02-Visualizing-Pandas-Time-Series-Data

October 5, 2024

[ ]:

1 Visualizing Time Series Data

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

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

[2]:
```
df = pd.read_csv('COST.csv')
```

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

[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2016-09-06	158.130005	158.149994	156.020004	158.059998	140.896622	2716900
1	2016-09-07	157.639999	157.869995	155.399994	155.639999	138.739395	2984100
2	2016-09-08	155.190002	155.490005	152.940002	153.470001	136.805038	2993100
3	2016-09-09	152.589996	152.789993	150.699997	150.699997	134.335831	2993900
4	2016-09-12	150.500000	151.990005	150.259995	151.690002	135.218338	2982400

[4]:
```
df = pd.read_csv('COST.csv',index_col='Date',parse_dates=True)
```

[5]:
```
df.head(10)
```

[5]:

Date	Open	High	Low	Close	Adj Close	Volume
2016-09-06	158.130005	158.149994	156.020004	158.059998	140.896622	2716900
2016-09-07	157.639999	157.869995	155.399994	155.639999	138.739395	2984100
2016-09-08	155.190002	155.490005	152.940002	153.470001	136.805038	2993100

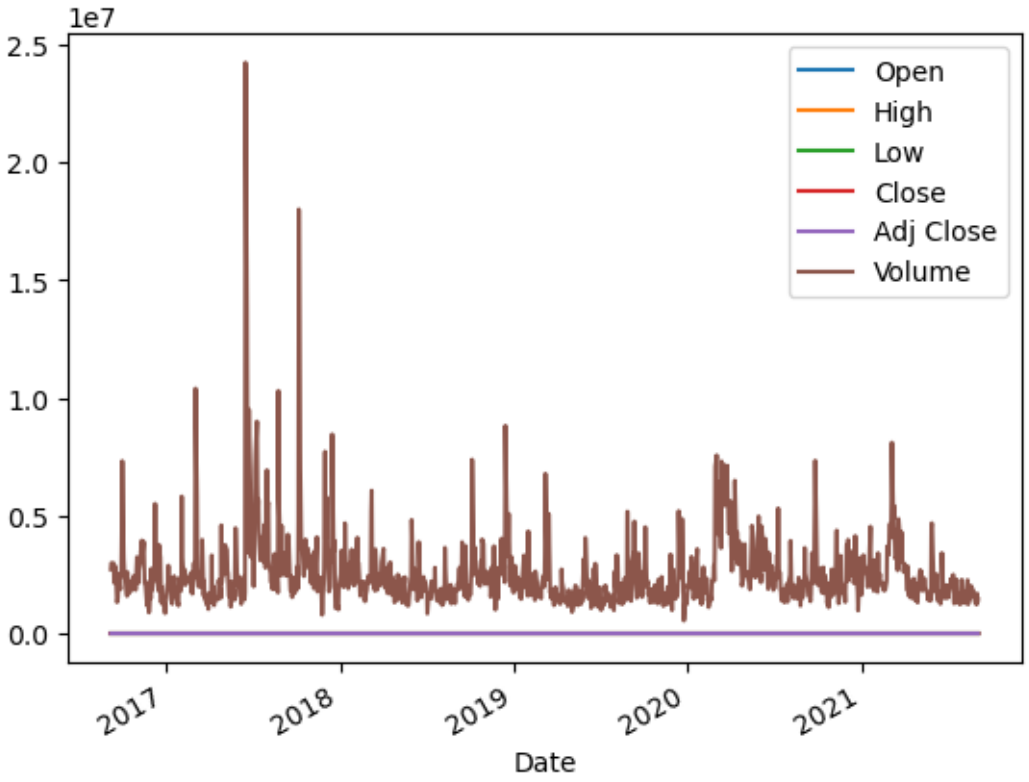
```
2016-09-09  152.589996  152.789993  150.699997  150.699997  134.335831
2016-09-12  150.500000  151.990005  150.259995  151.690002  135.218338
2016-09-13  151.179993  151.580002  150.070007  150.740005  134.371490
2016-09-14  150.710007  152.250000  150.199997  150.690002  134.326889
2016-09-15  150.720001  152.970001  150.110001  152.669998  136.091919
2016-09-16  151.960007  152.660004  151.139999  152.350006  135.806625
2016-09-19  152.130005  153.029999  151.600006  151.789993  135.307449

              Volume
Date
2016-09-06  2716900
2016-09-07  2984100
2016-09-08  2993100
2016-09-09  2993900
2016-09-12  2982400
2016-09-13  2148600
2016-09-14  2188900
2016-09-15  2284300
2016-09-16  2842000
2016-09-19  1321300
```

[6]:
```
# To show that dates are already parsed
df.index
```

[6]:
```
DatetimeIndex(['2016-09-06', '2016-09-07', '2016-09-08', '2016-09-09',
               '2016-09-12', '2016-09-13', '2016-09-14', '2016-09-15',
               '2016-09-16', '2016-09-19',
               …
               '2021-08-20', '2021-08-23', '2021-08-24', '2021-08-25',
               '2021-08-26', '2021-08-27', '2021-08-30', '2021-08-31',
               '2021-09-01', '2021-09-02'],
              dtype='datetime64[ns]', name='Date', length=1258, freq=None)
```

