Hands-on Activity 9.1 Data Visualization using Pandas and Matplotlib

Name: John Rome A. Belocora

Section: CPE22S3

9.1 Getting Started with Matplotlib

We need matplotlib.pyplot for plotting.

```
import matplotlib.pyplot as plt
import pandas as pd
```

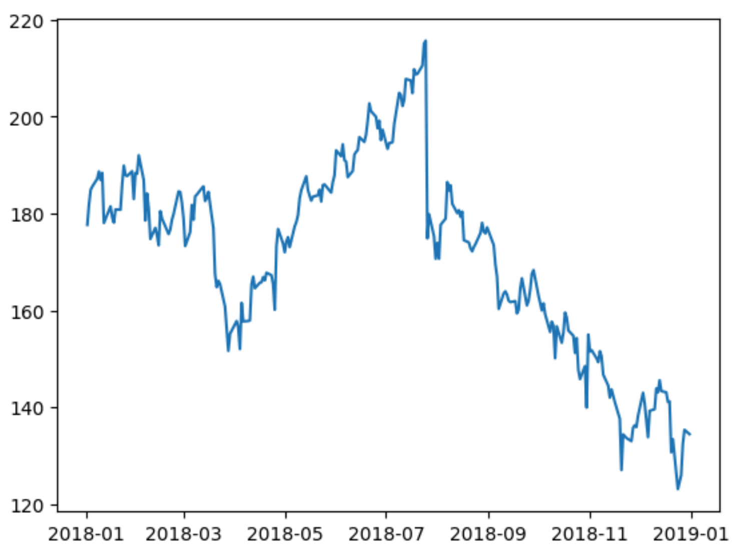
About the Data

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)

Plotting lines

```
fb = pd.read_csv(
    'data/fb_stock_prices_2018.csv', index_col='date', parse_dates=True #Inserting the dates in a column
)

plt.plot(fb.index, fb.open)
plt.show() #Showing the output of the graph
```



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.

```
#The %matplotlib inline will automatically show the plot without using th plt.show()
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd

fb = pd.read_csv(
    'data/fb_stock_prices_2018.csv', index_col='date', parse_dates=True
)
plt.plot(fb.index, fb.open)
```

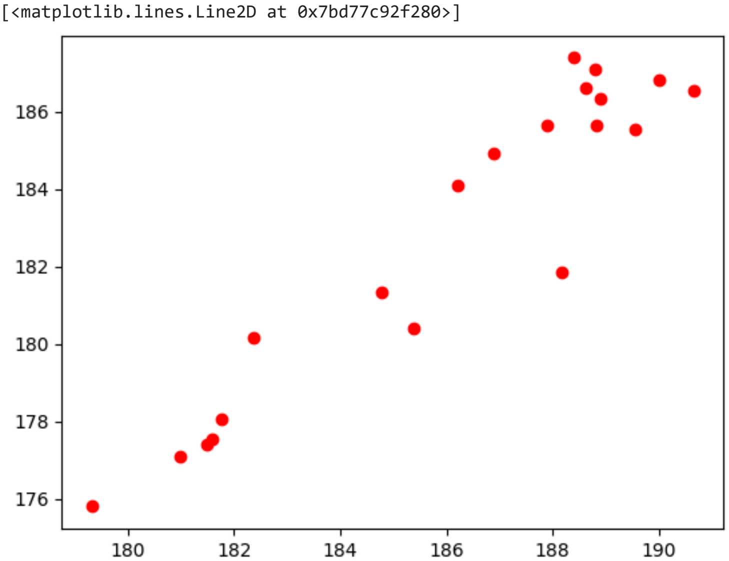
```
[<matplotlib.lines.Line2D at 0x7bd775ec3670>]
```

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':

```
# The scatter plots will show an output of the plot with just dots and no lines.
plt.plot('high', 'low', 'ro', data=fb.head(20))
```

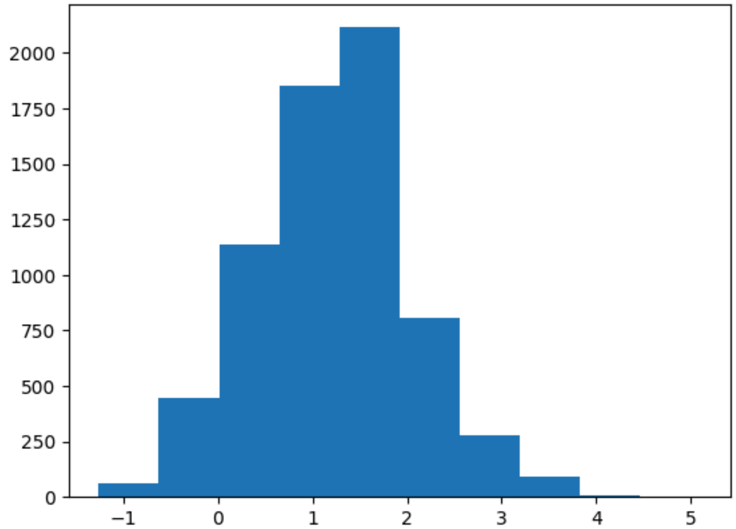
```
[<matplotlib.lines.Line2D at 0x7bd77c92f280>]
```



Histograms

```
# Histogram is a graph that shows the frequency of numerical data using rectangles.
quakes = pd.read_csv('data/earthquakes-1.csv')

# The plt.hist will show the output of the plot into Histogram format
plt.hist(quakes.query('magType == "ml"').mag)
```

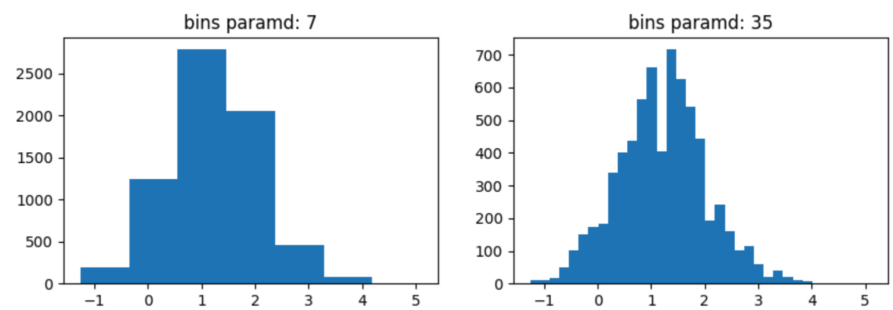
```
(array([6.400e+01, 4.450e+02, 1.137e+03, 1.853e+03, 2.114e+03, 8.070e+02,
        2.800e+02, 9.200e+01, 9.000e+00, 2.000e+00]),
 array([-1.26 , -0.624,  0.012,  0.648,  1.284,  1.92 ,  2.556,  3.192,
         3.828,  4.464,  5.1  ]),
 <BarContainer object of 10 artists>)
```



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):

```
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 paramd: {bins}')
```



Plot components

Figure

Top-level object that holds the other plot components.

```
fig = plt.figure()
```

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

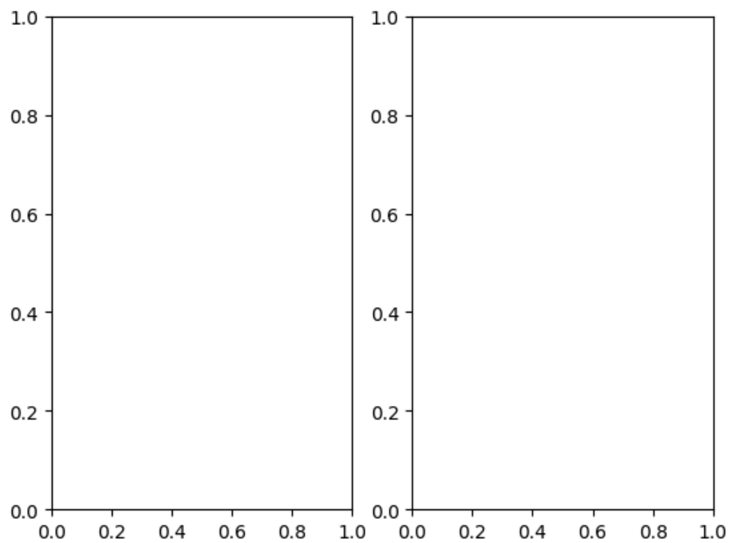
Axes

Individual plots contained within the Figure.

