# De_Guzman_Hands_On_Activity_9_1_Data_Visualization_using_Pandas

April 1, 2024

## 1 Hands-on Activity 9.1 Data Visualization using Pandas and Matplotlib

### 1.1 Procedure:

#### 1.1.1 9.1 Getting Started with Matplotlib

We need matplotlib.pyplot for plotting.

[6]:
```
import matplotlib.pyplot as plt
import pandas as pd
```

#### 1.1.2 About the Data

In this notebook, we will be working with 2 datasets:

- Facebook's stock price throughout 2018 (obtained using the stock_analysis package)
- Earthquake data from September 18, 2018 - October 13, 2018 (obtained from the US Geological Survey (USGS) using the USGS API)

#### 1.1.3 Plotting Lines

[7]:
```
fb = pd.read_csv('/content/fb_stock_prices_2018.csv',
                 index_col='date',
                 parse_dates=True)

plt.plot(fb.index, fb.open)
plt.show()
```


Since we are working in a Jupyter notebook, we can use the magic command %matplotlib inline once and not have to call plt.show() for each plot.

[8]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd

fb = pd.read_csv('/content/fb_stock_prices_2018.csv',
                 index_col = 'date',
                 parse_dates=True)

plt.plot(fb.index, fb.open)
```

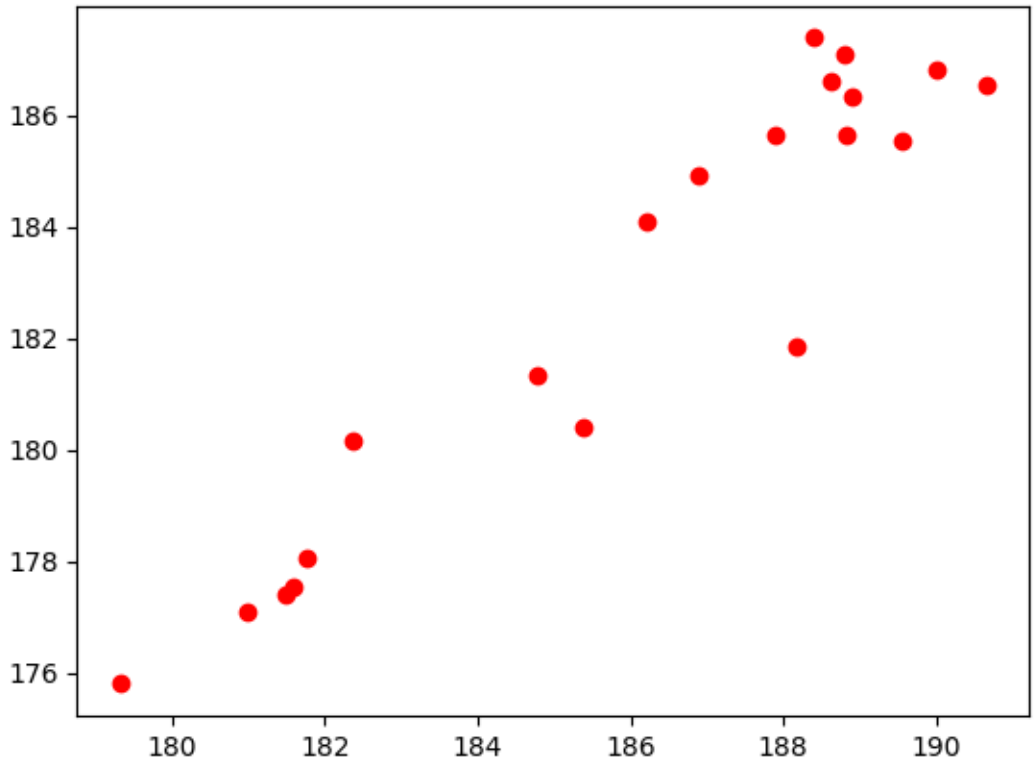
[8]:
```
[<matplotlib.lines.Line2D at 0x7e7dce888100>]
```


### 1.1.4 Scatter Plots

We can pass in a string specifying the style of the plot. This is of the form '[color][marker][linestyle]'. For example, we can make a black dashed line with 'k–' or a red scatter plot with 'ro' :

[9]:
```
plt.plot('high', 'low', 'ro', data=fb.head(20))
```

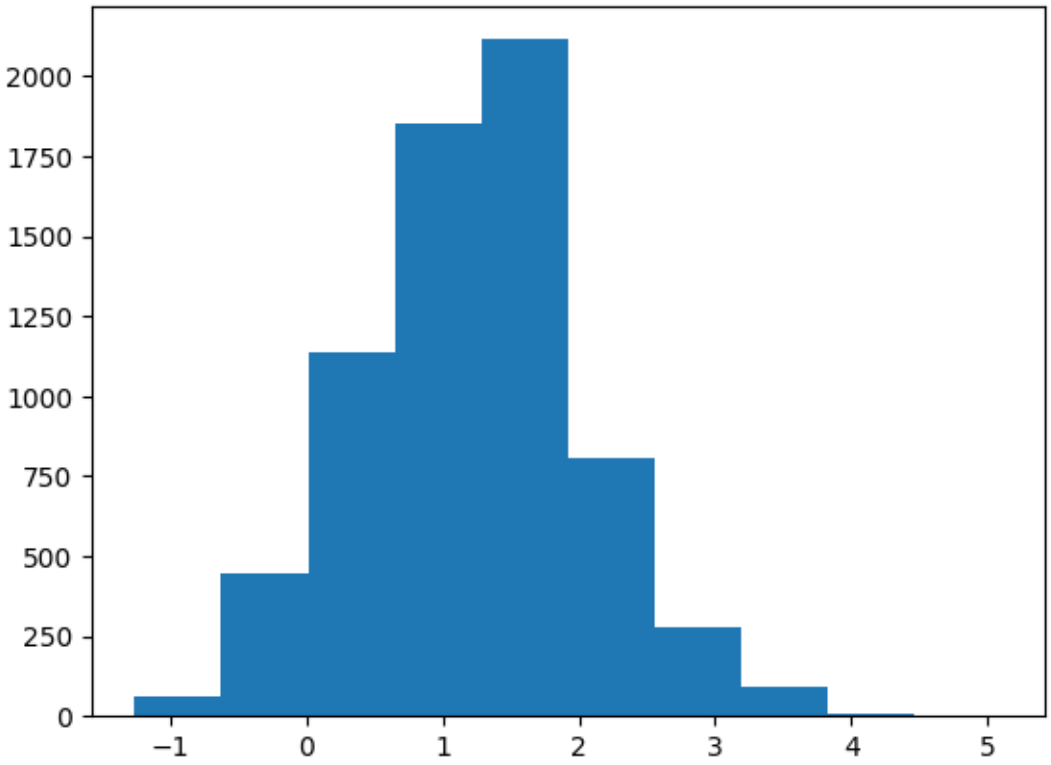
[9]:
```
[<matplotlib.lines.Line2D at 0x7e7dcc73c580>]
```




### 1.1.5 Histograms

[10]:
```
quakes = pd.read_csv('/content/earthquakes-1.csv')
plt.hist(quakes.query('magType == "ml"').mag)
```

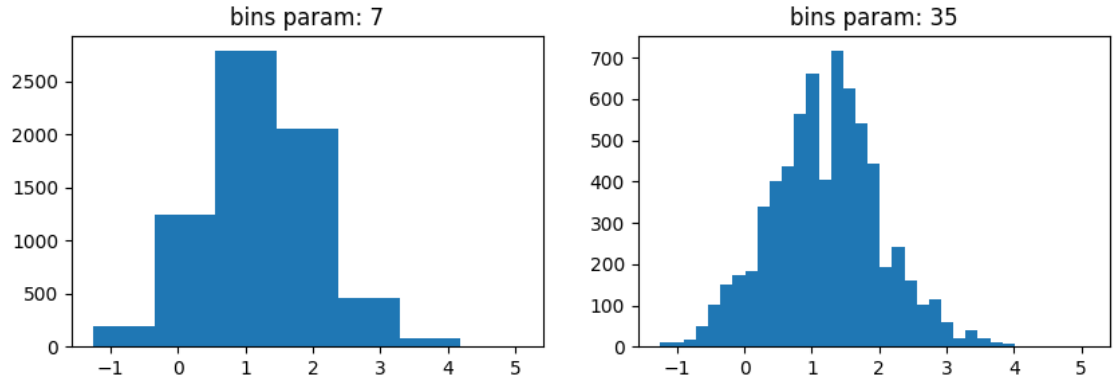
[10]:
```
(array([6.400e+01, 4.450e+02, 1.137e+03, 1.853e+03, 2.114e+03, 8.070e+02,
        2.800e+02, 9.200e+01, 9.000e+00, 2.000e+00]),
 array([-1.26 , -0.624,  0.012,  0.648,  1.284,  1.92 ,  2.556,  3.192,
         3.828,  4.464,  5.1  ]),
 <BarContainer object of 10 artists>)
```




### 1.1.6 Bin size matters

Notice how our assumptions of the distribution of the data can change based on the number of bins (look at the drop between the two highest peaks on the righthand plot):

[11]:
```
x = quakes.query('magType == "ml"').mag
fig, axes = plt.subplots(1, 2, figsize=(10, 3))
for ax, bins in zip(axes, [7, 35]):
  ax.hist(x, bins=bins)
  ax.set_title(f'bins param: {bins}')
```



### 1.1.7 Plot Components

**Figure** Top-level object that holds the other plot components

[12]:
```
fig = plt.figure()
```

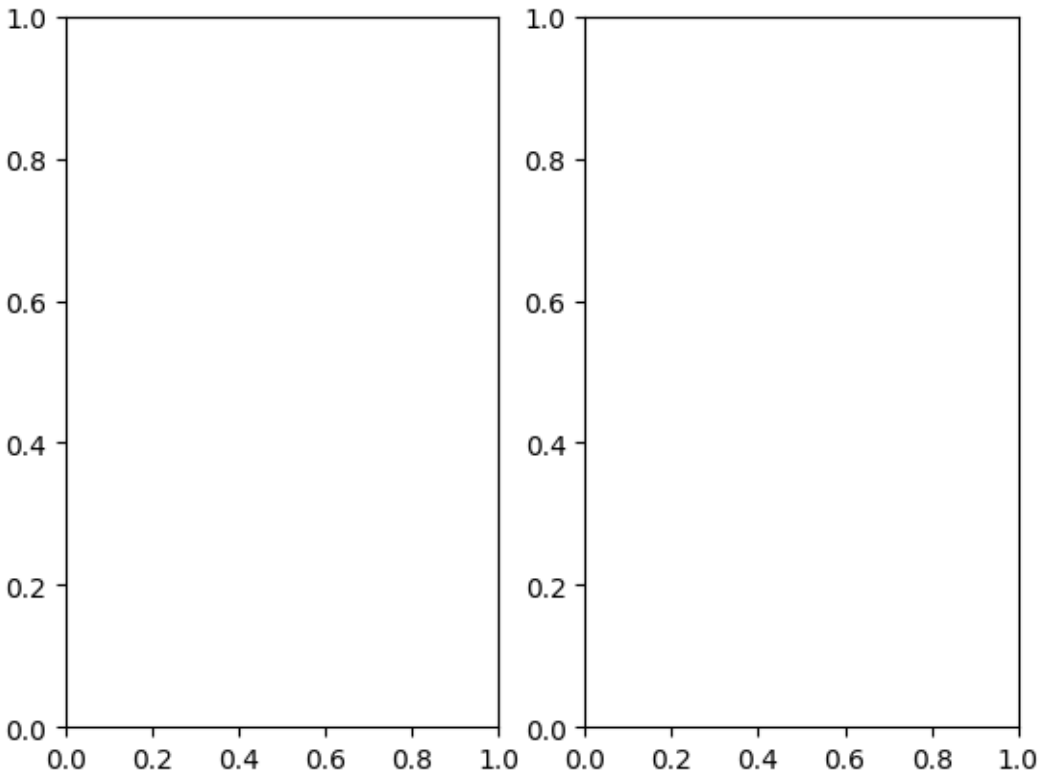
```
<Figure size 640x480 with 0 Axes>
```

**Axes** Individual plots contained within the Figure

### 1.1.8 Creating subplots

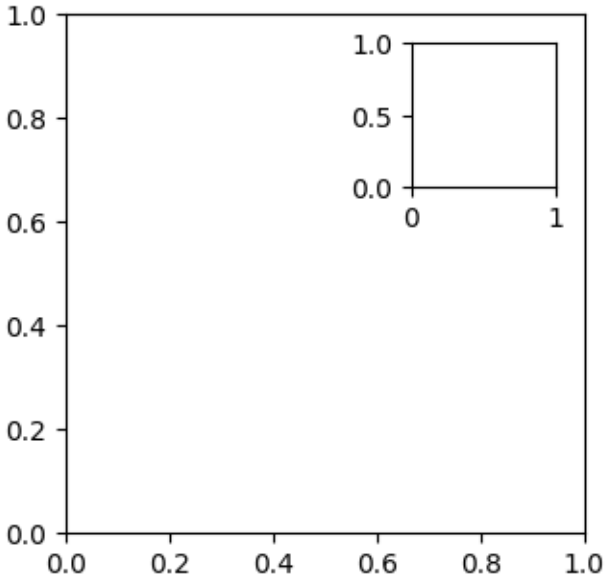
Simply specify the number of rows and columns to create:

[13]:
```
fig, axes = plt.subplots(1, 2)
```




As an alternative to using plt.subplots() we can add the Axes to the Figure on our own. This allows for some more complex layouts, such as picture in picture:

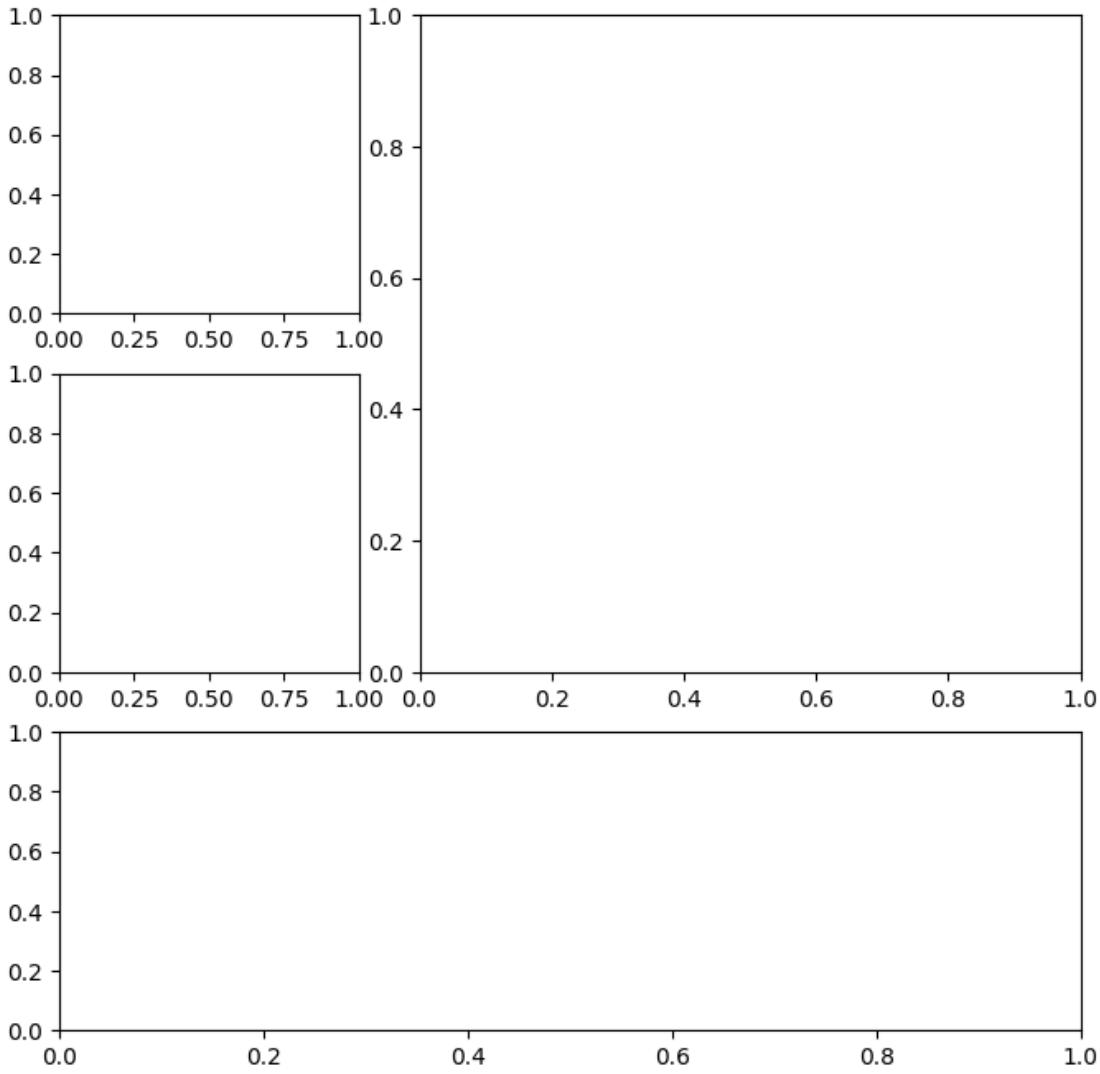
[14]:
```
fig = plt.figure(figsize=(3, 3))
outside = fig.add_axes([0.1, 0.1, 0.9, 0.9])
inside = fig.add_axes([0.7, 0.7, 0.25, 0.25])
```




### 1.1.9 Creating Plot Layouts with gridspec

We can create subplots with varying sizes as well:

[15]:
```
fig = plt.figure(figsize=(8, 8))
gs = fig.add_gridspec(3, 3)
top_left = fig.add_subplot(gs[0, 0])
mid_left = fig.add_subplot(gs[1, 0])
top_right = fig.add_subplot(gs[:2, 1:])
bottom = fig.add_subplot(gs[2,:])
```




### 1.1.10 Saving Plots

Use plt.savefig() to save the last created plot. To save a specific Figure object, use its savefig() method.

[16]:
```
fig.savefig('empty.png')
```

### 1.1.11 Cleaning up

It's important to close resources when we are done with them. We use plt.close() to do so. If we pass in nothing, it will close the last plot, but we can pass the specific Figure to close or say 'all' to close all Figure objects that are open. Let's close all the Figure objects that are open with plt.close():

[17]:
```
plt.close('all')
```

### 1.1.12 Additional plotting options

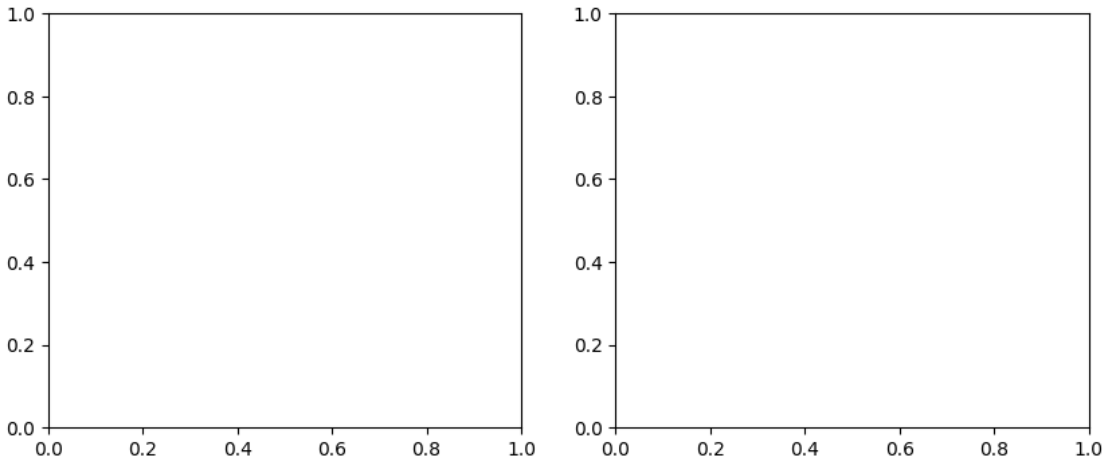
**Specifying figure size** Just pass the figsize parameter to plt.figure() . It's a tuple of (width, height):

[18]:
```
fig = plt.figure(figsize=(10, 4))
```

