Formatting Plots

Name : Jann Moises Nyll B. De los Reyes

Section : CPE22S3

Submitted to: Engr. Roman Richard

Date : March 28, 2024

About data

In this notebook, we will be working with Facebook's stock price throughout 2018 (obtained using the `stock_analysis` package)

Setup

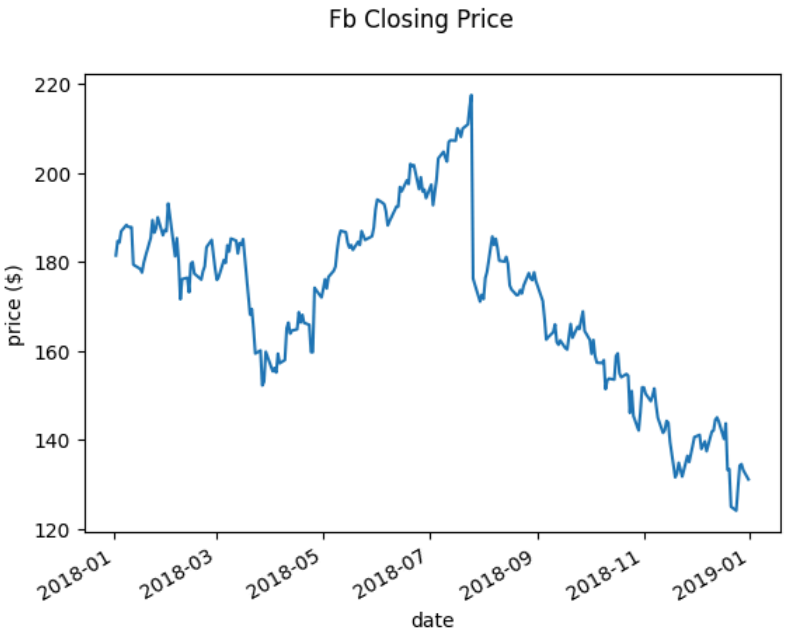
In [4]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sn

