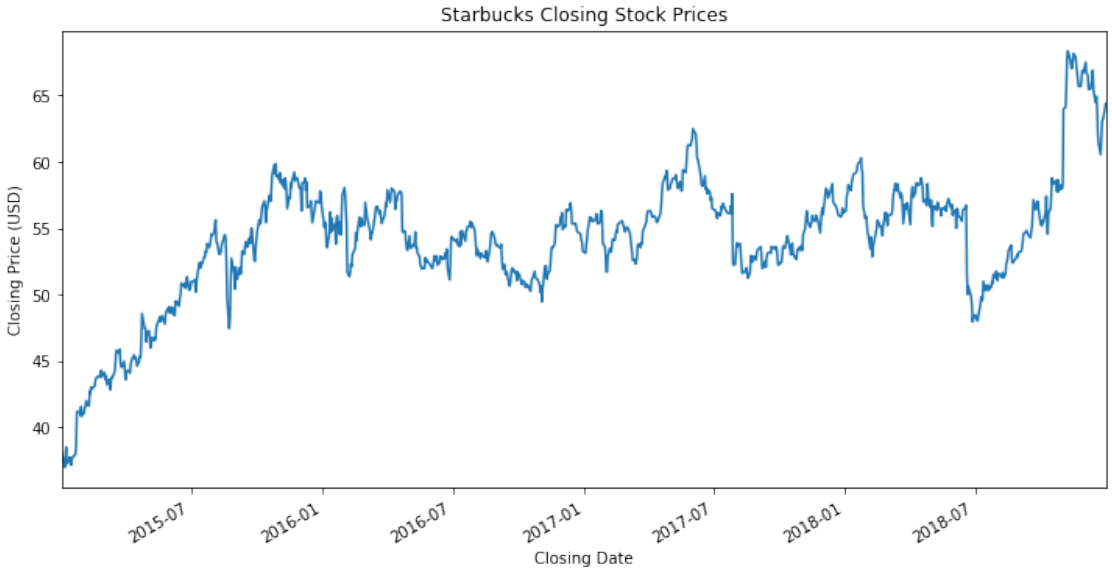


2 Plot Formatting

2.1 Adding a title and axis labels

NOTE: While we can pass a title into the pandas .plot() function, we can't pass x- and y-axis labels.However, since .plot() returns a matplotlib.axes.AxesSubplot object, we can set the labelson that object so long as we do it in the same jupyter cell. Setting an autoscale is done the same way.

[8]:
```
title='Starbucks Closing Stock Prices'
ylabel='Closing Price (USD)'
xlabel='Closing Date'

ax = df['Close'].plot(figsize=(12,6),title=title)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel);
```



2.2 X Limits

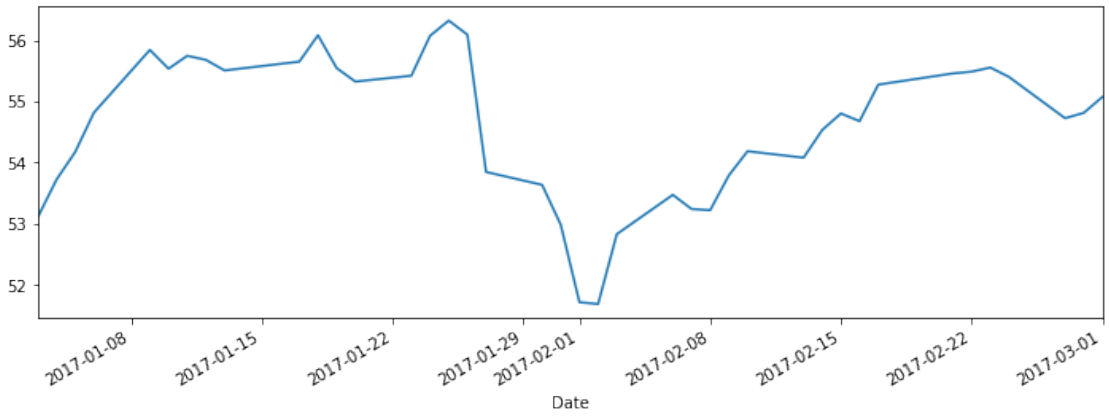
There are two ways we can set a specific span of time as an x-axis limit. We can plot a slice of the dataset, or we can pass x-limit values as an argument into df.plot().