```
<Figure size 1000x400 with 0 Axes>
```

This can be specified when creating subplots as well:

[19]:
```
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
```




**rcParams** A small subset of all the available plot settings (shuffling to get a good variation of options):

[20]:
```
import random
import matplotlib as mpl

rcparams_list = list(mpl.rcParams.keys())
random.seed(20) # make this repeatable
random.shuffle(rcparams_list)
sorted(rcparams_list[:20])
```

[20]:
```
['animation.convert_args',
 'axes.edgecolor',
 'axes.formatter.use_locale',
 'axes.spines.right',
 'boxplot.meanprops.markersize',
```

```
 'boxplot.showfliers',
 'keymap.home',
 'lines.markerfacecolor',
 'lines.scale_dashes',
 'mathtext.rm',
 'patch.force_edgecolor',
 'savefig.facecolor',
 'svg.fonttype',
 'text.hinting_factor',
 'xtick.alignment',
 'xtick.minor.top',
 'xtick.minor.width',
 'ytick.left',
 'ytick.major.left',
 'ytick.minor.width']
```

We can check the current default figsize using rcParams :

[21]:
```
mpl.rcParams['figure.figsize']
```

[21]:
```
[6.4, 4.8]
```

We can also update this value to change the default (until the kernel is restarted):

[22]:
```
mpl.rcParams['figure.figsize'] = (300, 10)
mpl.rcParams['figure.figsize']
```

[22]:
```
[300.0, 10.0]
```

Use rcdefaults() to restore the defaults:

[23]:
```
mpl.rcdefaults()
mpl.rcParams['figure.figsize']
```

[23]:
```
[6.4, 4.8]
```

This can also be done via pyplot:

[24]:
```
plt.rc('figure', figsize=(20,20)) # change figsize default to (20, 20)
plt.rcdefaults() # reset the default
```

### 1.1.13 Comments and Insights:

From this procedure, the MatPlotLib library is introduced. It is a very essential library to be able to use, especially in statistics. It allows us to plot graphs of our data and gives us a graphical representation of it. It allows us to properly visual our data and makes it easier for us to interpret and analyze the data.

### 1.1.14 9.2 Plotting with Pandas

The plot() method is available on Series and DataFrame objects. Many of the parameters get passed down to matplotlib. The kind argument let's us vary the plot type.

### 1.1.15 About the Data

In this notebook, we will be working with 2 datasets: * Facebook's stock price throughout 2018 (obtained using the stock_analysis package) * Earthquake data from September 18, 2018 - October 13, 2018 (obtained from the US Geological Survey (USGS) using the USGS API)

### 1.1.16 Setup

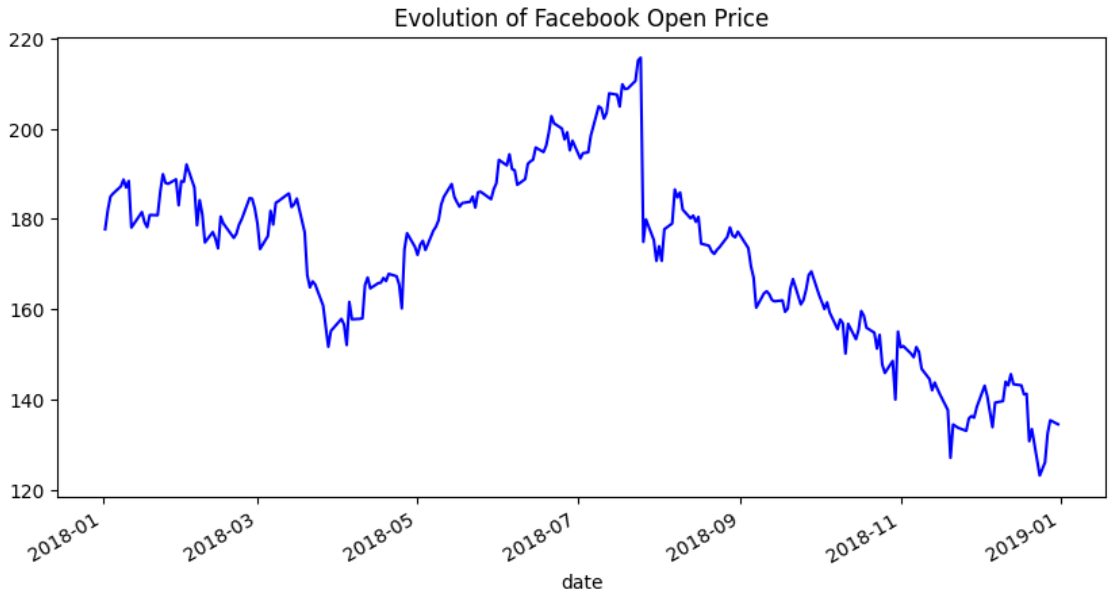
[25]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

fb = pd.read_csv('/content/fb_stock_prices_2018.csv',
                 index_col='date',
                 parse_dates=True
)
quakes = pd.read_csv('/content/earthquakes-1.csv')
```

### 1.1.17 Evolution Over Time

Line plots help us see how a variable changes over time. They are the default for the kind argument, but we can pass kind='line' to be explicit in our intent:

[26]:
```
fb.plot(
    kind='line',
    y='open',
    figsize=(10, 5),
    style='b-',
    legend=False,
    title='Evolution of Facebook Open Price'
)
```

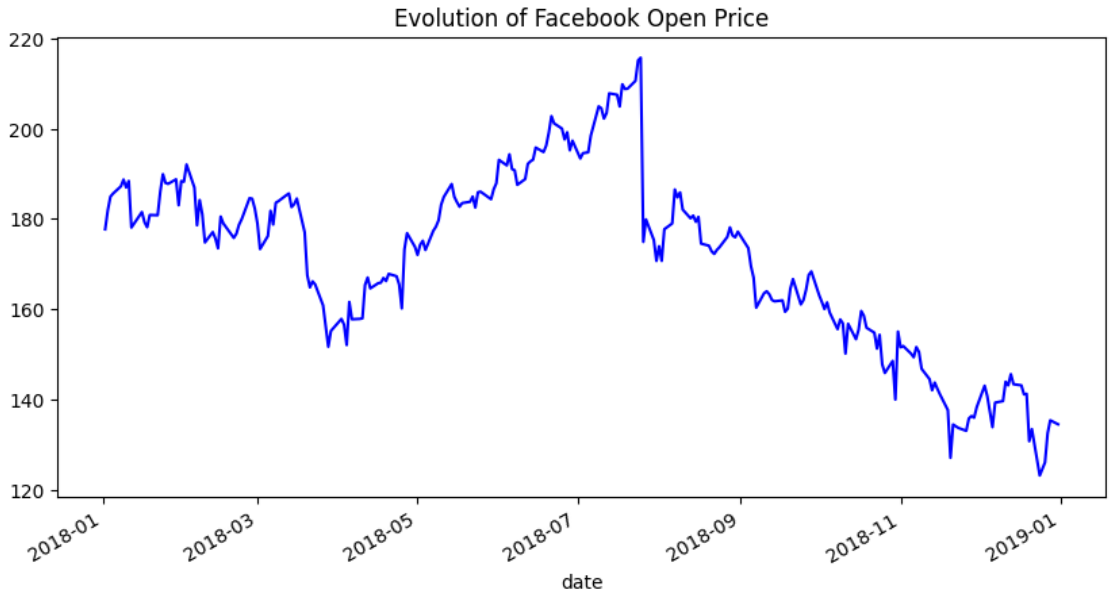
[26]:
```
<Axes: title={'center': 'Evolution of Facebook Open Price'}, xlabel='date'>
```




We provided the style argument in the previous example; however, we can use the color and linestyle arguments to get the same result:

[27]:
```
fb.plot(
    kind='line',
    y='open',
    figsize=(10, 5),
    color='blue',
    linestyle='solid',
    legend=False,
    title='Evolution of Facebook Open Price'
)
```

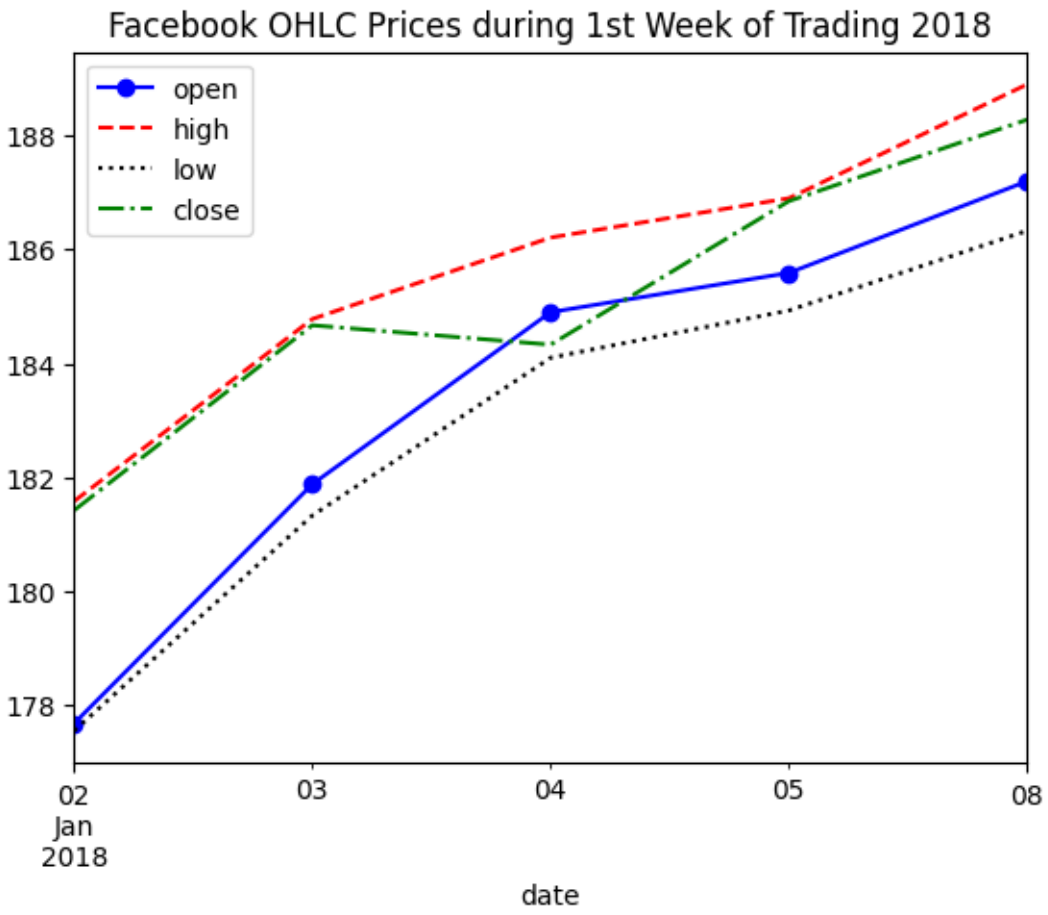
[27]:
```
<Axes: title={'center': 'Evolution of Facebook Open Price'}, xlabel='date'>
```




We can also plot many lines at once by simply passing a list of the columns to plot:

[28]:
```
fb.iloc[:5].plot(
    y=['open', 'high', 'low', 'close'],
    style=['b-o', 'r--', 'k:', 'g-.'],
    title='Facebook OHLC Prices during 1st Week of Trading 2018'
)
```

[28]:
```
<Axes: title={'center': 'Facebook OHLC Prices during 1st Week of Trading 2018'},
xlabel='date'>
```

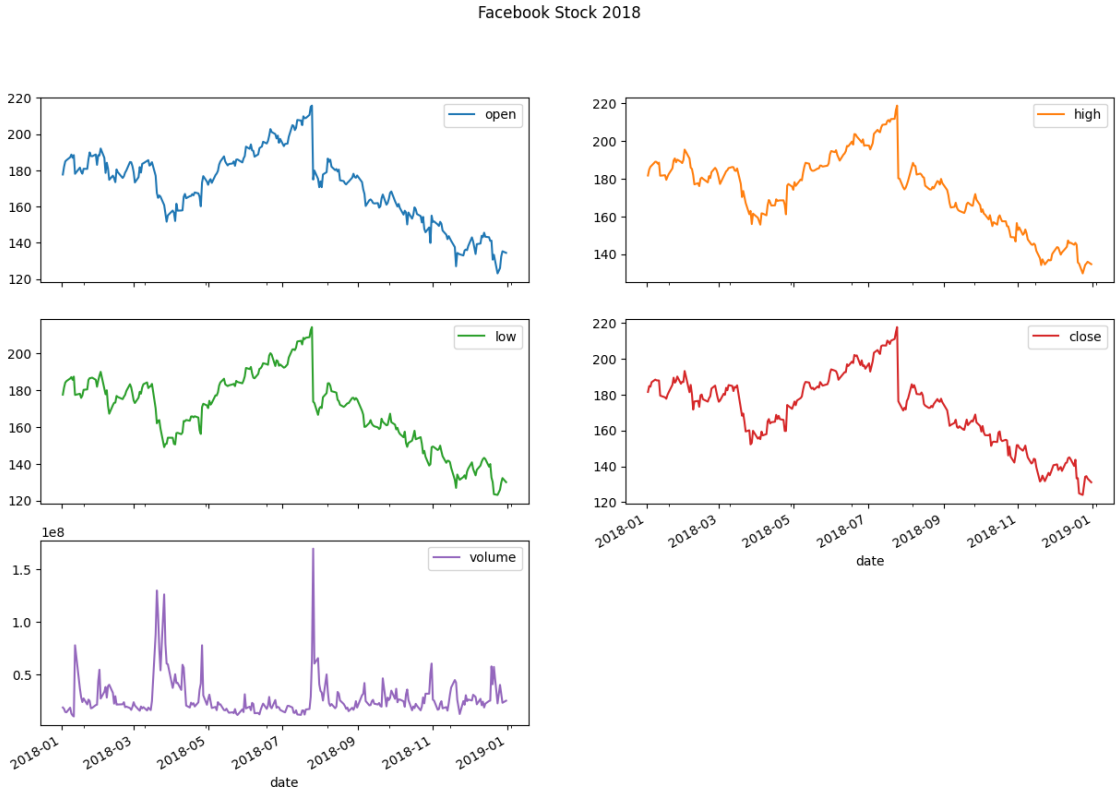



### 1.1.18 Creating subplots

When plotting with pandas, creating subplots is simply a matter of passing subplots=True to the plot() method, and (optionally) specifying the layout in a tuple of (rows, columns):

[29]:
```
fb.plot(
    kind='line',
    subplots=True,
    layout=(3,2),
    figsize=(15,10),
    title='Facebook Stock 2018'
)
```

[29]:
```
array([[<Axes: xlabel='date'>, <Axes: xlabel='date'>],
       [<Axes: xlabel='date'>, <Axes: xlabel='date'>],
       [<Axes: xlabel='date'>, <Axes: xlabel='date'>]], dtype=object)
```



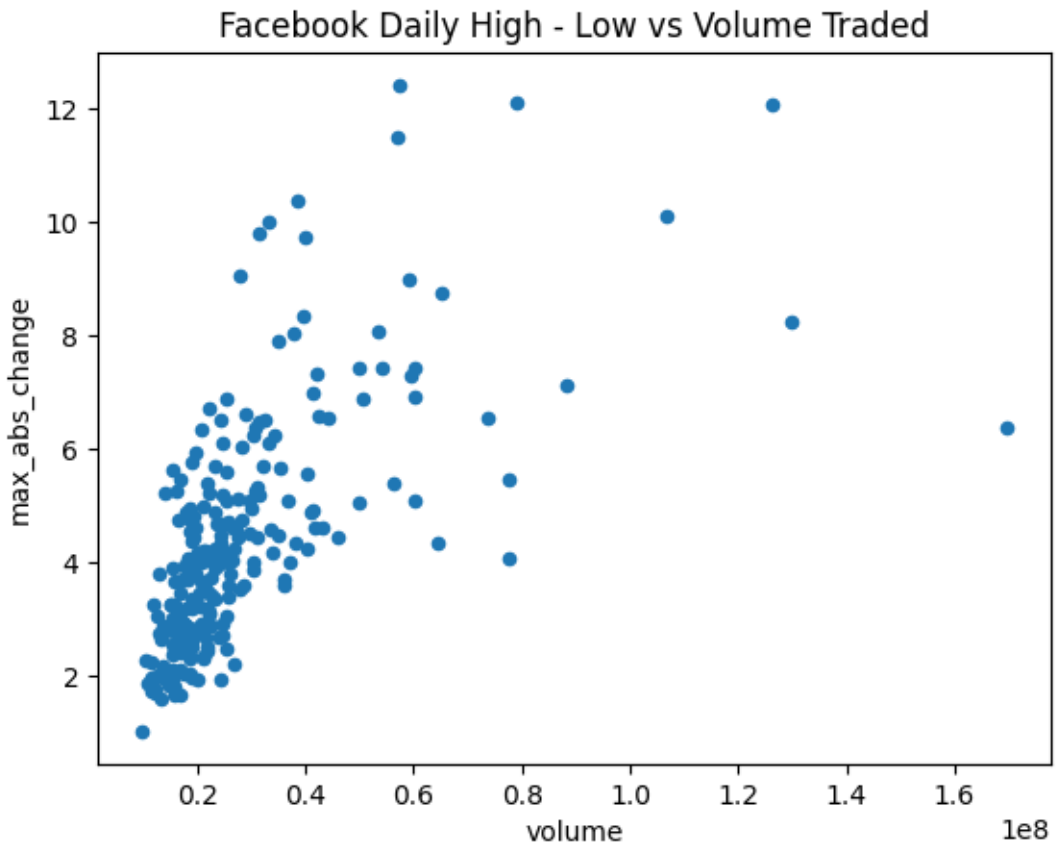

*Note that we didn't provide a specific column to plot and pandas plotted all of them for us*

### 1.1.19 Visualizing relationships between variables

**Scatter plots** We make scatter plots to help visualize the relationship between two variables. Creating scatter plots requires we pass in kind='scatter' along with a column for the x- axis and a column for the y-axis:

[30]:
```
fb.assign(
    max_abs_change=fb.high-fb.low
).plot(
    kind="scatter",
    x="volume",
    y="max_abs_change",
    title='Facebook Daily High - Low vs Volume Traded'
)
```

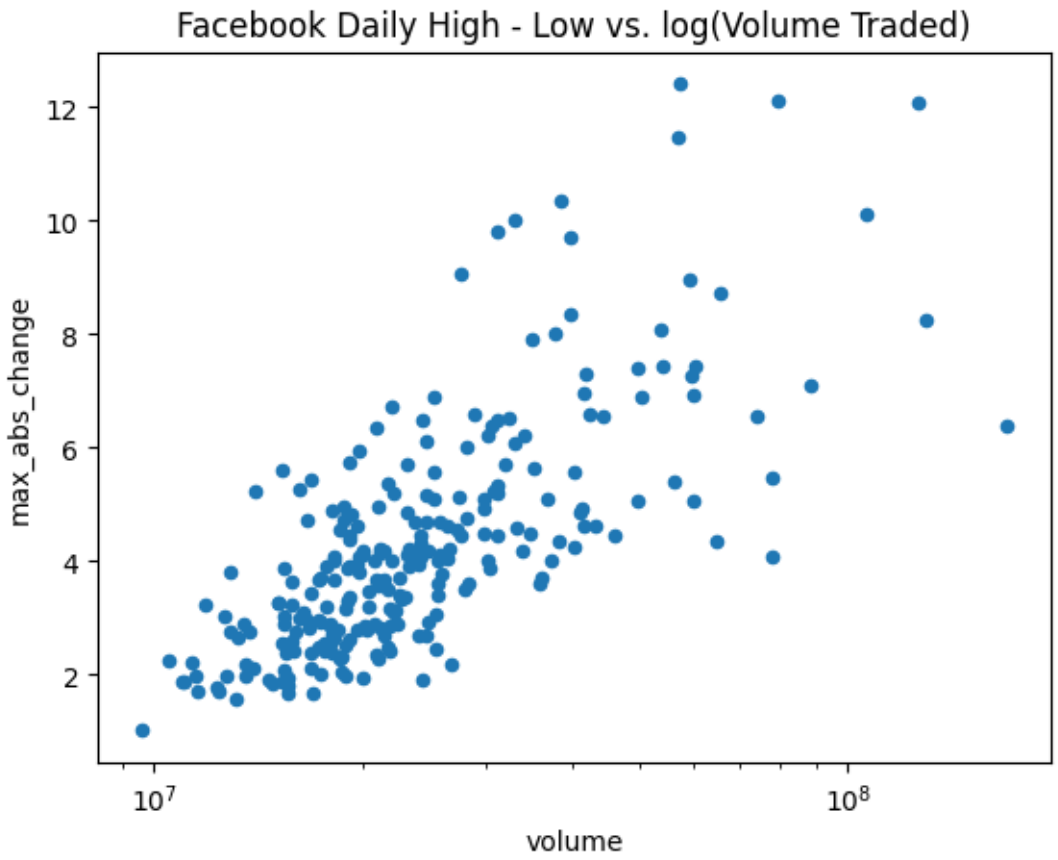
[30]:
```
<Axes: title={'center': 'Facebook Daily High - Low vs Volume Traded'},
xlabel='volume', ylabel='max_abs_change'>
```