Creating Subplots

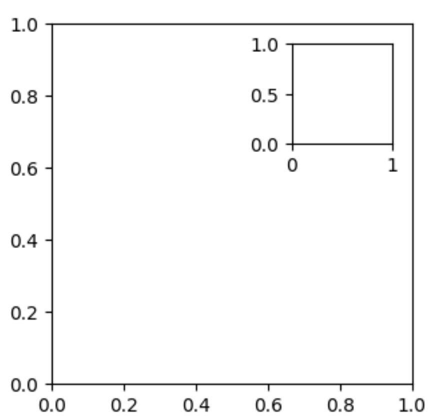
Simple specify the number of rows and columns to create:

```
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:

```
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])
```

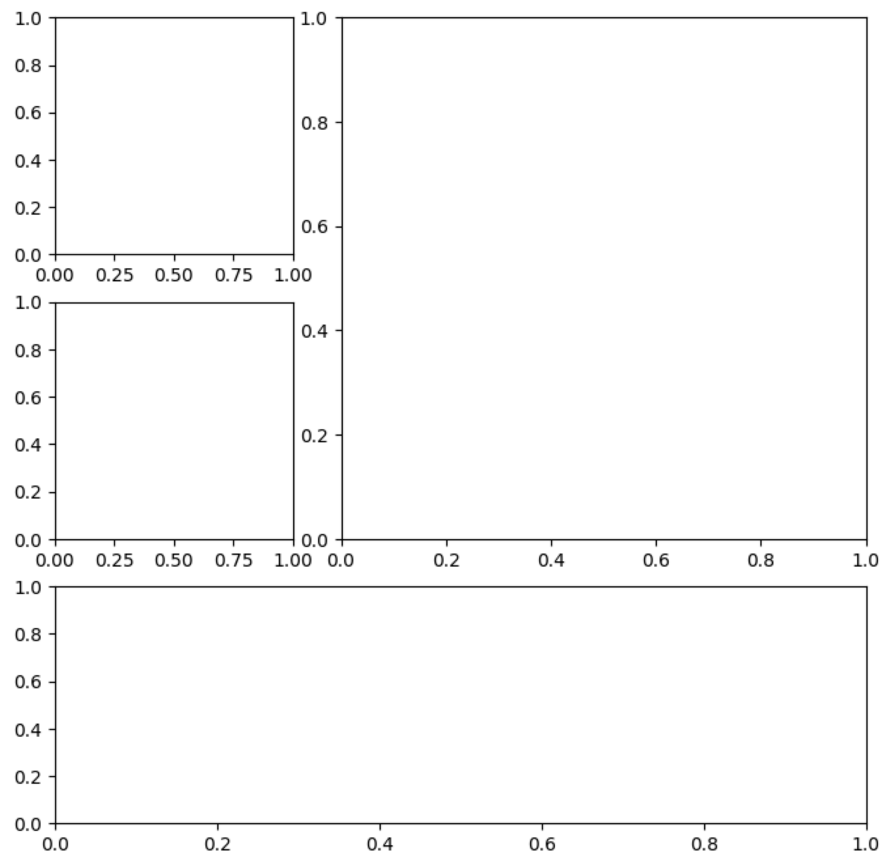


Creating Plot Layouts with gridspec

we can create subplots with varying sizes as well:

```
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:])
```

```
top_right = fig.add_subplot(gs[:2, 1:])
bottom = fig.add_subplot(gs[2,:])
```



Saving plots

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

```
fig.savefig('empty.png') #Used to save the created plot
```

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():

```
plt.close('all') #Closing all Figure objects that are open
```

Additional plotting options

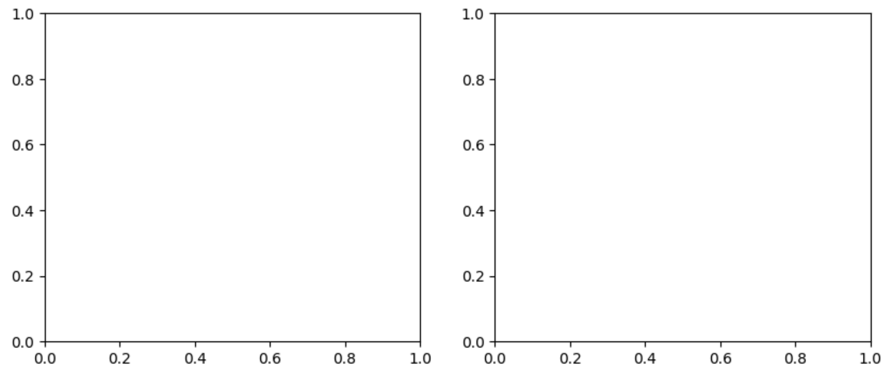
Specifying figure size Just pass the **figsize** parameter to plt.figure(). it's a tuple of (width, height):

```
fig = plt.figure(figsize=(10,4))
```

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

This can be specified when creating subplots as well:

```
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
```



rcParams

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

```
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])
```

```
['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**:

```
mpl.rcParams['figure.figsize'] #Checking the default figure size using rcParams:
```

```
[6.4, 4.8]
```

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

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

```
[300.0, 10.0]
```

Use **rcdefaults()** to restore the defaults:

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

```
[6.4, 4.8]
```

This can also be done via **pyplot**:

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

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.

Setup

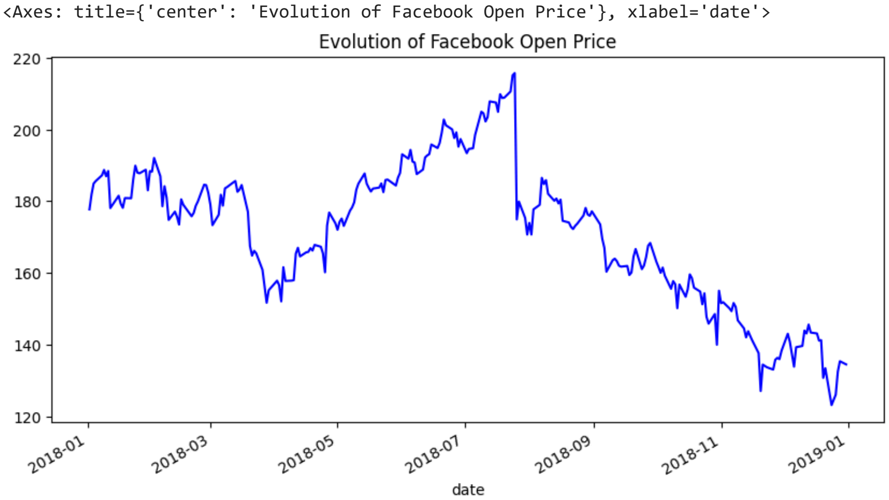
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
fb = pd.read_csv(
'data/fb_stock_prices_2018.csv', index_col='date', parse_dates=True
)
quakes = pd.read_csv('data/earthquakes.csv')
```

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:

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

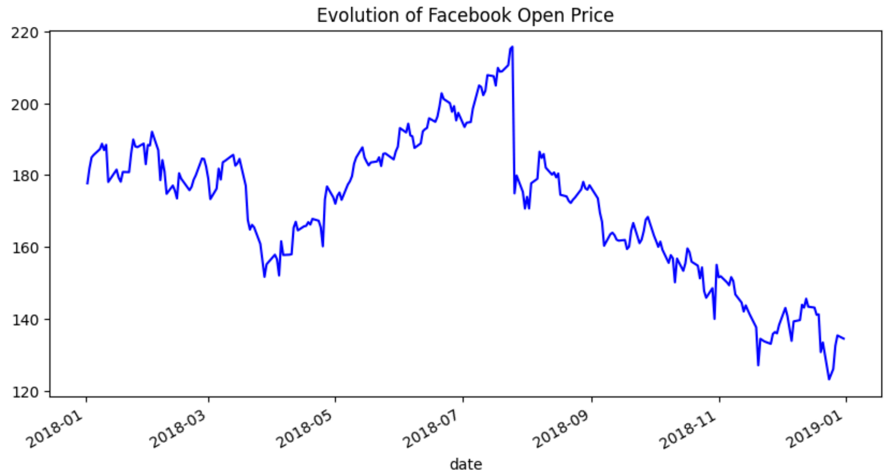
```
<Axes: title={'center': 'Evolution of Facebook Open Price'}, xlabel='date'>
```



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

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

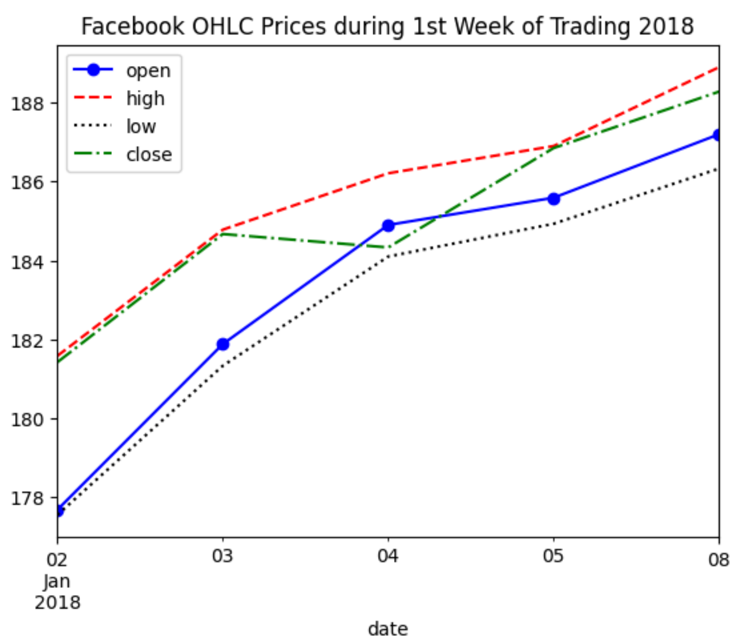
<Axes: title={'center': 'Evolution of Facebook Open Price'}, xlabel='date'>



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

```
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'
)
```

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

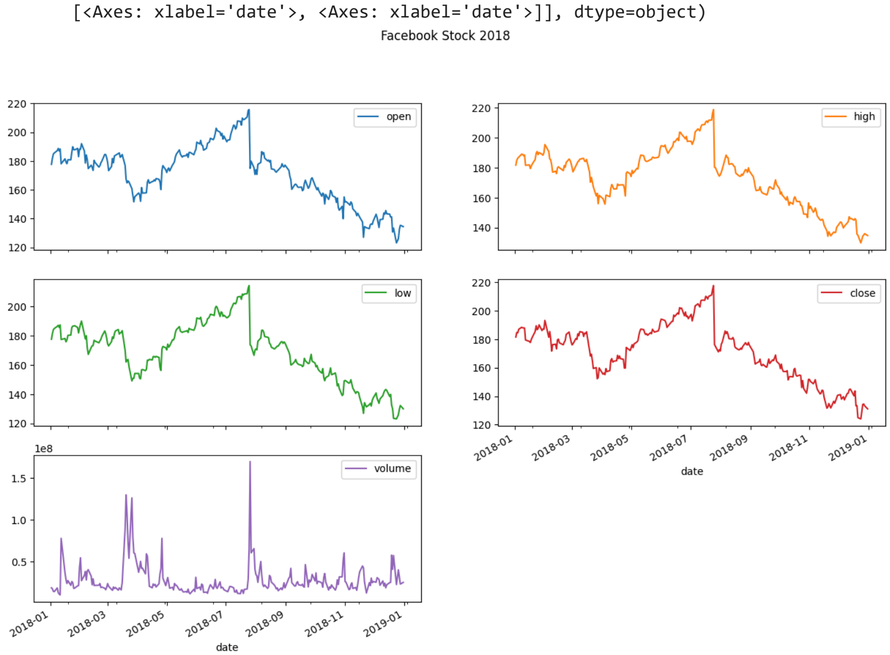


Creating subplots

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) :

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

```
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.