The advantage of using a slice is that pandas automatically adjusts the y-limits accordingly.

The advantage of passing in arguments is that pandas automatically tightens the x-axis. Plus, if we're also setting y-limits this can improve readability.

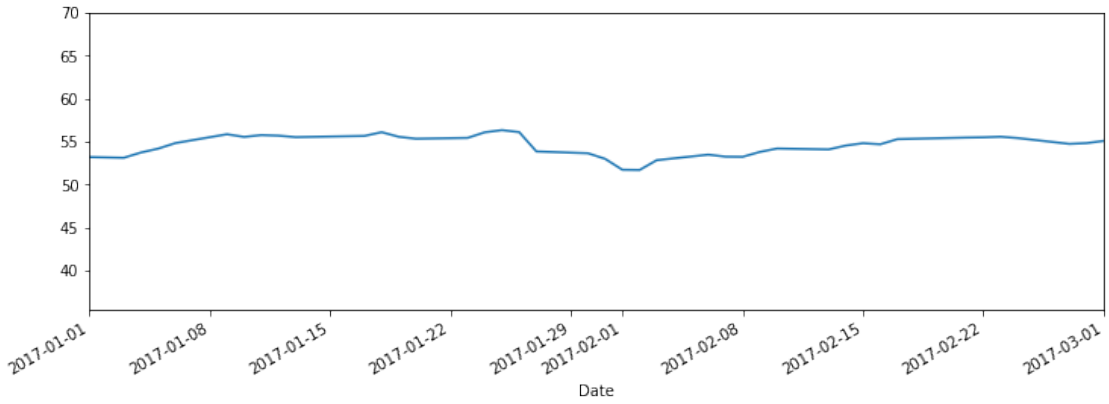
2.2.1 Choosing X Limits by Slice:

[9]:
```
# Dates are separated by a colon:
df['Close']['2017-01-01':'2017-03-01'].plot(figsize=(12,4)).
 ↪autoscale(axis='x',tight=True);
```



2.2.2 Choosing X Limits by Argument:

[10]:
```
# Dates are separated by a comma:
df['Close'].plot(figsize=(12,4),xlim=['2017-01-01','2017-03-01']);
```



NOTE: It's worth noting that the limit values do not have to appear in the index. Pandas will plot the actual dates based on their location in time.Also, another advantage of slicing over arguments is that it's easier to include the upper/lower bound as a limit.That is, df['column']['2017-01-01':].plot() is easier to type than df['column'].plot(xlim=('2017-01-01',df.index.max()))

Now let's focus on the y-axis limits to get a better sense of the shape of the data.First we'll find out what upper and lower limits to use.

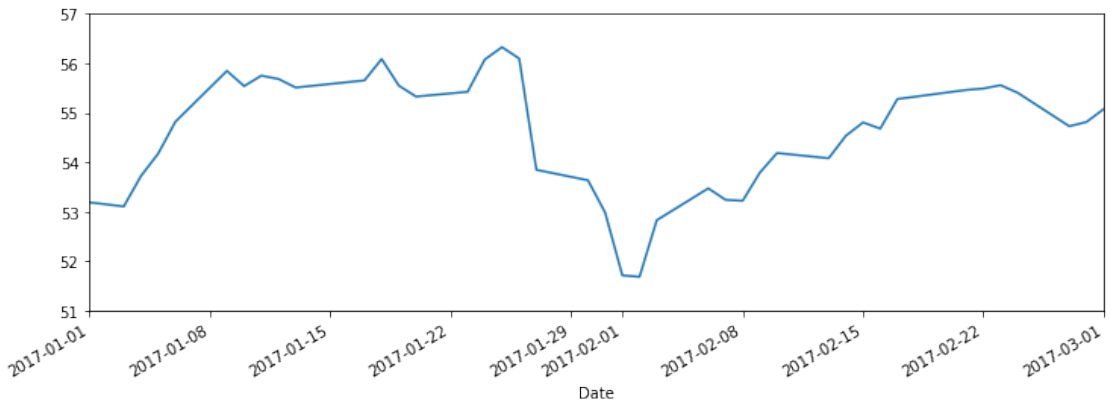
[11]:
```
# FIND THE MINIMUM VALUE IN THE RANGE:
df.loc['2017-01-01':'2017-03-01']['Close'].min()
```