The relationship doesn't seem to be linear, but we can try a log transform on the x-axis since the scales of the axes are very different. With pandas, we simply pass in logx=True:

[31]:
```
fb.assign(
    max_abs_change=fb.high - fb.low
).plot(
    kind='scatter',
    x='volume',
    y='max_abs_change',
    title='Facebook Daily High - Low vs. log(Volume Traded)',
    logx=True
)
```

[31]:
```
<Axes: title={'center': 'Facebook Daily High - Low vs. log(Volume Traded)'},
xlabel='volume', ylabel='max_abs_change'>
```



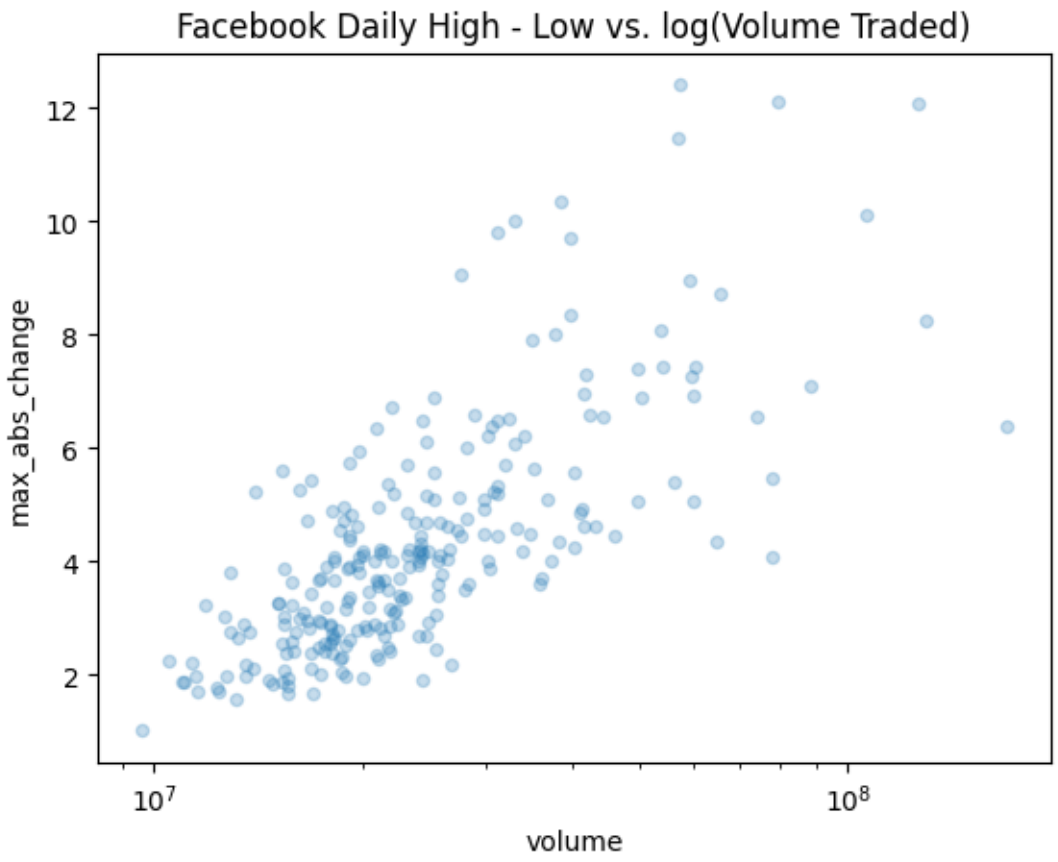

With matplotlib, we could use plt.xscale('log') to do the same thing.

### 1.1.20 Adding Transparency to Plots with alpha

Sometimes our plots have many overlapping values, but this can be impossible to see. This can be addressed by increasing the transparency of what we are plotting using the alpha parameter. It is a float on [0, 1] where 0 is completely transparent and 1 is completely opaque. By default this is 1, so let's put in a lower value and re-plot the scatter plot:

[32]:
```
fb.assign(
    max_abs_change=fb.high - fb.low
).plot(
    kind='scatter',
    x='volume',
    y='max_abs_change',
    title='Facebook Daily High - Low vs. log(Volume Traded)',
    logx=True,
    alpha=0.25
)
```

[32]:
```
<Axes: title={'center': 'Facebook Daily High - Low vs. log(Volume Traded)'},
xlabel='volume', ylabel='max_abs_change'>
```



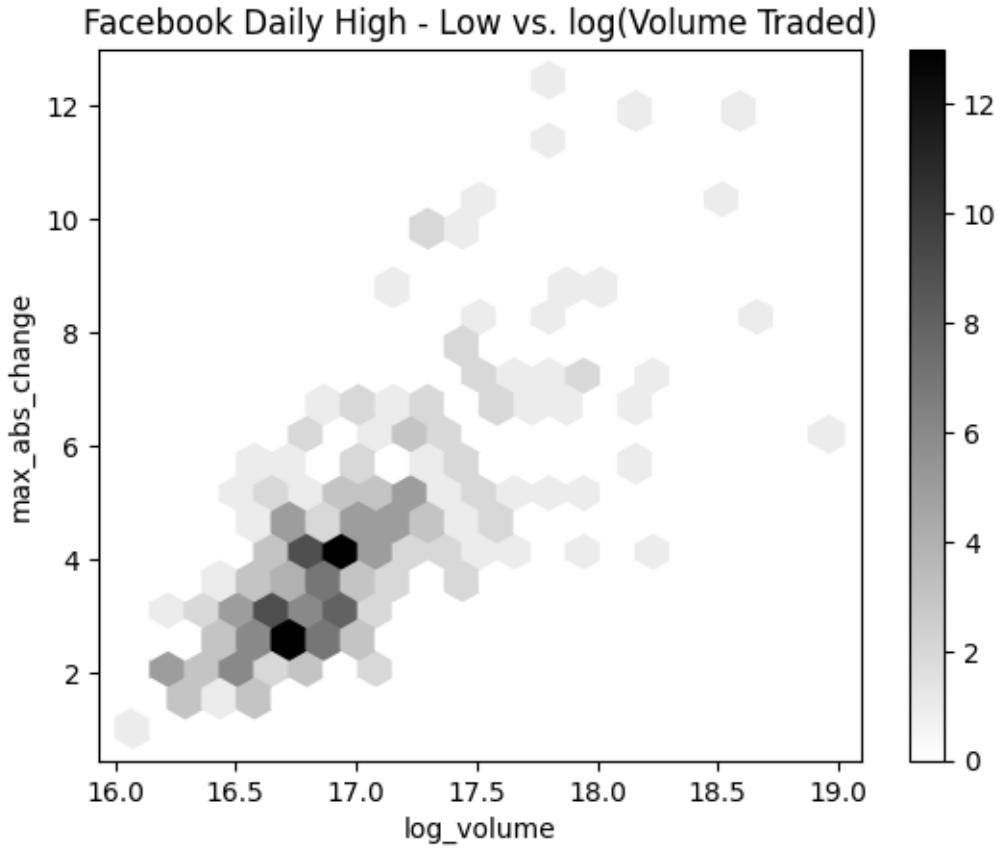

### 1.1.21 Hexbins

In the previous example, we can start to see the overlaps, but it is still difficult. Hexbins are another plot type that divide up the plot into hexagons, which are shaded according to the density of points there. With pandas, this is the hexbin value for the kind argument. It can also be important to tweak the gridsize , which determines the number of hexagons along the y-axis:

[33]:
```
fb.assign(
    log_volume=np.log(fb.volume),
    max_abs_change=fb.high - fb.low
).plot(
    kind='hexbin',
    x='log_volume',
    y='max_abs_change',
    title='Facebook Daily High - Low vs. log(Volume Traded)',
    colormap='gray_r',
    gridsize=20,
```

```
    sharex=False # we have to pass this to see the x-axis due to a bug in this␣
 ↪version of pandas
)
```

[33]:
```
<Axes: title={'center': 'Facebook Daily High - Low vs. log(Volume Traded)'},
xlabel='log_volume', ylabel='max_abs_change'>
```




### 1.1.22 Visualizing Correlations with Heatmaps

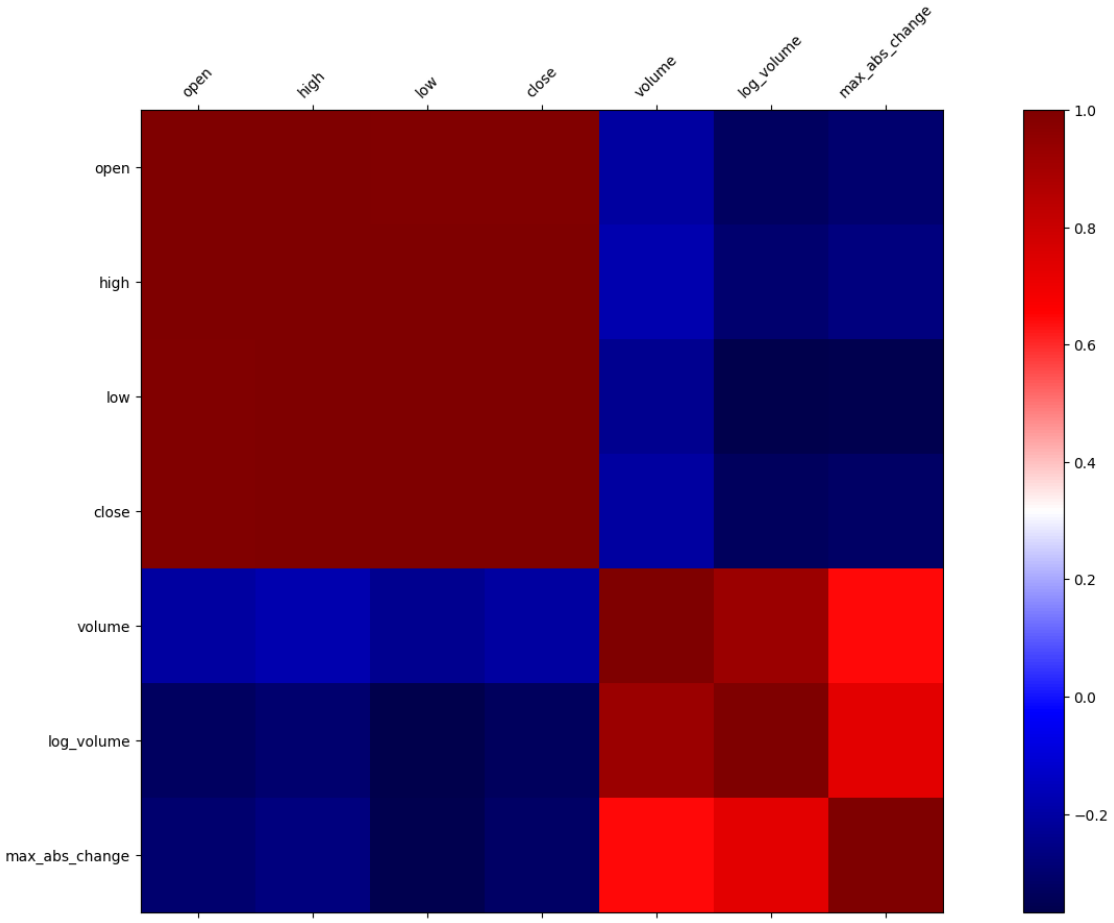
Pandas doesn't offer heatmaps; however, if we are able to get our data into a matrix, we can use matshow() from matplotlib:

[34]:
```
fig, ax = plt.subplots(figsize=(20, 10))

fb_corr = fb.assign(
    log_volume=np.log(fb.volume),
    max_abs_change=fb.high - fb.low
).corr()

im = ax.matshow(fb_corr, cmap='seismic')
```

```
fig.colorbar(im)

labels = [col.lower() for col in fb_corr.columns]
ax.set_xticklabels([''] + labels, rotation=45)
ax.set_yticklabels([''] + labels)
```

```
<ipython-input-34-3f78d5e545fb>:12: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels([''] + labels, rotation=45)
<ipython-input-34-3f78d5e545fb>:13: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels([''] + labels)
```

[34]:
```
[Text(0, -1.0, ''),
 Text(0, 0.0, 'open'),
 Text(0, 1.0, 'high'),
 Text(0, 2.0, 'low'),
 Text(0, 3.0, 'close'),
 Text(0, 4.0, 'volume'),
 Text(0, 5.0, 'log_volume'),
 Text(0, 6.0, 'max_abs_change'),
 Text(0, 7.0, '')]
```




[35]:
```
fb_corr.loc['max_abs_change', ['volume', 'log_volume']]
```

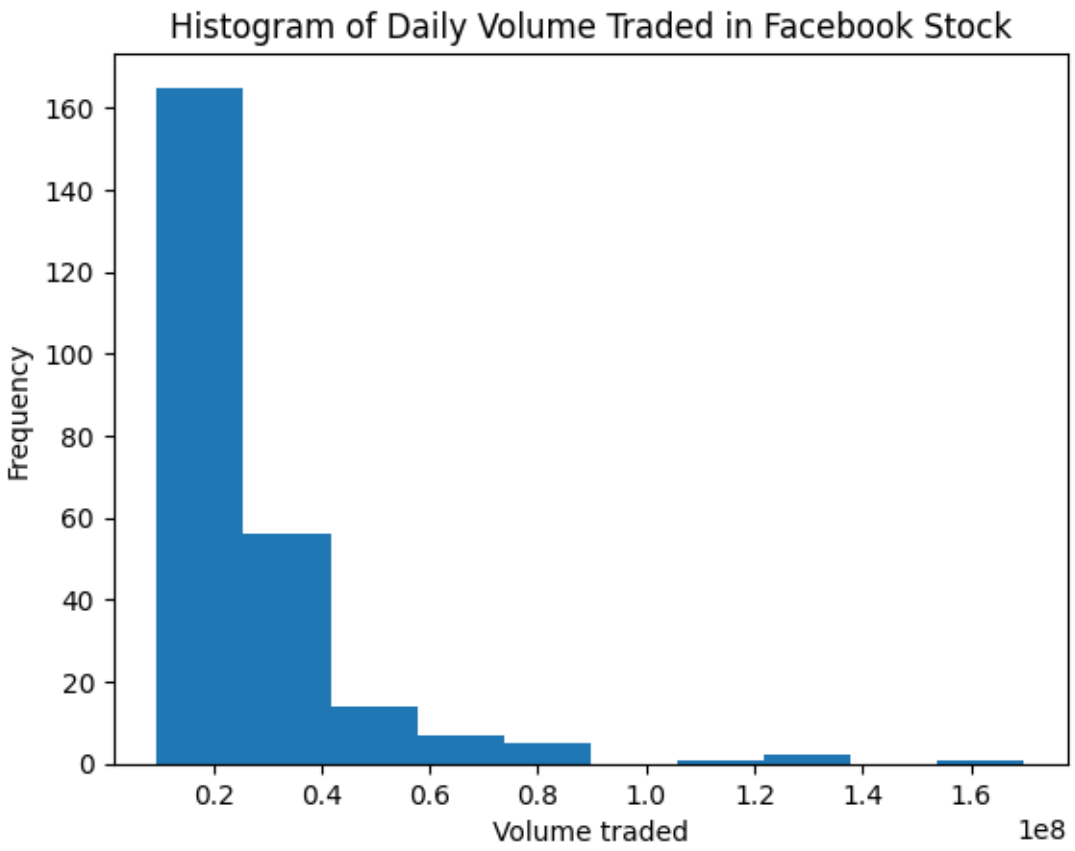
[35]:
```
volume        0.642027
log_volume    0.731542
Name: max_abs_change, dtype: float64
```

### 1.1.23 Visualizing distributions

**Histograms** With the pandas plot() method, making histograms is as easy as passing in kind='hist' :

[36]:
```
fb.volume.plot(
    kind='hist',
    title='Histogram of Daily Volume Traded in Facebook Stock'
)

plt.xlabel('Volume traded') # label the x-axis (discussed in chapter 6)
```

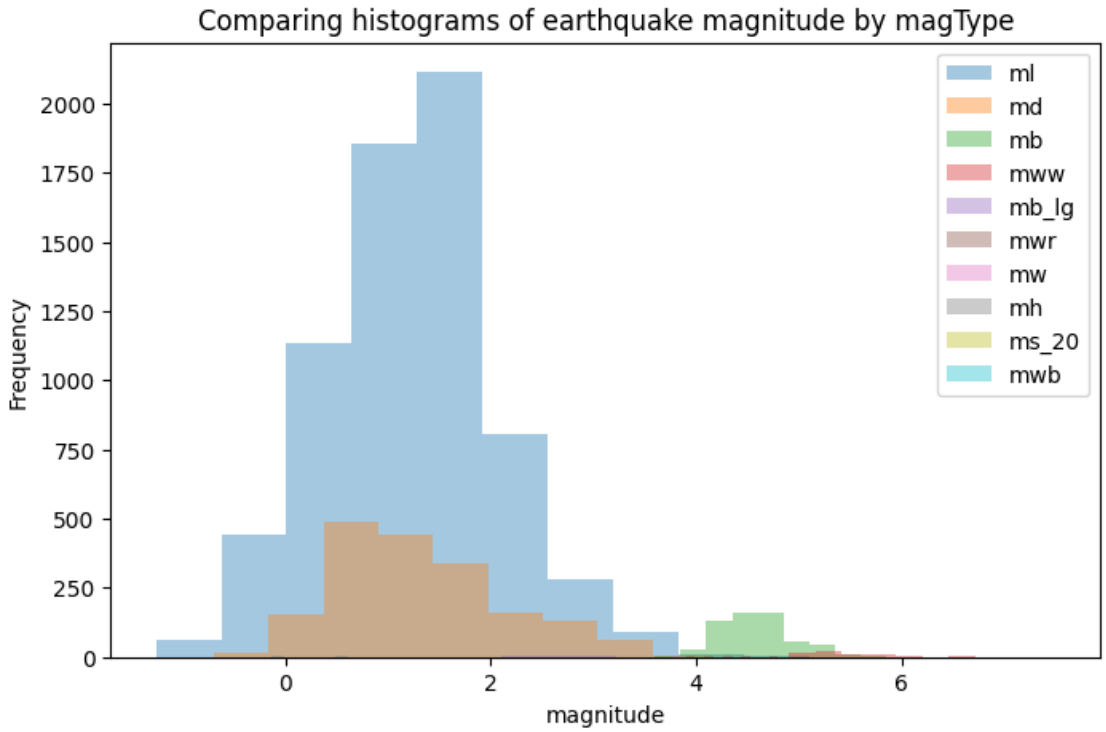
[36]: Text(0.5, 0, 'Volume traded')




We can overlap histograms to compare distributions provided we use the alpha parameter. For example, let's compare the usage and magnitude of the various magTypes in the data:

[37]:
```
fig, axes = plt.subplots(figsize=(8, 5))
for magtype in quakes.magType.unique():
  data = quakes.query(f'magType == "{magtype}"').mag
  if not data.empty:
    data.plot(
        kind='hist', ax=axes, alpha=0.4,
        label=magtype, legend=True,
        title='Comparing histograms of earthquake magnitude by magType'
    )

plt.xlabel('magnitude') # label the x-axis (discussed in chapter 6)
```

[37]: Text(0.5, 0, 'magnitude')

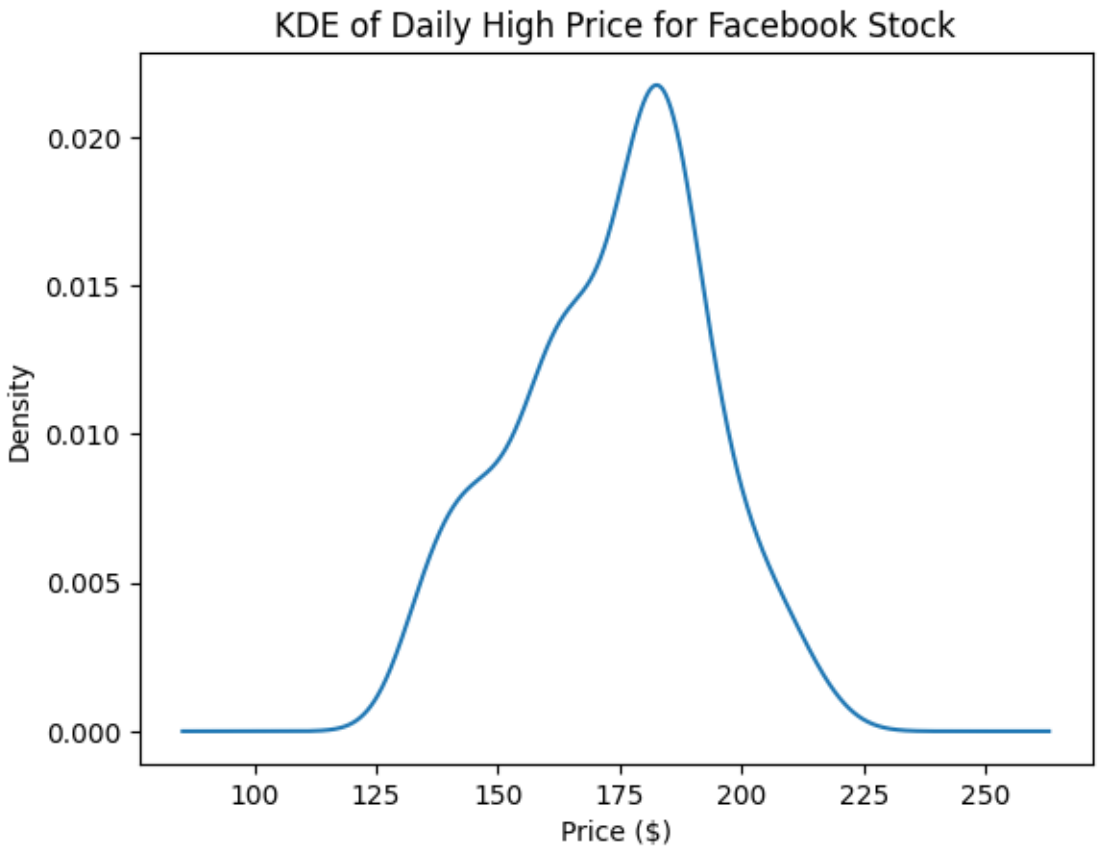



### 1.1.24 Kernel Density Estimation (KDE)

We can pass kind='kde' for a probability density function (PDF), which tells us the probability of getting a particular value:

[38]:
```
fb.high.plot(
    kind='kde',
    title='KDE of Daily High Price for Facebook Stock'
)

plt.xlabel('Price ($)') # label the x-axis (discussed in chapter 6)
```

[38]:
```
Text(0.5, 0, 'Price ($)')
```



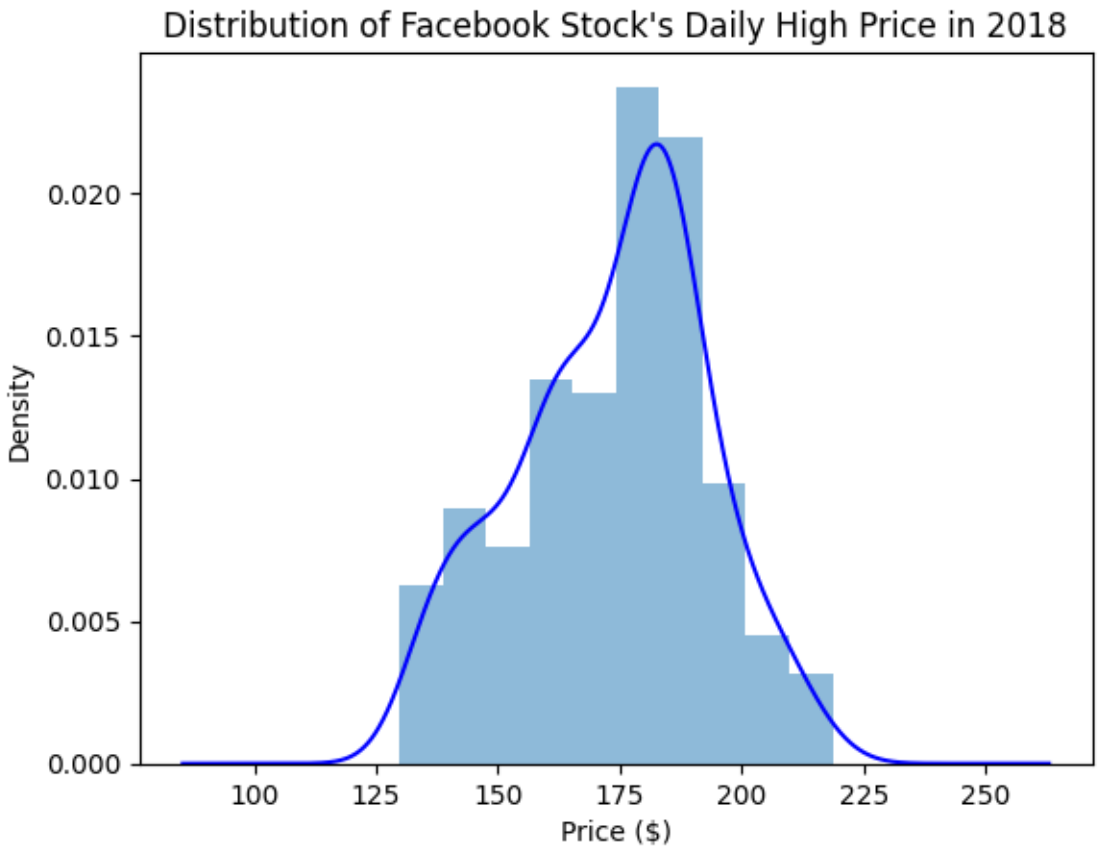

### 1.1.25 Adding to the result of plot()

The plot() method returns a matplotlib Axes object. We can store this for additional customization of the plot, or we can pass this into another call to plot() as the ax argument to add to the original plot.

It can often be helpful to view the KDE superimposed on top of the histogram, which can be achieved with this strategy:

[39]:
```
ax = fb.high.plot(kind='hist', density=True, alpha=0.5)
fb.high.plot(
    ax=ax, kind='kde', color='blue',
    title='Distribution of Facebook Stock\'s Daily High Price in 2018'
)

plt.xlabel('Price ($)') # label the x-axis (discussed in chapter 6)
```

[39]:
```
Text(0.5, 0, 'Price ($)')
```

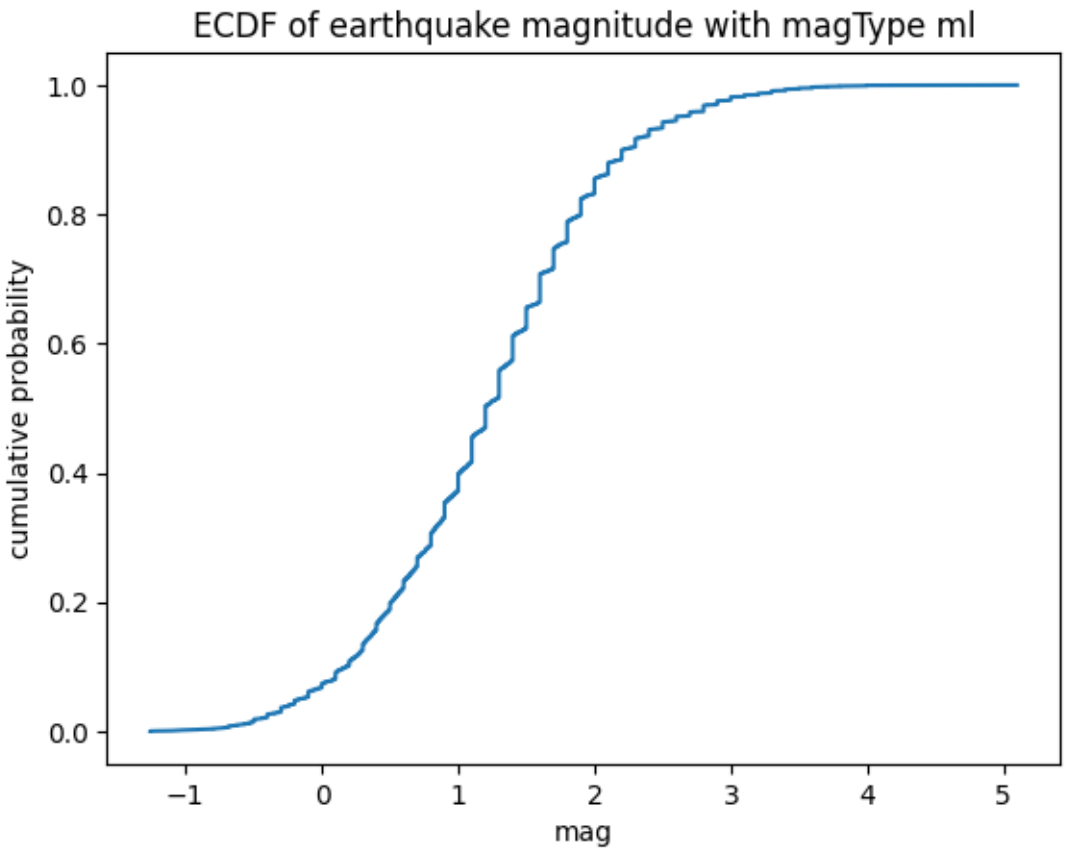



### 1.1.26 Plotting the ECDF

In some cases, we are more interested in the probability of getting less than or equal to that value (or greater than or equal), which we can see with the cumulative disribution function (CDF). Using the statsmodels package, we can estimate the CDF giving us the empirical cumulative distribution function (ECDF):

[40]:
```
from statsmodels.distributions.empirical_distribution import ECDF

ecdf = ECDF(quakes.query('magType == "ml"').mag)
plt.plot(ecdf.x, ecdf.y)

# axis labels (we will cover this in chapter 6)
plt.xlabel('mag') # add x-axis label
plt.ylabel('cumulative probability') # add y-axis label

# add title (we will cover this in chapter 6)
plt.title('ECDF of earthquake magnitude with magType ml')
```

[40]:
```
Text(0.5, 1.0, 'ECDF of earthquake magnitude with magType ml')
```

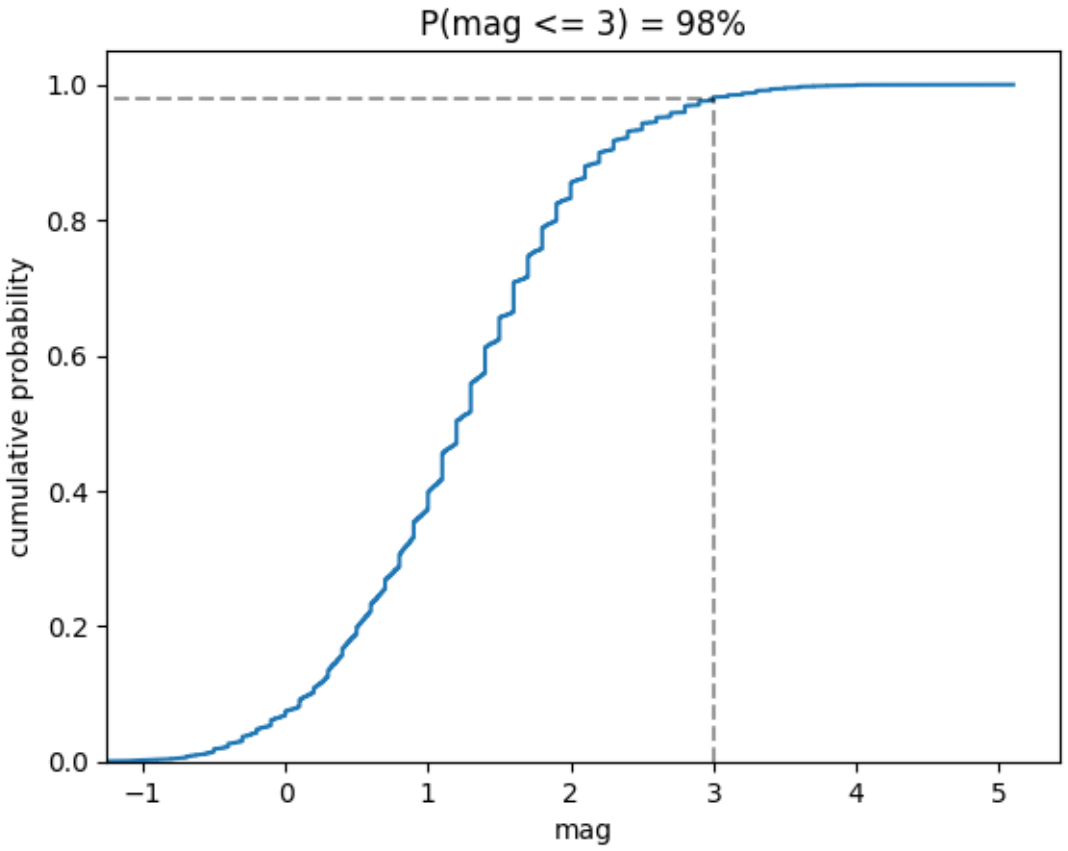



This ECDF tells us the probability of getting an earthquake with magnitude of 3 or less using the ml scale is 98%:

[41]:
```
from statsmodels.distributions.empirical_distribution import ECDF

ecdf = ECDF(quakes.query('magType == "ml"').mag)
plt.plot(ecdf.x, ecdf.y)

# formatting below will all be covered in chapter 6
# axis labels
plt.xlabel('mag') # add x-axis label
plt.ylabel('cumulative probability') # add y-axis label

# add reference lines for interpreting the ECDF for mag <= 3
plt.plot(
    [3, 3], [0, .98], 'k--',
     [-1.5, 3], [0.98, 0.98], 'k--', alpha=0.4
)

# set axis ranges
```

```
plt.ylim(0, None)
plt.xlim(-1.25, None)

# add a title
plt.title('P(mag <= 3) = 98%')
```

[41]:
```
Text(0.5, 1.0, 'P(mag <= 3) = 98%')
```

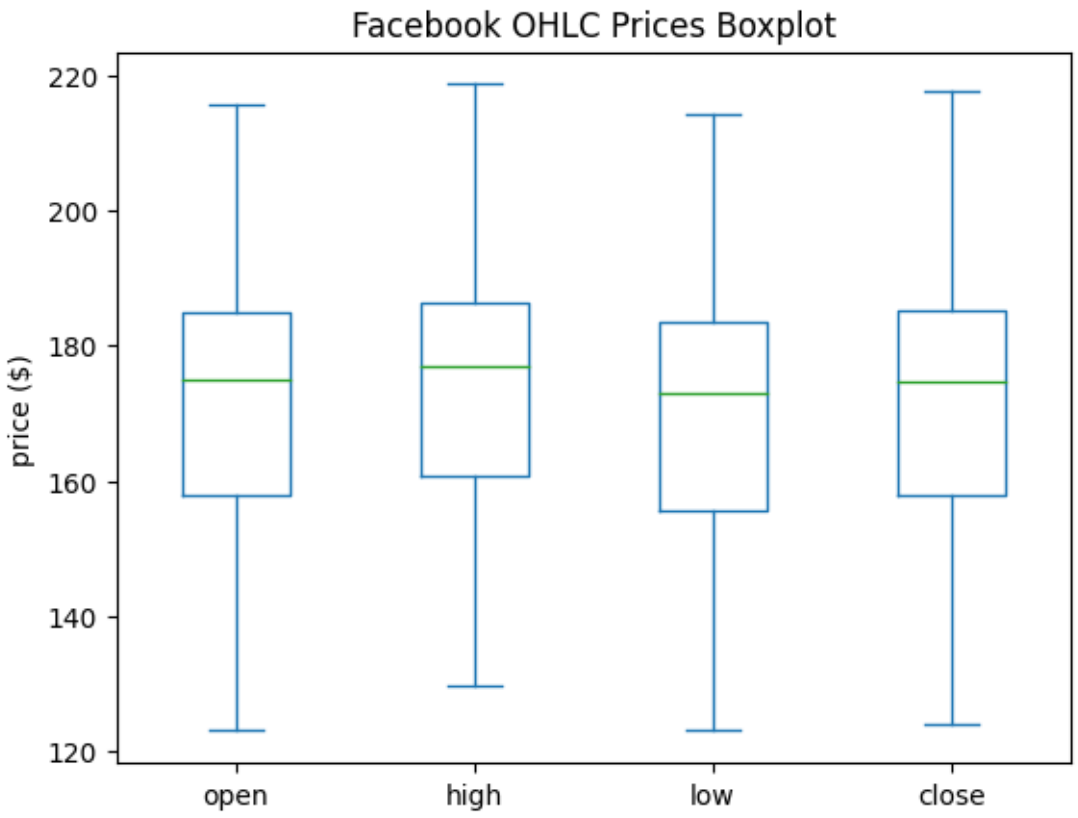



### 1.1.27 Box plots

To make box plots with pandas, we pass kind='box' to the plot() method:

[42]:
```
fb.iloc[:,:4].plot(kind='box', title='Facebook OHLC Prices Boxplot')
plt.ylabel('price ($)') # label the x-axis (discussed in chapter 6)
```

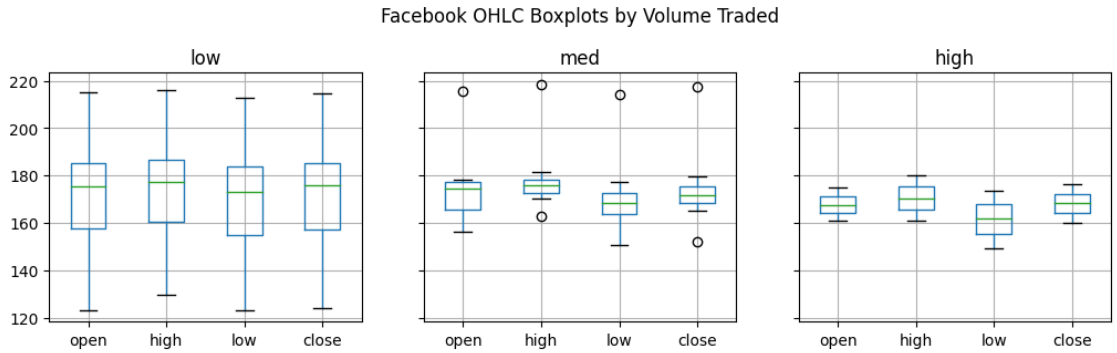
[42]:
```
Text(0, 0.5, 'price ($)')
```




This can also be combined with a groupby()

[43]:
```
fb.assign(
    volume_bin=pd.cut(fb.volume, 3, labels=['low', 'med', 'high'])
).groupby('volume_bin').boxplot(
    column=['open', 'high', 'low', 'close'],
    layout=(1, 3), figsize=(12, 3)
)

plt.suptitle('Facebook OHLC Boxplots by Volume Traded', y=1.1)
```