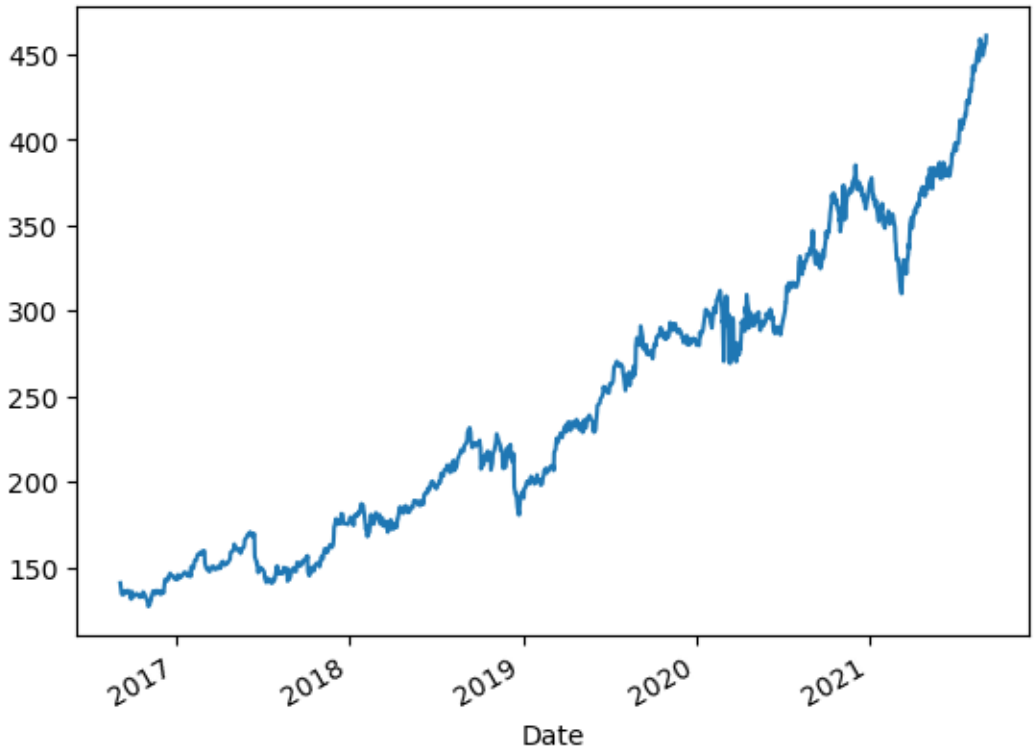
First we'll create a line plot that puts both 'Close' and 'Volume' on the same graph.Remember that we can use df.plot() in place of df.plot.line()

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

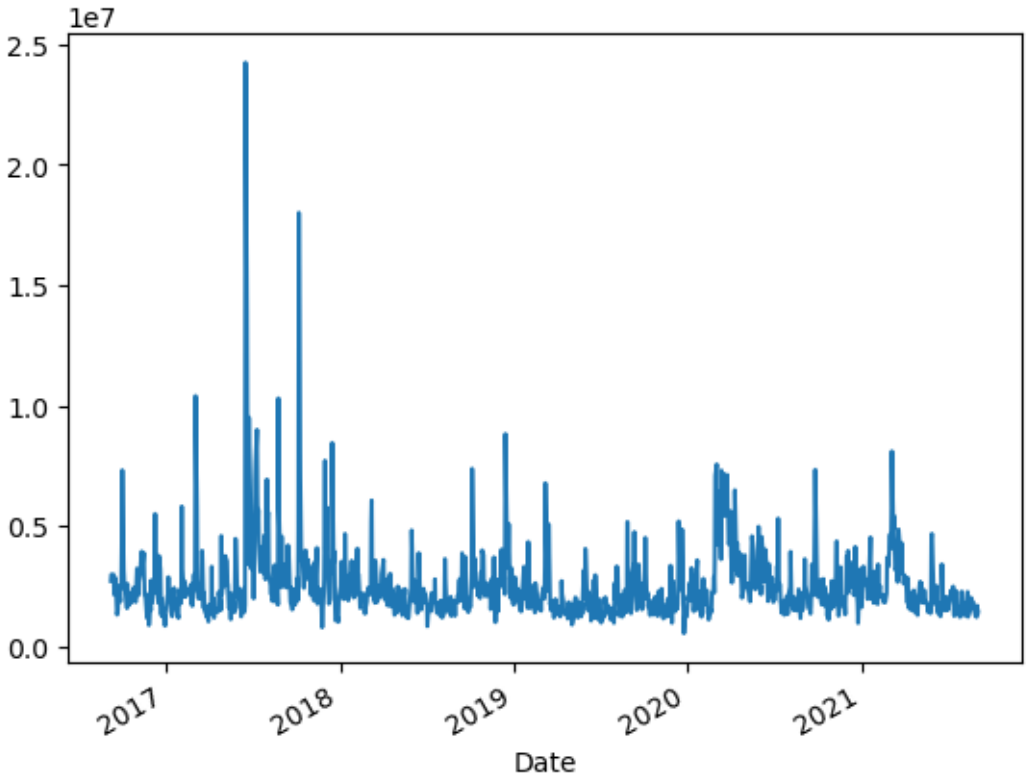


This isn't very helpful due to the difference in y-values, so we'll split them up.

[8]:
```
df['Adj Close'].plot();
```



[9]:
```
df['Volume'].plot();
```



2 Time Series Plot Formatting

2.1 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().

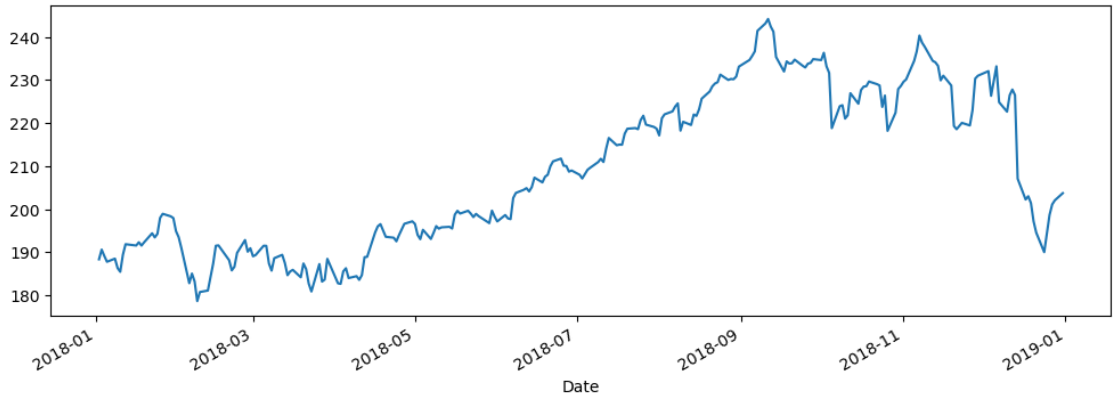
The advantage of using a slice is that pandas automatically adjusts the y-limits accordingly.

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.