[11]: 51.6899

[12]:
```
# FIND THE MAXIMUM VALUE IN THE RANGE:
df.loc['2017-01-01':'2017-03-01']['Close'].max()
```

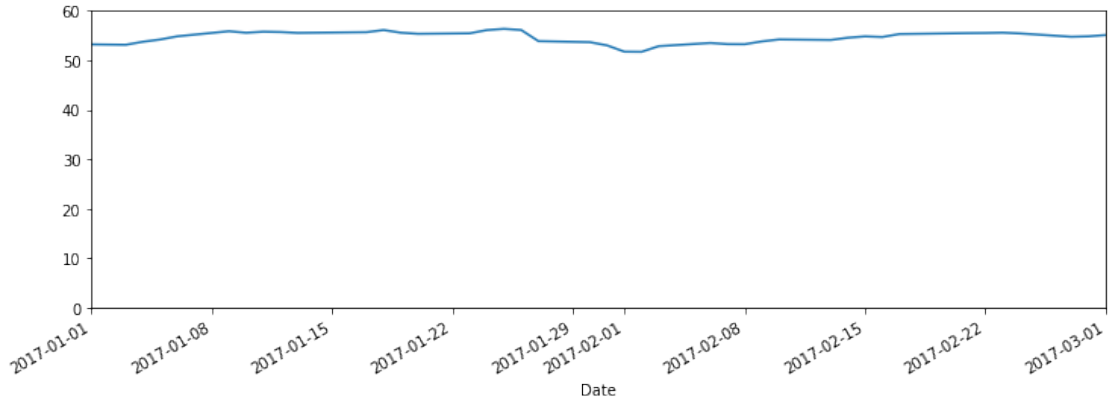
[12]: 56.3244

[13]:
```
# PLUG THESE IN AS Y-LIMIT VALUES:
df['Close'].plot(figsize=(12,4),xlim=['2017-01-01','2017-03-01'],ylim=[51,57]);
```



NOTE: Be careful when setting y-axis limits! Setting too narrow a slice can make graphs appear overly volatile. The above chart might lead you to believe that stocks were many times more valuable in January 2017 than in early February, but a look at them with the y-axis minimum set to zero tells a different story:

[14]:
```
df['Close'].plot(figsize=(12,4),xlim=['2017-01-01','2017-03-01'],ylim=[0,60]);
```

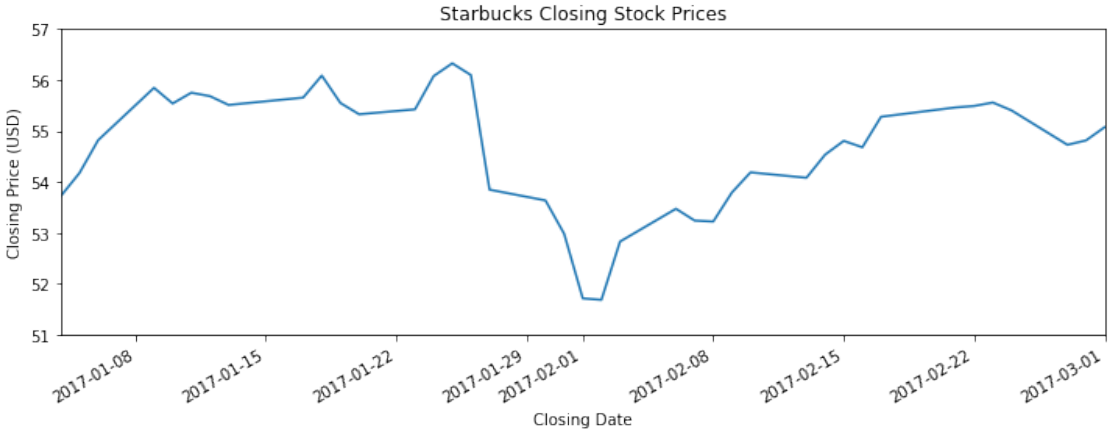


2.3 Title and axis labels

Let's add a title and axis labels to our subplot.

REMEMBER: ax.autoscale(axis='both',tight=True) is unnecessary if axis limits have been passed into .plot(). If we were to add it, autoscale would revert the axis limits to the full dataset.