fb = pd.read_csv('/content/drive/MyDrive/Module 9: Data Visualization using Pandas, Matplotlib and Seaborn/fb_stock_prices_201
                 index_col = 'date', parse_dates =True)
```

Titles and Axis Labels

- `plt.suptitle()` adds a title to plots and subplots
- `plt.title()` adds a title to a single plot. Note if you use subplots, it will only put the title on the last subplot, so you will need to use `plt.suptitle()`
- `plt.xlabel()` labels the x-axis
- `plt.ylabel()` labels the y-axis

In [5]:
```
fb.close.plot()
plt.suptitle('Fb Closing Price')
plt.xlabel('date')
plt.ylabel('price ($)')
```

Out[5]:
```
Text(0, 0.5, 'price ($)')
```

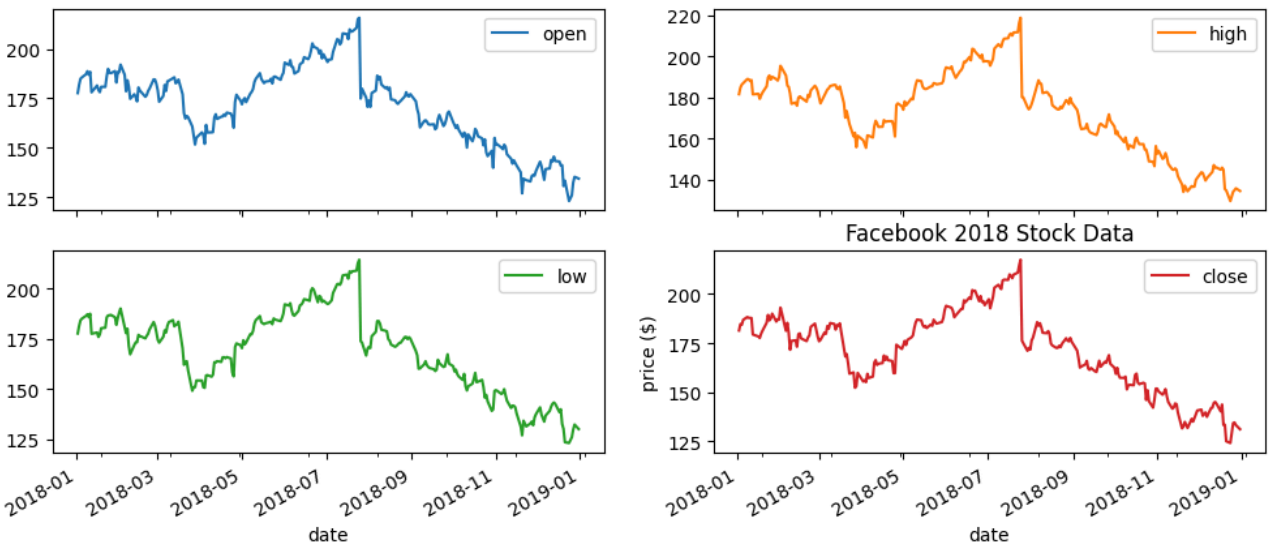


`plt.suptitle()` vs. `plt.title()`

Check out what happens when we call `plt.title()` with subplots:

In [6]:
```
fb.iloc[:,:4].plot(subplots=True, layout=(2, 2), figsize=(12, 5))
plt.title('Facebook 2018 Stock Data')
plt.xlabel('date')
plt.ylabel('price ($)')
```

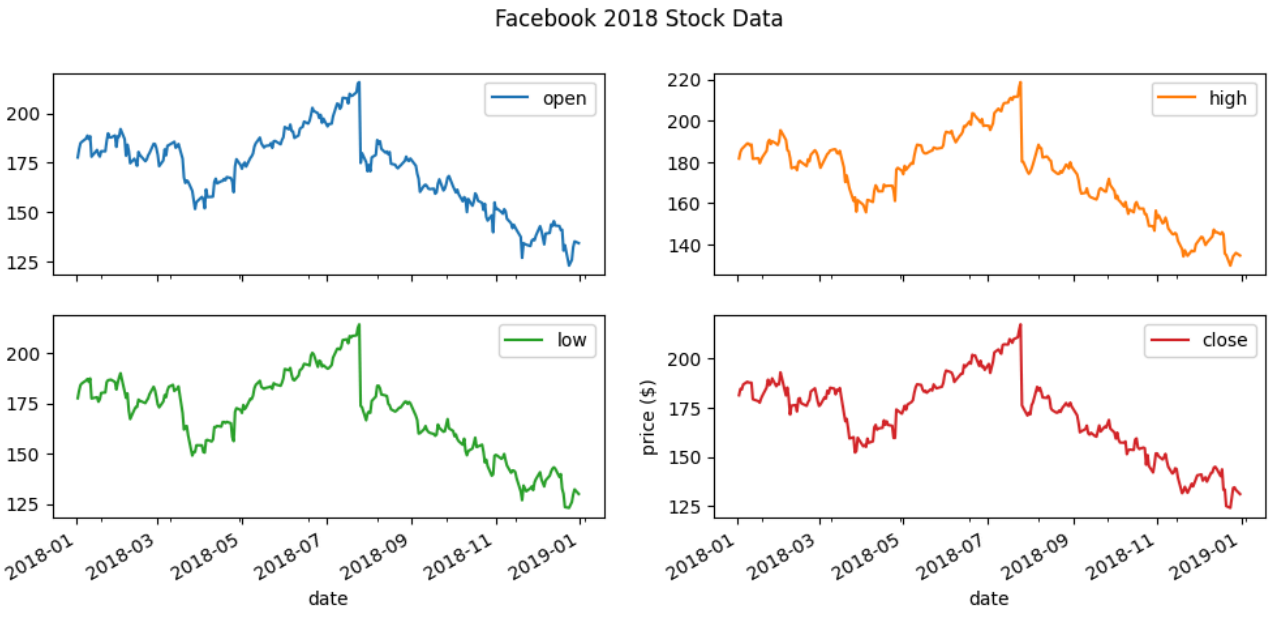
Out[6]: `Text(0, 0.5, 'price ($)')`



Simply getting into the habit of using `plt.suptitle()` instead of `plt.title()` will save you this confusion:

In [7]:
```
fb.iloc[:,:4].plot(subplots=True, layout=(2, 2), figsize=(12, 5))