2.1.1 Choosing X Limits by Slice:

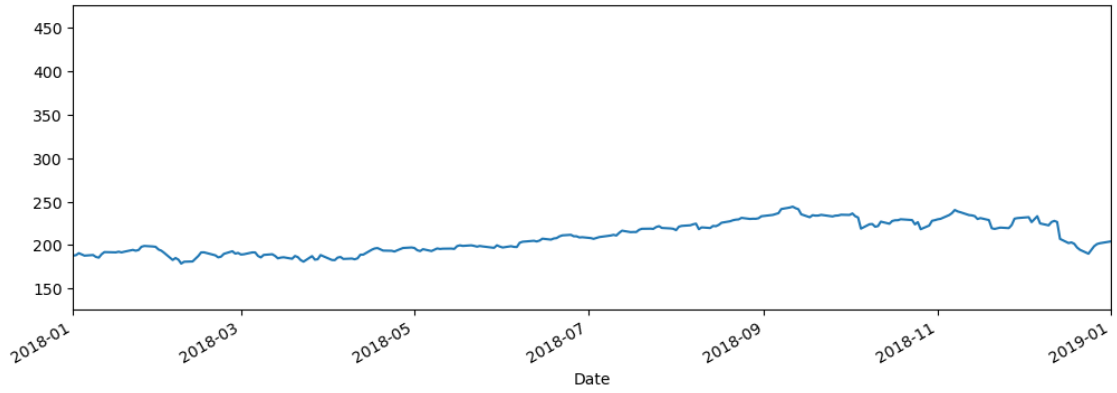
[10]:
```
# Dates are separated by a colon:
df['Close']['2018-01-01':'2019-01-01'].plot(figsize=(12,4))
```

[10]:
```
<Axes: xlabel='Date'>
```



2.1.2 Choosing X Limits by Argument:

[11]:
```
# Why is the y-axis so different?
# Because pandas first plotted the whole thing, then it narrowed
# down the plot, unlike above, where we narrowed the df first!
df['Close'].plot(figsize=(12,4),xlim=['2018-01-01','2019-01-01']);
```



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.

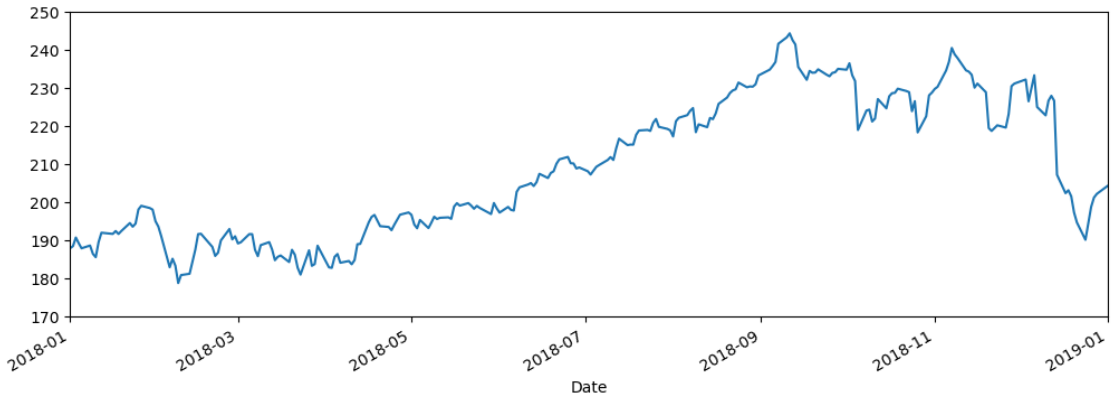
[12]:
```
# FIND THE MINIMUM VALUE IN THE RANGE:
df.loc['2018-01-01':'2019-01-01']['Close'].min()
```

[12]: 178.610001

[13]:
```
# FIND THE MAXIMUM VALUE IN THE RANGE:
df.loc['2018-01-01':'2019-01-01']['Close'].max()
```

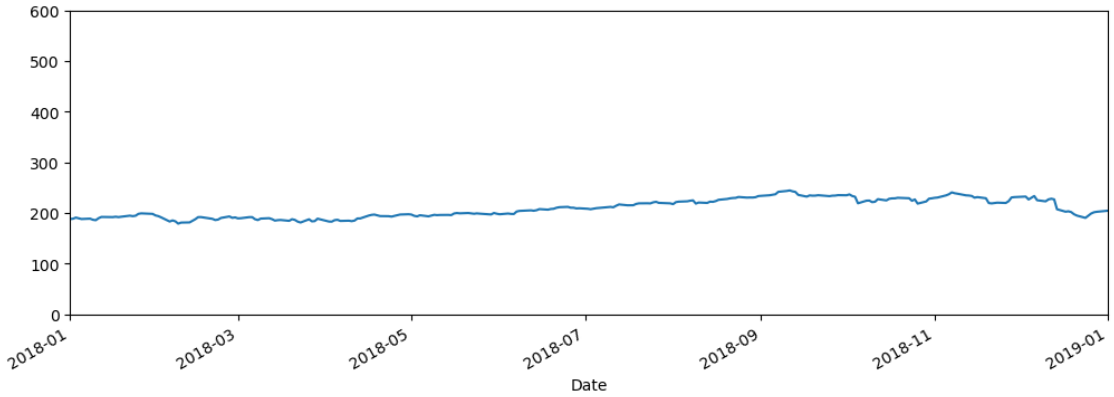
[13]:
```
244.210007
```

[14]:
```
# PLUG THESE IN AS Y-LIMIT VALUES:
df['Close'].
 ↪plot(figsize=(12,4),xlim=['2018-01-01','2019-01-01'],ylim=[170,250]);
```



NOTE: Be careful when setting y-axis limits! Setting too narrow a slice can make graphs appear overly volatile.

[15]:
```
df['Close'].plot(figsize=(12,4),xlim=['2018-01-01','2019-01-01'],ylim=[0,600]);
```



2.2 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.