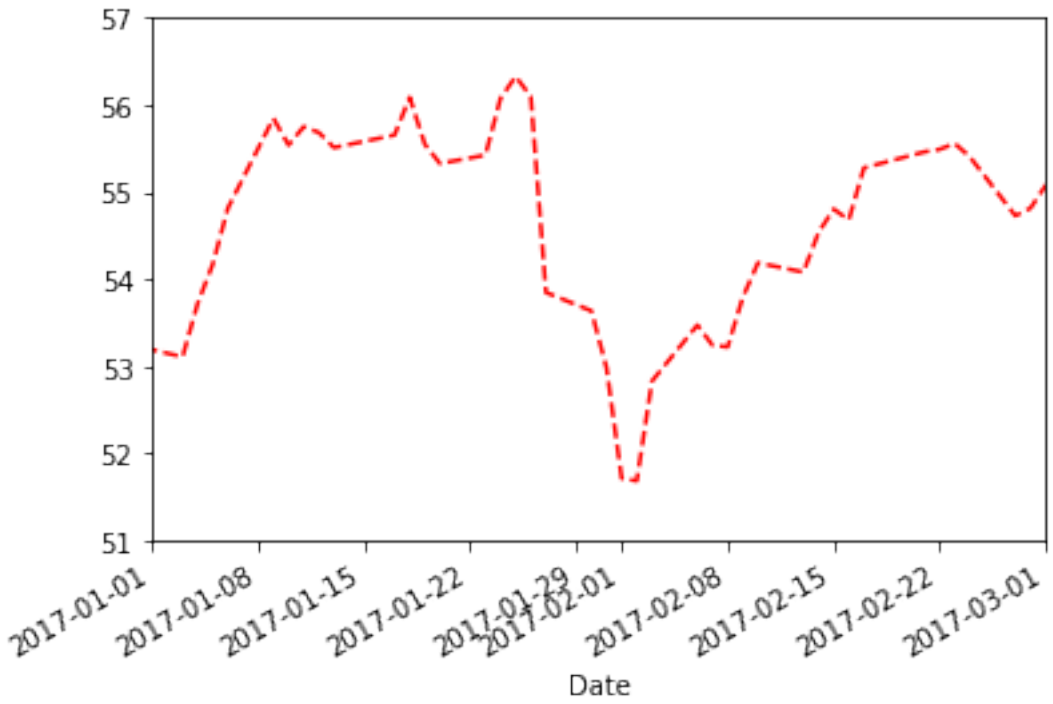
[15]:
```
title='Starbucks Closing Stock Prices'
ylabel='Closing Price (USD)'
xlabel='Closing Date'

ax = df['Close'].
 ↪plot(xlim=['2017-01-04','2017-03-01'],ylim=[51,57],figsize=(12,4),title=title)
ax.set(xlabel=xlabel, ylabel=ylabel);
```



2.4 Color and Style

We can pass arguments into .plot() to change the linestyle and color. Refer to the Customizing Plots lecture from the previous section for more options.

[16]:
```
df['Close'].plot(xlim=['2017-01-01','2017-03-01'],ylim=[51,57],ls='--',c='r');
```



2.5 X Ticks

In this section we'll look at how to change the format and appearance of dates along the x-axis. To do this, we'll borrow a tool from matplotlib called dates.