plt.suptitle('Facebook 2018 Stock Data')
plt.xlabel('date')
plt.ylabel('price ($)')
```

Out[7]: `Text(0, 0.5, 'price ($)')`

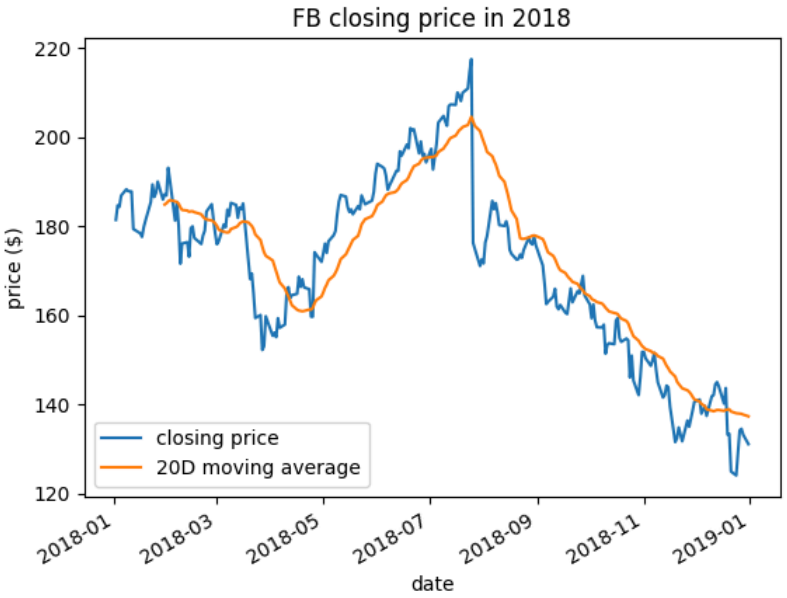


Legends

`plt.legend()` adds a legend to the plot. We can specify where to place it with the loc parameter:

In [8]:
```
fb.assign(
    ma=lambda x: x.close.rolling(20).mean()
).plot(
    y=['close', 'ma'],
    title='FB closing price in 2018',
    label=['closing price', '20D moving average']
)
plt.legend(loc='lower left')
plt.ylabel('price ($)')
```

Out[8]:
```
Text(0, 0.5, 'price ($)')
```



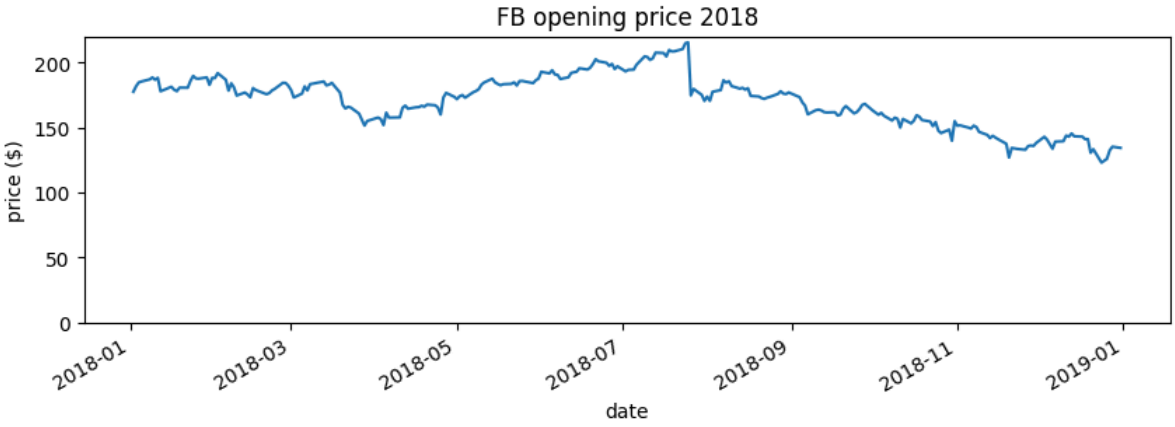
Formatting Axes

Specifying axis limits

`plt.xlim()` and `plt.ylim()` can be used to specify the minimum and maximum values for the axis. Passing None will have matplotlib determine the limit.

In [9]:
```
fb.open.plot(figsize=(10, 3), title='FB opening price 2018')
plt.ylim(0, None)
plt.ylabel('price ($)')
```

Out[9]:
```
Text(0, 0.5, 'price ($)')
```

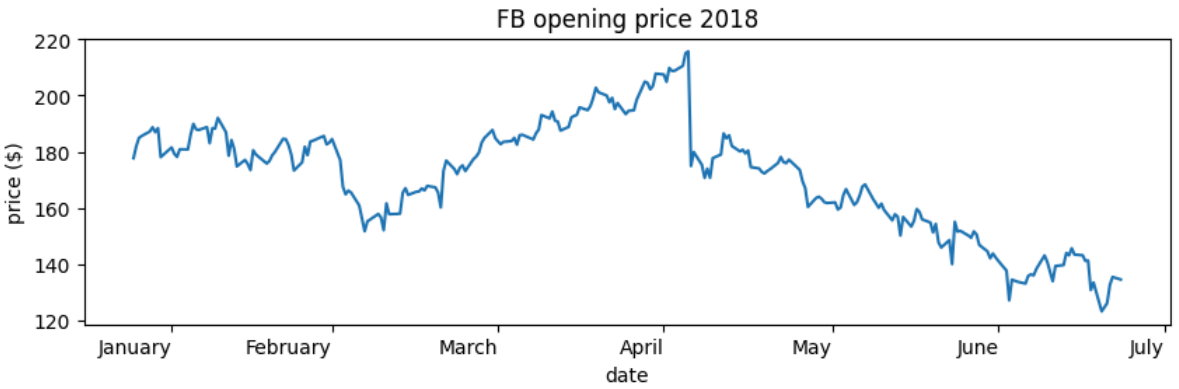


Formatting the Axis Ticks

We can use `plt.xticks()` and `plt.yticks()` to provide tick labels and specify, which ticks to show. Here, we show every other month:

In [41]:
```
import calendar