Visualizing relationships between variables

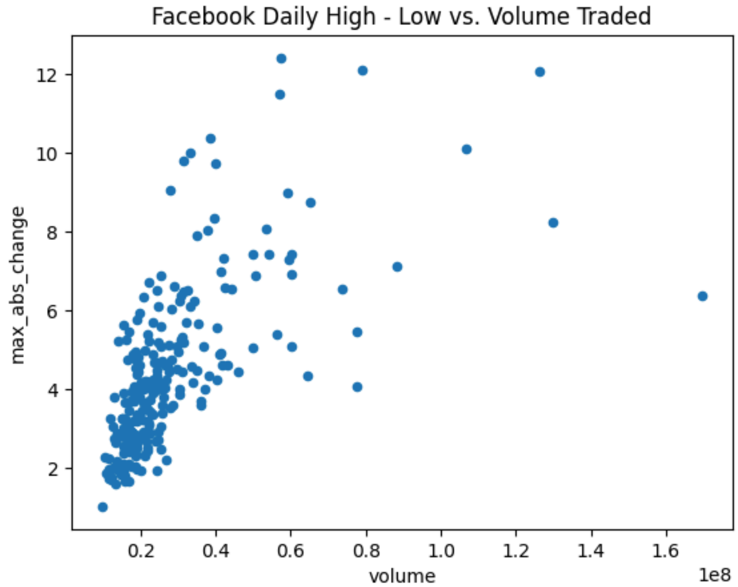
Scatter plots

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 xaxis and a column for the y-axis:

```
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'
)
```

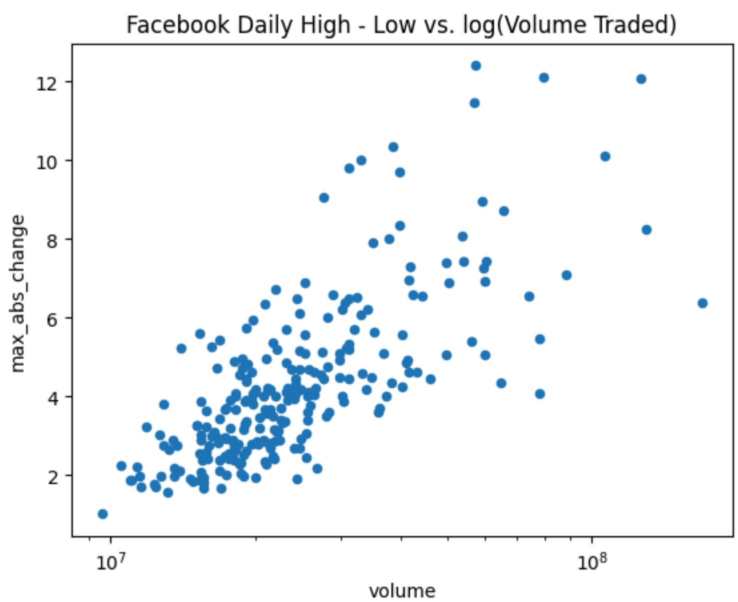
```
<Axes: title={'center': 'Facebook Daily High - Low vs. Volume Traded'},
xlabel='volume', ylabel='max_abs_change'>
```



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 :

```
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
)
```

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



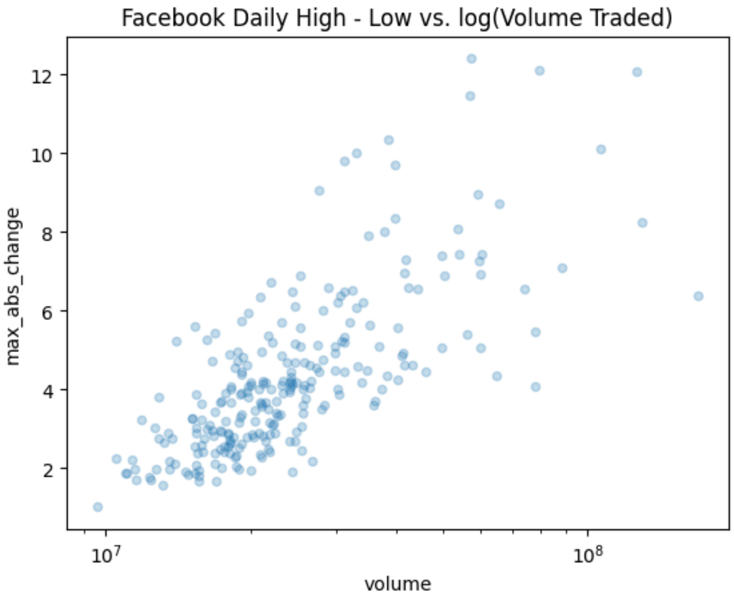
With matplotlib, we could use plt.xscale('log') to do the same thing.

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:

```
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
)
```

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

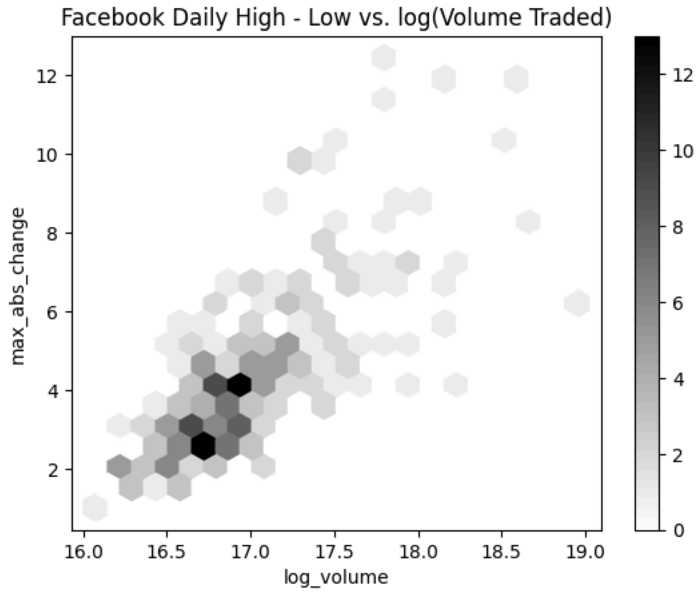


Hexbins

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:

```
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
)
```

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

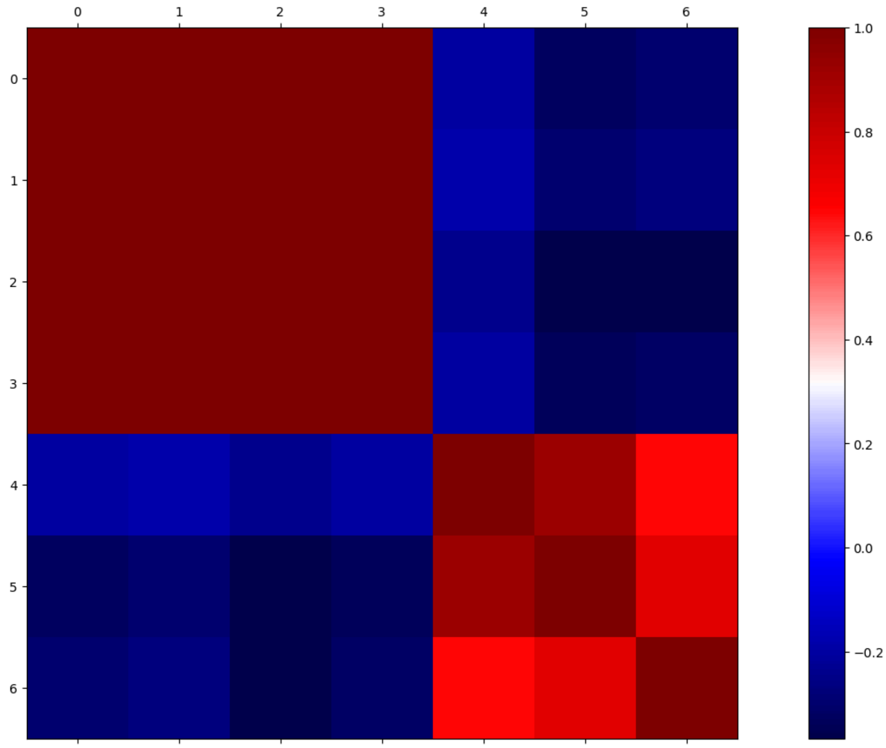


Visualizing Correlations with Heatmaps

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

```
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).set_clim(-1, 1)

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

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-88-e3d32d707d2b> in <cell line: 9>()
      7
      8 im = ax.matshow(fb_corr, cmap='seismic')
----> 9 fig.colorbar(im).set_clim(-1, 1)
     10
     11 labels = [col.lower() for col in fb_corr.columns]

AttributeError: 'Colorbar' object has no attribute 'set_clim'
```