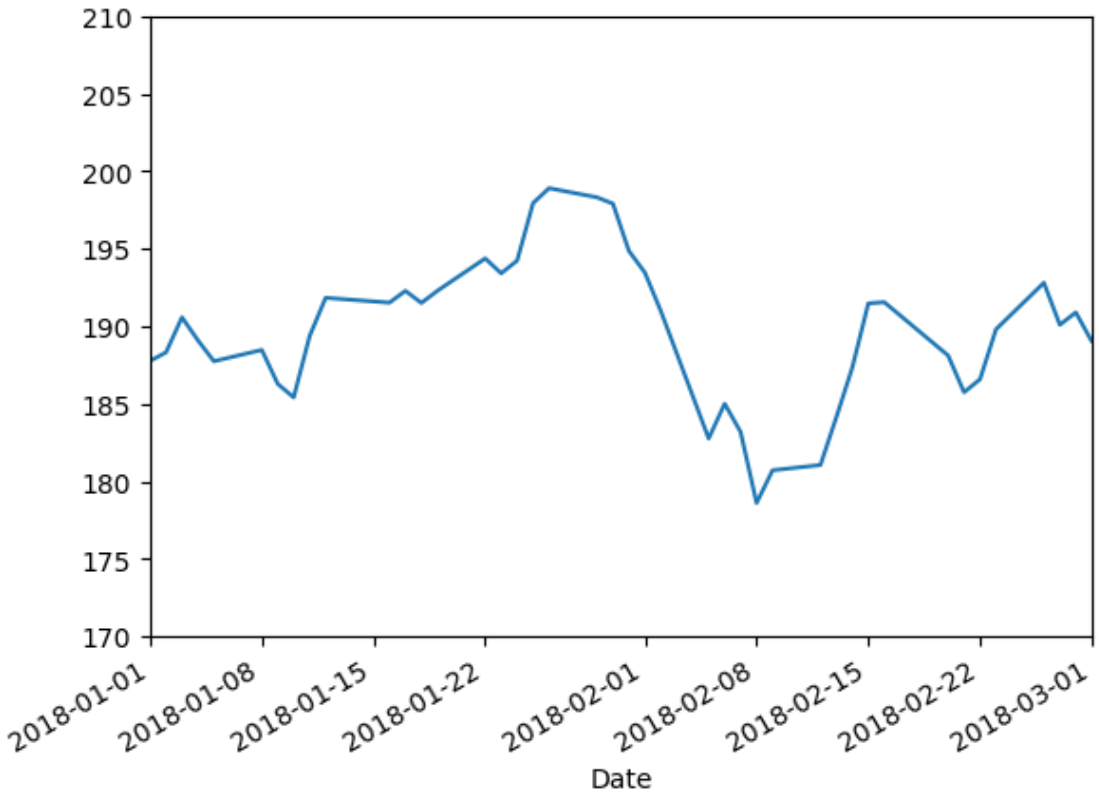
[16]:
```
from matplotlib import dates
```

2.2.1 Set the spacing

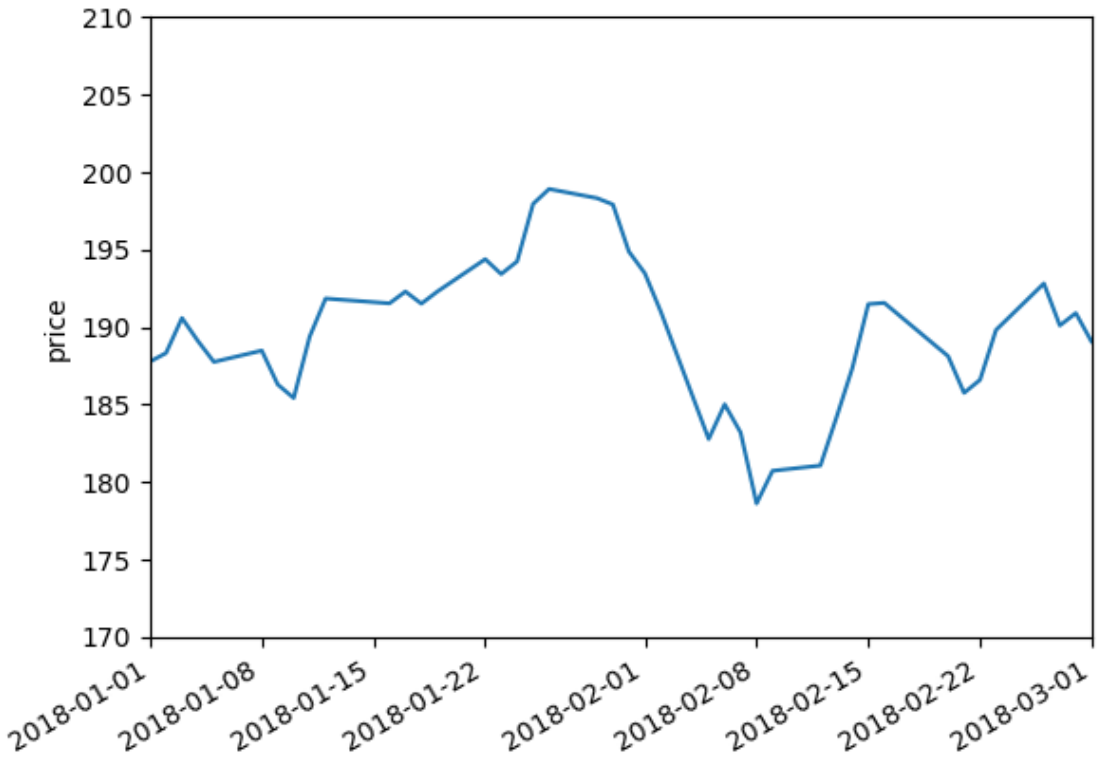
The x-axis values can be divided into major and minor axes. For now, we'll work only with the major axis and learn how to set the spacing with .set_major_locator().

[17]:
```
#Notice the strange "jump" in the middle due to the month change!
df['Close'].plot(xlim=['2018-01-01','2018-03-01'],ylim=(170,210))
```