fb.open.plot(figsize=(10, 3), rot=0, title='FB opening price 2018')
locs, labels = plt.xticks()
plt.xticks(locs + 15 , calendar.month_name[1:-5])
plt.ylabel('price ($)')
```

Out[41]:
```
Text(0, 0.5, 'price ($)')
```



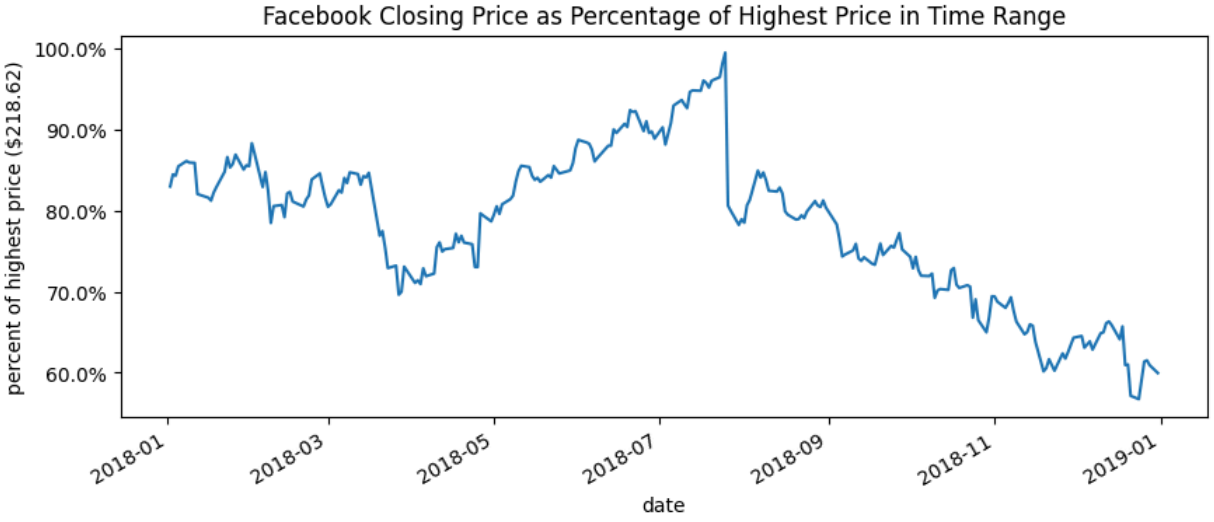
Using `ticker`

`PercentFormatter`

We can use `ticker.PercentFormatter` and specify the denominator (`xmax`) to use when calculating the percentages. This gets passed to the `set_major_formatter()` method of the `xaxis` or `yaxis` on the `Axes.`

In [42]:
```
import matplotlib.ticker as ticker