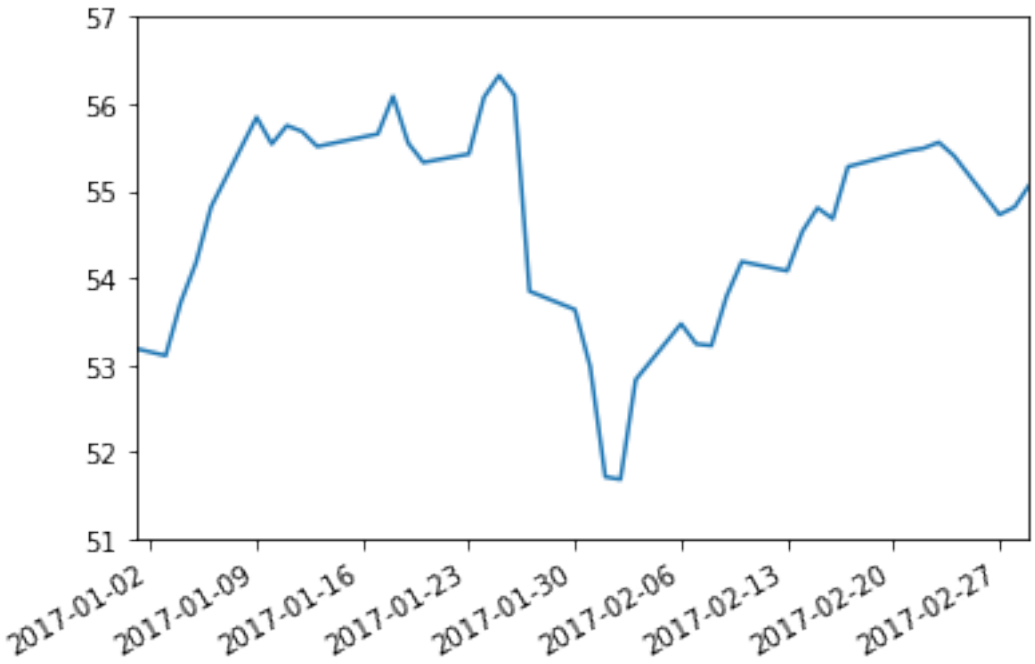
[17]:
```
from matplotlib import dates
```

2.5.1 Set the spacing

The x-axis values can be divided into major and minor axes. For now, we'll work only with the major axis and learn how to set the spacing with .set_major_locator().

[18]:
```
# CREATE OUR AXIS OBJECT
ax = df['Close'].plot(xlim=['2017-01-01','2017-03-01'],ylim=[51,57])

# REMOVE PANDAS DEFAULT "Date" LABEL
```

```
ax.set(xlabel='')

# SET THE TICK LOCATOR AND FORMATTER FOR THE MAJOR AXIS
ax.xaxis.set_major_locator(dates.WeekdayLocator(byweekday=0))
```



Notice that dates are spaced one week apart. The dates themselves correspond with byweekday=0, or Mondays. For a full list of locator options available from matplotlib.dates visit https://matplotlib.org/api/dates_api.html#date-tickers

2.5.2 Date Formatting

Formatting follows the Python datetime strftime codes. The following examples are based on datetime.datetime(2001, 2, 3, 16, 5, 6):

CODE

MEANING

EXAMPLE

%Y

Year with century as a decimal number.

2001

%y

Year without century as a zero-padded decimal number.

01

%m

Month as a zero-padded decimal number.

02

%B

Month as locale's full name.

February

%b

Month as locale's abbreviated name.

Feb

%d

Day of the month as a zero-padded decimal number.

03

%A

Weekday as locale's full name.

Saturday

%a

Weekday as locale's abbreviated name.

Sat

%H

Hour (24-hour clock) as a zero-padded decimal number.

16

%I

Hour (12-hour clock) as a zero-padded decimal number.

04

%p

Locale's equivalent of either AM or PM.

PM

%M

Minute as a zero-padded decimal number.

05

%S

Second as a zero-padded decimal number.

06

CODE

MEANING

EXAMPLE

%#m

Month as a decimal number. (Windows)

2

%-m

Month as a decimal number. (Mac/Linux)