[17]:
```
<Axes: xlabel='Date'>
```



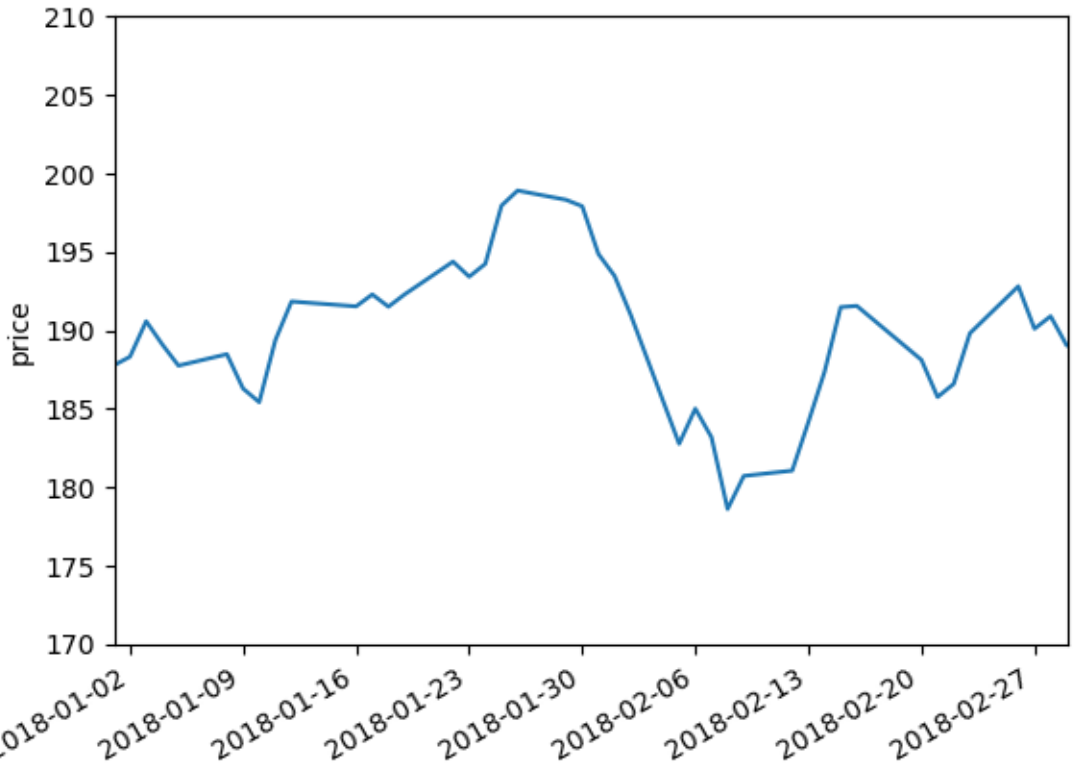
[18]:
```
# you can remove "date" index name label
ax = df['Close'].plot(xlim=['2018-01-01','2018-03-01'],ylim=(170,210),
                      xlabel='',ylabel='price')
```



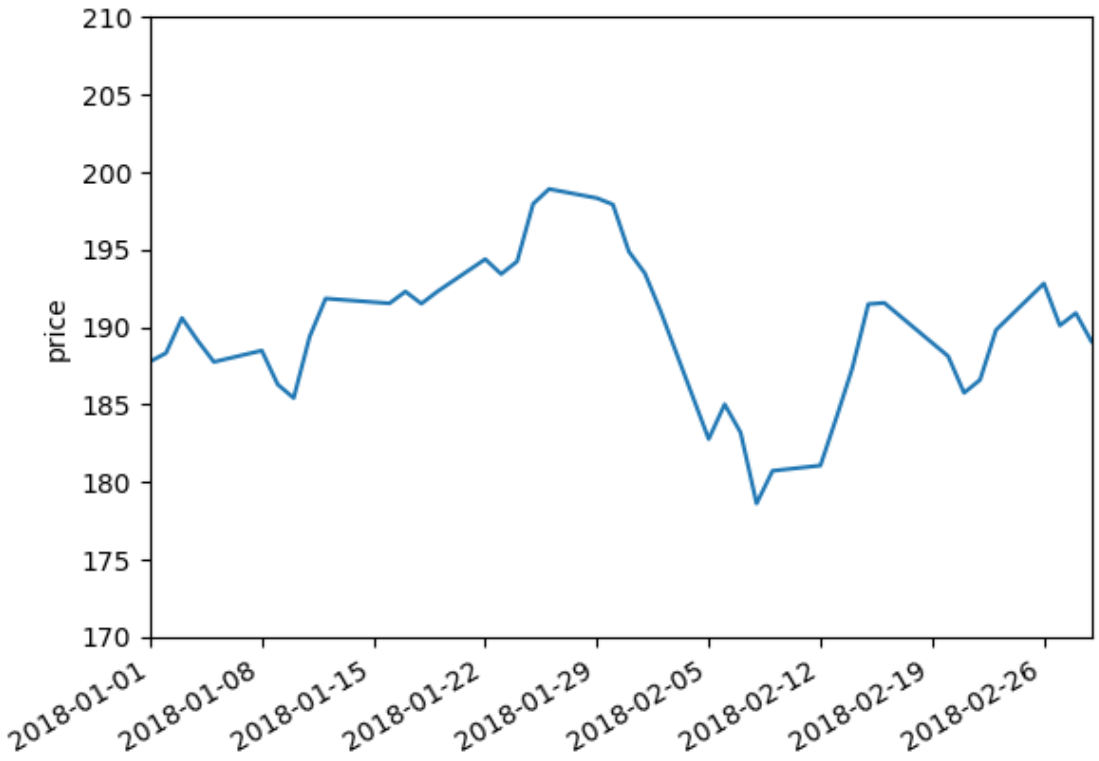
2.2.2 Using Tick Locator

[19]:
```
from matplotlib import dates
```

[20]:
```
ax = df['Close'].plot(xlim=['2018-01-01','2018-03-01'],ylim=(170,210),
                      xlabel='',ylabel='price')

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

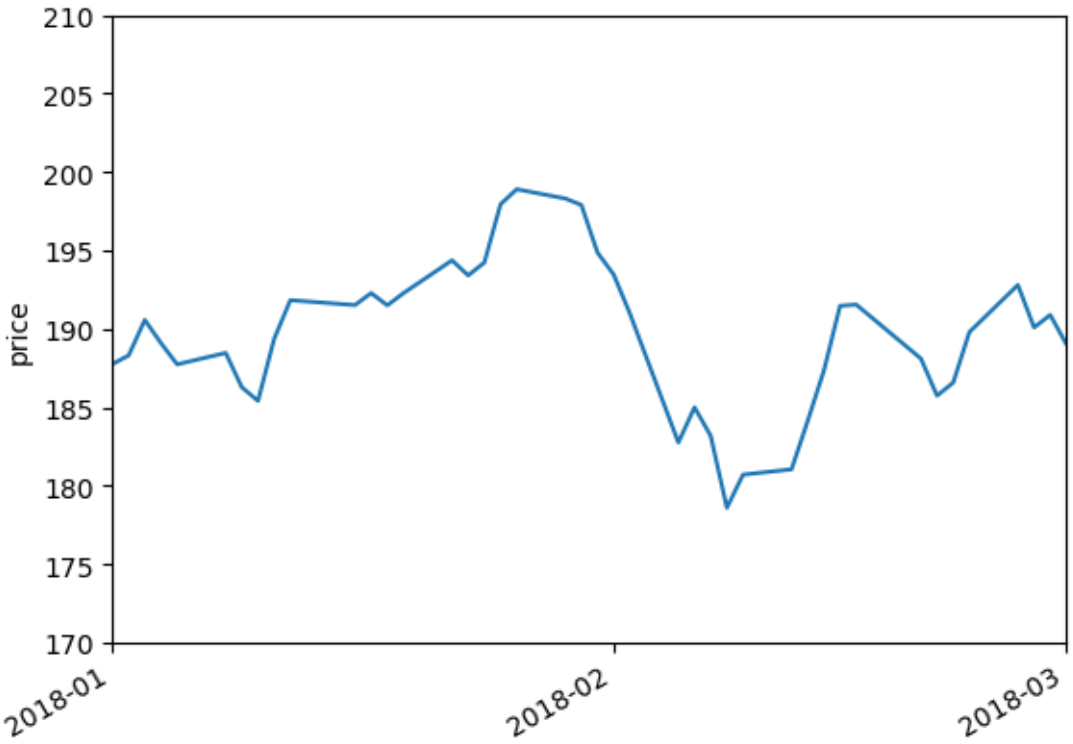


[21]:
```
ax = df['Close'].plot(xlim=['2018-01-01','2018-03-01'],ylim=(170,210),
                      xlabel='',ylabel='price')