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

```
volume        0.642027
log_volume    0.731542
Name: max_abs_change, dtype: float64
```

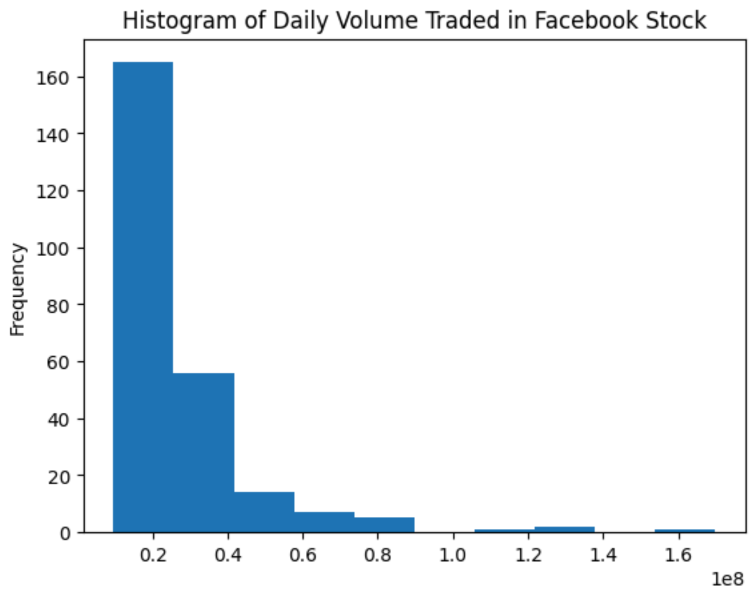
Visualizing distributions

Histograms

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

```
import matplotlib.pyplot as plt
label_text= 'Price ($)'
# Assuming fb is your DataFrame containing the data
fb.volume.plot( kind='hist', title='Histogram of Daily Volume Traded in Facebook Stock')  # Label the x-axis
plt.xlabel(label_text)
```

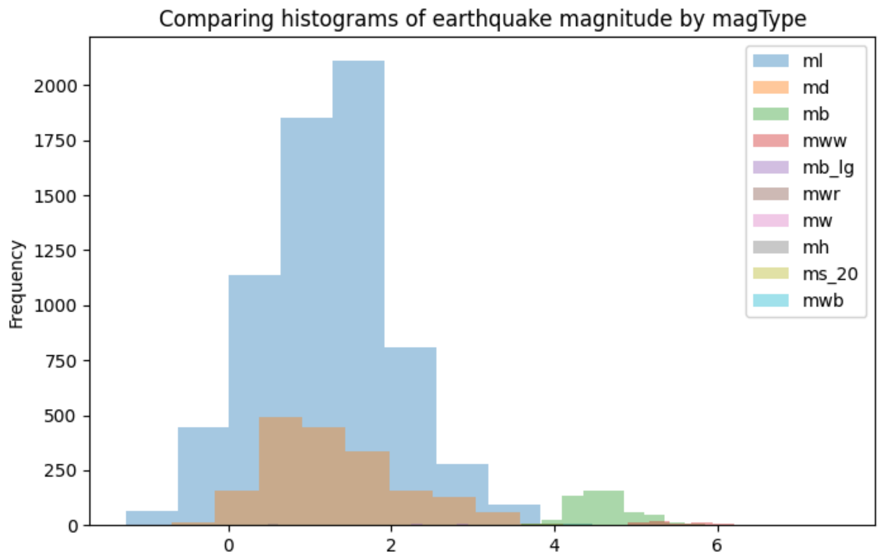
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-116-2bbbe7a9887a> in <cell line: 5>()
      3 # Assuming fb is your DataFrame containing the data
      4 fb.volume.plot( kind='hist', title='Histogram of Daily Volume Traded in
Facebook Stock')  # Label the x-axis
----> 5 plt.xlabel(label_text)

TypeError: 'str' object is not callable
```



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:

```
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)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-97-ad92217f2f43> in <cell line: 10>()
      8         title='Comparing histograms of earthquake magnitude by magType'
      9     )
---> 10 plt.xlabel('magnitude') # label the x-axis (discussed in chapter 6)

TypeError: 'str' object is not callable
```

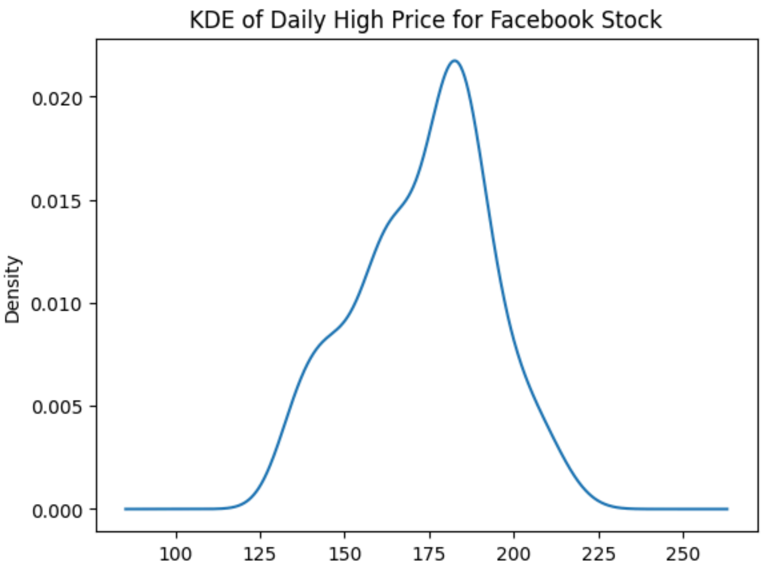


Kernel Density Estimation (KDE)

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

```
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)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-74-0aa784e0a1d2> in <cell line: 5>()
      3     title='KDE of Daily High Price for Facebook Stock'
      4 )
----> 5 plt.xlabel('Price ($)') # label the x-axis (discussed in chapter 6)

TypeError: 'str' object is not callable
```

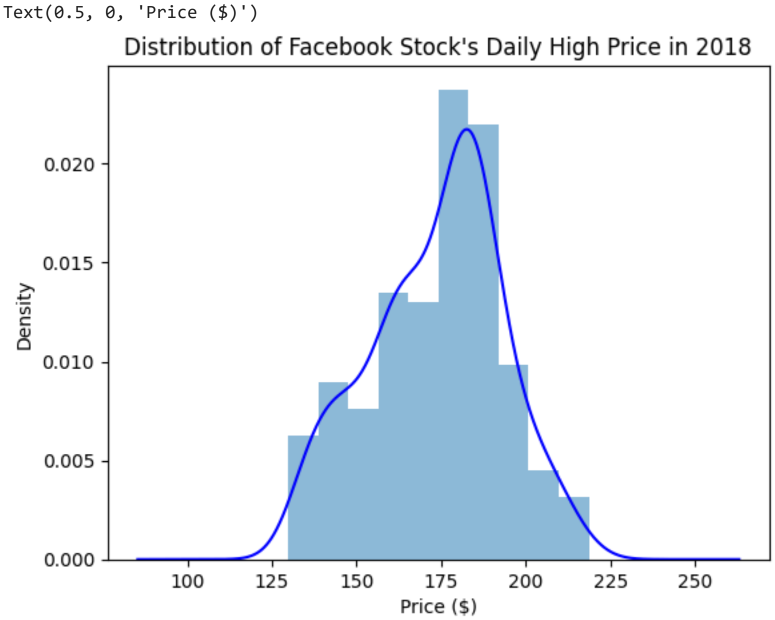


Adding to the result of plot()

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:

```
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)
```

```
Text(0.5, 0, 'Price ($)')
```

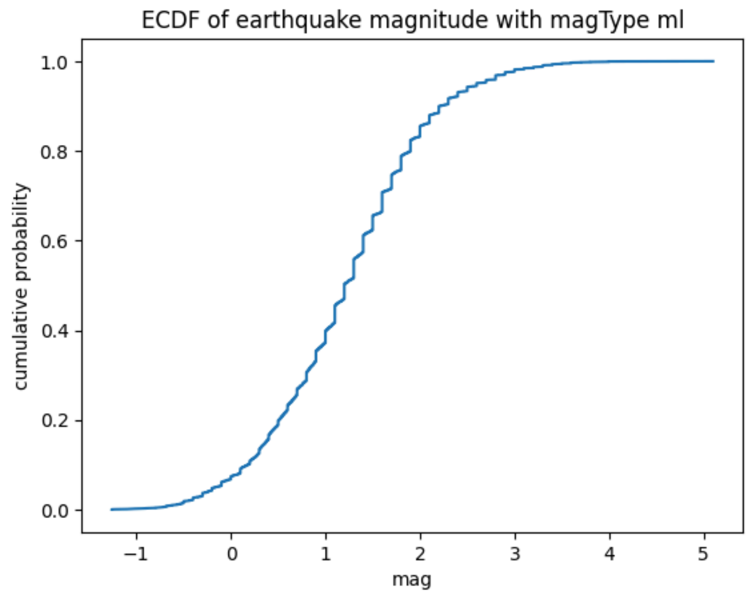


Plotting the ECDF

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):