2

%#x

Long date

Saturday, February 03, 2001

%#c

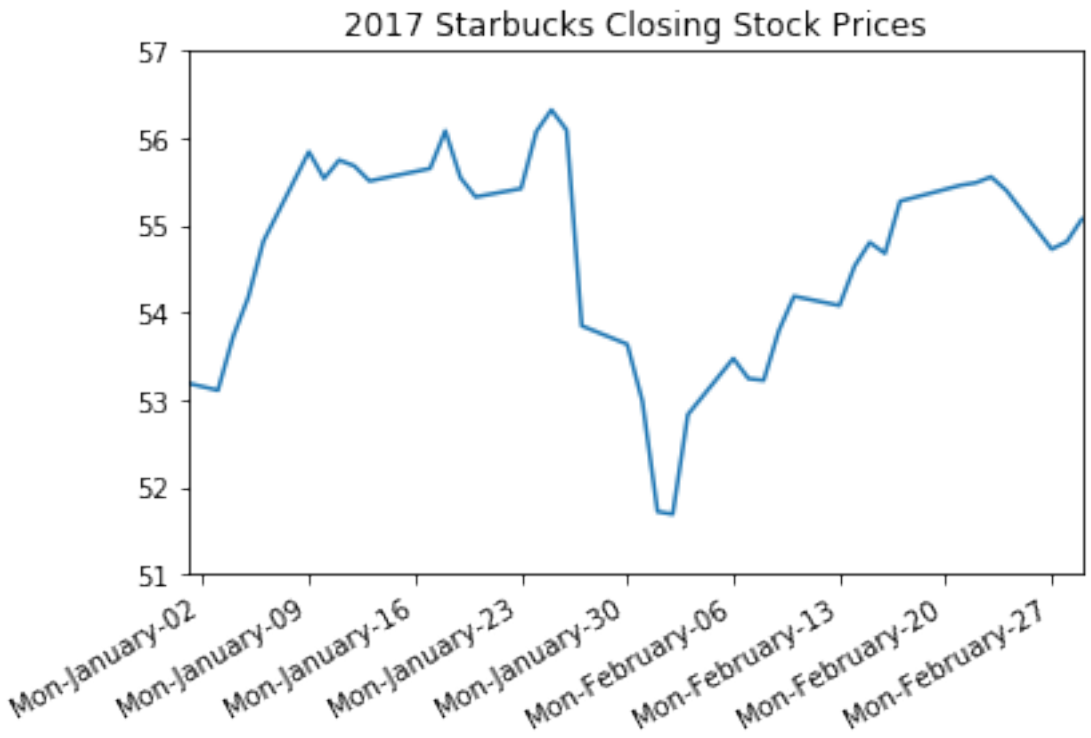
Long date and time

Saturday, February 03, 2001 16:05:06

[19]:
```
# USE THIS SPACE TO EXPERIMENT WITH DIFFERENT FORMATS
from datetime import datetime
datetime(2001, 2, 3, 16, 5, 6).strftime("%A, %B %d, %Y  %I:%M:%S %p")
```

[19]:
```
'Saturday, February 03, 2001  04:05:06 PM'
```

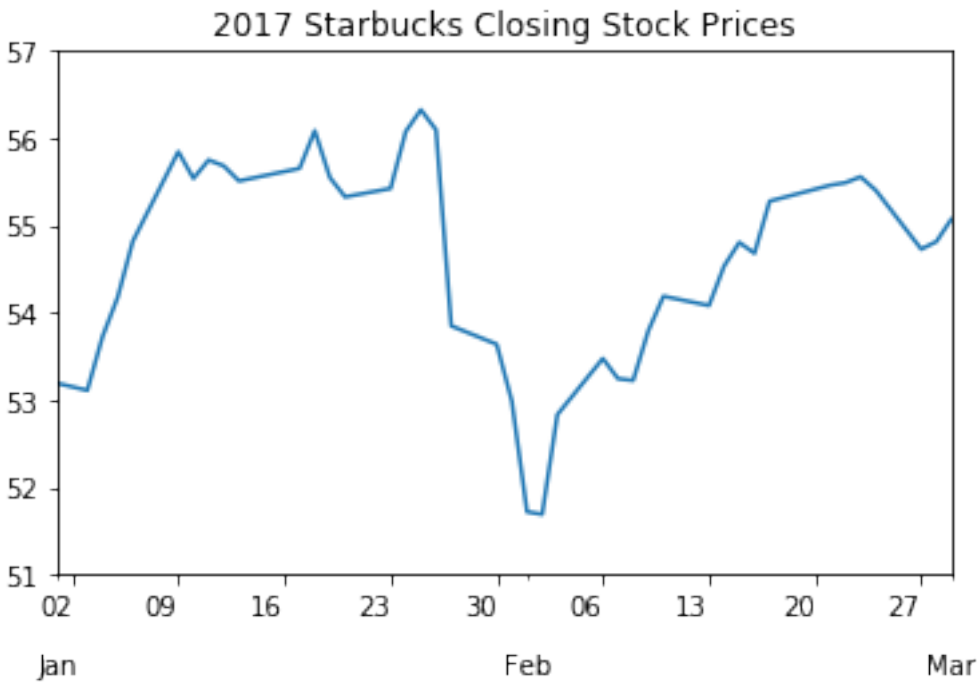
[20]:
```
ax = df['Close'].plot(xlim=['2017-01-01','2017-03-01'],ylim=[51,57],title='2017␣
 ↪Starbucks Closing Stock Prices')
ax.set(xlabel='')

ax.xaxis.set_major_locator(dates.WeekdayLocator(byweekday=0))
ax.xaxis.set_major_formatter(dates.DateFormatter("%a-%B-%d"))
```



2.6 Major vs. Minor Axis Values

All of the tick marks we've used so far have belonged to the major axis. We can assign another level called the minor axis, perhaps to separate month names from days of the month.