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

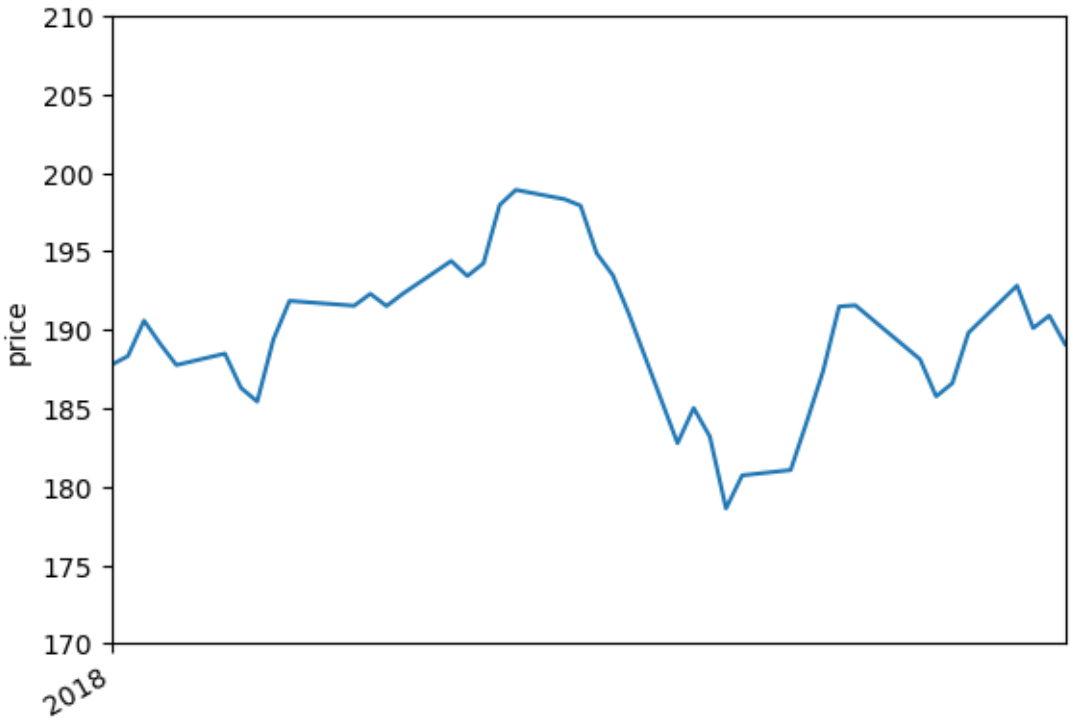


Notice that dates are spaced one week apart. The dates themselves correspond with byweekday=0, or Mondays. For a full list of locator options available from matplotlib.dates visit https://matplotlib.org/api/dates_api.html#date-tickers

[22]:
```
ax = df['Close'].plot(xlim=['2018-01-01','2018-03-01'],ylim=(170,210),
                      xlabel='',ylabel='price')

# SET THE TICK LOCATOR AND FORMATTER FOR THE MAJOR AXIS
ax.xaxis.set_major_locator(dates.MonthLocator())
```



[23]:
```
ax = df['Close'].plot(xlim=['2018-01-01','2018-03-01'],ylim=(170,210),
                      xlabel='',ylabel='price')

# SET THE TICK LOCATOR AND FORMATTER FOR THE MAJOR AXIS
ax.xaxis.set_major_locator(dates.YearLocator()) # DayLocator()
```



2.3 Formator

2.3.1 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

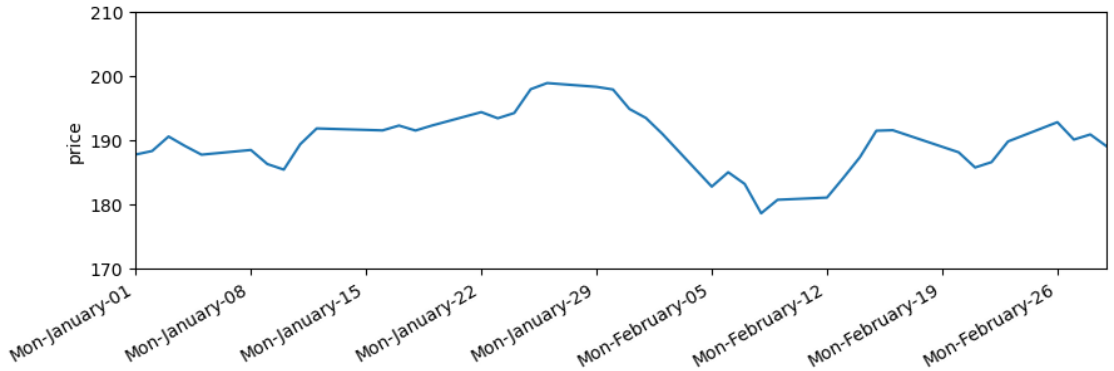
Saturday, February 03, 2001 16:05:06

[24]:
```
# 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")
```

[24]:
```
'Saturday, February 03, 2001  04:05:06 PM'
```

2.3.2 Combine Locator with Formatter

[25]:
```
plt.figure(dpi=100,figsize=(10,3))
ax = df['Close'].plot(xlim=['2018-01-01','2018-03-01'],ylim=(170,210),
                      xlabel='',ylabel='price')

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

# ADD IN THE FORMATTER
ax.xaxis.set_major_formatter(dates.DateFormatter("%a-%B-%d"))
```