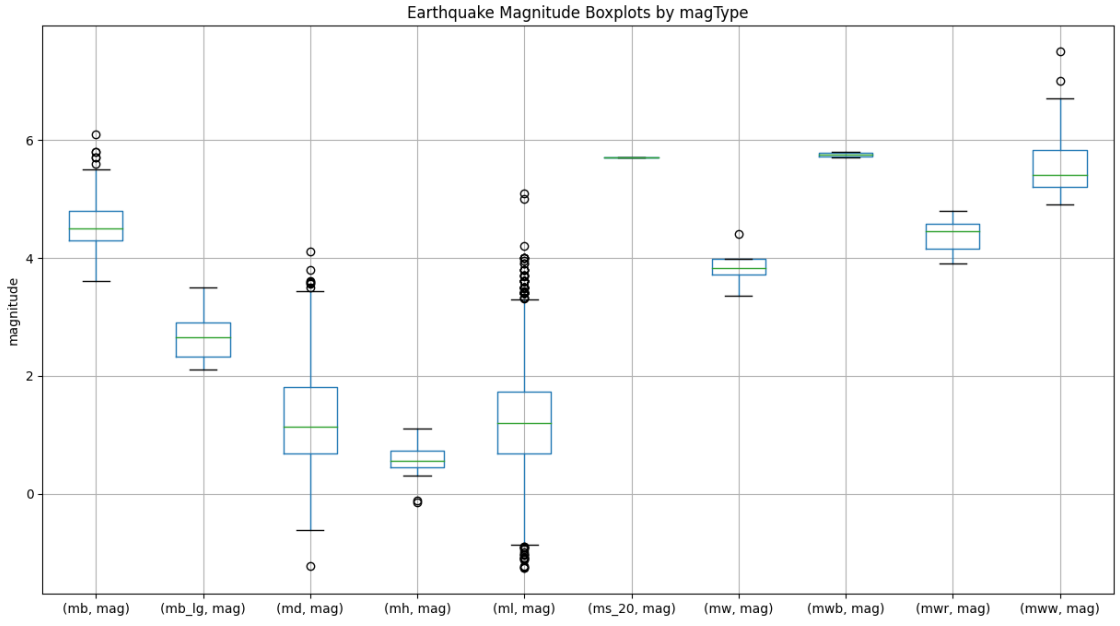
[43]:
```
Text(0.5, 1.1, 'Facebook OHLC Boxplots by Volume Traded')
```




We can use this to see the distribution of magnitudes across the different measurement methods for earthquakes:

[44]:
```
quakes[['mag', 'magType']].groupby('magType').boxplot(
    figsize=(15, 8), subplots=False
)

plt.title('Earthquake Magnitude Boxplots by magType')
plt.ylabel('magnitude') # label the y-axis (discussed in chapter 6)
```

[44]:
```
Text(0, 0.5, 'magnitude')
```

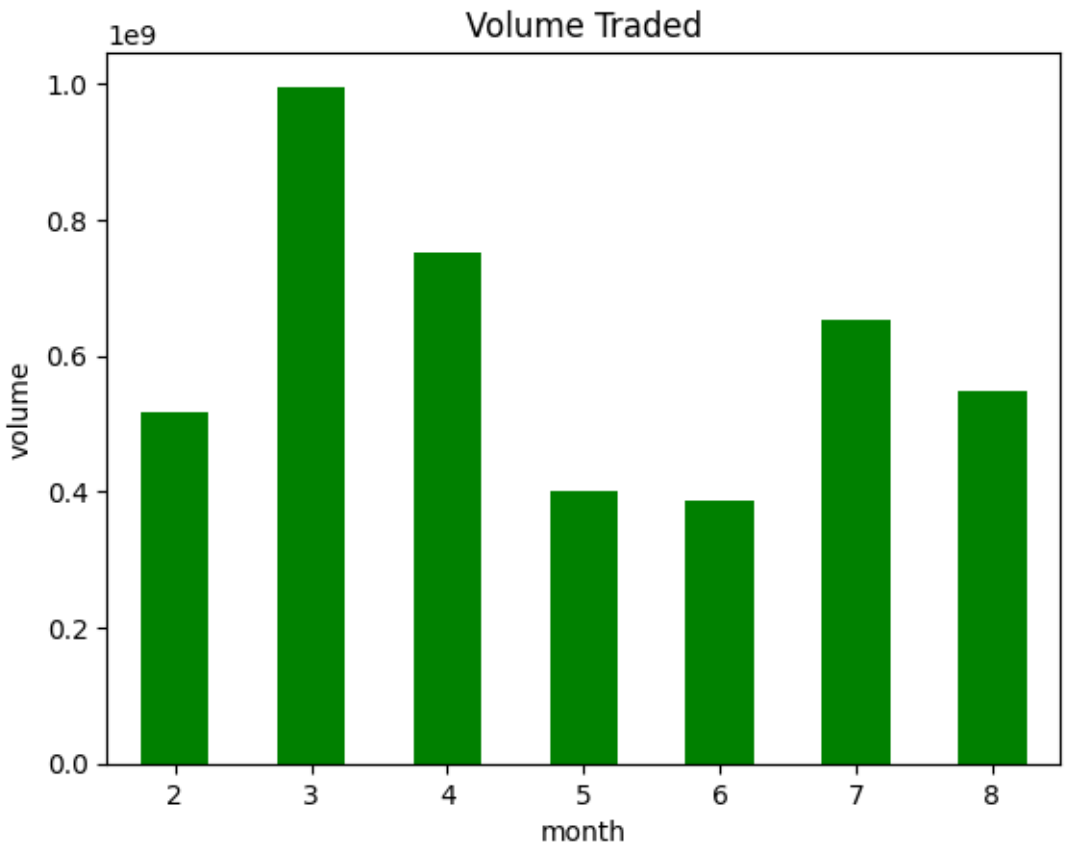


### 1.1.28 Counts and Frequencies

**Bar charts** With pandas, we have the option of using the kind argument or using plot.() . Let's use plot.bar() here to show the evolution of monthly volume traded in Facebook stock over time:

[45]:
```
fb['2018-02':'2018-08'].assign(
    month=lambda x: x.index.month
).groupby('month').sum().volume.plot.bar(
    color='green', rot=0, title='Volume Traded'
  )

plt.ylabel('volume') # label the y-axis (discussed in chapter 6)
```

[45]:
```
Text(0, 0.5, 'volume')
```

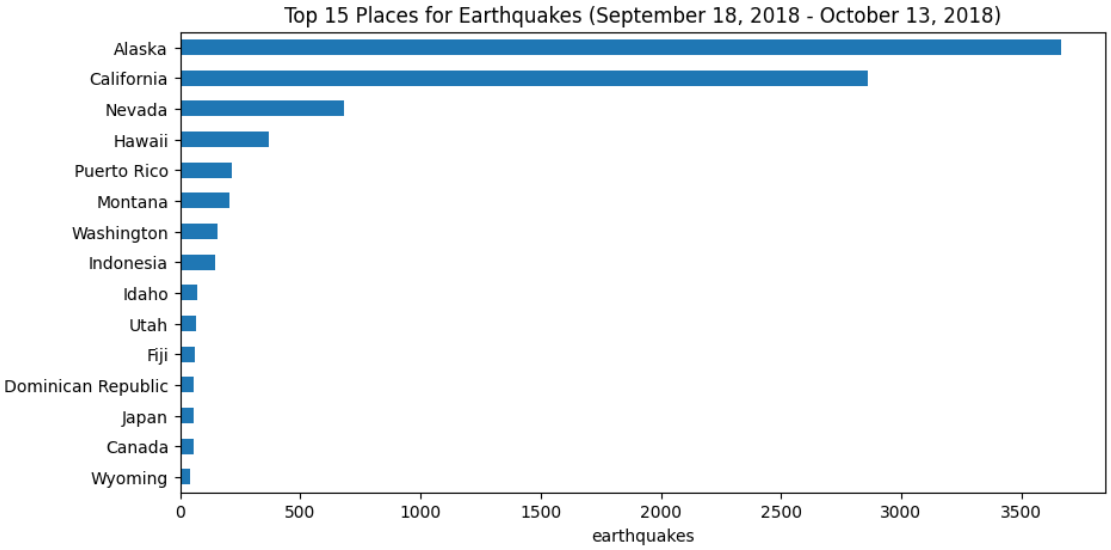



We can also change the orientation of the bars. Passing kind='barh' gives us horizontal bars instead of vertical ones. Let's use this to look at the top 15 places for earthquakes in our data:

[46]:
```
quakes.parsed_place.value_counts().iloc[14::-1,].plot(
    kind='barh', figsize=(10, 5),
    title='Top 15 Places for Earthquakes '\
```

```
          '(September 18, 2018 - October 13, 2018)'
)

plt.xlabel('earthquakes') # label the x-axis (discussed in chapter 6)
```

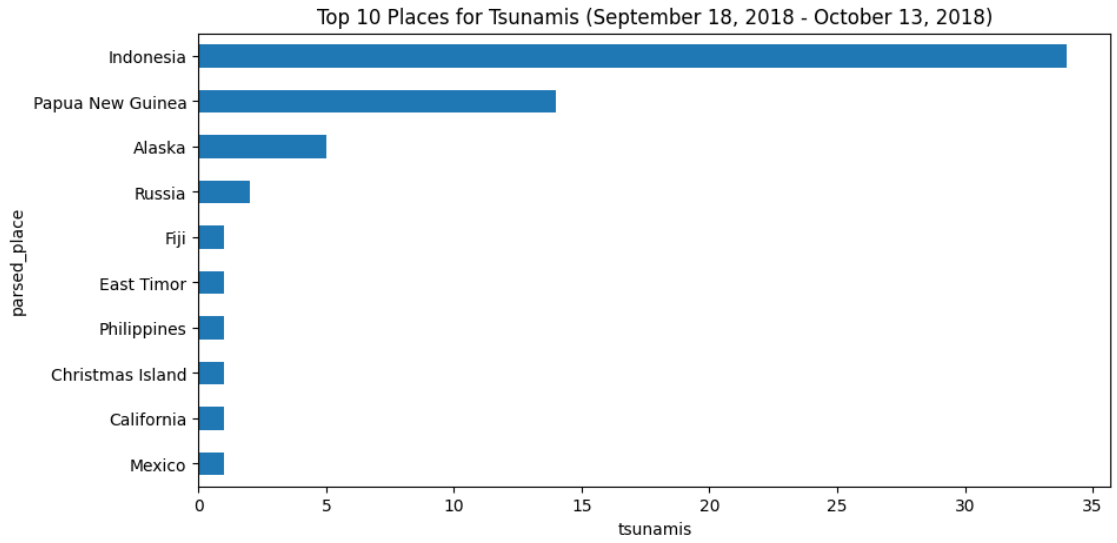
[46]:
```
Text(0.5, 0, 'earthquakes')
```




We also have data on whether earthquakes were accompanied by tsunamis. Let's see what the top places for tsunamis are:

[47]:
```
quakes.groupby('parsed_place').tsunami.sum().sort_values().iloc[-10::,].plot(
    kind='barh', figsize=(10, 5),
    title='Top 10 Places for Tsunamis '\
          '(September 18, 2018 - October 13, 2018)'
)

plt.xlabel('tsunamis') # label the x-axis (discussed in chapter 6)
```

[47]:
```
Text(0.5, 0, 'tsunamis')
```



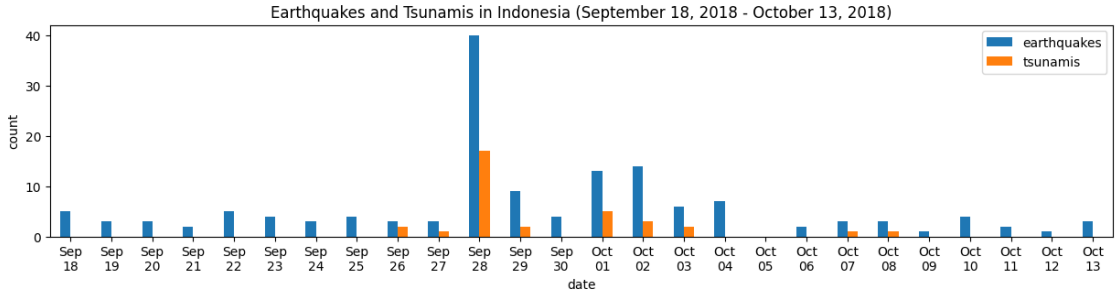

Seeing that Indonesia is the top place for tsunamis during the time period we are looking at, we may want to look how many earthquakes and tsunamis Indonesia gets on a daily basis. We could show this as a line plot or with bars; since this section is about bars, we will use bars here

[48]:
```
indonesia_quakes = quakes.query('parsed_place == "Indonesia"').assign(
    time=lambda x: pd.to_datetime(x.time, unit='ms'),
    earthquake=1
).set_index('time').resample('1D').sum()

indonesia_quakes.index = indonesia_quakes.index.strftime('%b\n%d')

indonesia_quakes.plot(
    y=['earthquake', 'tsunami'],
    kind='bar',
    figsize=(15, 3),
    rot=0,
    label=['earthquakes', 'tsunamis'],
    title='Earthquakes and Tsunamis in Indonesia '\
          '(September 18, 2018 - October 13, 2018)'
)

# label the axes (discussed in chapter 6)
plt.xlabel('date')
plt.ylabel('count')
```

```
<ipython-input-48-12e8b941ade3>:4: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
```

```
).set_index('time').resample('1D').sum()
```

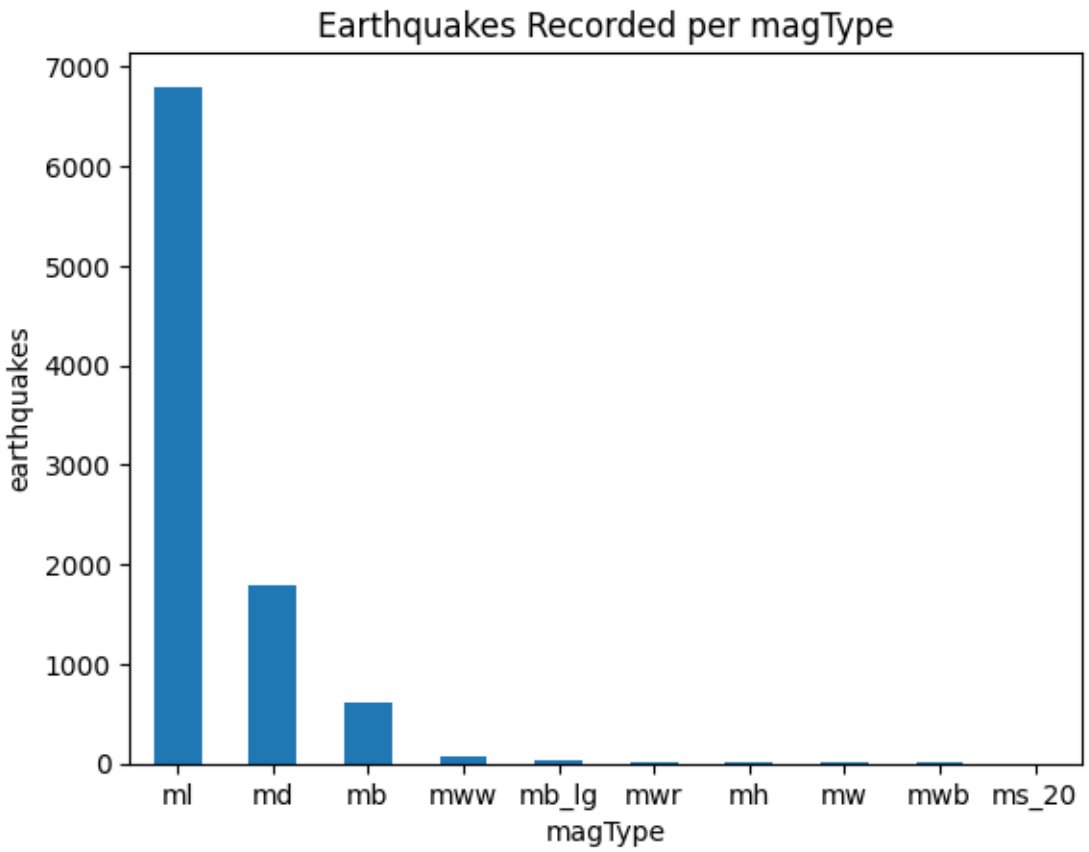
[48]:
```
Text(0, 0.5, 'count')
```




Using the kind arugment for vertical bars when the labels for each bar are shorter:

[49]:
```
quakes.magType.value_counts().plot(
    kind='bar',
    title='Earthquakes Recorded per magType',
    rot=0
)

# label the axes (discussed in chapter 6)
plt.xlabel('magType')
plt.ylabel('earthquakes')
```

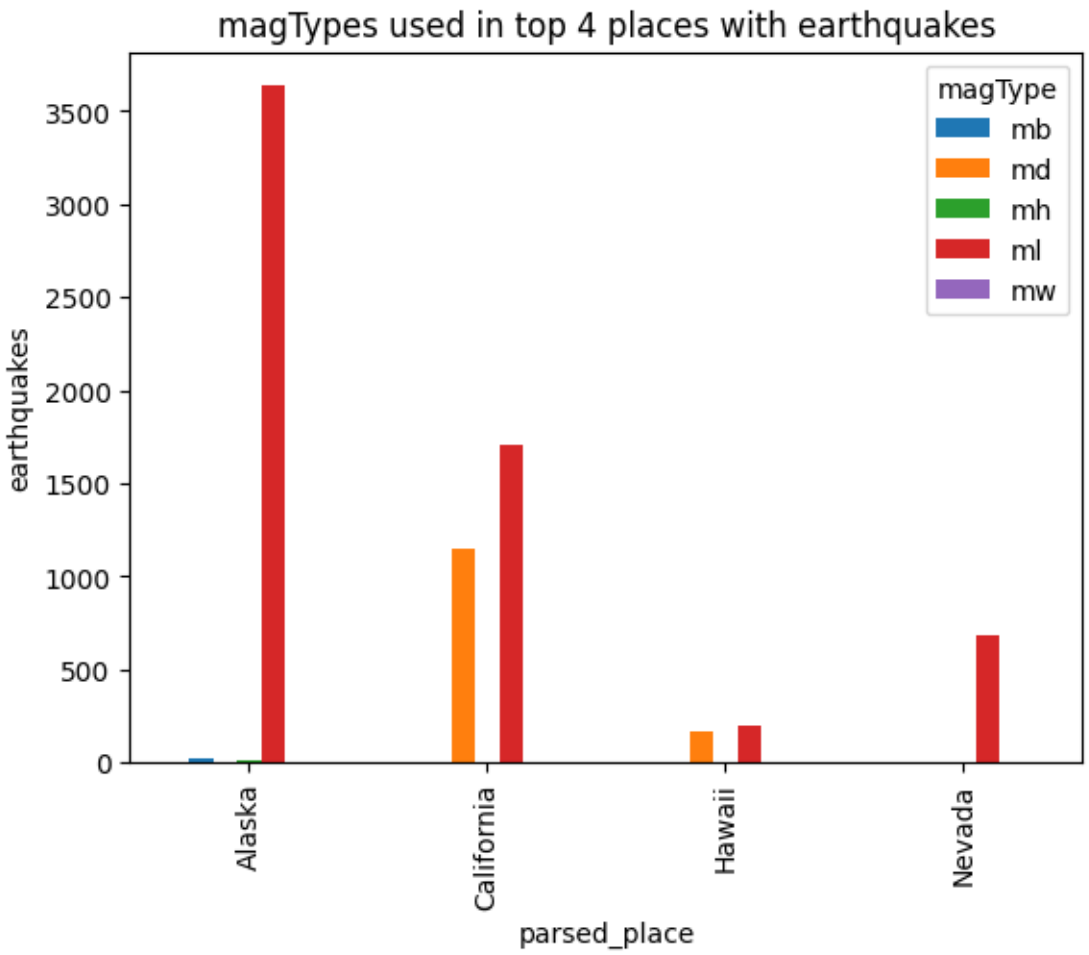
[49]:
```
Text(0, 0.5, 'earthquakes')
```




Top 4 places with earthquakes:

[50]:
```
quakes[
    quakes.parsed_place.isin(['California','Alaska','Nevada','Hawaii'])
].groupby(['parsed_place', 'magType']).mag.count().unstack().plot.bar(
    title='magTypes used in top 4 places with earthquakes'
)
plt.ylabel('earthquakes') # label the axes (discussed in chapter 6)
```

[50]:
```
Text(0, 0.5, 'earthquakes')
```

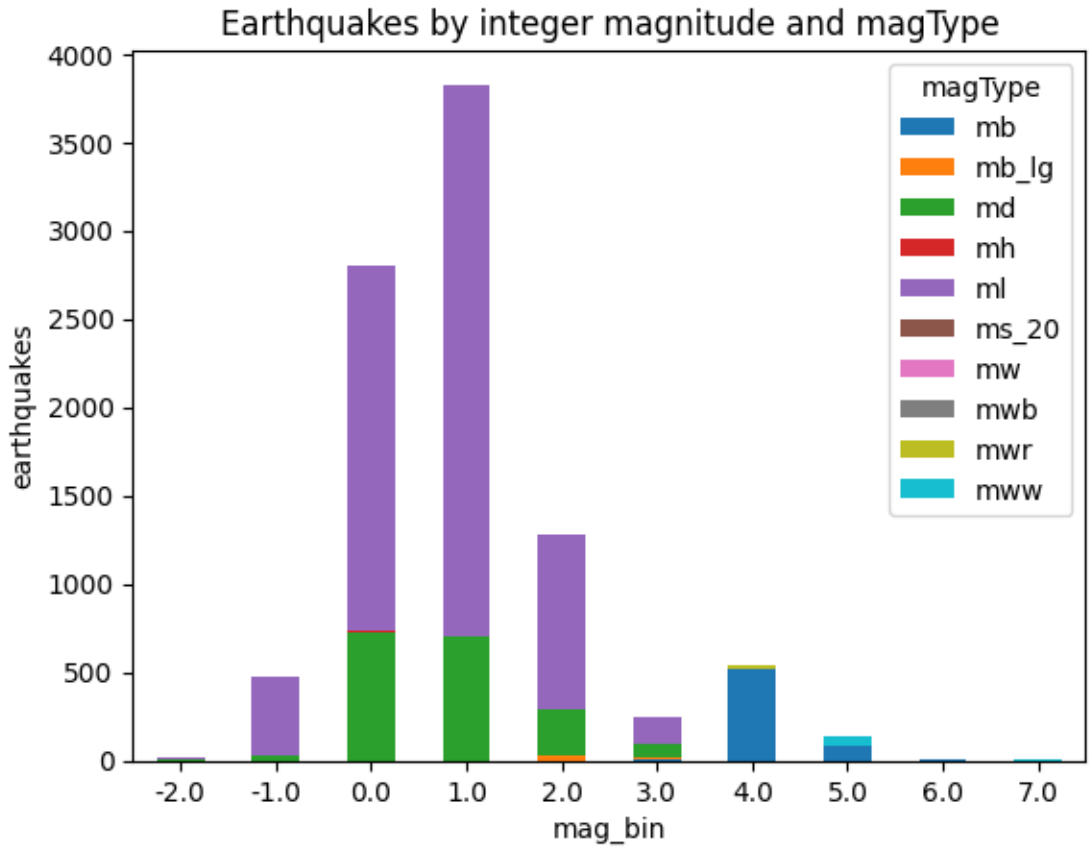



### 1.1.29 Stacked bar chart

[51]:
```
pivot = quakes.assign(
    mag_bin=lambda x: np.floor(x.mag)
).pivot_table(
    index='mag_bin',
    columns='magType',
    values='mag',
    aggfunc='count'
)

pivot.plot.bar(
    stacked=True,
    rot=0,
    title='Earthquakes by integer magnitude and magType'
)
```

```
plt.ylabel('earthquakes') # label the axes (discussed in chapter 6)
```

[51]:
```
Text(0, 0.5, 'earthquakes')
```

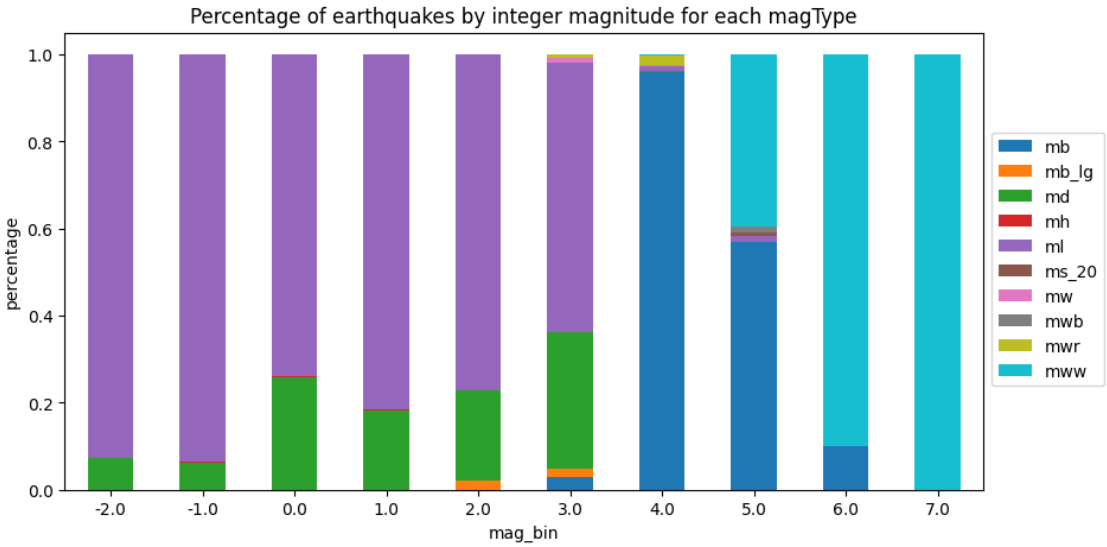



### 1.1.30 Normalized stacked bars

Plot the percentages to be better able to see the different magTypes.

[52]:
```
normalized_pivot = pivot.fillna(0).apply(lambda x: x/x.sum(), axis=1)
ax = normalized_pivot.plot.bar(
    stacked=True, rot=0, figsize=(10, 5),
    title='Percentage of earthquakes by integer magnitude for each magType'
)

ax.legend(bbox_to_anchor=(1, 0.8)) # move legend to the right of the plot
plt.ylabel('percentage') # label the axes (discussed in chapter 6)
```

[52]:
```
Text(0, 0.5, 'percentage')
```




### 1.1.31 Comments and Insights:

From the activity, I was introduced to pandas' ability to plot its data and the various plots that we can create using both pandas and matplotlib. The plot or graph type that we need to use will depend on the type of data and how we what to present the data. Line graphs usually present data over a period of time, while bar graphs represent individual portions of data for different subgroups, and a lot of these plots can be used to compare 2 variables. Being able to present data using the proper graph and visuals is important as it makes our interpretations and analyses better and more accurate.

### 1.1.32 9.3 Pandas Plotting Subpackage

Pandas provides some extra plotting functions for a few select plot types.

### 1.1.33 About the Data

In this notebook, we will be working with Facebook's stock price throughout 2018.

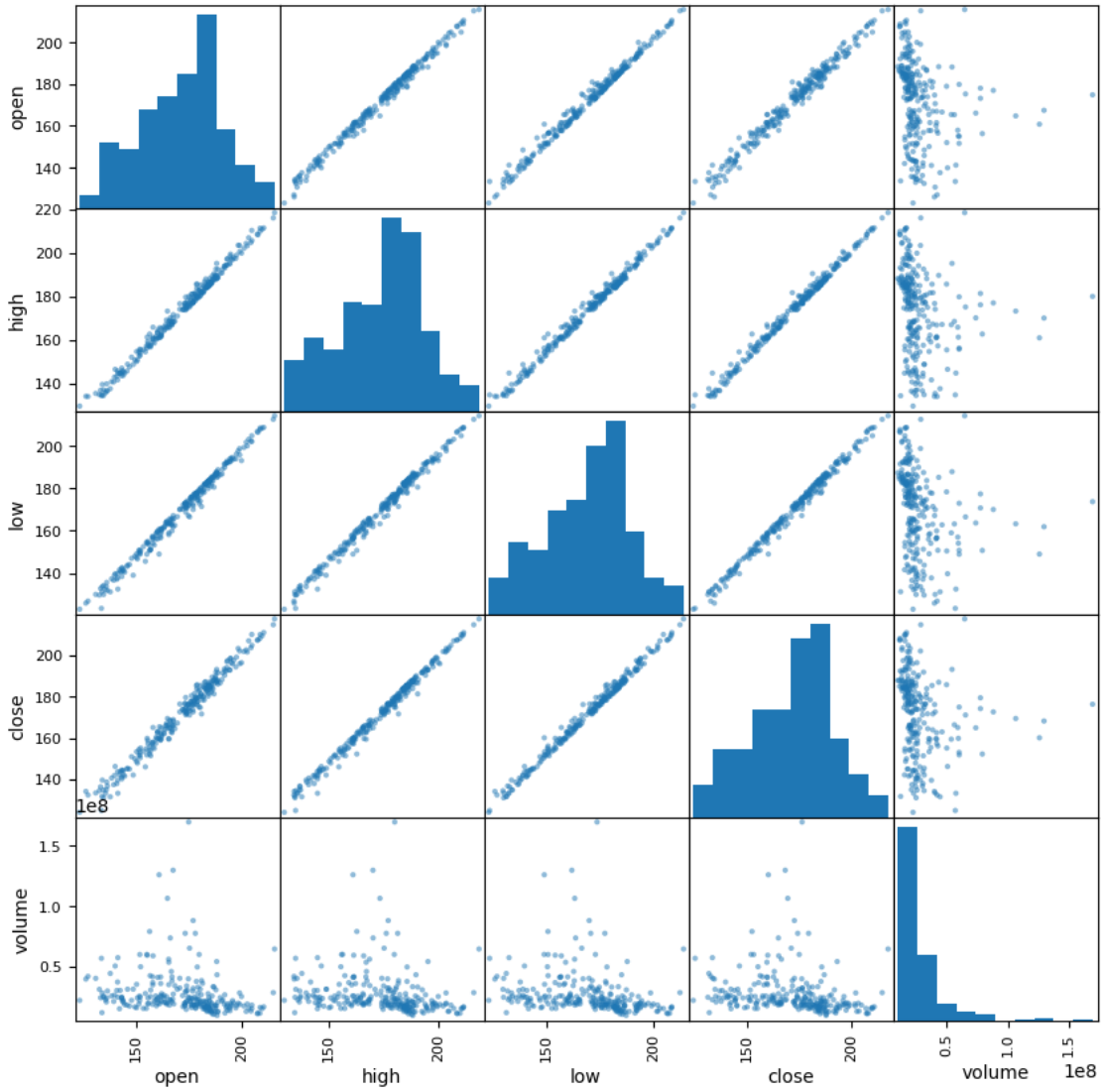
### 1.1.34 Setup

[53]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

fb = pd.read_csv(
    '/content/fb_stock_prices_2018.csv', index_col='date', parse_dates=True
)
```

### 1.1.35 Scatter matrix

[54]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(fb, figsize=(10, 10))
```

[54]:
```
array([[<Axes: xlabel='open', ylabel='open'>,
        <Axes: xlabel='high', ylabel='open'>,
        <Axes: xlabel='low', ylabel='open'>,
        <Axes: xlabel='close', ylabel='open'>,
        <Axes: xlabel='volume', ylabel='open'>],
       [<Axes: xlabel='open', ylabel='high'>,
        <Axes: xlabel='high', ylabel='high'>,
        <Axes: xlabel='low', ylabel='high'>,
        <Axes: xlabel='close', ylabel='high'>,
        <Axes: xlabel='volume', ylabel='high'>],
       [<Axes: xlabel='open', ylabel='low'>,
        <Axes: xlabel='high', ylabel='low'>,
        <Axes: xlabel='low', ylabel='low'>,
        <Axes: xlabel='close', ylabel='low'>,
        <Axes: xlabel='volume', ylabel='low'>],
       [<Axes: xlabel='open', ylabel='close'>,
        <Axes: xlabel='high', ylabel='close'>,
        <Axes: xlabel='low', ylabel='close'>,
        <Axes: xlabel='close', ylabel='close'>,
        <Axes: xlabel='volume', ylabel='close'>],
       [<Axes: xlabel='open', ylabel='volume'>,
        <Axes: xlabel='high', ylabel='volume'>,
        <Axes: xlabel='low', ylabel='volume'>,
        <Axes: xlabel='close', ylabel='volume'>,
        <Axes: xlabel='volume', ylabel='volume'>]], dtype=object)
```

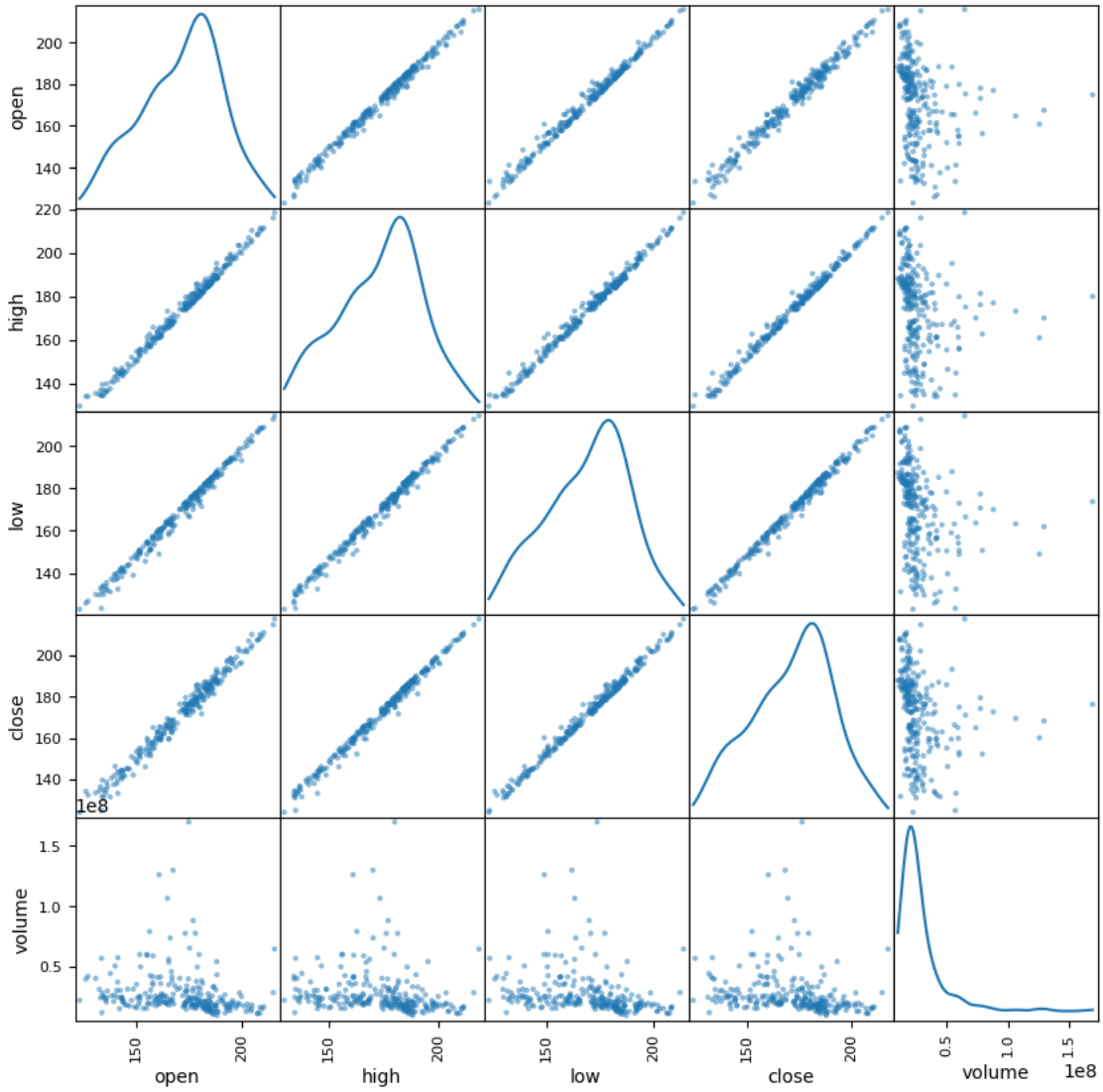



Changing the diagonal from histograms to KDE:

[55]:
```
scatter_matrix(fb, figsize=(10, 10), diagonal='kde')
```

[55]:
```
array([[<Axes: xlabel='open', ylabel='open'>,
        <Axes: xlabel='high', ylabel='open'>,
        <Axes: xlabel='low', ylabel='open'>,
        <Axes: xlabel='close', ylabel='open'>,
        <Axes: xlabel='volume', ylabel='open'>],
       [<Axes: xlabel='open', ylabel='high'>,
        <Axes: xlabel='high', ylabel='high'>,
        <Axes: xlabel='low', ylabel='high'>,
        <Axes: xlabel='close', ylabel='high'>,
```

```
 <Axes: xlabel='volume', ylabel='high'>],
[<Axes: xlabel='open', ylabel='low'>,
 <Axes: xlabel='high', ylabel='low'>,
 <Axes: xlabel='low', ylabel='low'>,
 <Axes: xlabel='close', ylabel='low'>,
 <Axes: xlabel='volume', ylabel='low'>],
[<Axes: xlabel='open', ylabel='close'>,
 <Axes: xlabel='high', ylabel='close'>,
 <Axes: xlabel='low', ylabel='close'>,
 <Axes: xlabel='close', ylabel='close'>,
 <Axes: xlabel='volume', ylabel='close'>],
[<Axes: xlabel='open', ylabel='volume'>,
 <Axes: xlabel='high', ylabel='volume'>,
 <Axes: xlabel='low', ylabel='volume'>,
 <Axes: xlabel='close', ylabel='volume'>,
 <Axes: xlabel='volume', ylabel='volume'>]], dtype=object)
```

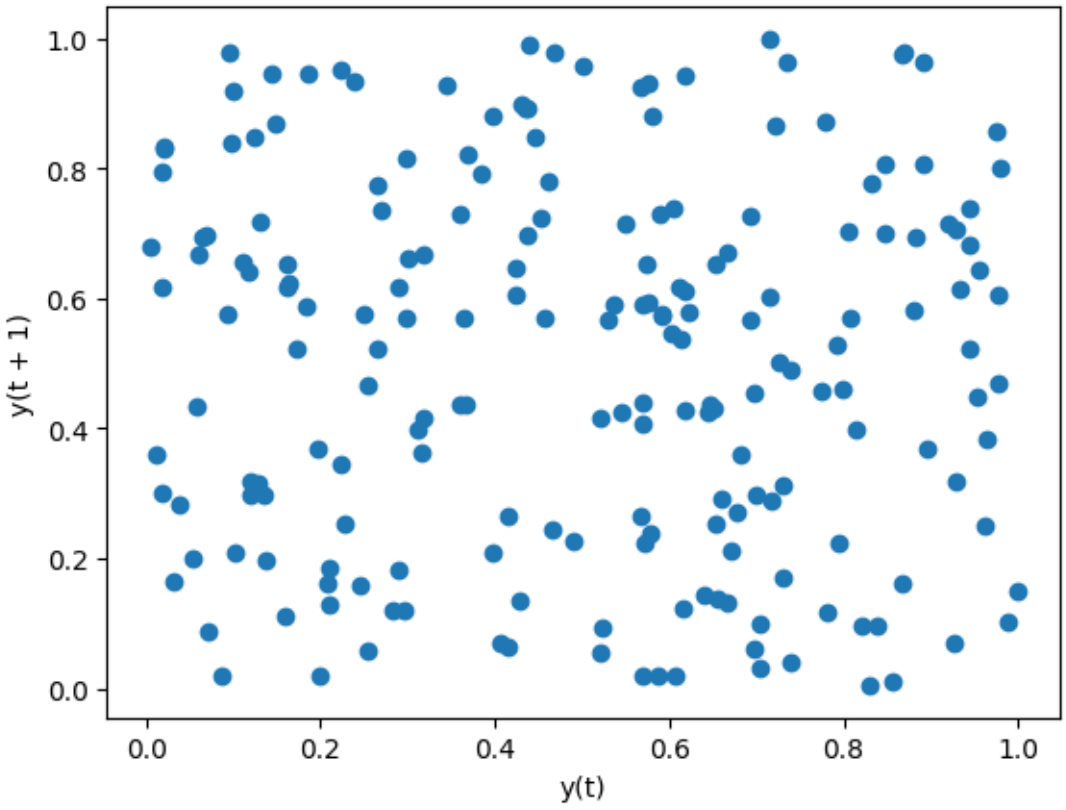



### 1.1.36 Lag plot

Lag plots let us see how the variable correlations with past observations of itself. Random data has no pattern:

[56]:
```
from pandas.plotting import lag_plot
np.random.seed(0) # make this repeatable
lag_plot(pd.Series(np.random.random(size=200)))
```

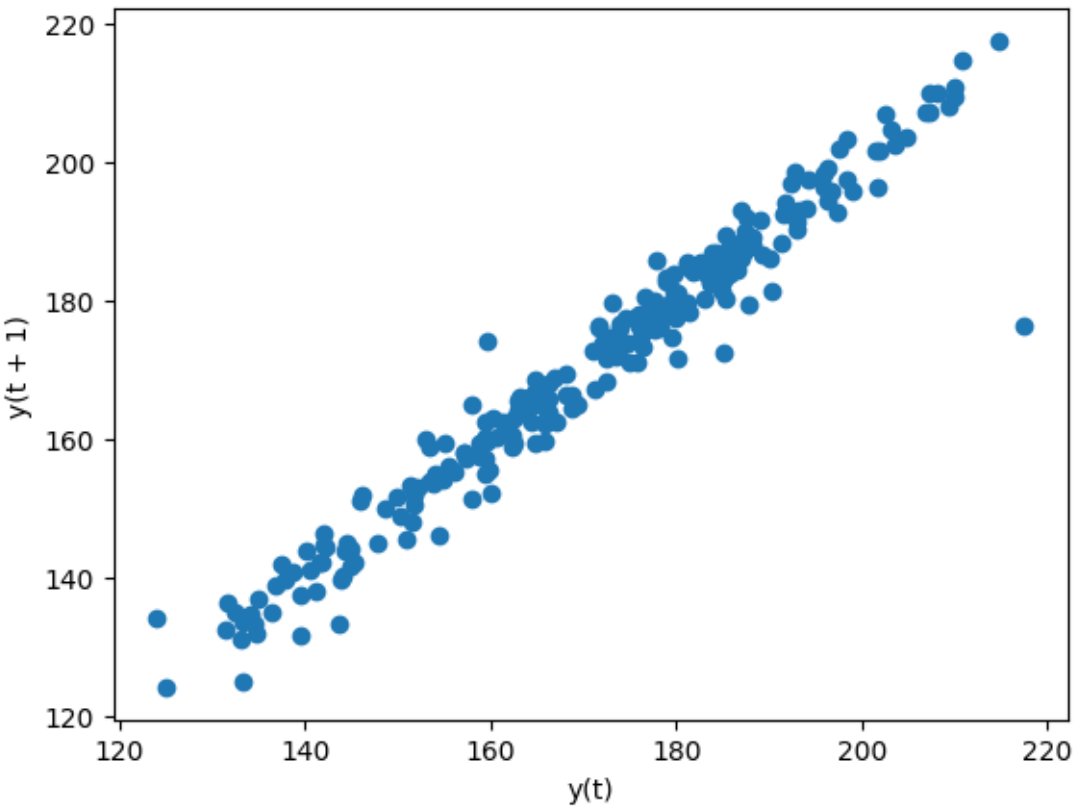
[56]:
```
<Axes: xlabel='y(t)', ylabel='y(t + 1)'>
```




Data with some level of correlation to itself (autocorrelation) may have patterns. Stock prices are highly auto-correlated:

[57]:
```
lag_plot(fb.close)
```

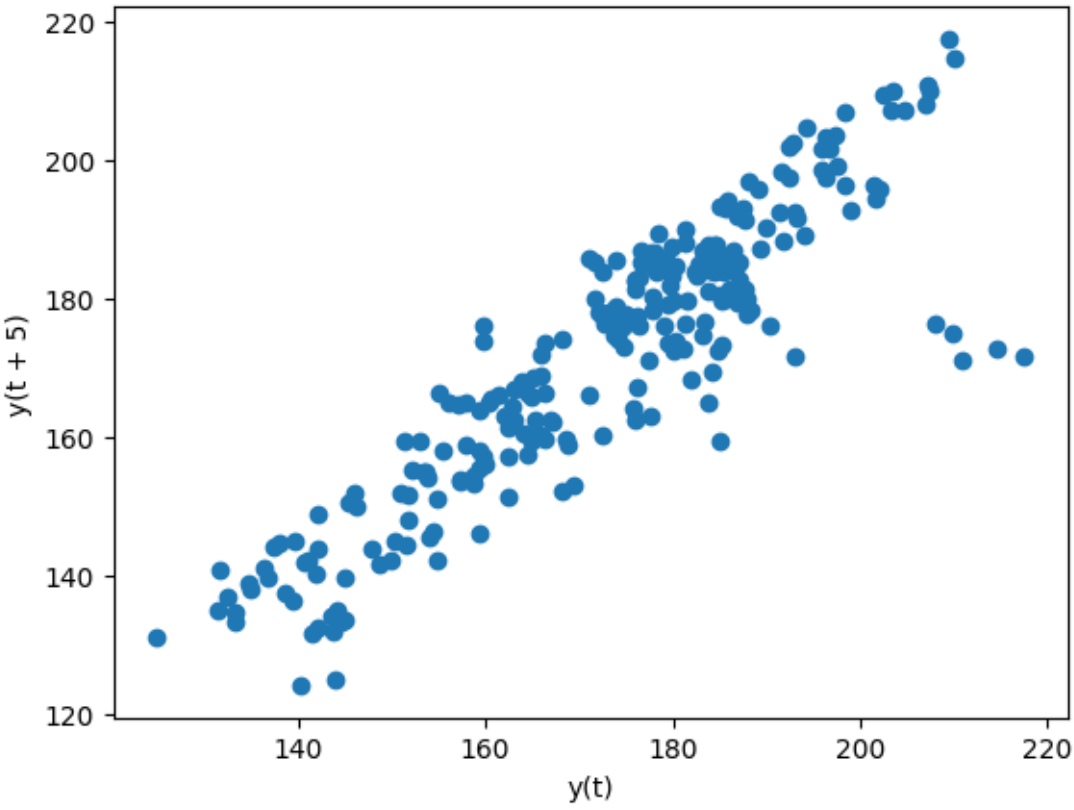
[57]:
```
<Axes: xlabel='y(t)', ylabel='y(t + 1)'>
```




The default lag is 1, but we can alter this with the lag parameter. Let's look at a 5 day lag (a week of trading activity):

[58]:
```
lag_plot(fb.close, lag=5)
```

[58]:
```
<Axes: xlabel='y(t)', ylabel='y(t + 5)'>
```

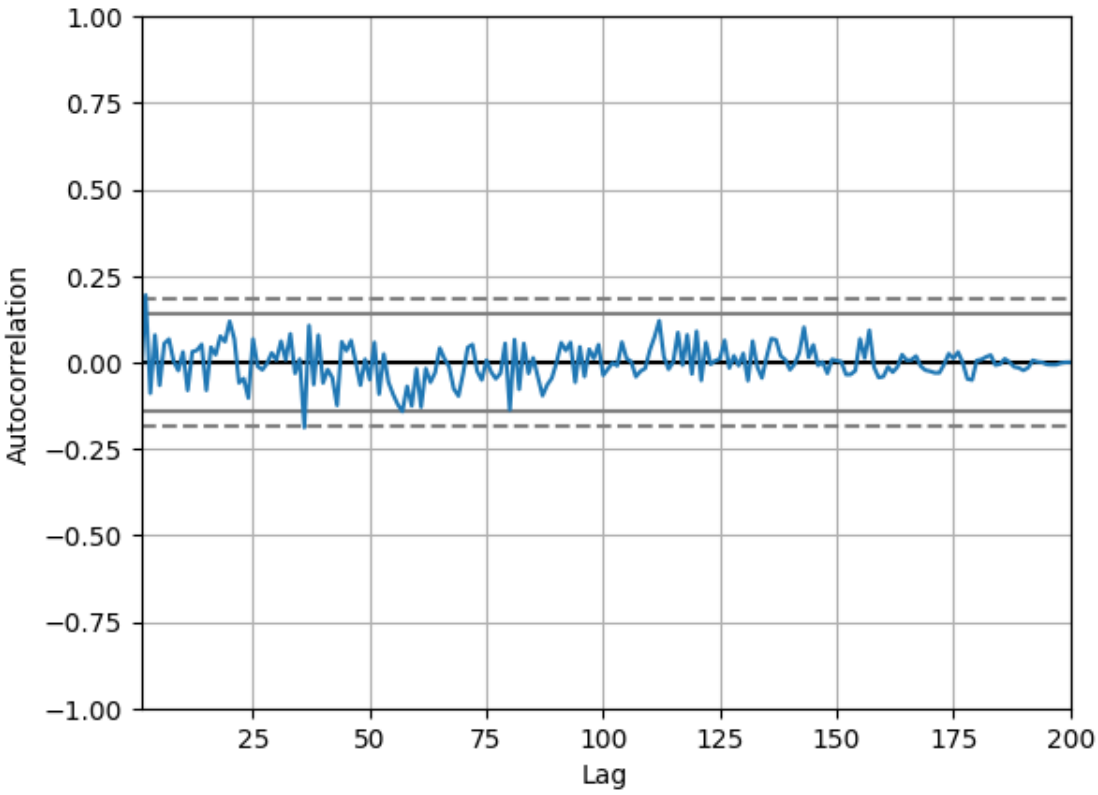



### 1.1.37 Autocorrelation plots

We can use the autocorrelation plot to see if this relationship may be meaningful or just noise. Random data will not have any significant autocorrelation (it stays within the bounds below):

[59]:
```
from pandas.plotting import autocorrelation_plot
np.random.seed(0) # make this repeatable
autocorrelation_plot(pd.Series(np.random.random(size=200)))
```

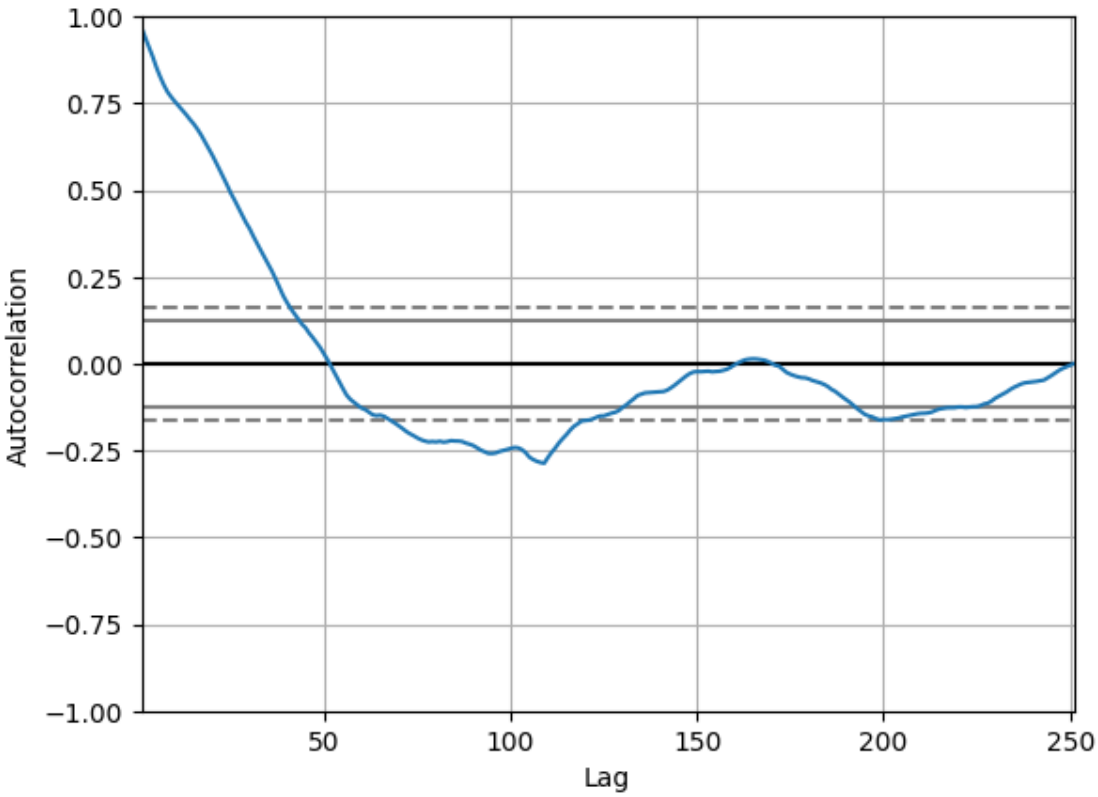
[59]:
```
<Axes: xlabel='Lag', ylabel='Autocorrelation'>
```




Stock data, on the other hand, does have significant autocorrelation:

[60]:
```
autocorrelation_plot(fb.close)
```

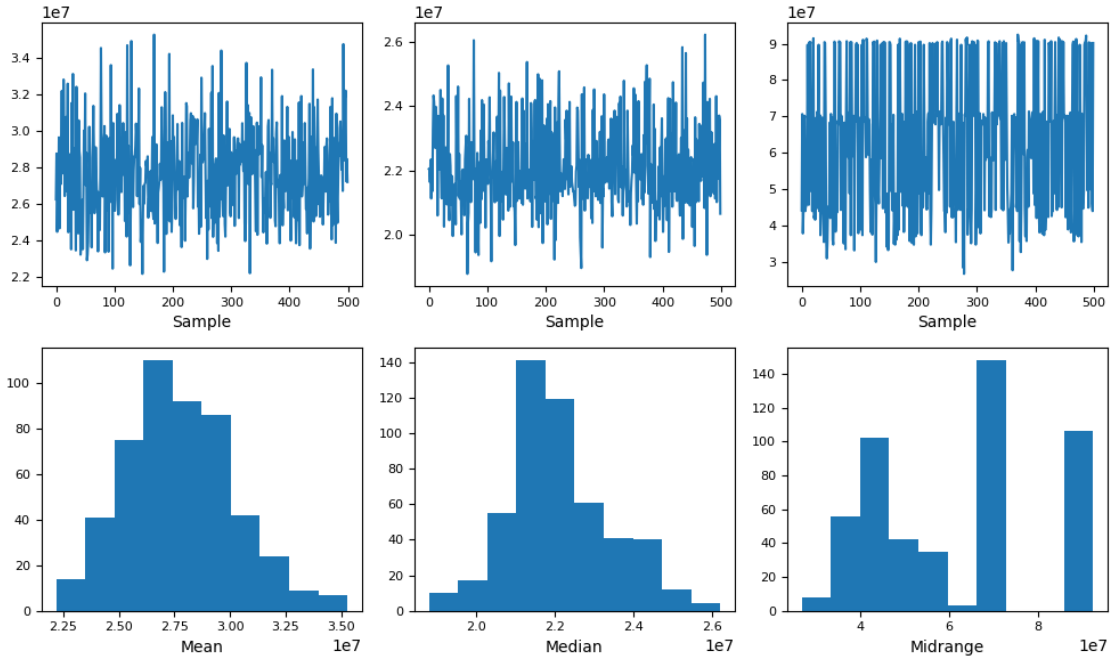
[60]:
```
<Axes: xlabel='Lag', ylabel='Autocorrelation'>
```




### 1.1.38 Bootstrap plot

This plot helps us understand the uncertainty in our summary statistics:

[61]:
```
from pandas.plotting import bootstrap_plot
fig = bootstrap_plot(fb.volume, fig=plt.figure(figsize=(10, 6)))
```




### 1.1.39 Comments and Insights

From this activity, I was given some more complex and additional plots which can also be used for data, specifically for multiple statistics for a given data. These plots are more complex and specific, which means that these plots are not used all the time and are only applicable for special cases and sets of data, as well as different statistics.

## 1.2 Supplementary Activity:

Using the CSV files provided and what we have learned so far in this module complete the following exercises:

1. Plot the rolling 20-day minimum of the Facebook closing price with the pandas plot() method.

[62]:
```
fb = pd.read_csv('/content/fb_stock_prices_2018.csv', index_col='date',␣
↪parse_dates=True)

# Creating 20D Rolling Close Minimum column
fb['rolling_close'] = fb['close'].rolling('20D').min()
fb.head()
```

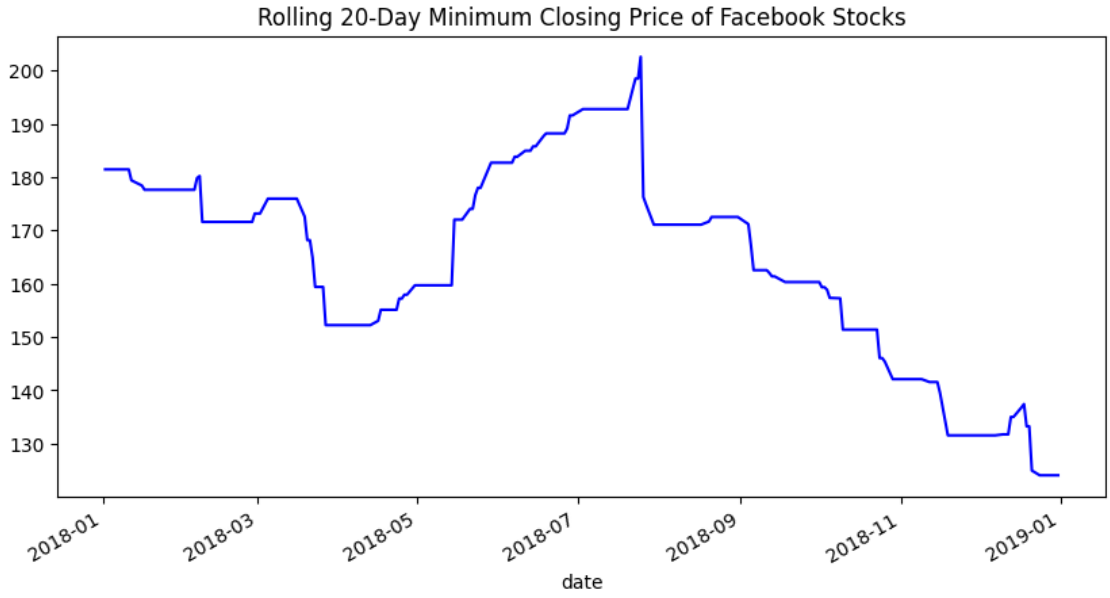
[62]:

| date | open | high | low | close | volume | rolling_close |
|---|---|---|---|---|---|---|
| 2018-01-02 | 177.68 | 181.58 | 177.5500 | 181.42 | 18151903 | 181.42 |
| 2018-01-03 | 181.88 | 184.78 | 181.3300 | 184.67 | 16886563 | 181.42 |
| 2018-01-04 | 184.90 | 186.21 | 184.0996 | 184.33 | 13880896 | 181.42 |

```
2018-01-05  185.59  186.90  184.9300  186.85  13574535         181.42
2018-01-08  187.20  188.90  186.3300  188.28  17994726         181.42
```

[63]:
```
# Plotting data (line graph)
fb.plot(
    kind='line',
    y='rolling_close',
    figsize=(10, 5),
    style='b-',
    legend=False,
    title='Rolling 20-Day Minimum Closing Price of Facebook Stocks'
)
```

[63]:
```
<Axes: title={'center': 'Rolling 20-Day Minimum Closing Price of Facebook
Stocks'}, xlabel='date'>
```




2. Create a histogram and KDE of the change from open to close in the price of Facebook stock.

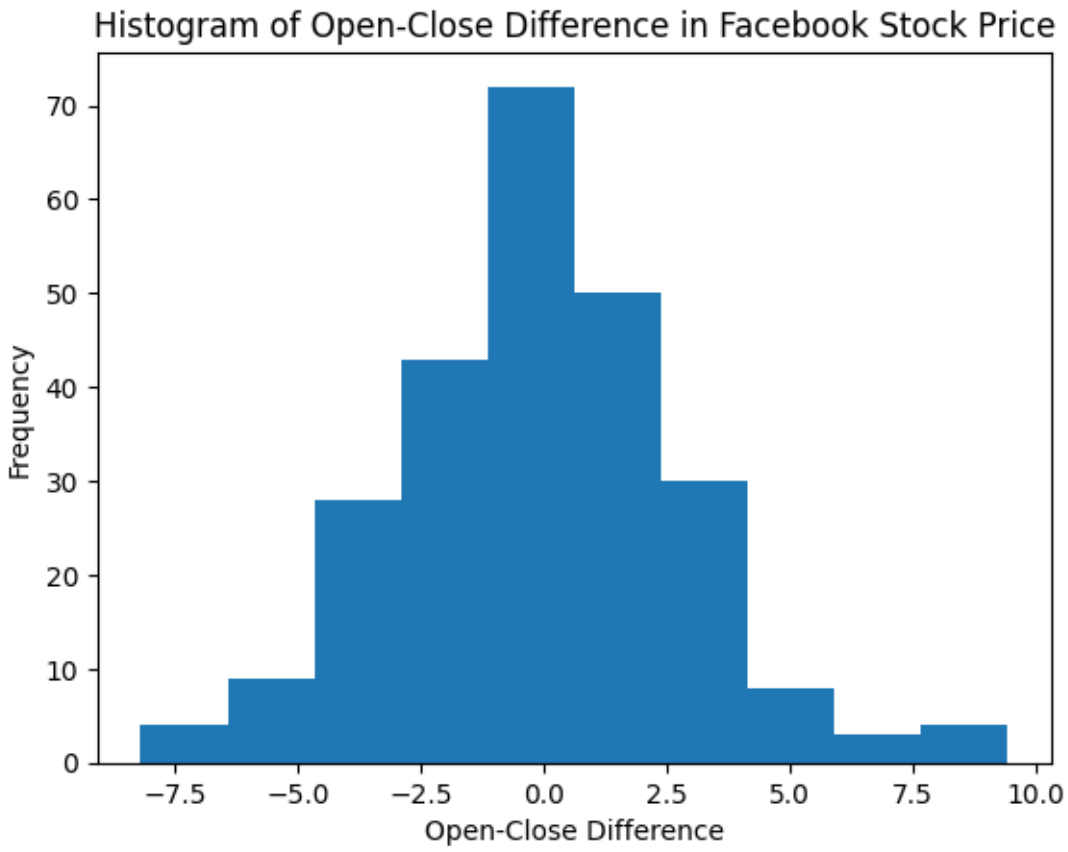
[64]:
```
# Create Open-Close Column
fb['open_close'] = fb['open'] - fb['close']
fb.head()
```

[64]:
```
              open    high       low   close    volume  rolling_close  \
date
2018-01-02  177.68  181.58  177.5500  181.42  18151903         181.42
2018-01-03  181.88  184.78  181.3300  184.67  16886563         181.42
2018-01-04  184.90  186.21  184.0996  184.33  13880896         181.42
```

```
2018-01-05  185.59  186.90  184.9300  186.85  13574535          181.42
2018-01-08  187.20  188.90  186.3300  188.28  17994726          181.42

            open_close
date
2018-01-02       -3.74
2018-01-03       -2.79
2018-01-04        0.57
2018-01-05       -1.26
2018-01-08       -1.08
```

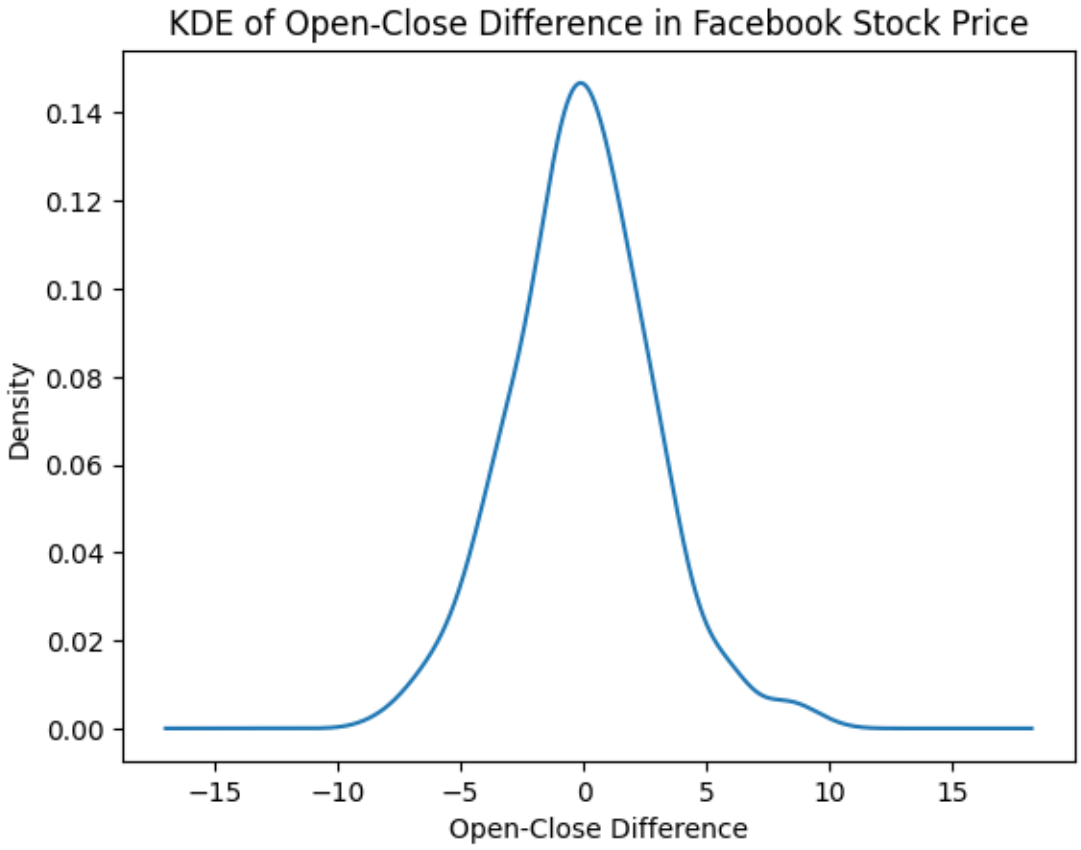
[65]:
```
# Creating Histogram
fb.open_close.plot(
    kind='hist',
    title='Histogram of Open-Close Difference in Facebook Stock Price'
)

plt.xlabel('Open-Close Difference')
```

[65]:
```
Text(0.5, 0, 'Open-Close Difference')
```



[66]:
```
# Creating KDE
fb.open_close.plot(
    kind='kde',
    title='KDE of Open-Close Difference in Facebook Stock Price'
)

plt.xlabel('Open-Close Difference')
```

[66]:
```
Text(0.5, 0, 'Open-Close Difference')
```



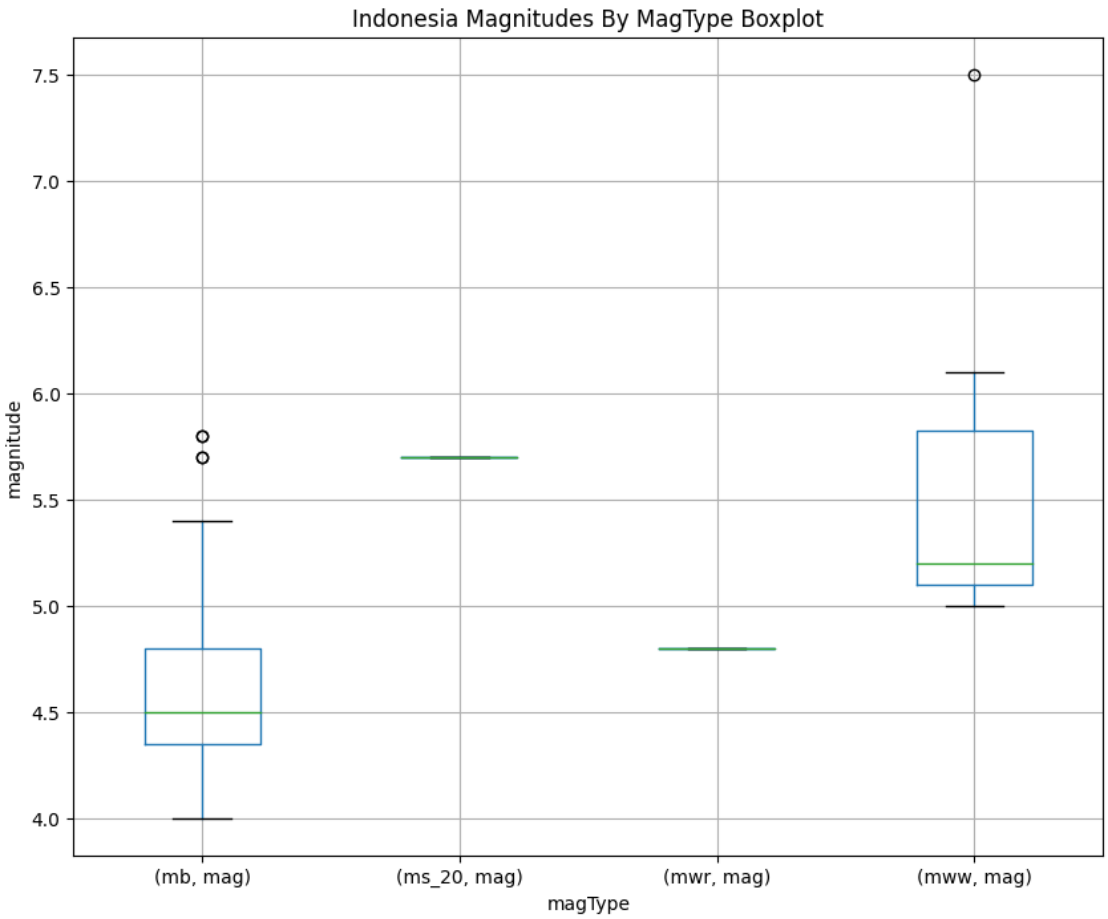

3. Using the earthquake data, create box plots for the magnitudes of each magType used in Indonesia.

[67]:
```
# Setup
earthquake = pd.read_csv('/content/earthquakes-1.csv')
```

[68]:
```
# Obtaining the Indonesia data using query()
mag_indo = earthquake.query("parsed_place == 'Indonesia'")

mag_indo[["mag"]].groupby(mag_indo["magType"]).boxplot(
```

```
    figsize=[10,8],
    subplots=False,
)

plt.title("Indonesia Magnitudes By MagType Boxplot")
plt.xlabel("magType")
plt.ylabel("magnitude")
```

[68]:
```
Text(0, 0.5, 'magnitude')
```

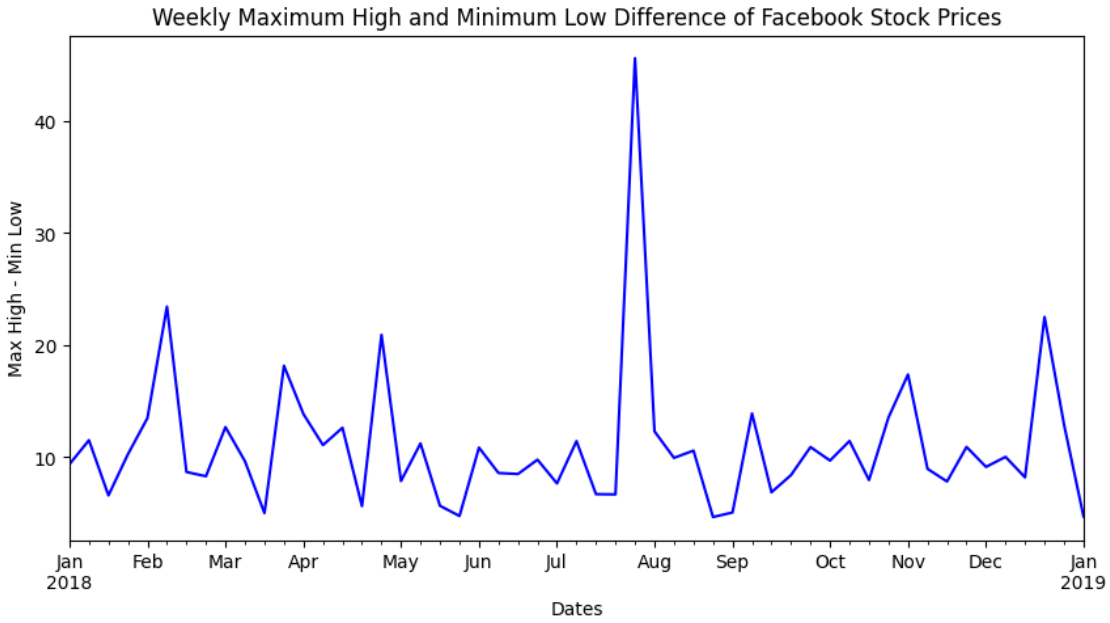



4. Make a line plot of the difference between the weekly maximum high price and the weekly minimum low price for Facebook. This should be a single line.

[69]:
```
# creating high-low difference per week column

open_close_rolling = fb.resample('W').apply(
    lambda x: x['high'].max() - x["low"].min()
)
```

[70]:
```
# plotting line graph
open_close_rolling.plot(
    kind='line',
    y='open-close rolling',
    figsize=(10, 5),
    style='b-',
    legend=False,
    title='Weekly Maximum High and Minimum Low Difference of Facebook Stock␣
 ↪Prices'
)

plt.xlabel("Dates")
plt.ylabel("Max High - Min Low")
```

[70]:
```
Text(0, 0.5, 'Max High - Min Low')
```




5. Using matplotlib and pandas, create two subplots side-by-side showing the effect that after-hours trading has had on Facebook's stock price:

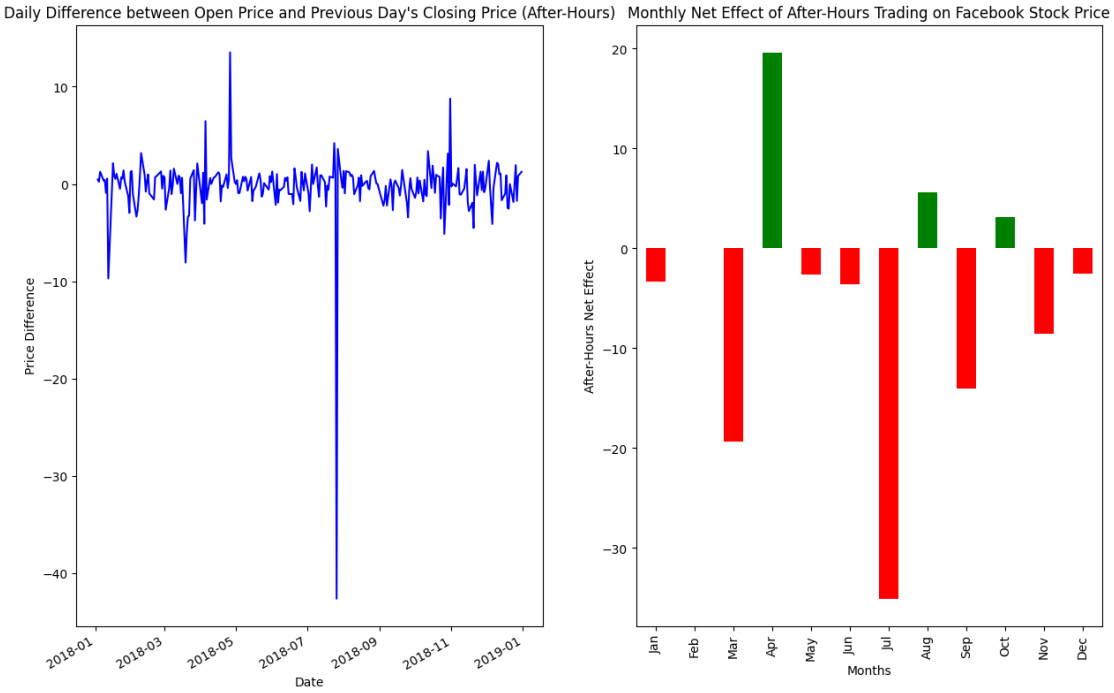
- The first subplot will contain a line plot of the daily difference between that day's opening price and the prior day's closing price (be sure to review the Time series section of Aggregating Pandas DataFrames for an easy way to do this).
- The second subplot will be a bar plot showing the net effect this had monthly, using resample().
- Bonus #1: Color the bars according to whether they are gains in the stock price (green) or drops in the stock price (red).

- Bonus #2: Modify the x-axis of the bar plot to show the threeletter abbreviation for the month.

[80]:
```
# creating data for first subplot
fb_1 = fb.assign(
    prior_close=lambda x: x.close.shift(),
    after_hours_change_in_price=lambda x: x.open - x.prior_close,
    abs_change=lambda x: x.after_hours_change_in_price.abs()
)

fig, axes = plt.subplots(1, 2, figsize=[15,10])
fb_1.after_hours_change_in_price.plot(
    kind='line',
    ax=axes[0],
    legend=False,
    style='b-',
    title="Daily Difference between Open Price and Previous Day's Closing Price␣
 ↪(After-Hours)"
)

net_effect = fb_1.after_hours_change_in_price.resample('M').sum()
net_effect.plot(
    kind='bar',
    ax=axes[1],
    legend=False,
    color = ['green' if value >= 0 else 'red' for value in fb_1.
 ↪after_hours_change_in_price.resample('M').sum()],
    title="Monthly Net Effect of After-Hours Trading on Facebook Stock Price"
)

axes[0].set_xlabel('Date')
axes[0].set_ylabel('Price Difference')

axes[1].set_xlabel('Months')
axes[1].set_ylabel('After-Hours Net Effect')

axes[1].set_xticklabels(net_effect.index.strftime('%b'))
```

[80]:
```
[Text(0, 0, 'Jan'),
 Text(1, 0, 'Feb'),
 Text(2, 0, 'Mar'),
 Text(3, 0, 'Apr'),
 Text(4, 0, 'May'),
 Text(5, 0, 'Jun'),
 Text(6, 0, 'Jul'),
 Text(7, 0, 'Aug'),
 Text(8, 0, 'Sep'),
```

```
Text(9, 0, 'Oct'),
Text(10, 0, 'Nov'),
Text(11, 0, 'Dec')]
```




## 1.3 Summary / Conclusion

From the activity, I was able to learn about the basics on matplotlib library, as well as plotting data using pandas.plot(). I learned how important it is to understand and know how to plot data because being able to represent data graphically makes it easier for us to analyze and draw conclusions. Creating a graphical plot of the data helps us see the data in a wider perspective because of the visuals compared to just seeing numbers. By learning how to customize and properly arrange a graphical plot of the data, we will be able to represent it properly and accurately. In turn, this will make our analysis more reliable and accurate as well since we are interpreting very well-presented data. Along with this, I also learned how important it is to understand which graph type to use because being able to choose the right graph type for the kind of data means that we can represent and interpret it well and we can show the data for how it should be shown and no important values or statistics would be left out.

[82]:
```
!cp /content/De_Guzman_Hands-On_Activity_9.
↪1_Data_Visualization_using_Pandas_and_Matplotlib.ipynb ./
```

```
cp: cannot stat '/content/De_Guzman_Hands-
On_Activity_9.1_Data_Visualization_using_Pandas_and_Matplotlib.ipynb': No such
file or directory
```