[21]:
```
ax = df['Close'].
 ↪plot(xlim=['2017-01-01','2017-03-01'],ylim=[51,57],rot=0,title='2017␣
 ↪Starbucks Closing Stock Prices')
ax.set(xlabel='')

ax.xaxis.set_major_locator(dates.WeekdayLocator(byweekday=0))
ax.xaxis.set_major_formatter(dates.DateFormatter('%d'))

ax.xaxis.set_minor_locator(dates.MonthLocator())
ax.xaxis.set_minor_formatter(dates.DateFormatter('\n\n%b'))
```

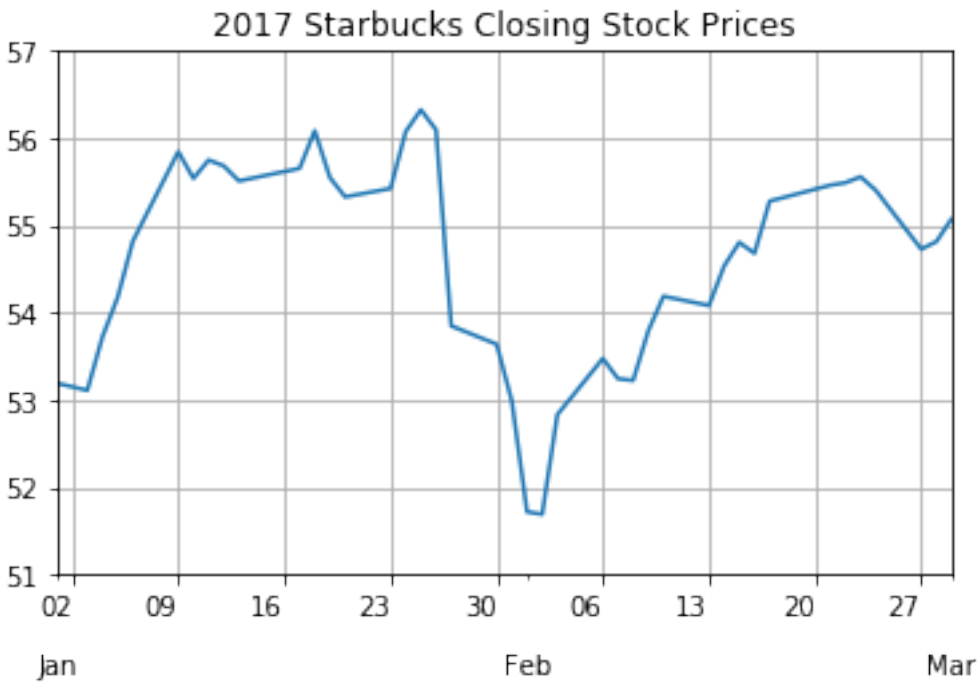


NOTE: we passed a rotation argument rot=0 into df.plot() so that the major axis values appear horizontal, not slanted.

2.7 Adding Gridlines

We can add x and y axis gridlines that extend into the plot from each major tick mark.

[22]:
```
ax = df['Close'].
 ↪plot(xlim=['2017-01-01','2017-03-01'],ylim=[51,57],rot=0,title='2017␣
 ↪Starbucks Closing Stock Prices')
ax.set(xlabel='')

ax.xaxis.set_major_locator(dates.WeekdayLocator(byweekday=0))
ax.xaxis.set_major_formatter(dates.DateFormatter('%d'))

ax.xaxis.set_minor_locator(dates.MonthLocator())
ax.xaxis.set_minor_formatter(dates.DateFormatter('\n\n%b'))

ax.yaxis.grid(True)
ax.xaxis.grid(True)
```


2017 Starbucks Closing Stock Prices
57
56
55
54
53
52
51
02
09
16
23
30
06
13
20
27
Jan
Feb
Mar

2.8 Great job!