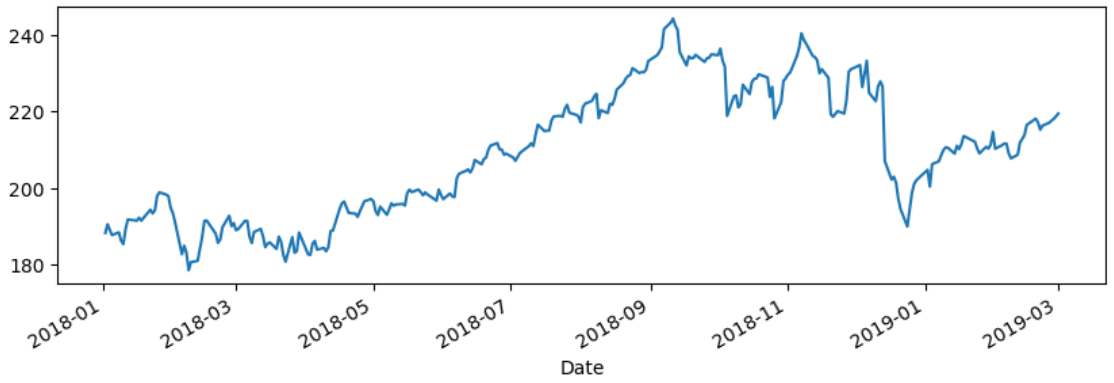


2.4 Major vs. Minor Axis Values

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.

DEFAULT PLOT

[26]:
```
plt.figure(dpi=100,figsize=(10,3))
ax = df['Close']['2018-01-01':'2019-03-01'].plot()
```

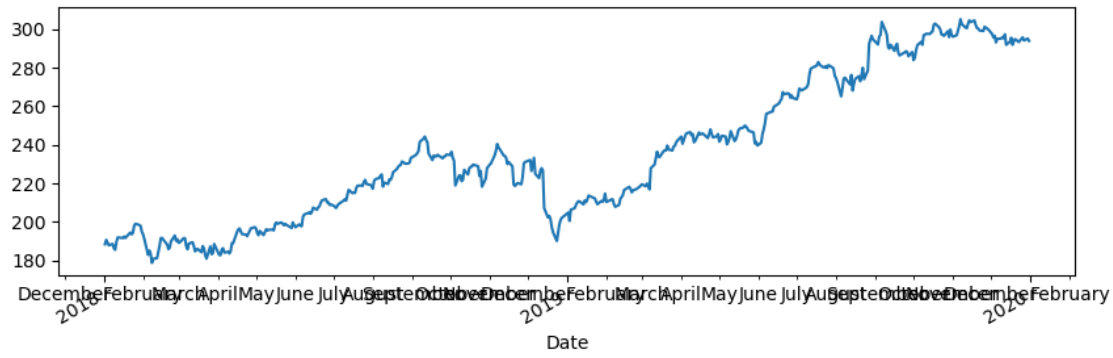


MAJOR TICK PLOT EDITS

[27]:
```
plt.figure(dpi=100,figsize=(10,3))
ax = df['Close']['2018-01-01':'2020-01-01'].plot()

# SET THE TICK LOCATOR AND FORMATTER FOR THE MAJOR AXIS
ax.xaxis.set_major_locator(dates.YearLocator())
ax.xaxis.set_major_formatter(dates.DateFormatter("%Y"))
```

MAJOR AND MINOR TICK PLOT EDITS

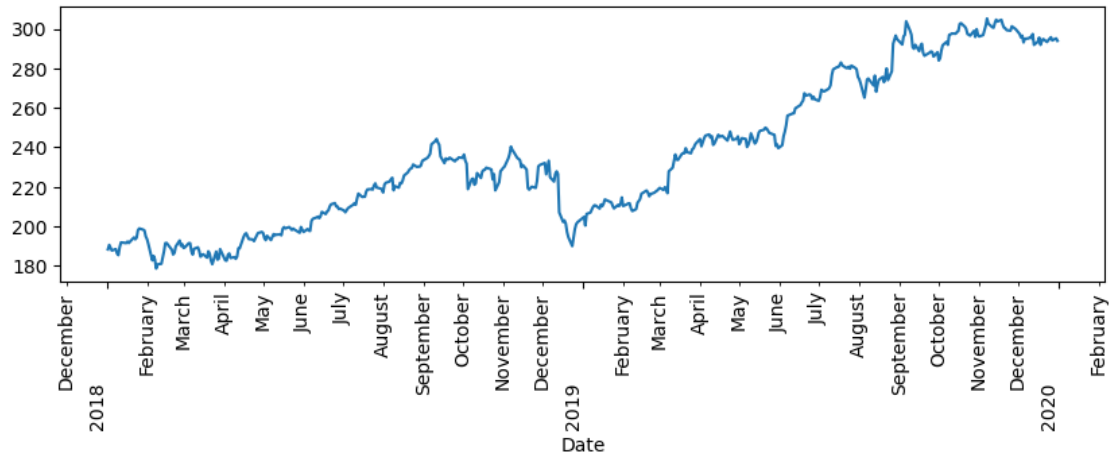
[28]:
```
plt.figure(dpi=100,figsize=(10,3))
ax = df['Close']['2018-01-01':'2020-01-01'].plot()

# SET THE TICK LOCATOR AND FORMATTER FOR THE MAJOR AXIS
ax.xaxis.set_major_locator(dates.YearLocator())
ax.xaxis.set_major_formatter(dates.DateFormatter("%Y"))

# SET THE TICK LOCATOR AND FORMATTER FOR THE MINOR AXIS
ax.xaxis.set_minor_locator(dates.MonthLocator())
ax.xaxis.set_minor_formatter(dates.DateFormatter("%B"))
```