```
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')
```

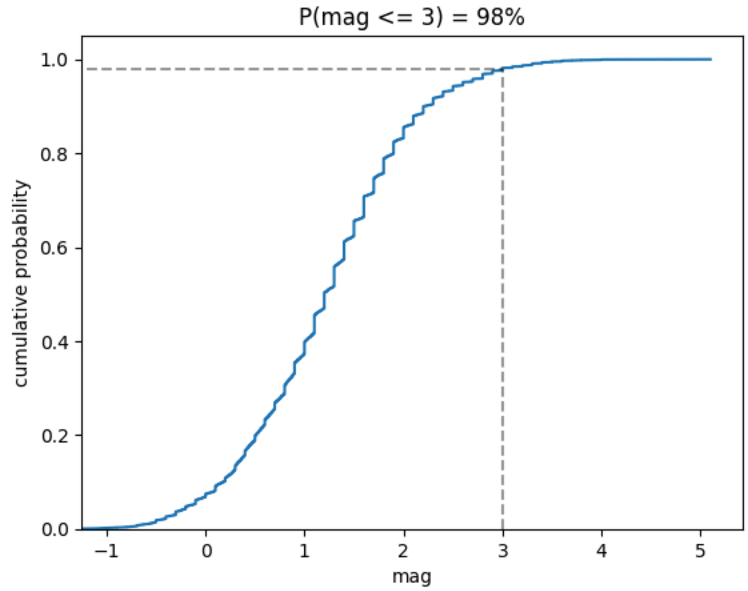
```
Text(0.5, 1.0, 'ECDF of earthquake magnitude with magType ml')
```



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

```
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%')
```

```
Text(0.5, 1.0, 'P(mag <= 3) = 98%')
```

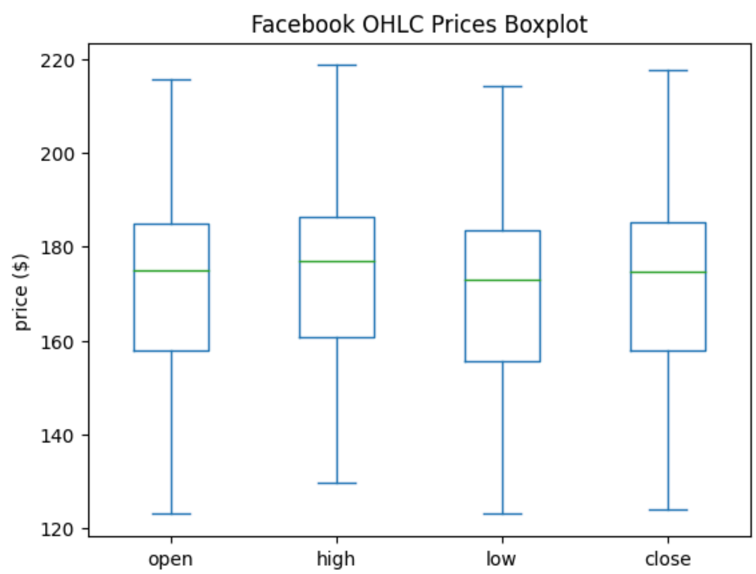


Box plots

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

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

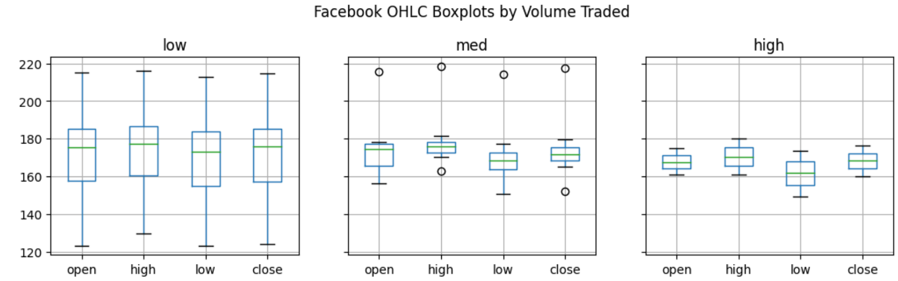
```
Text(0, 0.5, 'price ($)')
```



This can also be combined with a groupby() :

```
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)
```

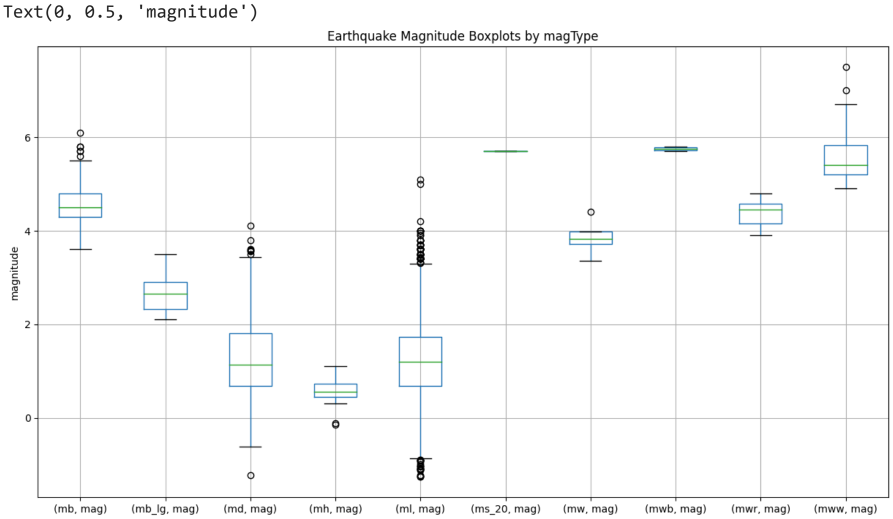
```
Text(0.5, 1.1, 'Facebook OHLC Boxplots by Volume Traded')
```



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

```
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)
```

```
Text(0, 0.5, 'magnitude')
```



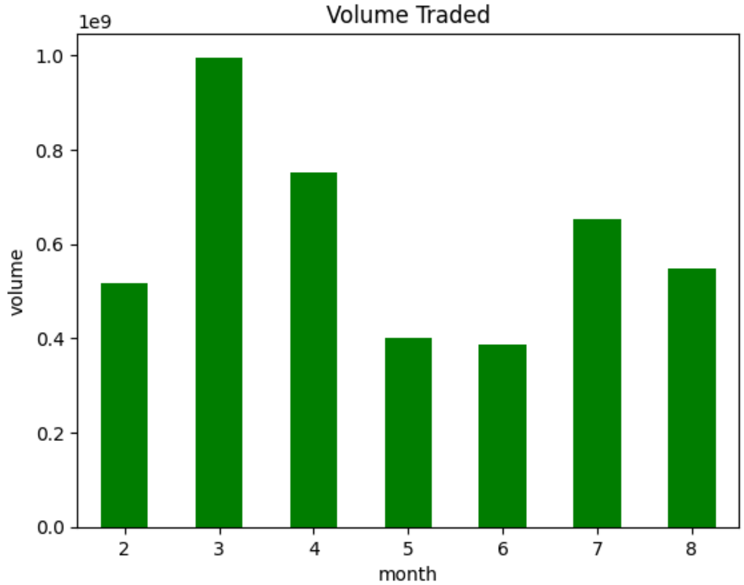
Counts and frequencies

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:

```
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)
```

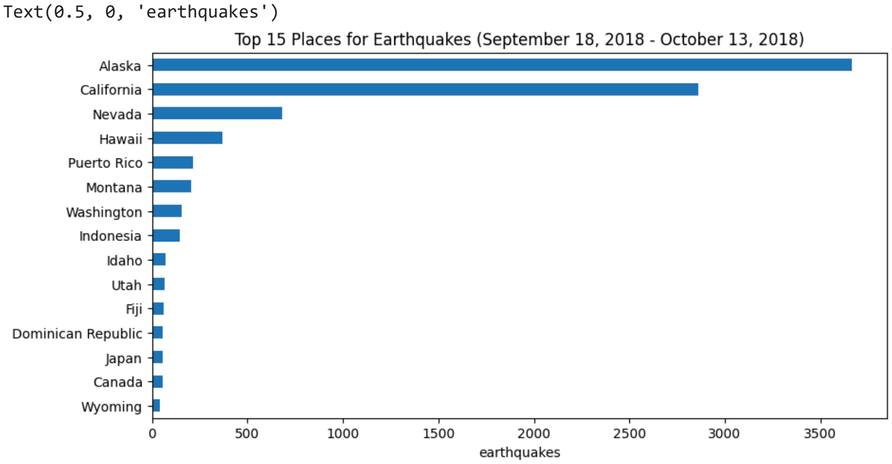
```
Text(0, 0.5, 'volume')
```



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:

```
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)
```

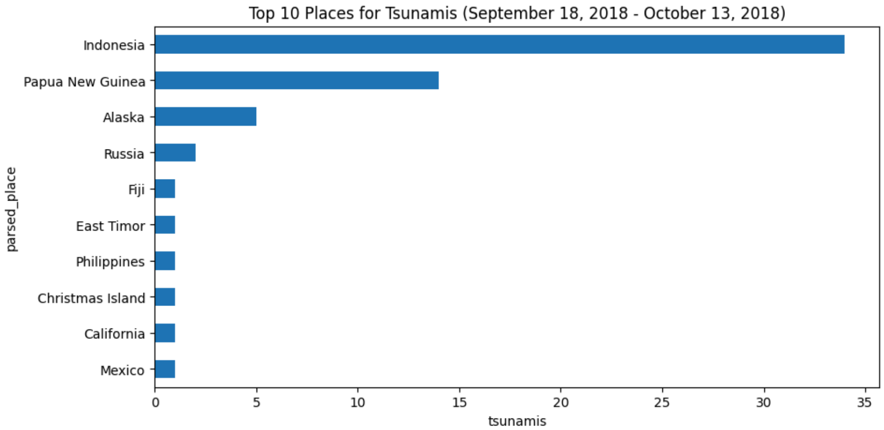
```
Text(0.5, 0, 'earthquakes')
```



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

```
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)
```

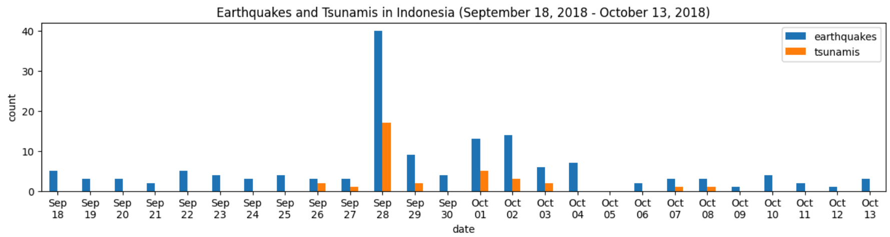
```
Text(0.5, 0, 'tsunamis')
```



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:

```
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')
```

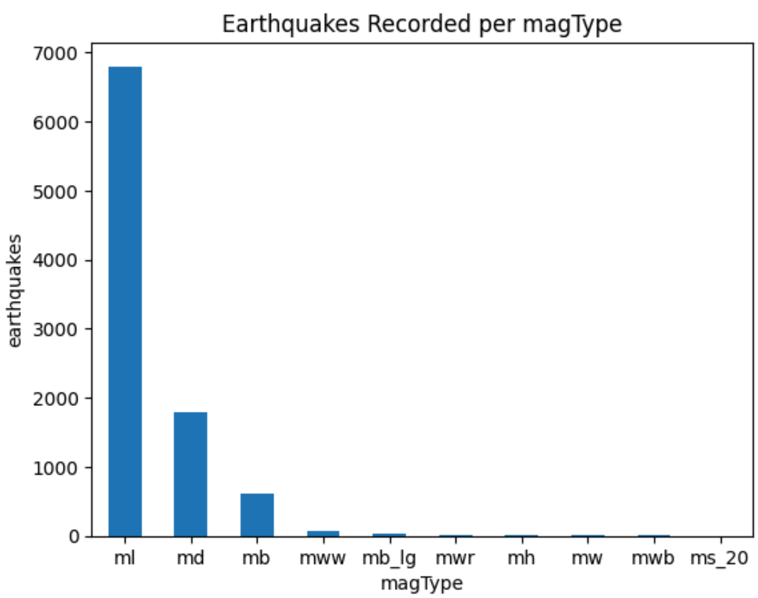
```
<ipython-input-27-79287d06555a>:4: FutureWarning: The default value of numeric_only in
  ).set_index('time').resample('1D').sum()
Text(0, 0.5, 'count')
```



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

```
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')
```

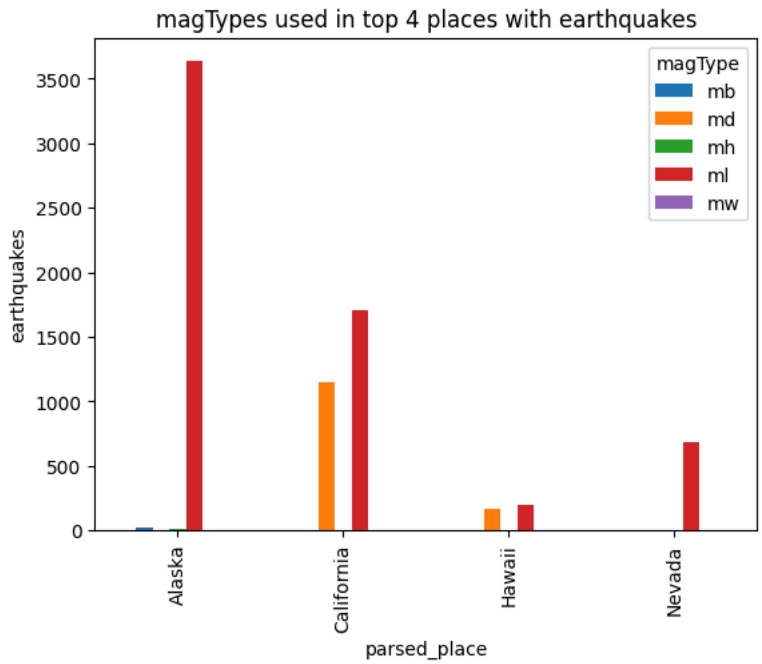
```
Text(0, 0.5, 'earthquakes')
```



Top 4 places with earthquakes:

```
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)
```

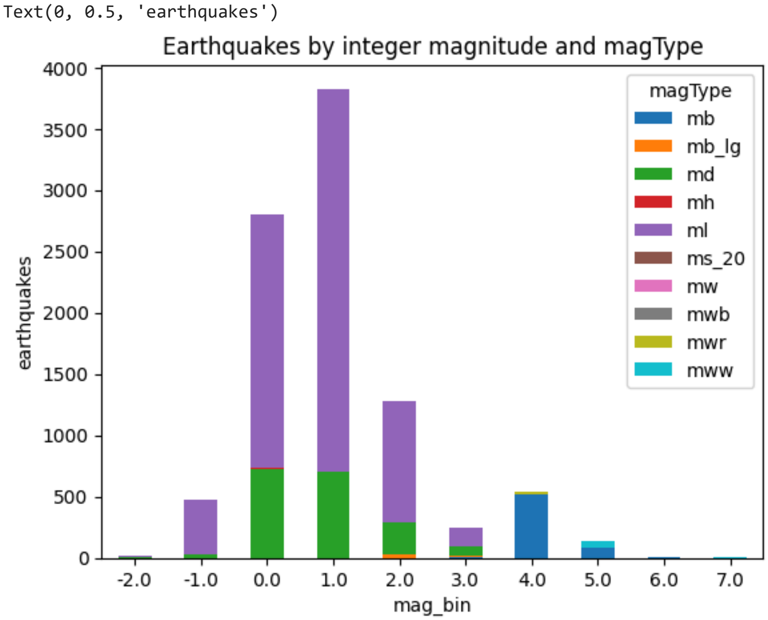
```
Text(0, 0.5, 'earthquakes')
```



Stacked bar chart

```
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)
```

```
Text(0, 0.5, 'earthquakes')
```

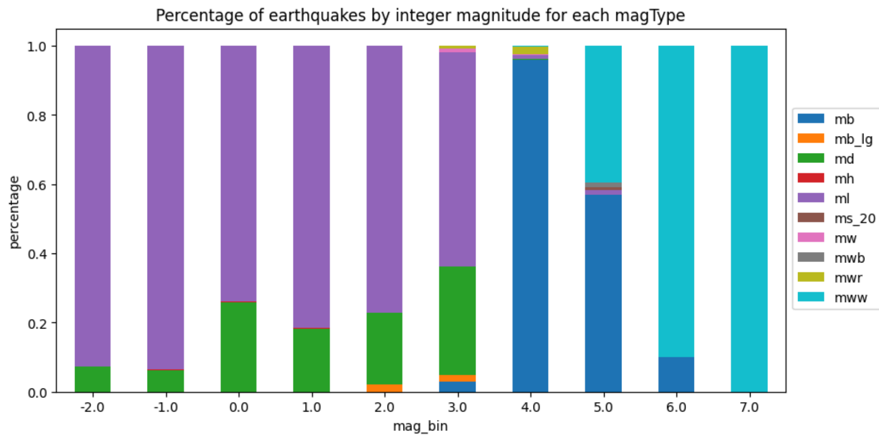


Normalized stacked bars

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

```
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)
```

```
Text(0, 0.5, 'percentage')
```



9.3 Pandas Plotting Subpackage

pandas.plotting subpackage

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

About the Data

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

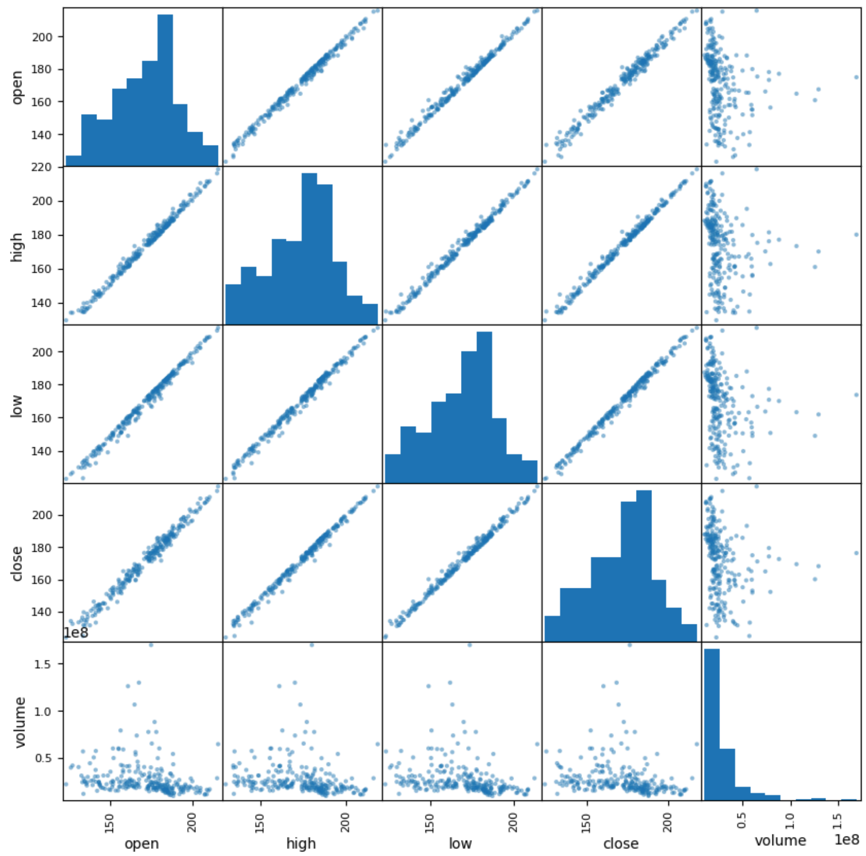
Setup

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

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

Scatter matrix

```
from pandas.plotting import scatter_matrix
scatter_matrix(fb, figsize=(10, 10))
```

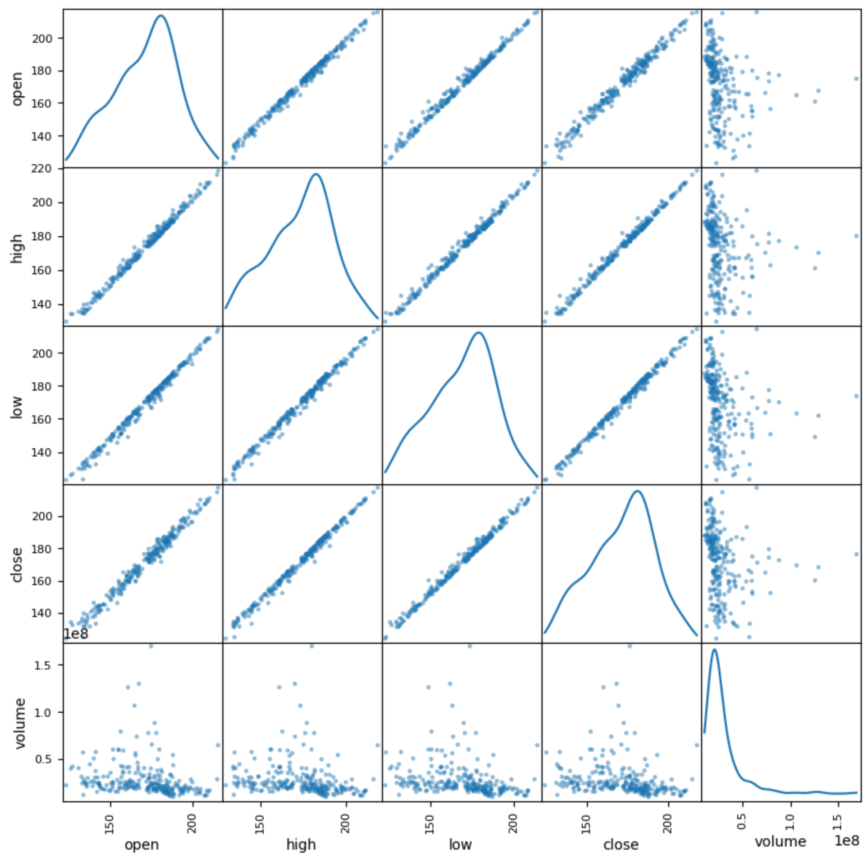
```
array([[<Axes: xlabel='open', ylabel='open'>,
        <Axes: xlabel='high', ylabel='open'>,
        <Axes: xlabel='low', ylabel='open'>,
        <Axes: xlabel='close', ylabel='open'>,
        <Axes: xlabel='volume', ylabel='open'>],
       [<Axes: xlabel='open', ylabel='high'>,
        <Axes: xlabel='high',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

```
scatter_matrix(fb, figsize=(10, 10), diagonal='kde')
```

```
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)
```



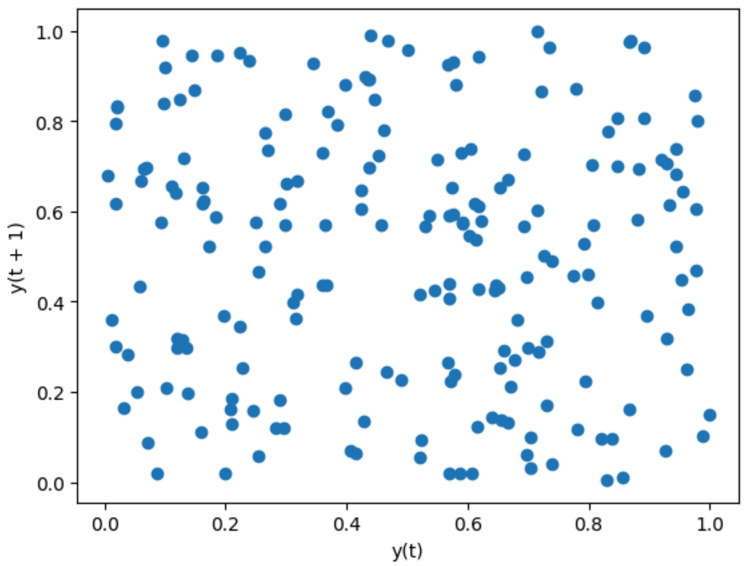
Lag plot

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

```
from pandas.plotting import lag_plot
np.random.seed(0) # make this repeatable
```

```
lag_plot(pd.Series(np.random.random(size=200)))
```

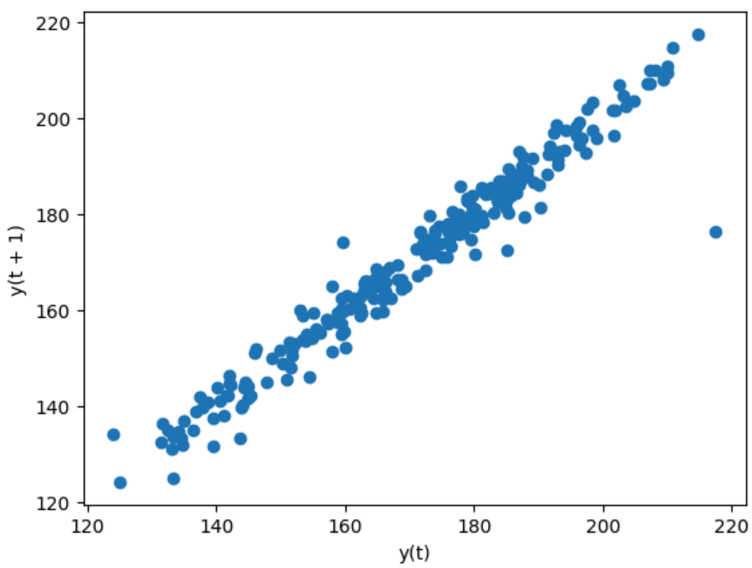
```
<Axes: xlabel='y(t)', ylabel='y(t + 1)'>
```



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

```
lag_plot(fb.close)
```

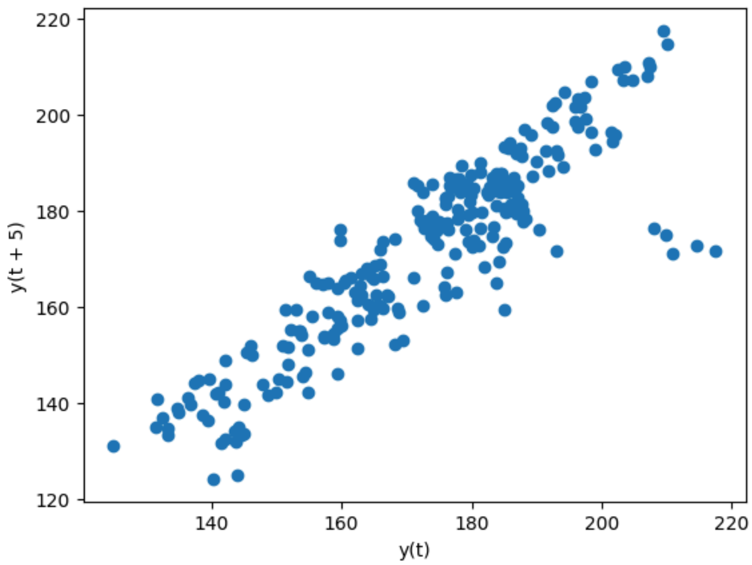
```
<Axes: xlabel='y(t)', ylabel='y(t + 1)'>
```



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):

```
lag_plot(fb.close, lag=5)
```

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

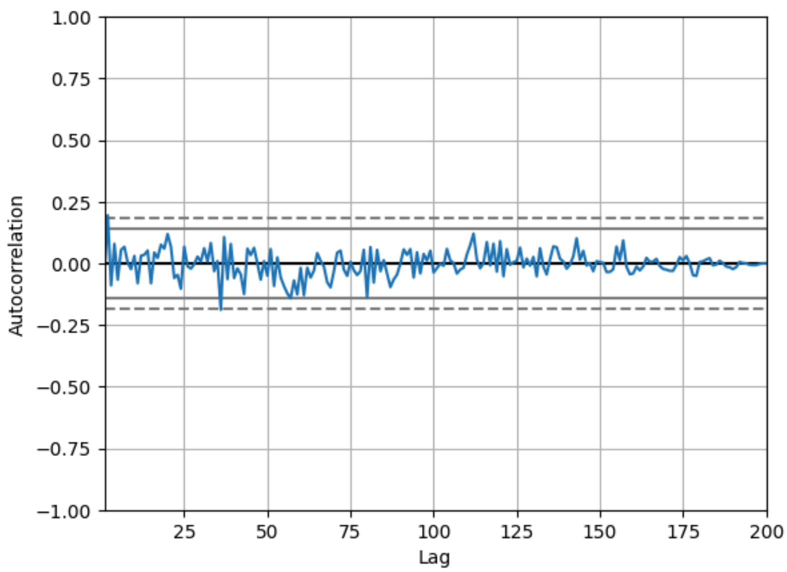


Autocorrelation plots

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):

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

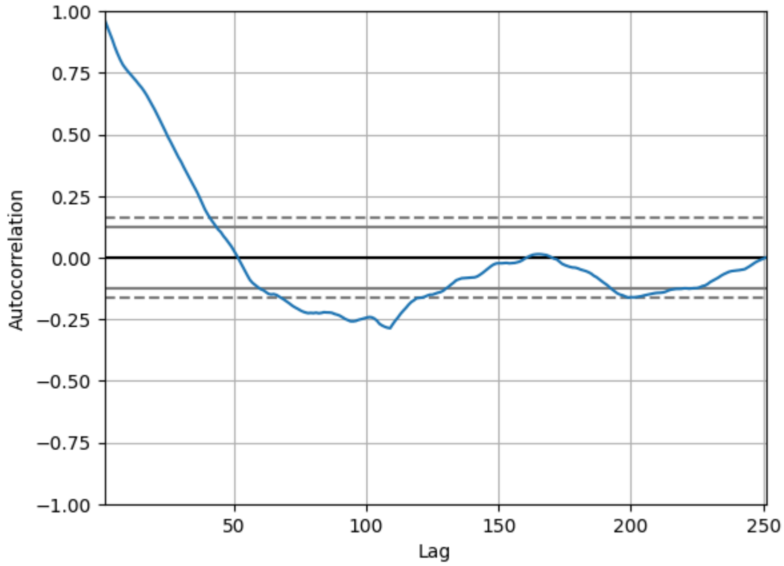
```
<Axes: xlabel='Lag', ylabel='Autocorrelation'>
```



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

```
autocorrelation_plot(fb.close)
```

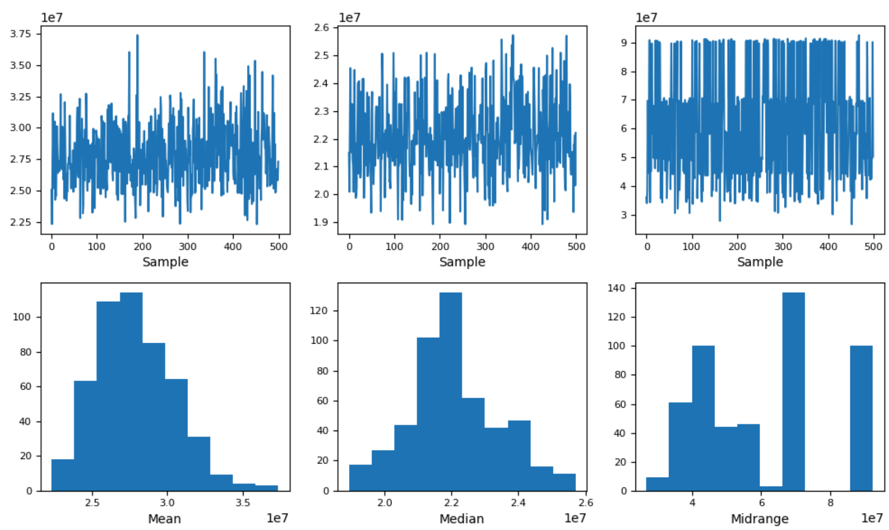
<Axes: xlabel='Lag', ylabel='Autocorrelation'>



Bootstrap plot

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

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



Data Analysis:

leveraging the diverse range of visualization techniques available in Python with Pandas enables comprehensive exploratory data analysis (EDA) and insightful data visualization. By utilizing tools such as Bootstrap plots, Lag plots, Scatter matrices, Stacked bar charts, Counts and frequencies analysis, Box plots, ECDF plots, and Kernel Density Estimation (KDE), analysts can effectively explore and communicate complex patterns, distributions, and relationships within their datasets.

We can efficiently explore different visualizations and refine the understanding of the data. By integrating these tools we can gain deeper insights into the datasets, and ultimately drive informed decision-making and innovation in the respective fields.

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.

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Converting the strings in date to a Date value
fb = pd.read_csv('fb_stock_prices_2018.csv', index_col = 'date', parse_dates = True)
fb
```

date	open	high	low	close	volume
2018-01-02	177.68	181.58	177.5500	181.42	18151903
2018-01-03	181.88	184.78	181.3300	184.67	16886563
2018-01-04	184.90	186.21	184.0996	184.33	13880896
2018-01-05	185.59	186.90	184.9300	186.85	13574535
2018-01-08	187.20	188.90	186.3300	188.28	17994726
...	...	...	...	...	...
2018-12-24	123.10	129.74	123.0200	124.06	22066002
2018-12-26	126.00	134.24	125.8900	134.18	39723370
2018-12-27	132.44	134.99	129.6700	134.52	31202509
2018-12-28	135.34	135.92	132.2000	133.20	22627569
2018-12-31	134.45	134.64	129.9500	131.09	24625308

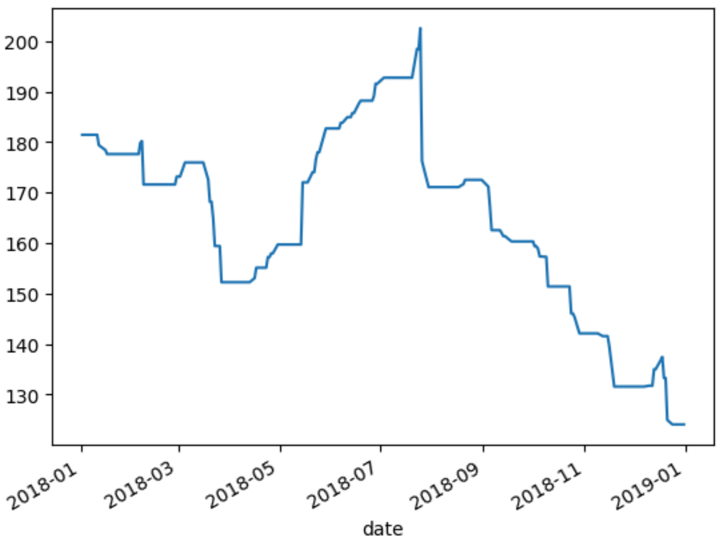
251 rows × 5 columns

```
# Rolling the 20 days minimum of Facebook closing price
fb_rg = fb['close'].rolling('20D').min()
fb_rg
```

```
date
2018-01-02    181.42
2018-01-03    181.42
2018-01-04    181.42
2018-01-05    181.42
2018-01-08    181.42
               ...
2018-12-24    124.06
2018-12-26    124.06
2018-12-27    124.06
2018-12-28    124.06
2018-12-31    124.06
Name: close, Length: 251, dtype: float64
```

```
# Plotting the 20 days minimum of Facebook closing price
fb_rg.plot()
```

```
<Axes: xlabel='date'>
```



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

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

diff = pd.DataFrame()
```

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

# Subtracting the open and close in the price of Facebook stock to get the difference
diff['Difference'] = (fb['open'] - fb['close'])
diff
```

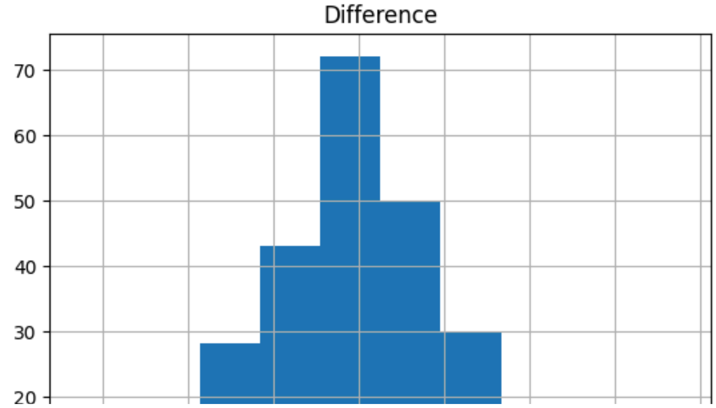
	Difference
date	
2018-01-02	-3.74
2018-01-03	-2.79
2018-01-04	0.57
2018-01-05	-1.26
2018-01-08	-1.08
...	...
2018-12-24	-0.96
2018-12-26	-8.18
2018-12-27	-2.08
2018-12-28	2.14
2018-12-31	3.36

251 rows × 1 columns

Next steps: View recommended plots

```
# Histogram output
diff.hist()
```

```
array([[<Axes: title={'center': 'Difference'}>]], dtype=object)
```




```
# KDE of the Daily High Price for Facebook Stock
diff.Difference.plot(
    kind='kde',
    title='KDE of Daily High Price for Facebook Stock'
)
```

```
<Axes: title={'center': 'KDE of Daily High Price for Facebook Stock'},
ylabel='Density'>
```