ax = fb.close.plot(
    figsize=(10, 4),
    title='Facebook Closing Price as Percentage of Highest Price in Time Range'
)
ax.yaxis.set_major_formatter(
    ticker.PercentFormatter(xmax=fb.high.max())
)
ax.set_yticks([
    fb.high.max()*pct for pct in np.linspace(0.6, 1, num=5)
]) # show round percentages only (60%, 80%, etc.)
ax.set_ylabel(f'percent of highest price (${fb.high.max()})')
```

Out[42]:
```
Text(0, 0.5, 'percent of highest price ($218.62)')
```

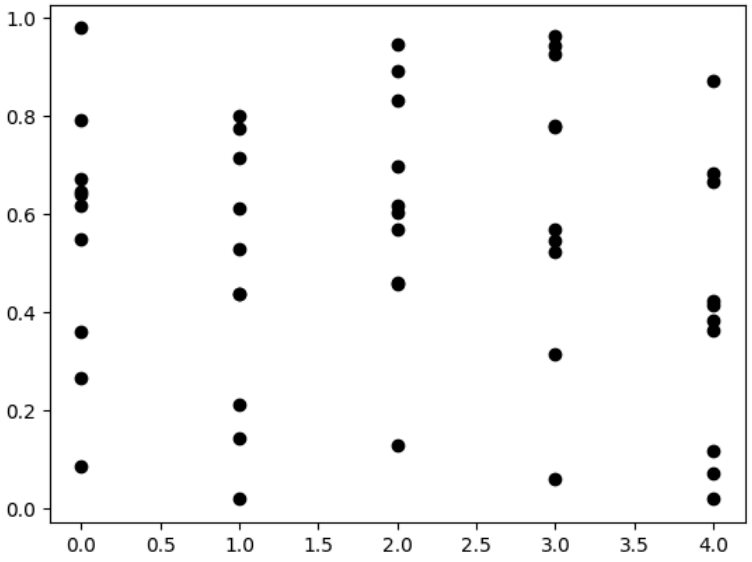


`MultipleLocator`

Say we have the following data. The points only take on integer values for `x`.

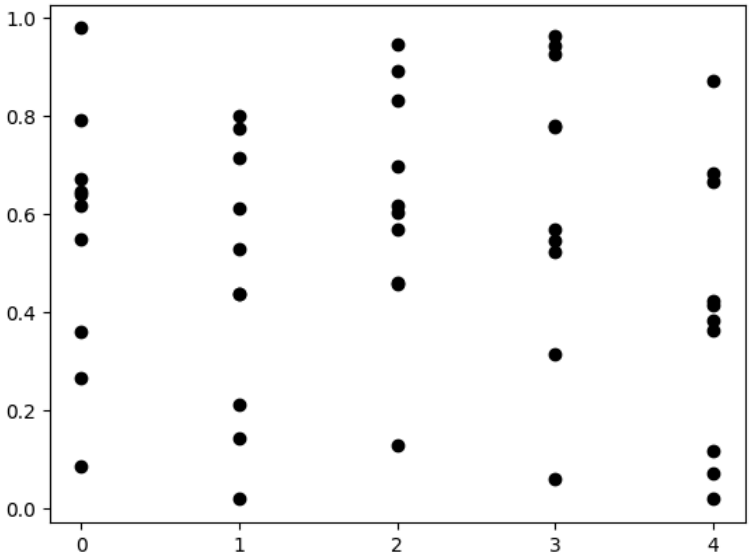
In [43]:
```
fig, ax = plt.subplots(1, 1)
np.random.seed(0)
ax.plot(np.tile(np.arange(0, 5), 10), np.random.rand(50), 'ko')
```

Out[43]:
```
[<matplotlib.lines.Line2D at 0x7c11905e08b0>]
```



If we don't want to show decimal values on the x-axis, we can use the `MultipleLocator`. This will give ticks for all multiples of a number specified with the `base` parameter. To get integer values, we use `base=1`:

In [44]:
```
fig, ax = plt.subplots(1, 1)
np.random.seed(0)
ax.plot(np.tile(np.arange(0, 5), 10), np.random.rand(50), 'ko')
ax.get_xaxis().set_major_locator(
    ticker.MultipleLocator(base=1)
)
```



In [ ]:
```

```