Rotate Minor Ticks

[29]:
```
plt.figure(dpi=100,figsize=(10,3))
ax = df['Close']['2018-01-01':'2020-01-01'].plot()

# SET THE TICK LOCATOR AND FORMATTER FOR THE MAJOR AXIS
ax.xaxis.set_major_locator(dates.YearLocator())
ax.xaxis.set_major_formatter(dates.DateFormatter("%Y"))
```

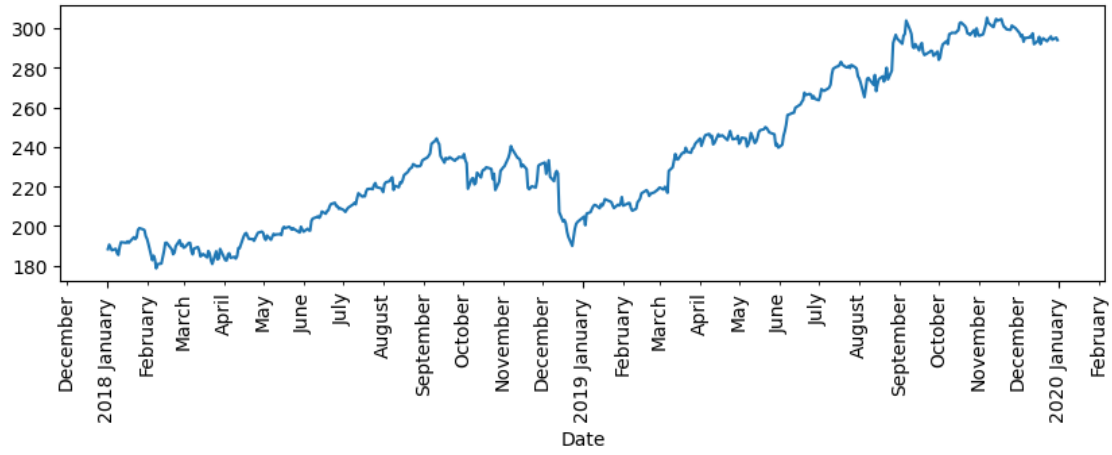
```
# SET THE TICK LOCATOR AND FORMATTER FOR THE MINOR AXIS
ax.xaxis.set_minor_locator(dates.MonthLocator())
ax.xaxis.set_minor_formatter(dates.DateFormatter("%B"))


# FURTHER EDITING TICK PROPERTIES

ax.tick_params(axis="x", which="major", rotation=90,pad=50)
ax.tick_params(axis="x", which="minor", rotation=90)
```



Include Janurary instead of just Year on Major Tick

[30]:
```
plt.figure(dpi=100,figsize=(10,3))
ax = df['Close']['2018-01-01':'2020-01-01'].plot()

# SET THE TICK LOCATOR AND FORMATTER FOR THE MAJOR AXIS
ax.xaxis.set_major_locator(dates.YearLocator())
ax.xaxis.set_major_formatter(dates.DateFormatter("%Y %B"))

# SET THE TICK LOCATOR AND FORMATTER FOR THE MINOR AXIS
ax.xaxis.set_minor_locator(dates.MonthLocator())
ax.xaxis.set_minor_formatter(dates.DateFormatter("%B"))


# FURTHER EDITING TICK PROPERTIES

ax.tick_params(axis="x", which="major",rotation=90, pad=5)
ax.tick_params(axis="x", which="minor",rotation=90)

# To get it exactly perfect is a lot more work:
```

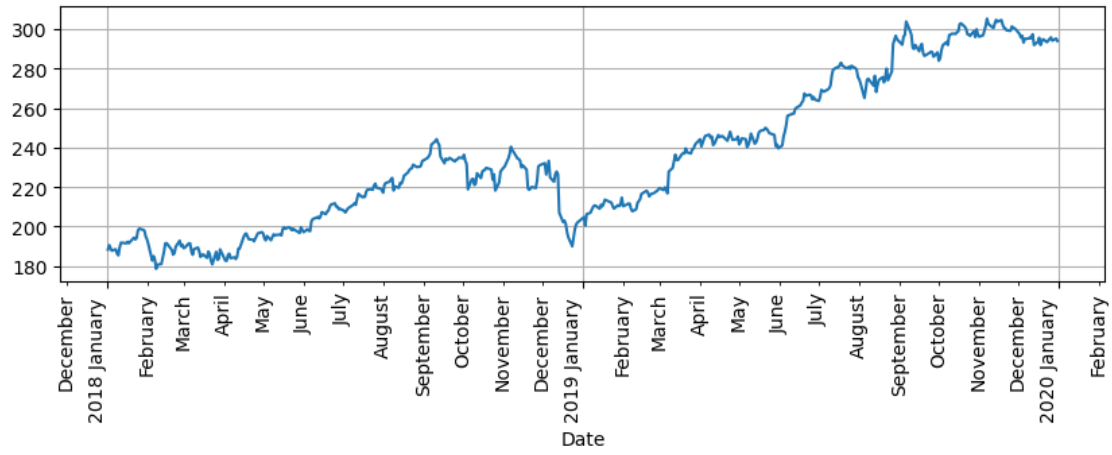
```
# https://stackoverflow.com/questions/28615887/
  ↪how-to-move-a-ticks-label-in-matplotlib
plt.xticks(ha='center');
```



2.5 Adding Gridlines

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

[31]:
```
plt.figure(dpi=100,figsize=(10,3))
ax = df['Close']['2018-01-01':'2020-01-01'].plot()

# SET THE TICK LOCATOR AND FORMATTER FOR THE MAJOR AXIS
ax.xaxis.set_major_locator(dates.YearLocator())
ax.xaxis.set_major_formatter(dates.DateFormatter("%Y %B"))

# SET THE TICK LOCATOR AND FORMATTER FOR THE MINOR AXIS
ax.xaxis.set_minor_locator(dates.MonthLocator())
ax.xaxis.set_minor_formatter(dates.DateFormatter("%B"))


# FURTHER EDITING TICK PROPERTIES

ax.tick_params(axis="x", which="major",rotation=90, pad=5)
ax.tick_params(axis="x", which="minor",rotation=90)

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


300
280
260
240
220
200
180
December
2018 January
February
March
April
May
June
July
August
September
October
November
December
2019 January
February
March
April
May
June
July
August
September
October
November
December
2020 January
February
Date

[ ]: