CPE311 Computational Thinking with Python

Name: Jhon Hendricks T. Bautista

Performed on: 04/13/2025

Submitted on: 04/13/2025

Submitted to: Engr. Roman M. Richard

Instructions:

Create a Python notebook to answer all shown procedures, exercises and analysis in this section

Resources:

Download the following datasets: fb_stock_prices_2018.csv Download fb_stock_prices_2018.csv, earthquakes-1.csv

Procedures:

```
9.4 Introduction to Seaborn

9.5 Formatting Plots

9.6 Customizing Visualizations
```

9.4 Introduction to Seaborn

Introduction to Seaborn

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)
```

Setup

In [1]:
```
import matplotlib.pyplot as plt # for plotting
import numpy as np # handling series data
import seaborn as sns # plotting better visuals
import pandas as pd # dataframe handling
%matplotlib inline

fb = pd.read_csv('fb_stock_prices_2018.csv', index_col='date', parse_dates=True
) # loading data set
quakes = pd.read_csv('earthquakes.csv') # loading dataset
```

Categorical data

A 7.5 magnitude earthquake on September 28, 2018 near Palu, Indonesia caused a devastating tsunami afterwards. Let's take a look at some visualizations to understand what magTypes are used in Indonesia, the range of magnitudes there, and how many of the earthquakes are accompanied by a tsunami.

In [2]:
```
quakes.assign(time=lambda x: pd.to_datetime(x.time, unit='ms')
).set_index('time').loc['2018-09-28'].query(
"parsed_place == 'Indonesia' and tsunami == 1 and mag == 7.5")
```

Out[2]:

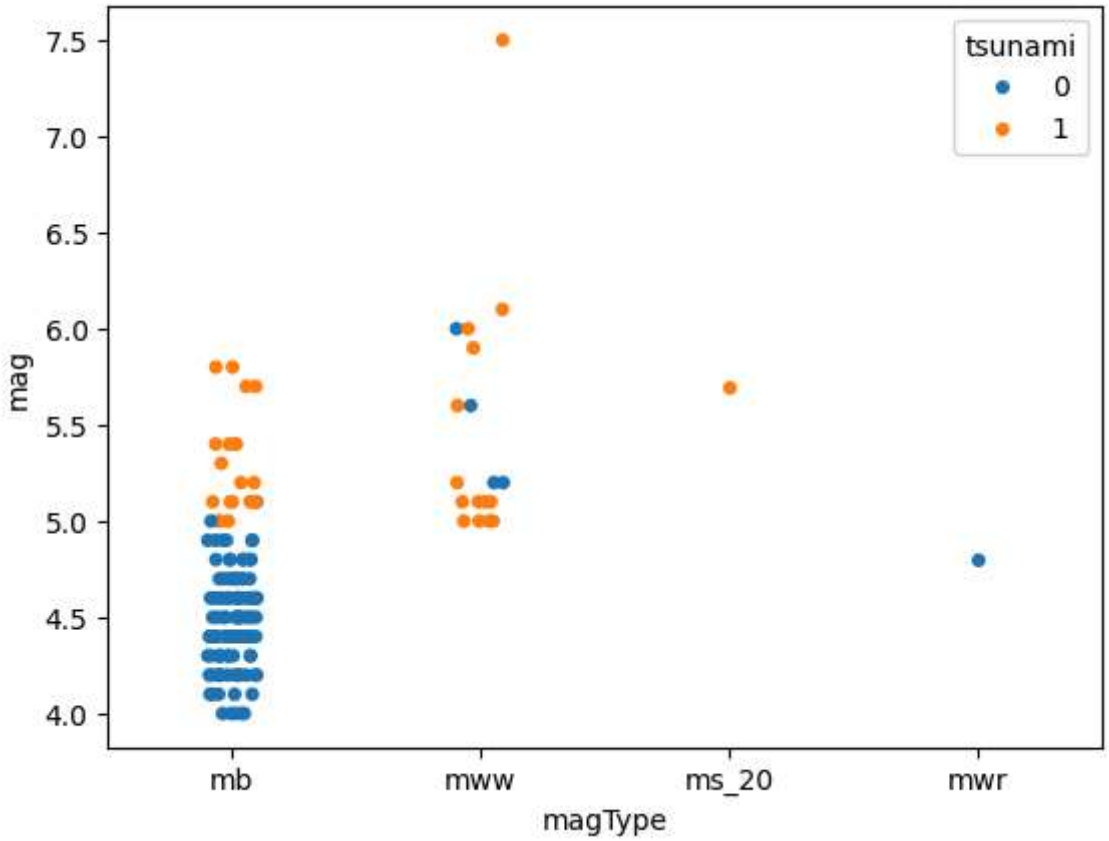
time	mag	magType	place	tsunami	parsed_place
2018-09-28 10:02:43.480	7.5	mww	78km N of Palu, Indonesia	1	Indonesia

stripplot()

The stripplot() function helps us visualize categorical data on one axis and numerical data on the other. We also now have the option of coloring our points using a column of our data (with the hue parameter). Using a strip plot, we can see points for each earthquake that was measured with a given magType and what its magnitude was; however, it isn't too easy to see density of the points due to overlap:

In [12]:
```
sns.stripplot(
 x='magType',
 y='mag',
 hue='tsunami',
 data=quakes.query('parsed_place == "Indonesia"')
```

```
)
plt.show()
```

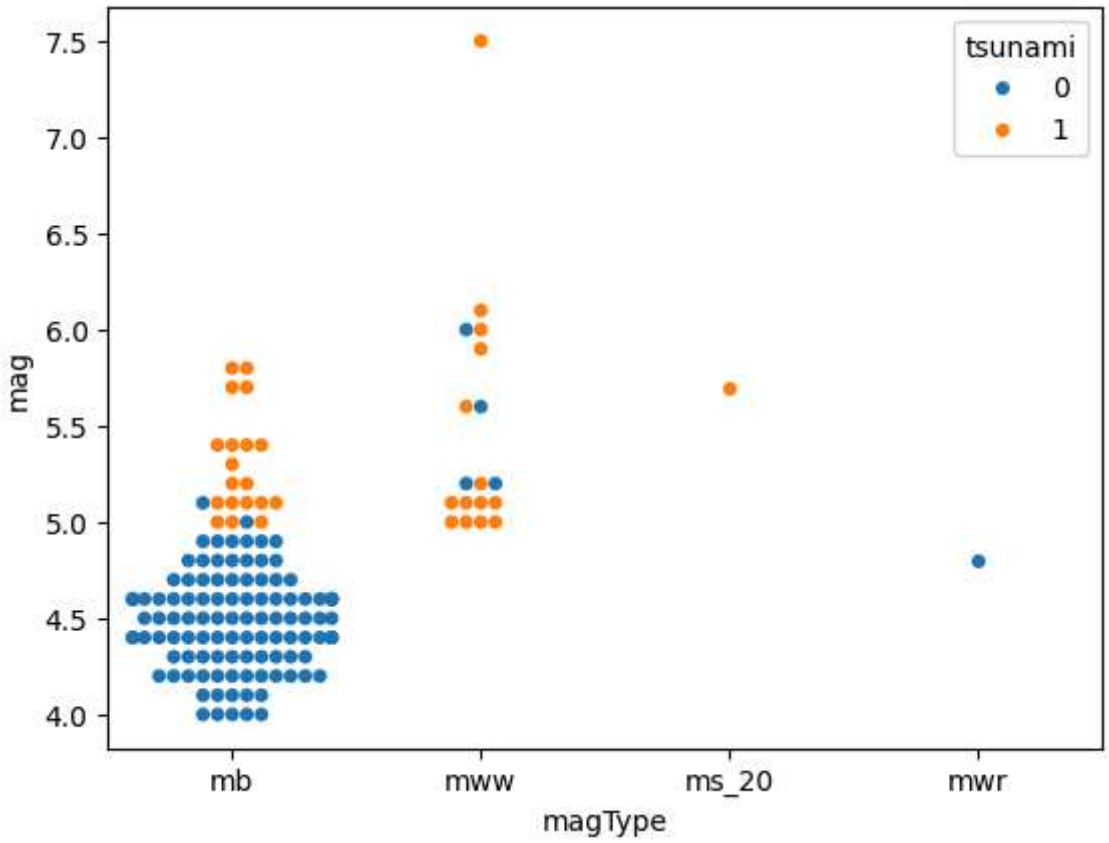


swarmplot()

The bee swarm plot helps address this issue be keeping the points from overlapping. Notice how many more points we can see for the blue section of the mb magType :

In [13]:
```
sns.swarmplot(
 x='magType',
 y='mag',
 hue='tsunami',
 data=quakes.query('parsed_place == "Indonesia"')
)
plt.show()
```

```
C:\Users\Arnel Bulambao\.conda\envs\CPE311_BULAMBAO\Lib\site-packages\seaborn\categorical.py:3399: UserWarning: 10.2% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

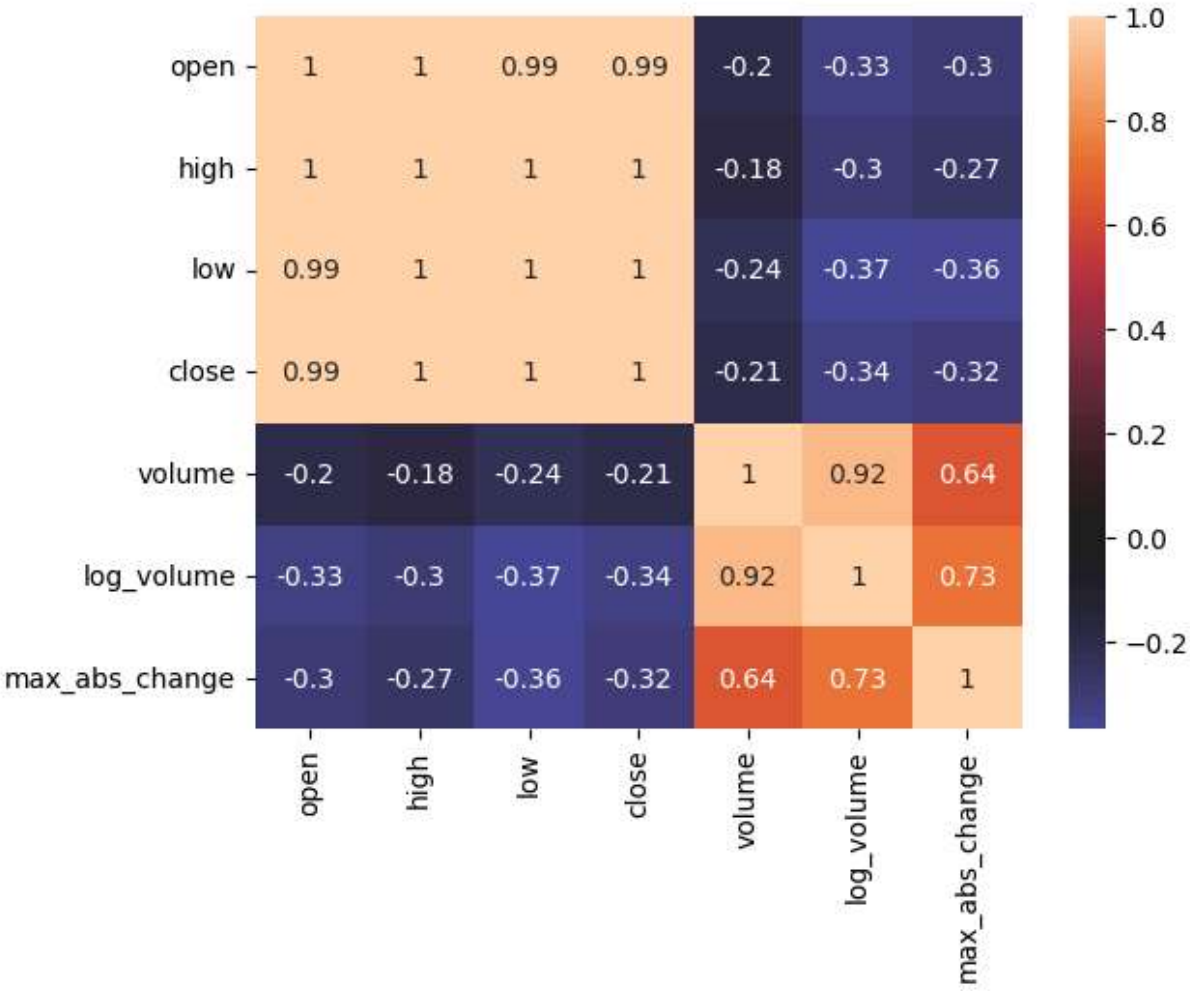


Correlations and Heatmaps

heatmap()

An easier way to create correlation matrix is to use seaborn :

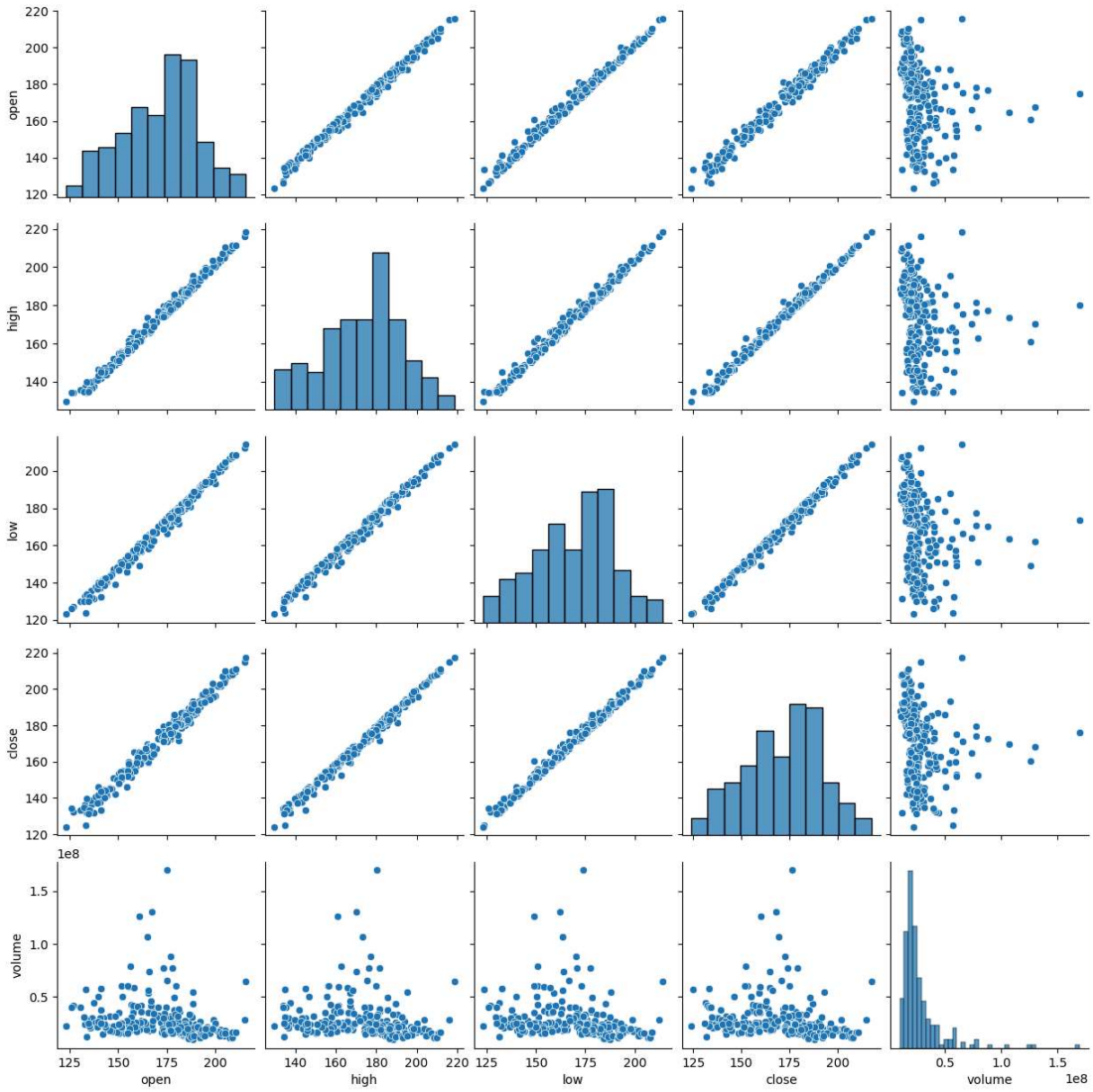
In [14]:
```
sns.heatmap(
 fb.sort_index().assign(
 log_volume=np.log(fb.volume),
 max_abs_change=fb.high - fb.low
).corr(),
 annot=True, center=0
)

plt.show()
```



pairplot()

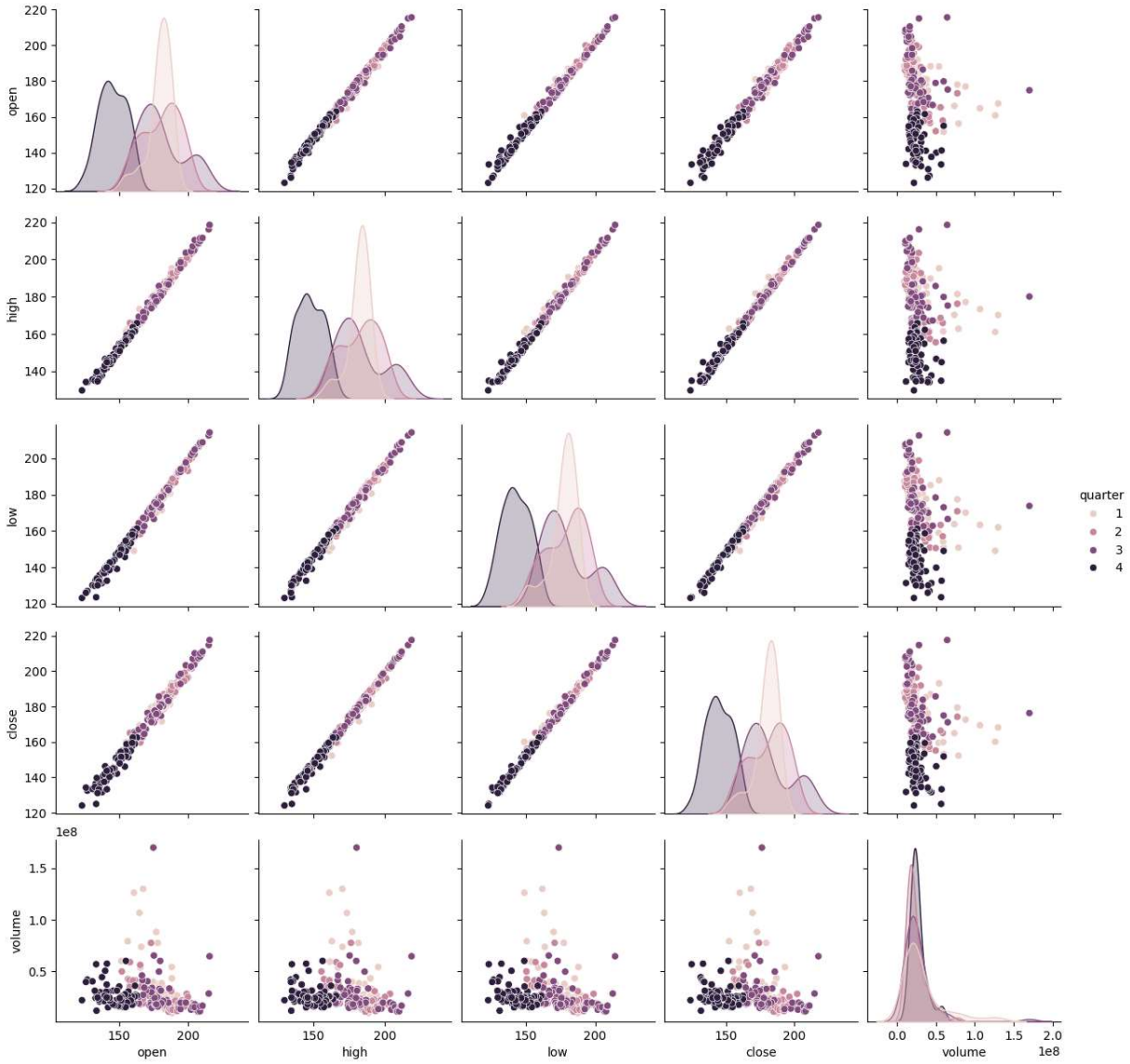
The pair plot is seaborn's answer to the scatter matrix we saw in the pandas subplotting notebook:

In [17]:
```
sns.pairplot(fb)
# creates plots relationships
plt.show()
```



Just as with pandas we can specify what to show along the diagonal; however, seaborn also allows us to color the data based on another column (or other data with the same shape):

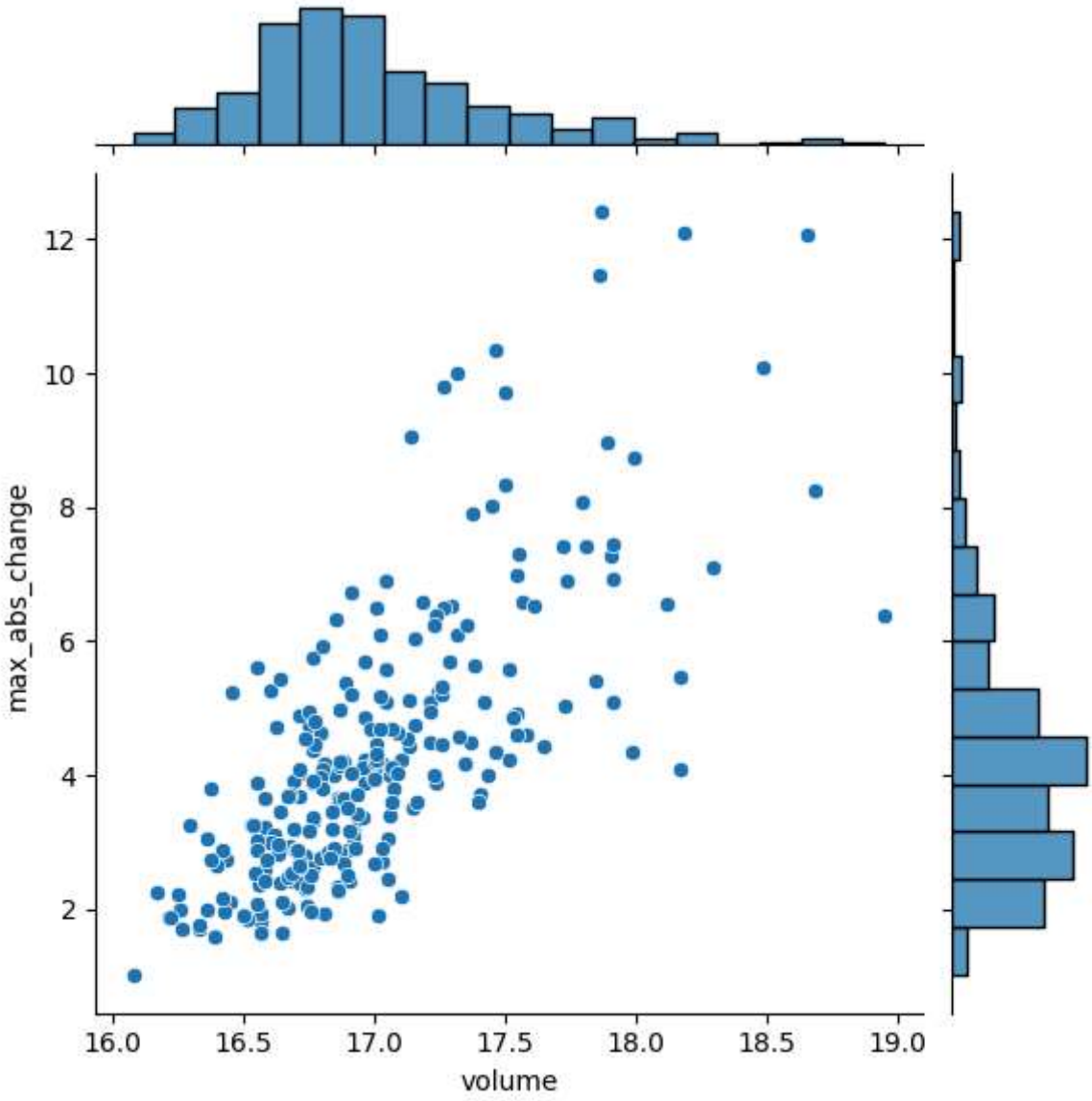
In [19]:
```
sns.pairplot(
 fb.assign(quarter=lambda x: x.index.quarter),
 diag_kind='kde',
 hue='quarter'
)
plt.show()
```



jointplot()

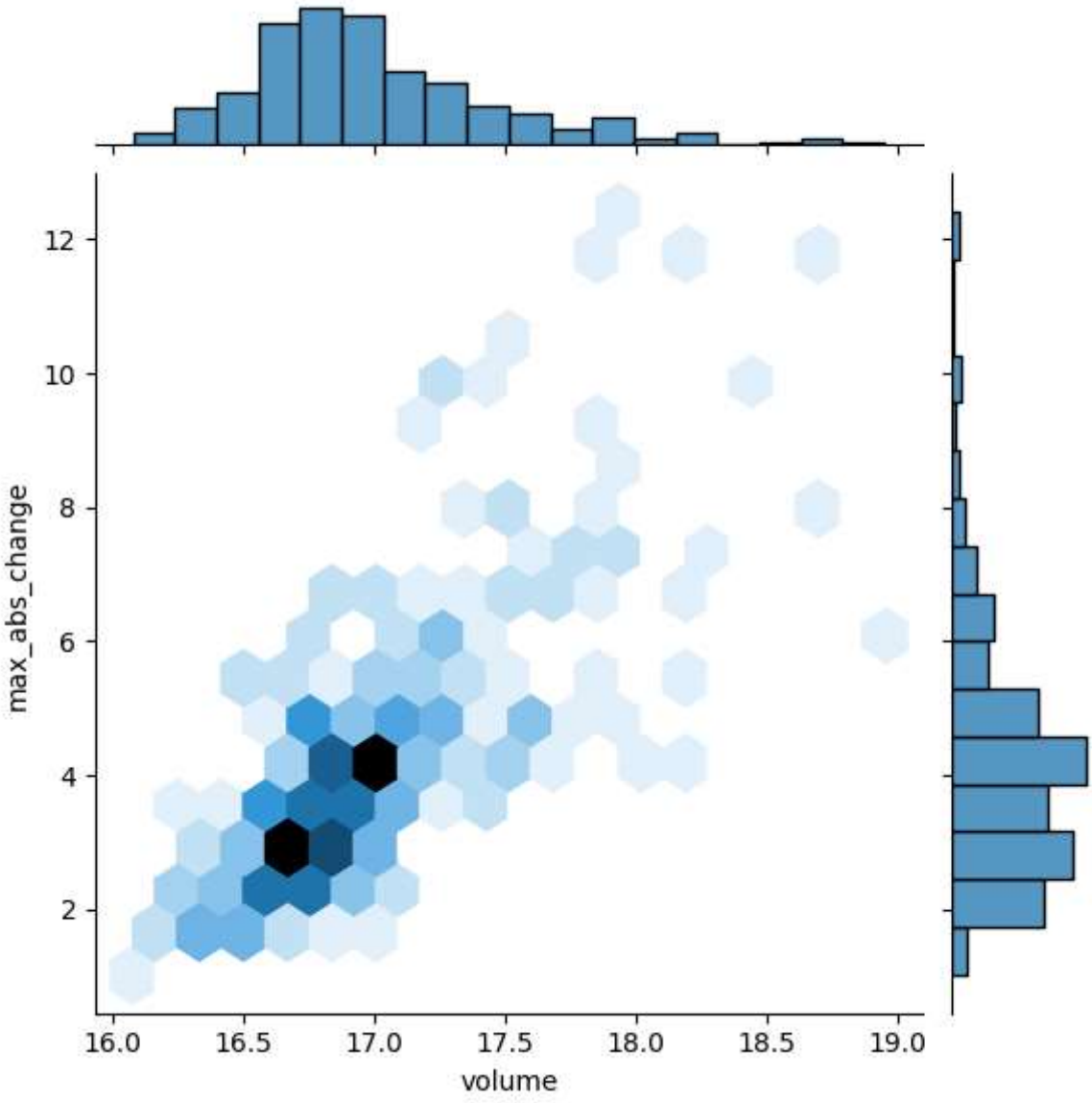
The joint plot allows us to visualize the relationship between two variables, like a scatter plot. However, we get the added benefit of being able to visualize their distributions at the same time (as a histogram or KDE). The default options give us a scatter plot in the center and histograms on the sides:

In [22]:
```
sns.jointplot(
 x='volume',
 y='max_abs_change',
 data=fb.assign(
 volume=np.log(fb.volume),
 max_abs_change=fb.high - fb.low
 )
)
plt.show()
```



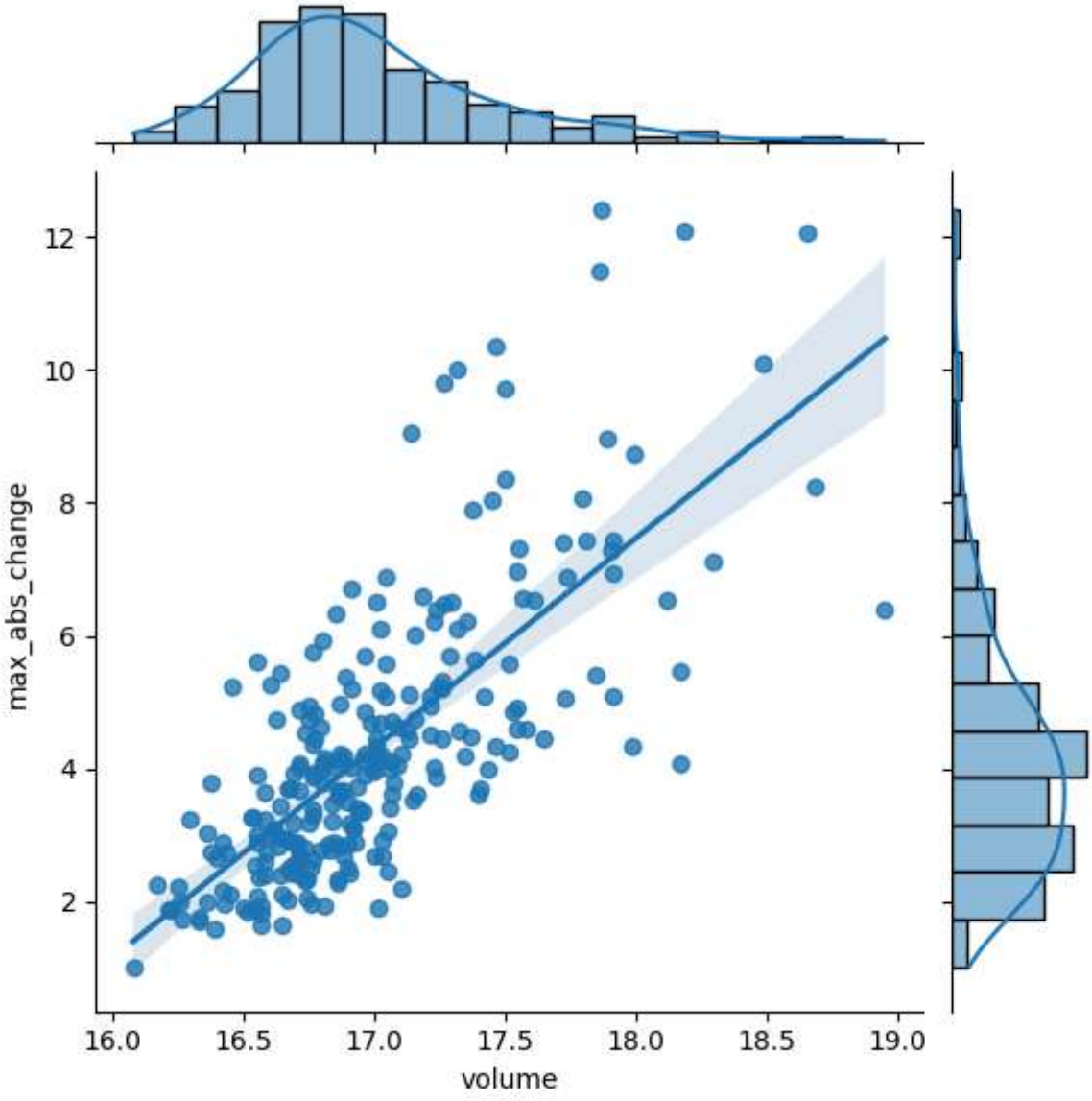
By changing the kind argument, we can change how the center of the plot is displayed. For example, we can pass kind='hex' for hexbins:

In [24]:
```
sns.jointplot(
 x='volume',
 y='max_abs_change',
 kind='hex',
 data=fb.assign(
 volume=np.log(fb.volume),
 max_abs_change=fb.high - fb.low
 )
)

plt.show()
```



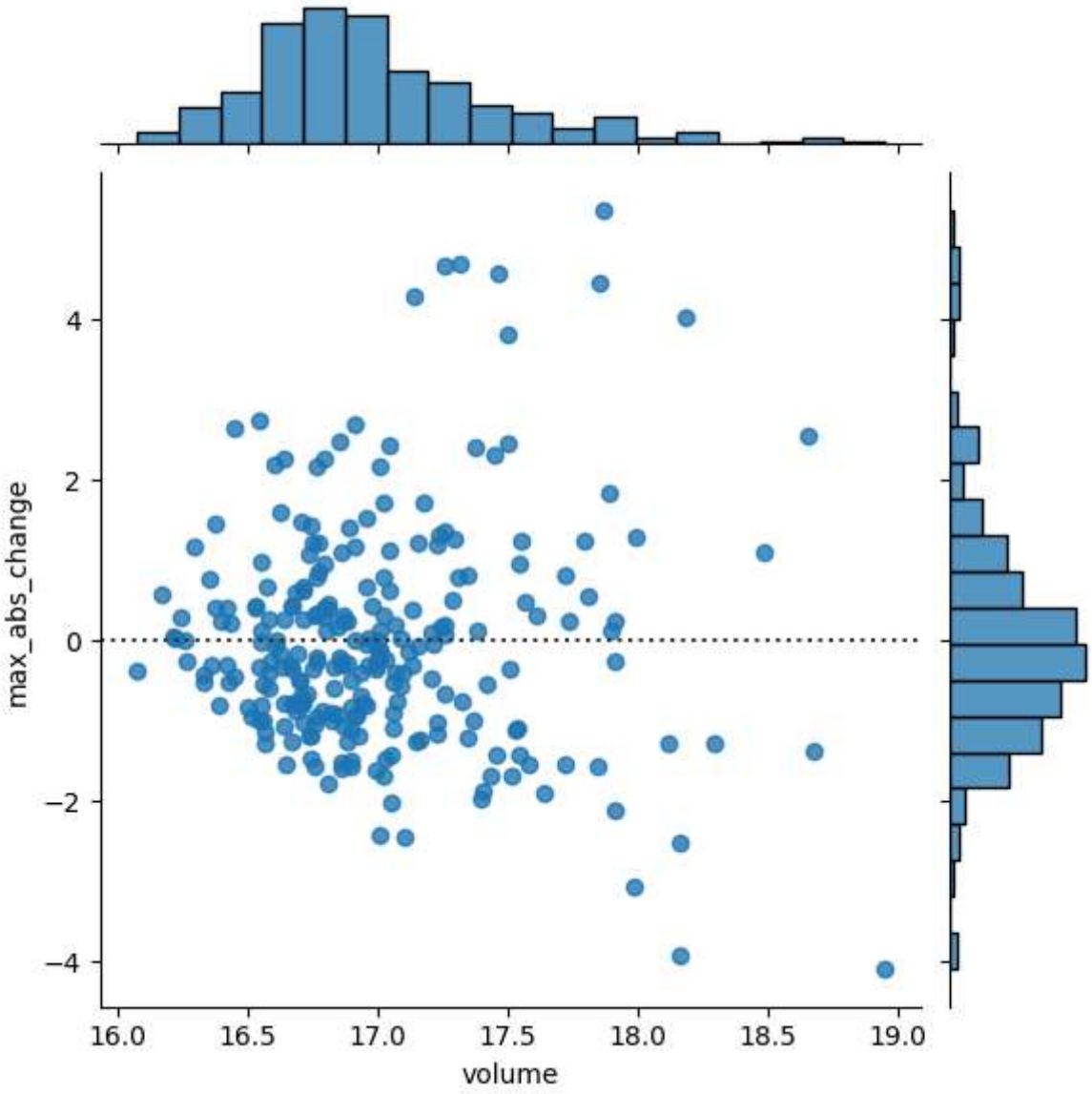
If we specify kind='reg' instead, we get a regression line in the center and KDEs on the sides:

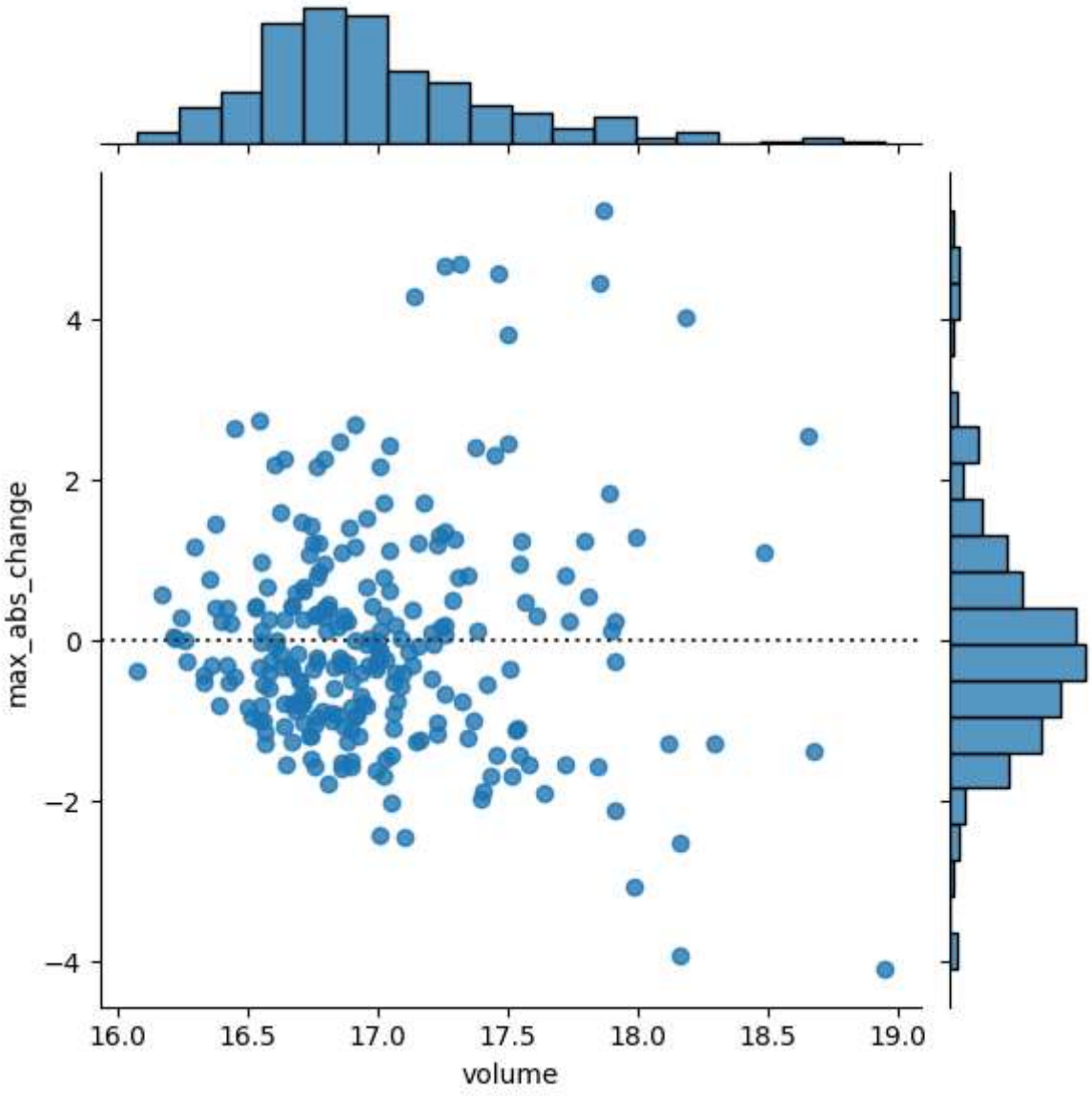
In [26]:
```
sns.jointplot(
 x='volume',
 y='max_abs_change',
 kind='reg',
 data=fb.assign(
 volume=np.log(fb.volume),
 max_abs_change=fb.high - fb.low
 )
)


plt.show()
```



If we pass kind='resid' , we get the residuals from the aforementioned regression:

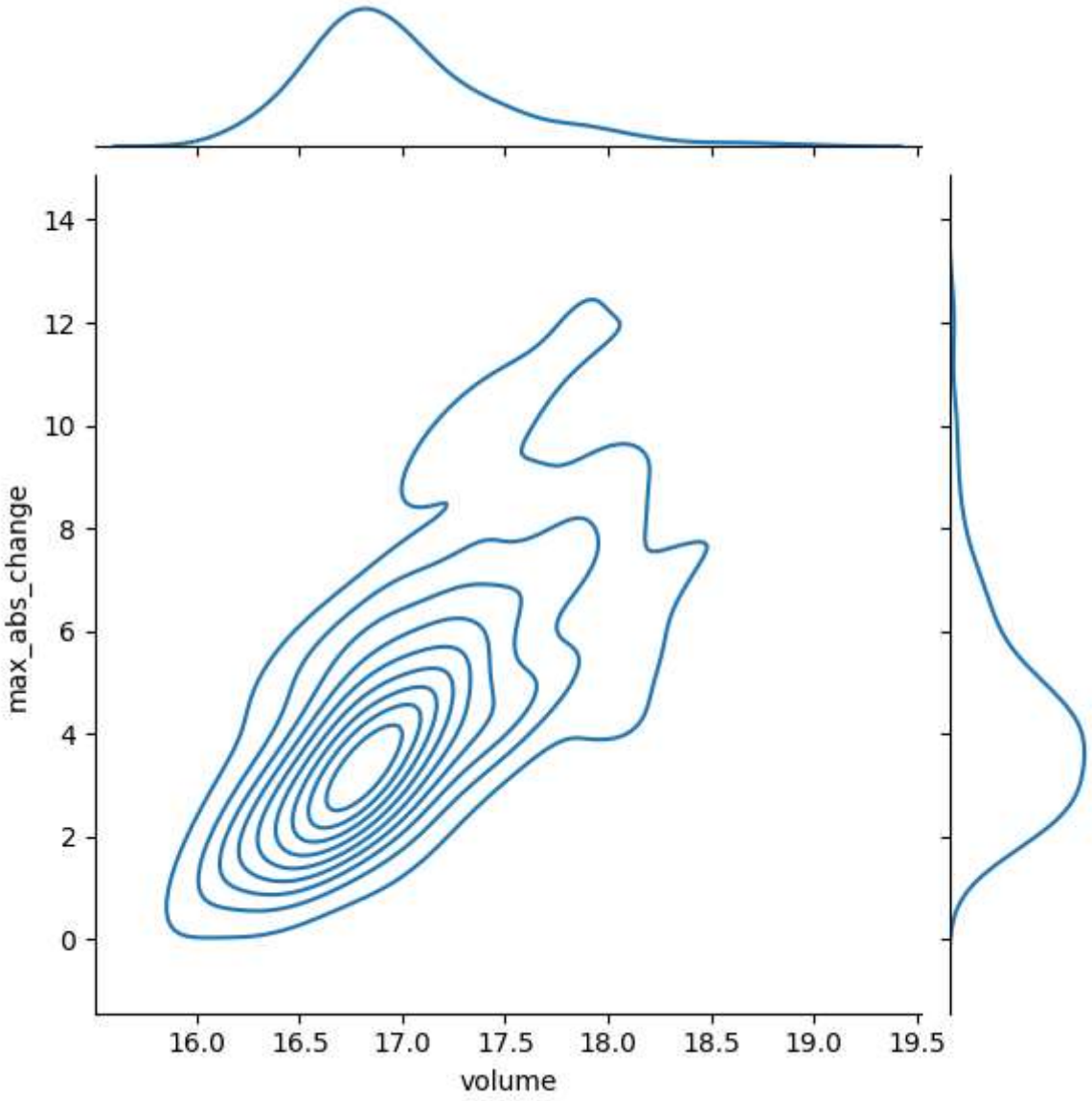
In [28]:
```
sns.jointplot(
 x='volume',
 y='max_abs_change',
 kind='resid',
 data=fb.assign(
 volume=np.log(fb.volume),
 max_abs_change=fb.high - fb.low
 )
)

plt.show()
```


max_abs_change
volume
4
2
0
−2
−4
16.0
16.5
17.0
17.5
18.0
18.5
19.0



Finally, if we pass kind='kde' , we get a contour plot of the joint density estimate with KDEs along the sides:

In [29]:
```
sns.jointplot(
 x='volume',
 y='max_abs_change',
 kind='kde',
 data=fb.assign(
 volume=np.log(fb.volume),
 max_abs_change=fb.high - fb.low
 )
)

plt.show()
```



Regression plots

We are going to use seaborn to visualize a linear regression between the log of the volume traded in Facebook stock and the maximum absolute daily change (daily high stock price - daily low stock price). To do so, we first need to isolate this data:

In [31]:
```
fb_reg_data = fb.assign(
 volume=np.log(fb.volume),
 max_abs_change=fb.high - fb.low
).iloc[:,-2:]
```

Since we want to visualize each column as the regressor, we need to look at permutations of their order. Permutations and combinations (among other things) are made easy in Python with itertools , so let's import it:

In [32]:
```
import itertools
```

itertools gives us efficient iterators. Iterators are objects that we loop over, exhausting them. This is an iterator from itertools ; notice how the second loop doesn't do anything:

In [33]:
```
iterable = list(itertools.repeat("I'm an iterable", 1))
for i in iterable:
 print(f'-->{i}')
print('This prints again because it\'s an iterable:')
for i in iterable:
 print(f'-->{i}')
```

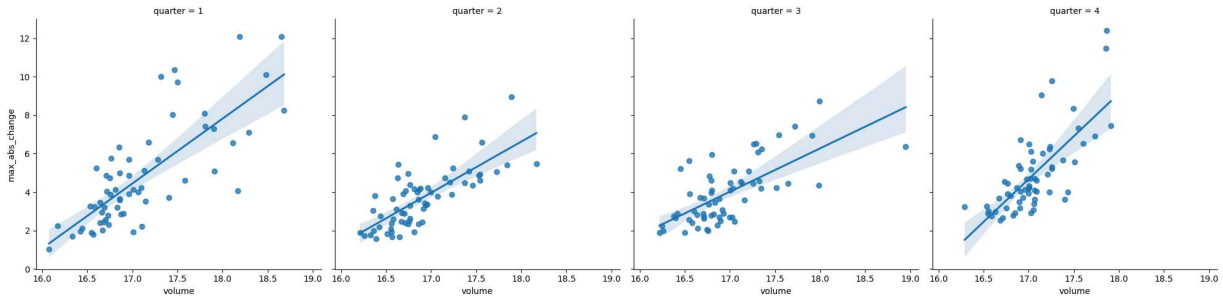
```
-->I'm an iterable
This prints again because it's an iterable:
-->I'm an iterable
```

The reg_resid_plots() function from the reg_resid_plot.py module in this folder uses regplot() and residplot() from seaborn along with itertools to plot the regression and residuals side-by-side:

In [ ]:
```
from reg_resid_plot import reg_resid_plots
reg_resid_plots(fb_reg_data)
```

We can use lmplot() to split our regression across subsets of our data. For example, we can perform a regression per quarter on the Facebook stock data:

In [39]:
```
sns.lmplot(
 x='volume',
 y='max_abs_change',
 data=fb.assign(
 volume=np.log(fb.volume),
 max_abs_change=fb.high - fb.low,
 quarter=lambda x: x.index.quarter
 ),
 col='quarter'
)

plt.show()
```



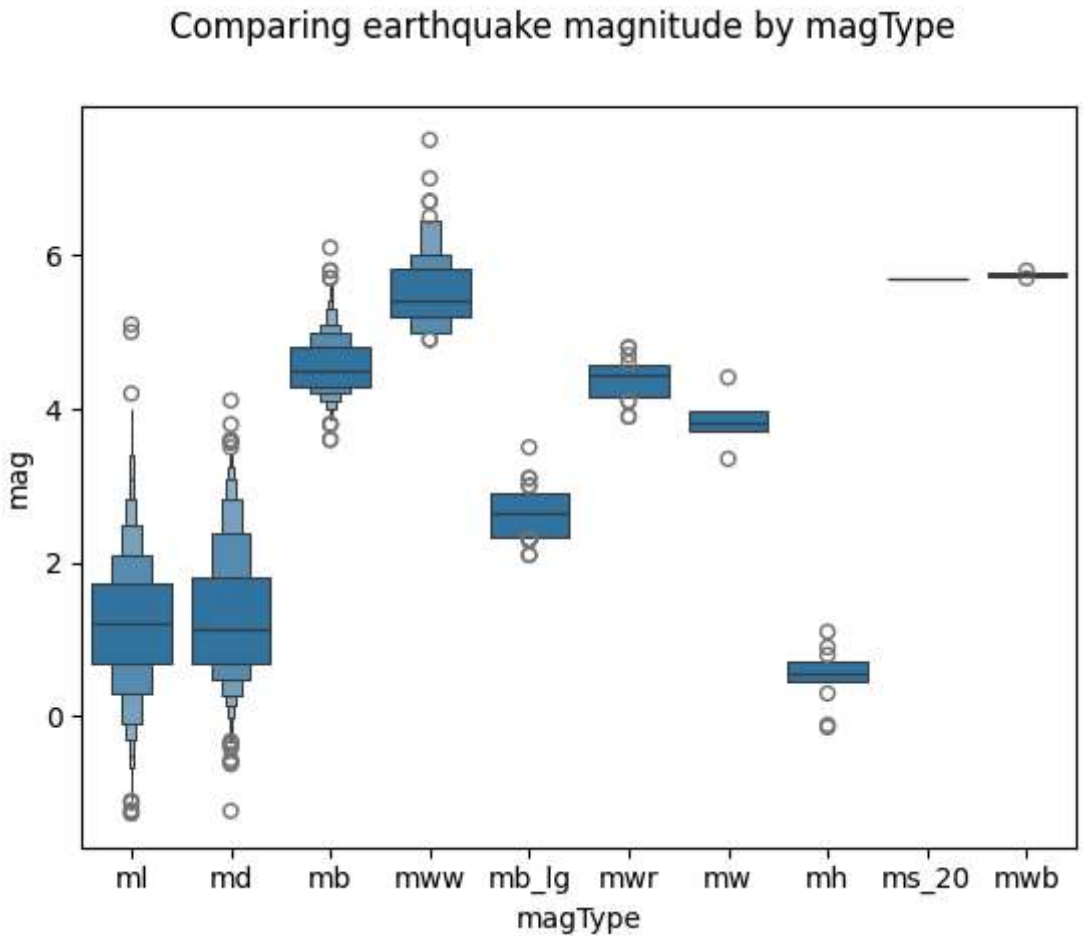
Distributions

Seaborn provides some new plot types for visualizing distributions in additional to its own versions of the plot types we discussed in chapter 5 (in this notebook).

boxenplot()

The boxenplot is a box plot that shows additional quantiles

In [42]:
```
sns.boxenplot(
 x='magType', y='mag', data=quakes[['magType', 'mag']]
)
plt.suptitle('Comparing earthquake magnitude by magType')
plt.show()
```



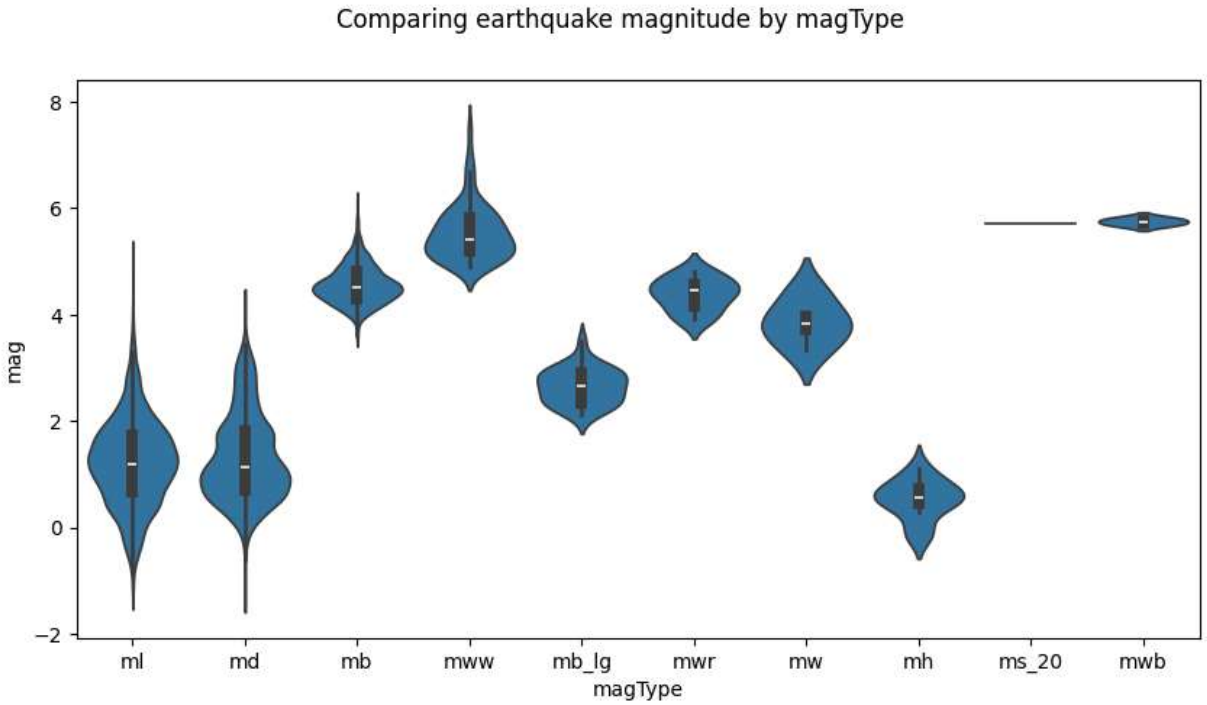
violinplot()

Box plots lose some information about the distribution, so we can use violin plots which combine box plots and KDEs:

In [45]:
```
fig, axes = plt.subplots(figsize=(10, 5)) #
sns.violinplot(
 x='magType', y='mag', data=quakes[['magType', 'mag']],
 ax=axes, scale='width' # all violins have same width
)
```

```
plt.suptitle('Comparing earthquake magnitude by magType')
plt.show()
```

```
C:\Users\Arnel Bulambao\AppData\Local\Temp\ipykernel_10320\577471595.py:2: FutureWar
ning:

The `scale` parameter has been renamed and will be removed in v0.15.0. Pass `density
_norm='width'` for the same effect.
  sns.violinplot(
```



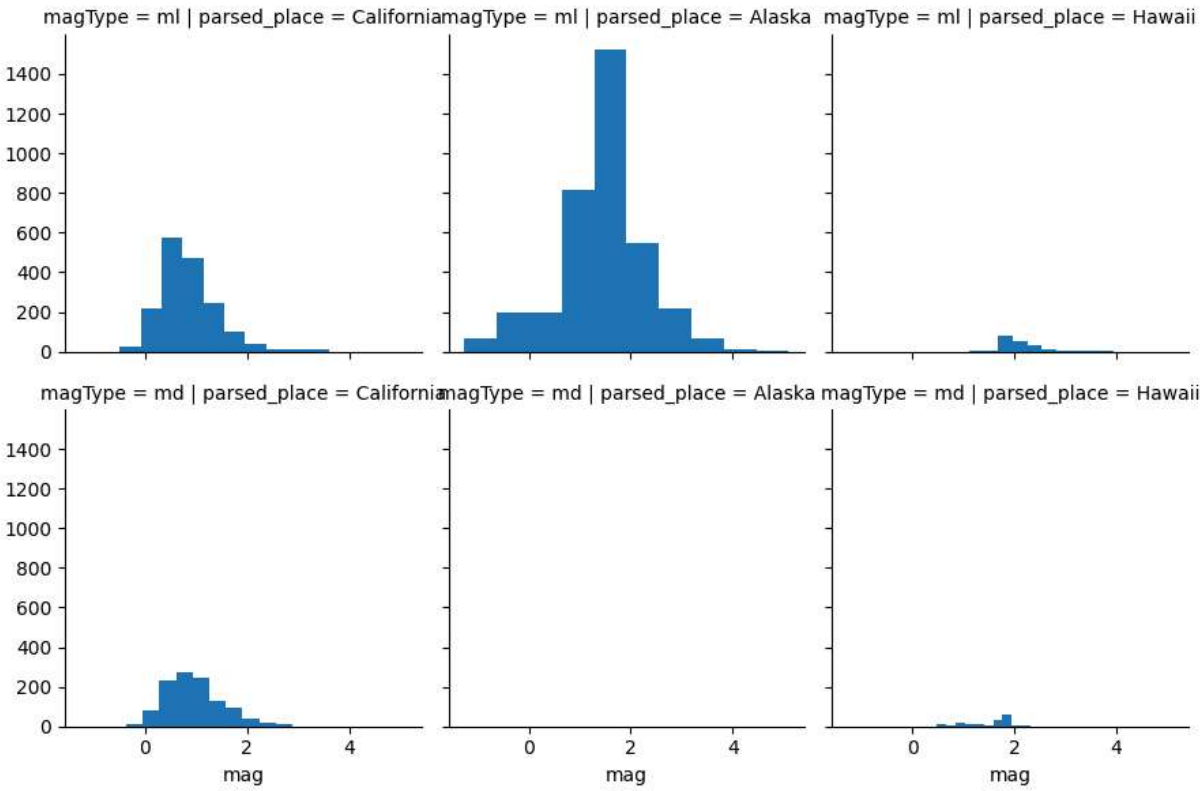
Faceting

We can create subplots across subsets of our data by faceting. First, we create a FacetGrid specifying how to layout the plots (which categorical column goes along the rows and which one along the columns). Then, we call the map() method of the FacetGrid and pass in the plotting function we want to use (along with any additional arguments).

Let's make histograms showing the distribution of earthquake magnitude in California, Alaska, and Hawaii faceted by magType and parse_placed :

In [46]:
```
g = sns.FacetGrid(
 quakes[
 (quakes.parsed_place.isin([
 'California', 'Alaska', 'Hawaii'
 ]))\
 & (quakes.magType.isin(['ml', 'md']))
 ],
 row='magType',
 col='parsed_place'
)
g = g.map(plt.hist, 'mag')
```

```
plt.show()
```



9.5 Formatting Plots

Formatting Plots

About the Data

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

Setup

In [47]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
fb = pd.read_csv(
 'fb_stock_prices_2018.csv', 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

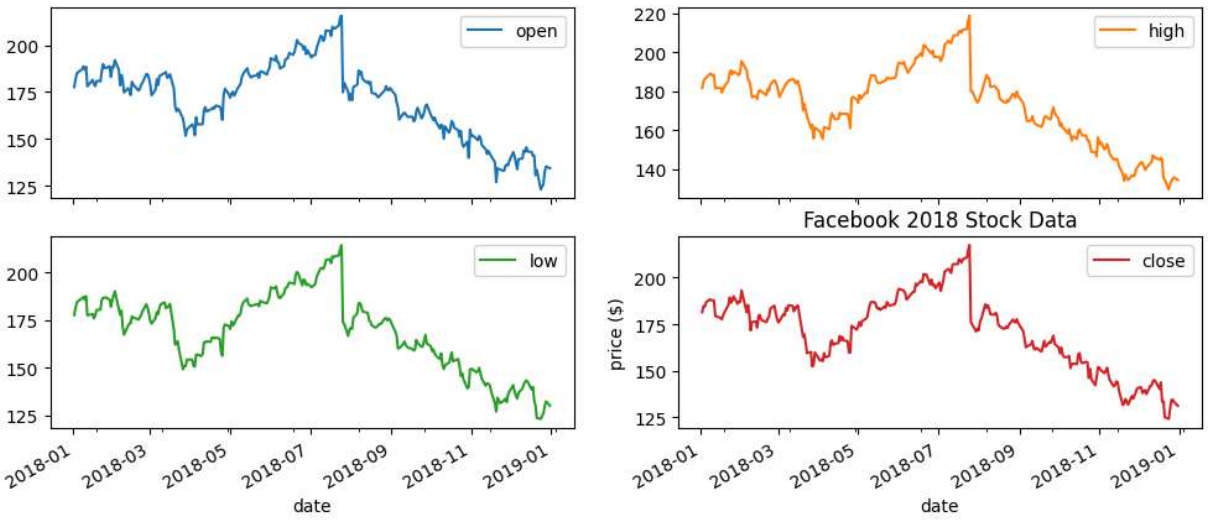
In [50]:
```
fb.close.plot()
plt.suptitle('FB Closing Price')
plt.xlabel('date')
plt.ylabel('price ($)')
plt.show()
```

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

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

In [53]:
```
fb.iloc[:,:4].plot(subplots=True, layout=(2, 2), figsize=(12, 5))
plt.title('Facebook 2018 Stock Data')
```

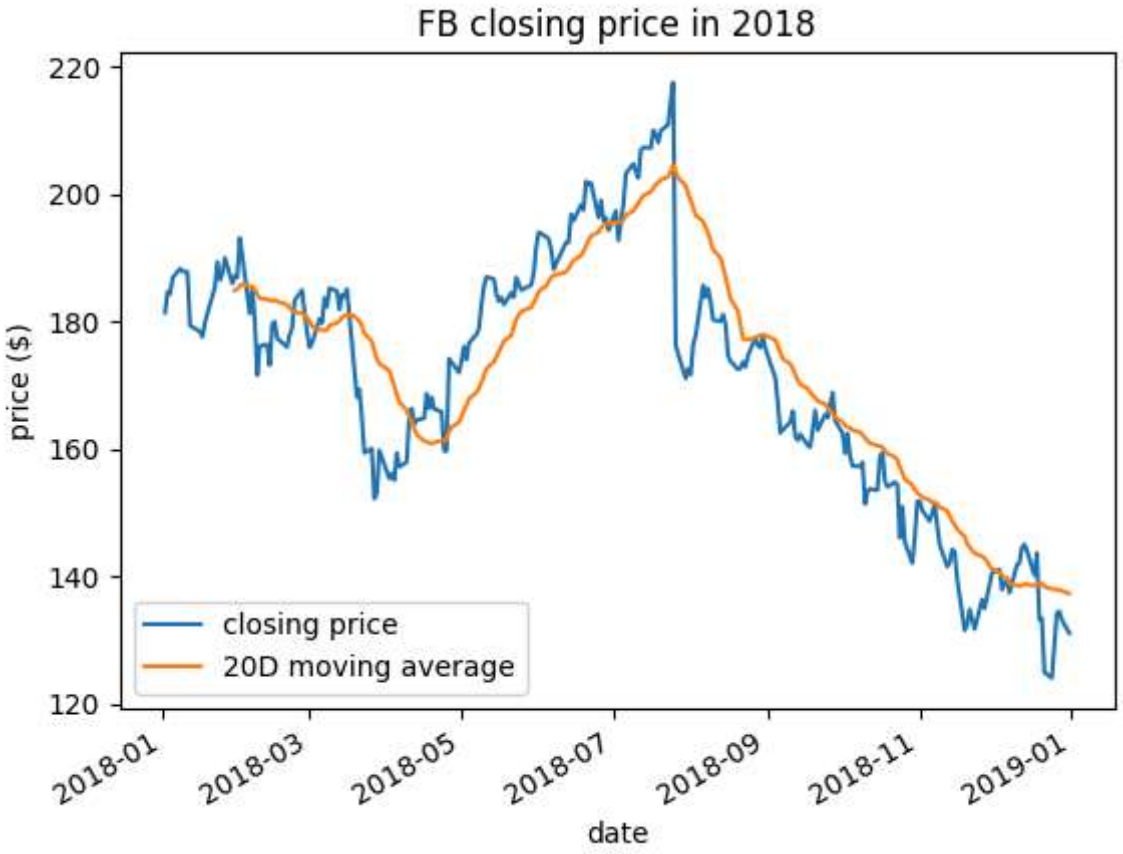
```
plt.xlabel('date')
plt.ylabel('price ($)')
plt.show()
```



Legends

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

In [56]:
```
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 ($)')
plt.show()
```

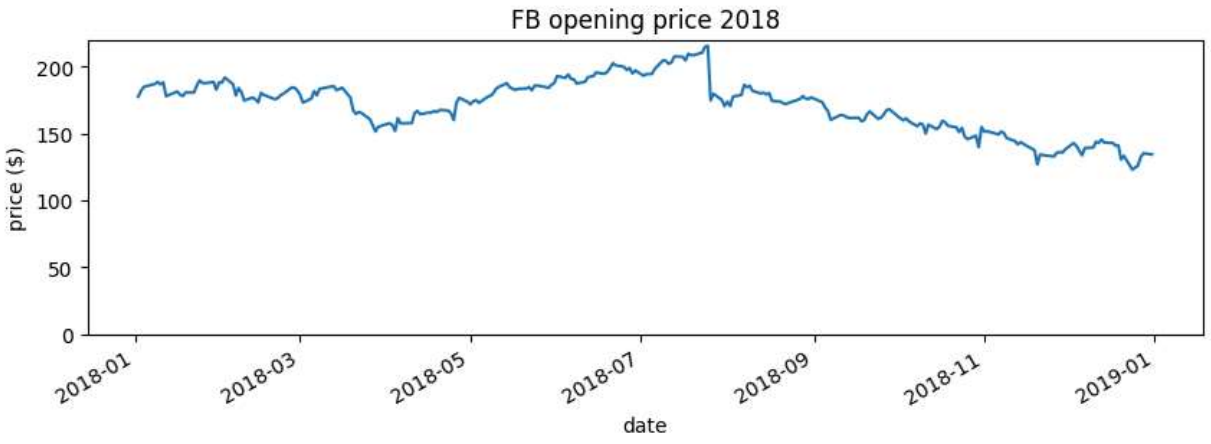


Formatting Axes

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 [58]:
```
fb.open.plot(figsize=(10, 3), title='FB opening price 2018')
plt.ylim(0, None)
plt.ylabel('price ($)')
plt.show()
```



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 [61]:
```
import calendar
fb.open.plot(figsize=(10, 3), rot=0, title='FB opening price 2018')
locs, labels = plt.xticks()
plt.xticks(locs[:6] + 15 , calendar.month_name[1::2])
plt.ylabel('price ($)')
plt.show()
```

PercentFormatter

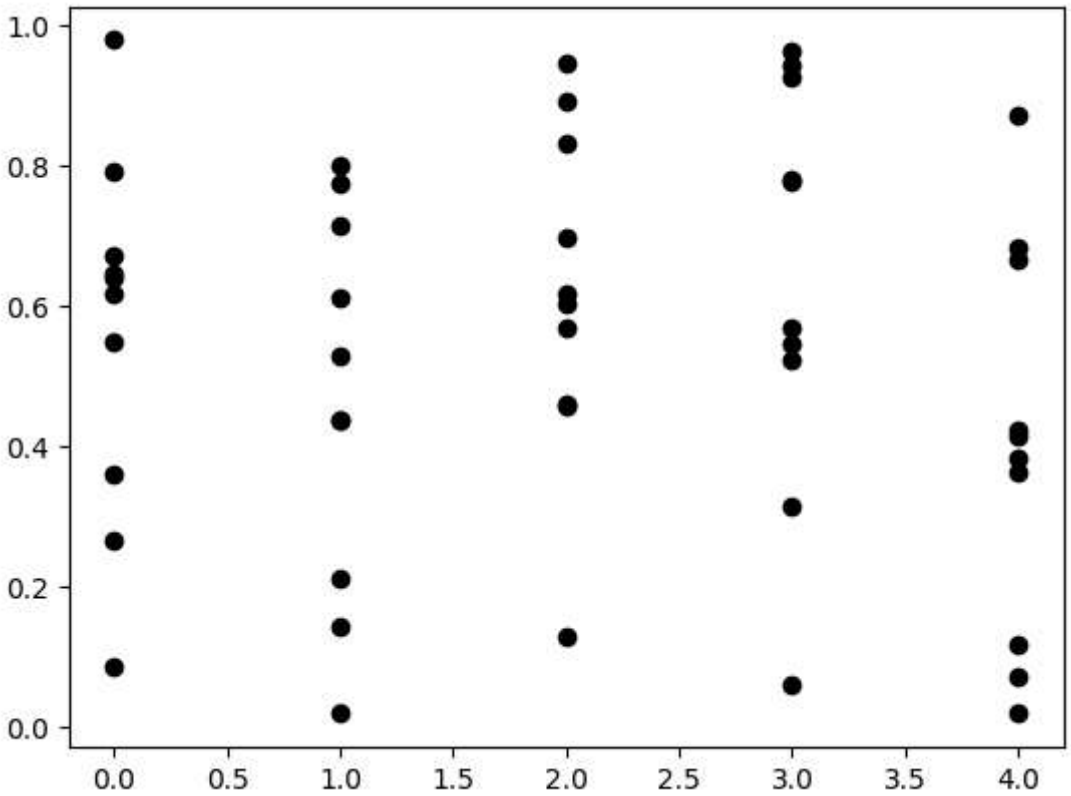
We can use ticker.PercentFormatter and specify the denominator (xmax) to use when calculating the percentages. This gets passed to the set_major_formatter() method of the xaxis or yaxis on the Axes .

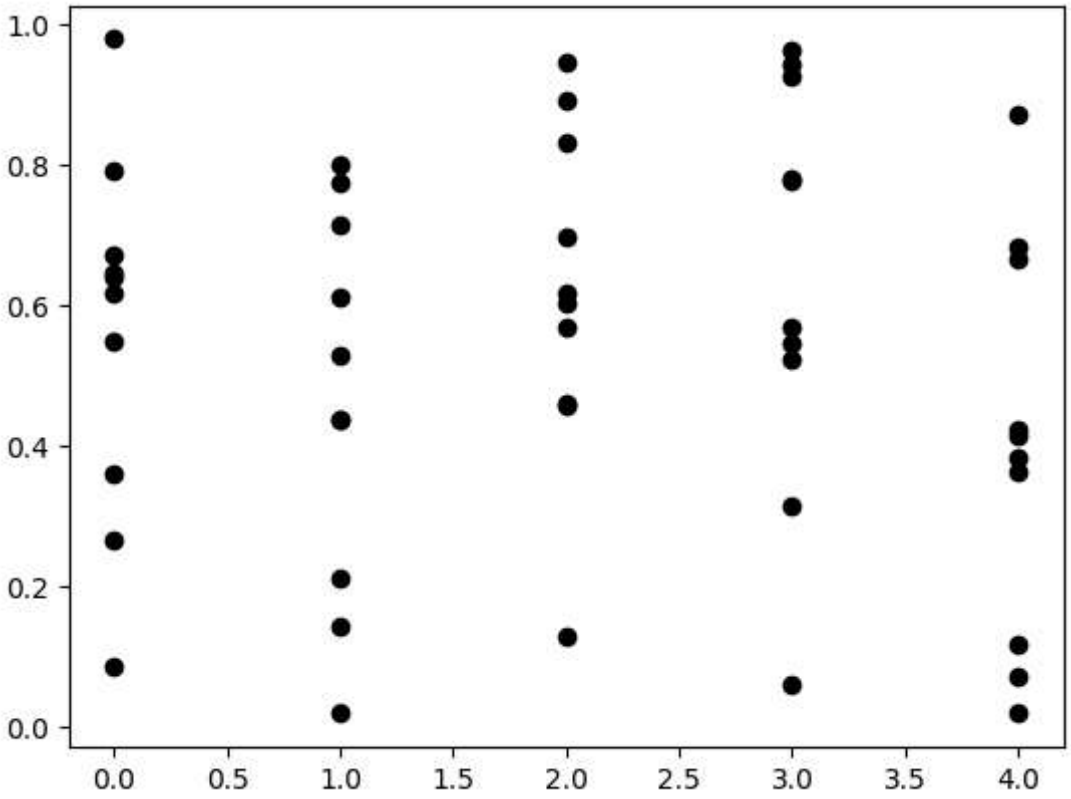
In [62]:
```
import matplotlib.ticker as ticker
ax = fb.close.plot(
 figsize=(10, 4),
 title='Facebook Closing Price as Percentage of Highest Price in Time Range'
)

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)
])
ax.set_ylabel(f'percent of highest price (${fb.high.max()})')

plt.show()
```

MultipleLocator

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

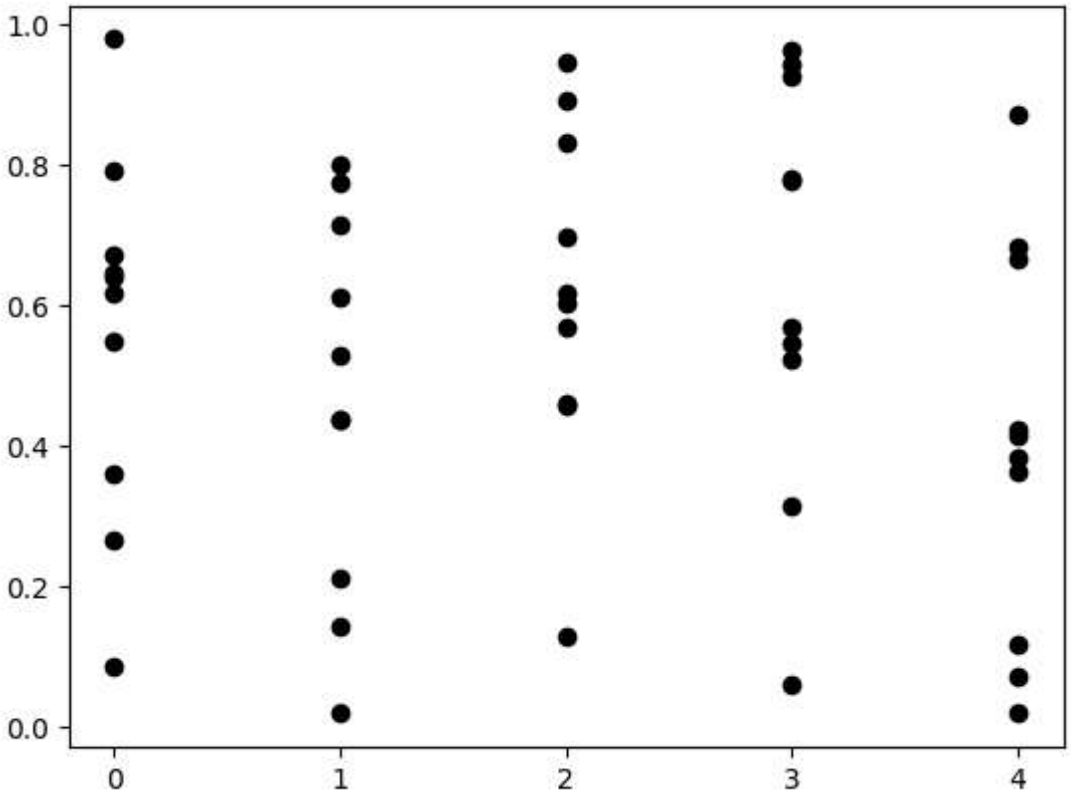
In [64]:
```
fig, ax = plt.subplots(1, 1)
np.random.seed(0)
ax.plot(np.tile(np.arange(0, 5), 10), np.random.rand(50), 'ko')

plt.show()
```





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 [65]:
```
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)
)


plt.show()
```



9.6 Customizing Visualizations

pandas.plotting subpackage

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

About the Data

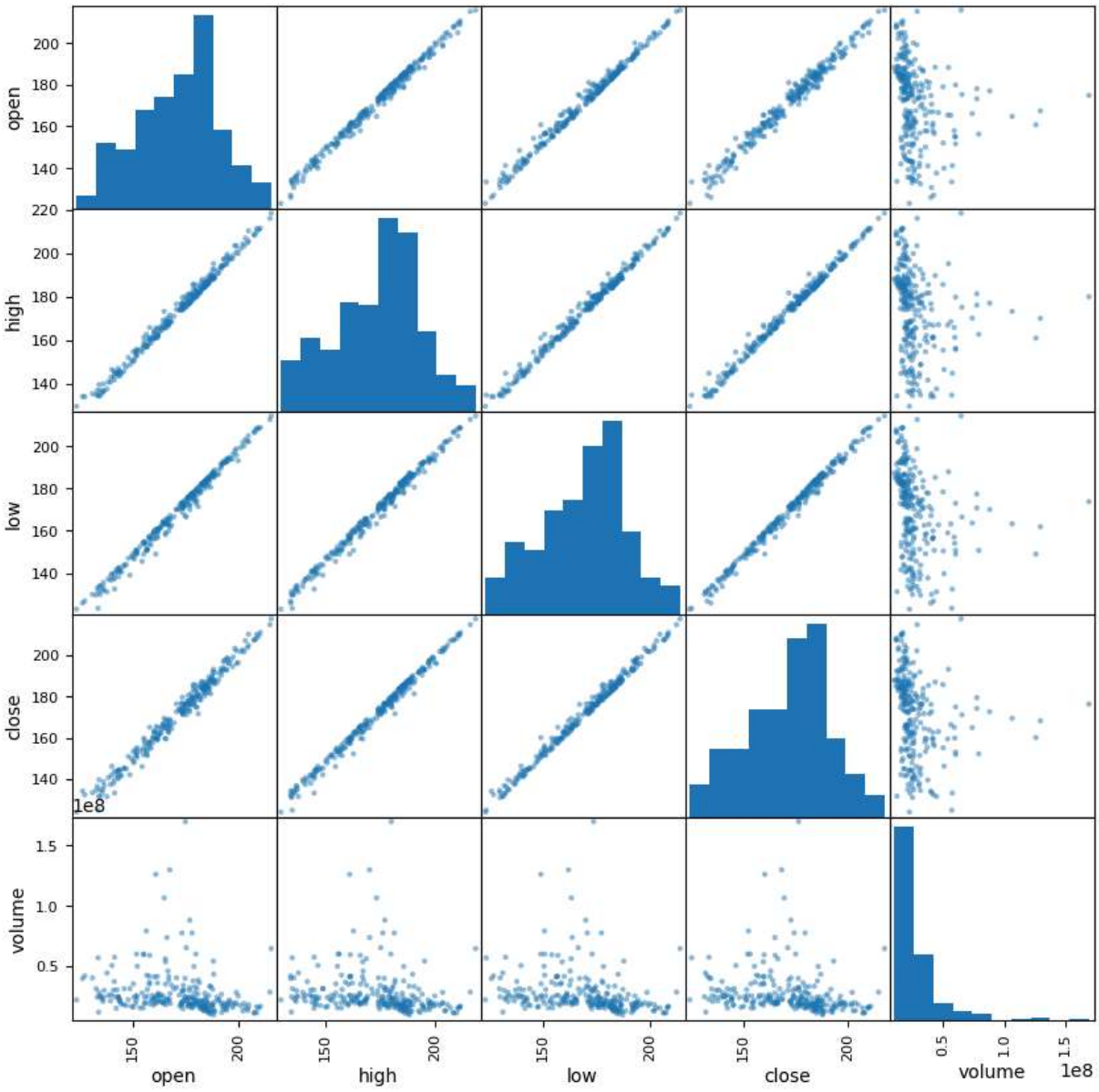
In this notebook, we will be working with Facebook's stock price throughout 2018 (obtained using the stock_analysis package).

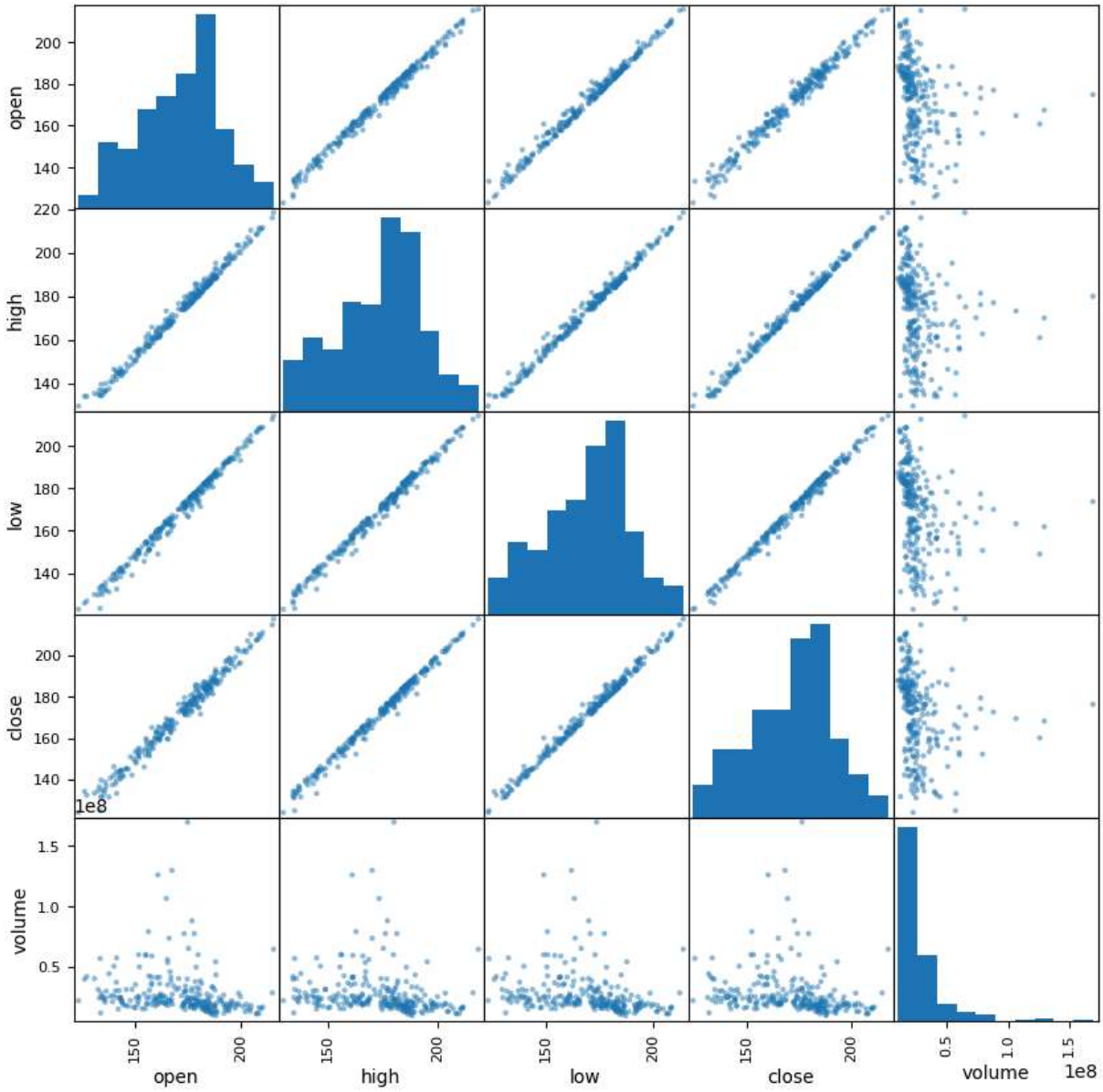
Setup

In [66]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
fb = pd.read_csv(
 'fb_stock_prices_2018.csv', index_col='date', parse_dates=True
)
```

In [68]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(fb, figsize=(10, 10))
```

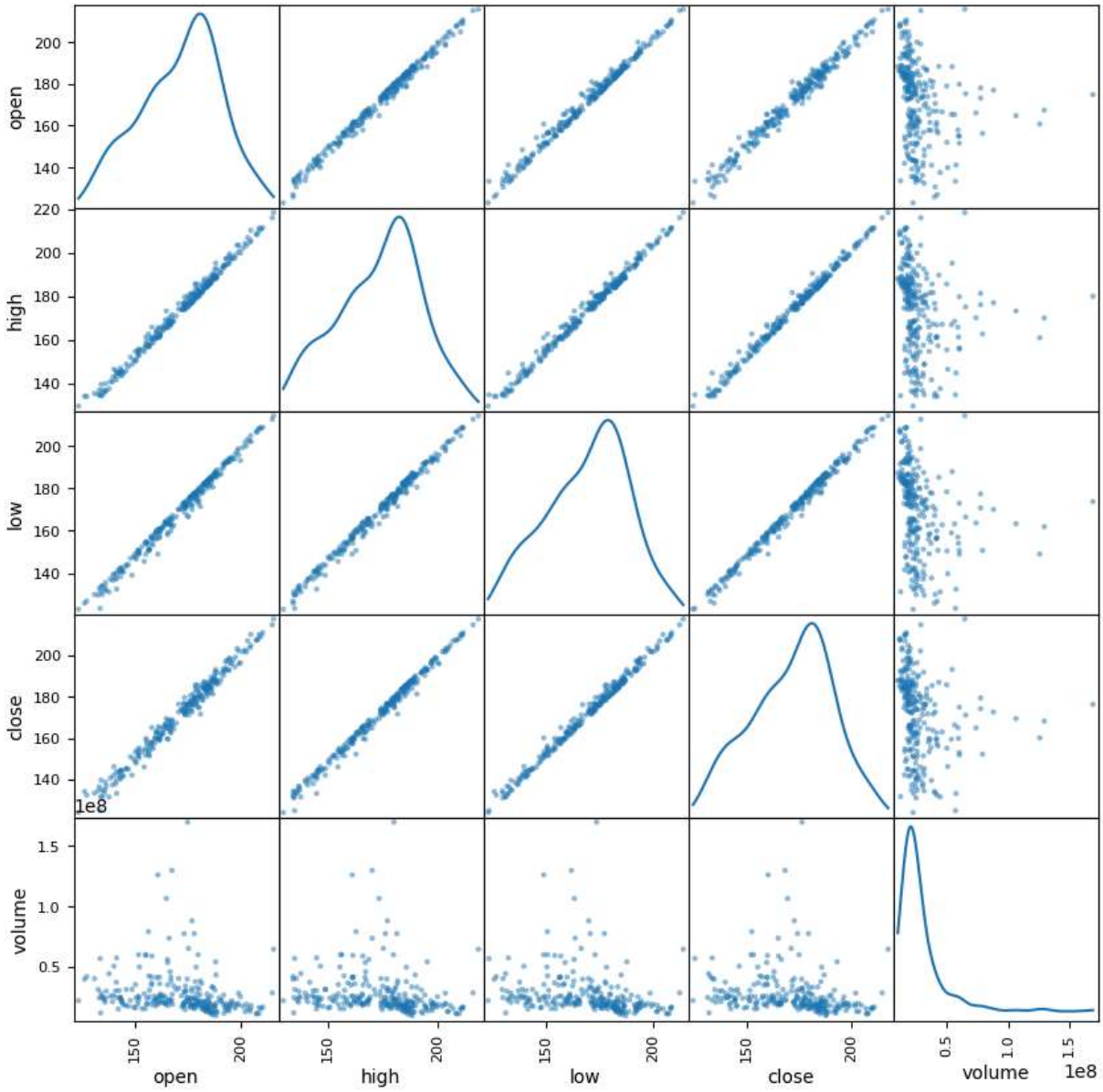
```
plt.show()
```





Changing the diagonal from histograms to KDE:

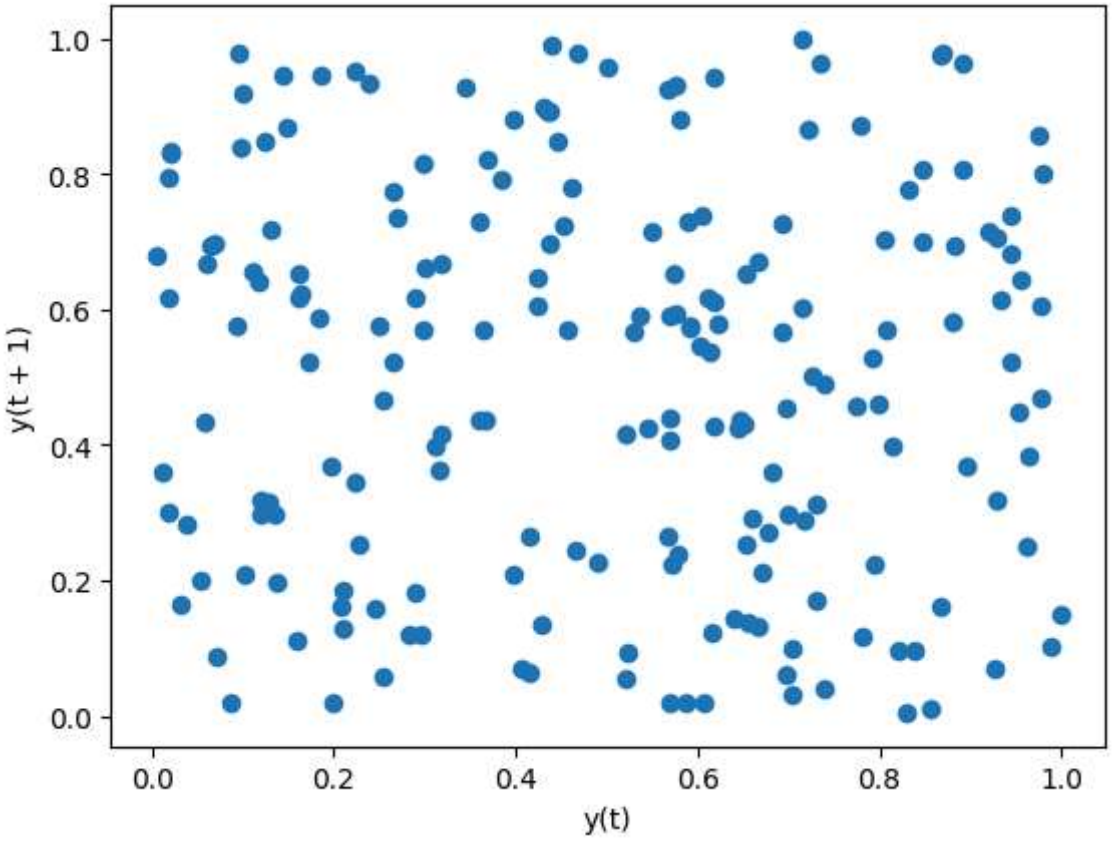
In [69]:
```
scatter_matrix(fb, figsize=(10, 10), diagonal='kde')

plt.show()
```



Lag plot

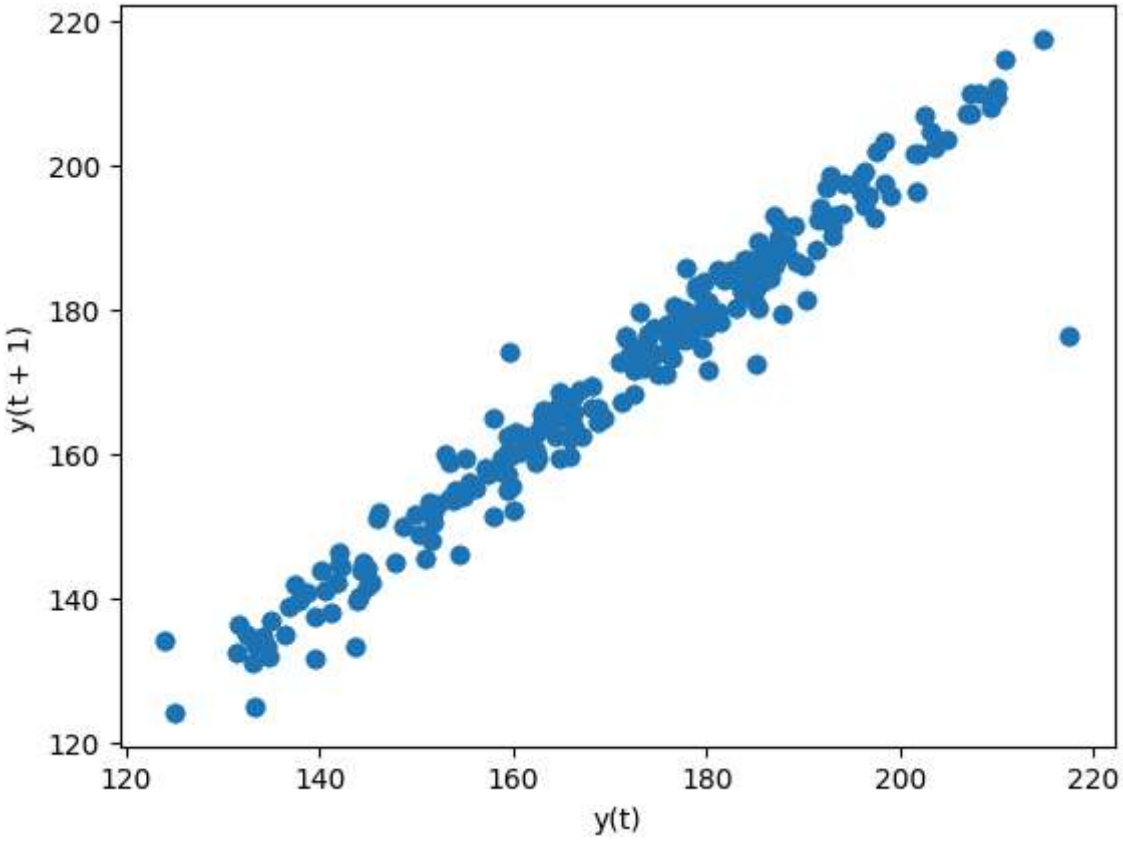
Lag plots let us see how the variable correlations with past observations of itself. Random data has no pattern:

In [70]:
```
from pandas.plotting import lag_plot
np.random.seed(0)
lag_plot(pd.Series(np.random.random(size=200)))

plt.show()
```



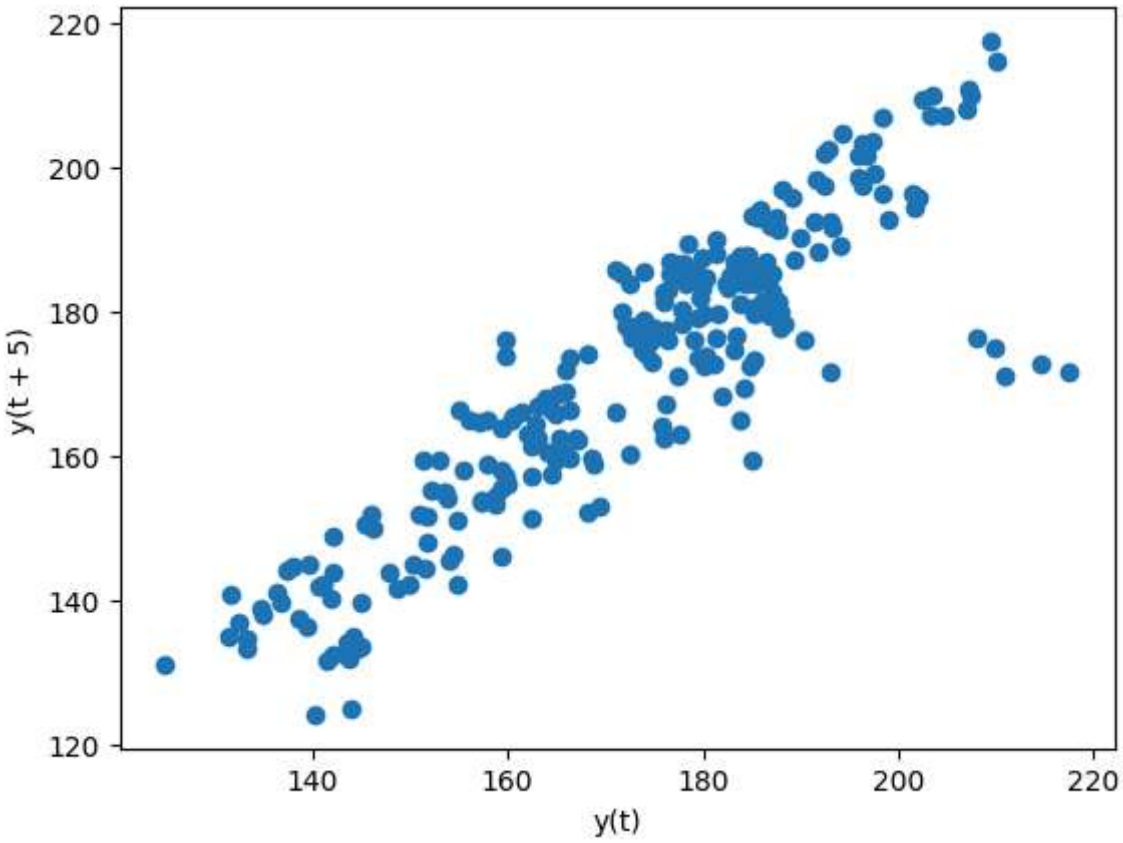
Data with some level of correlation to itself (autocorrelation) may have patterns. Stock prices are highly auto-correlated:

In [72]:
```
lag_plot(fb.close)

plt.show()
```



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):

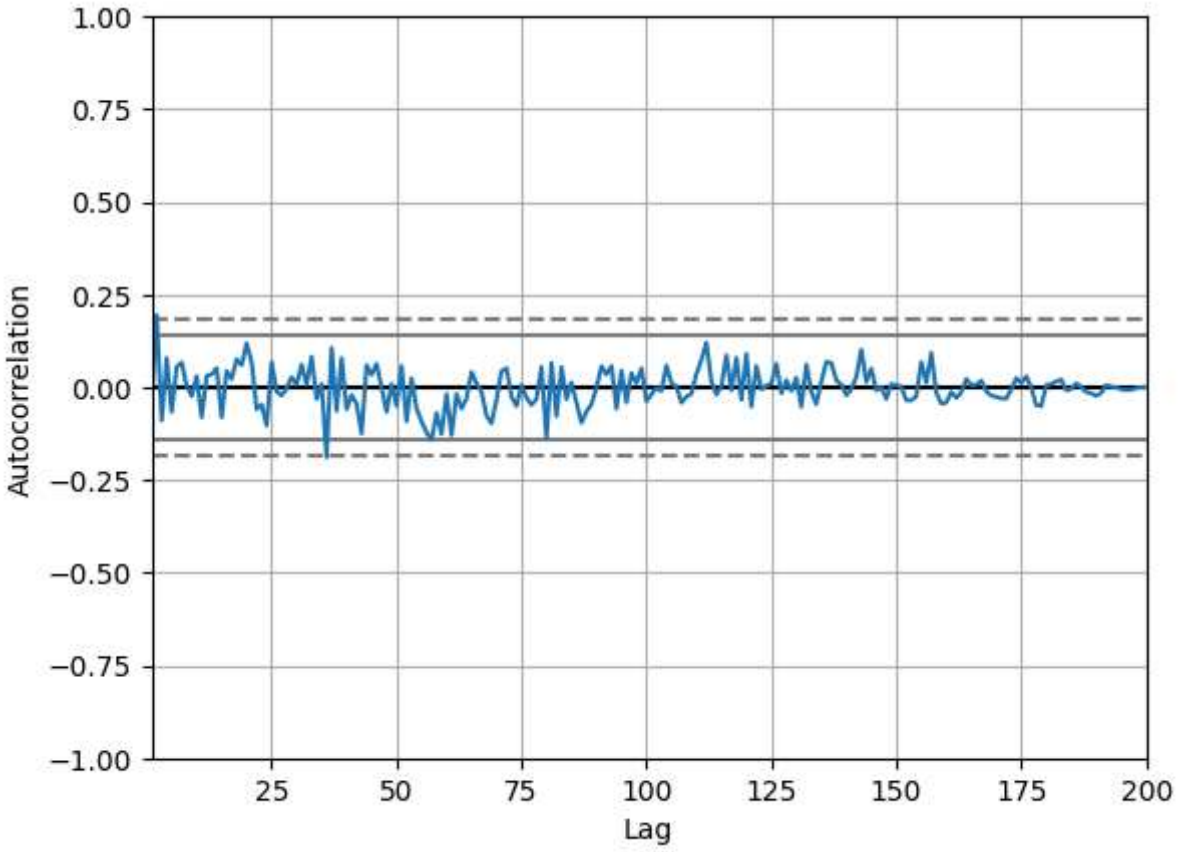
In [73]:
```
lag_plot(fb.close, lag=5)

plt.show()
```



Autocorrelation plots

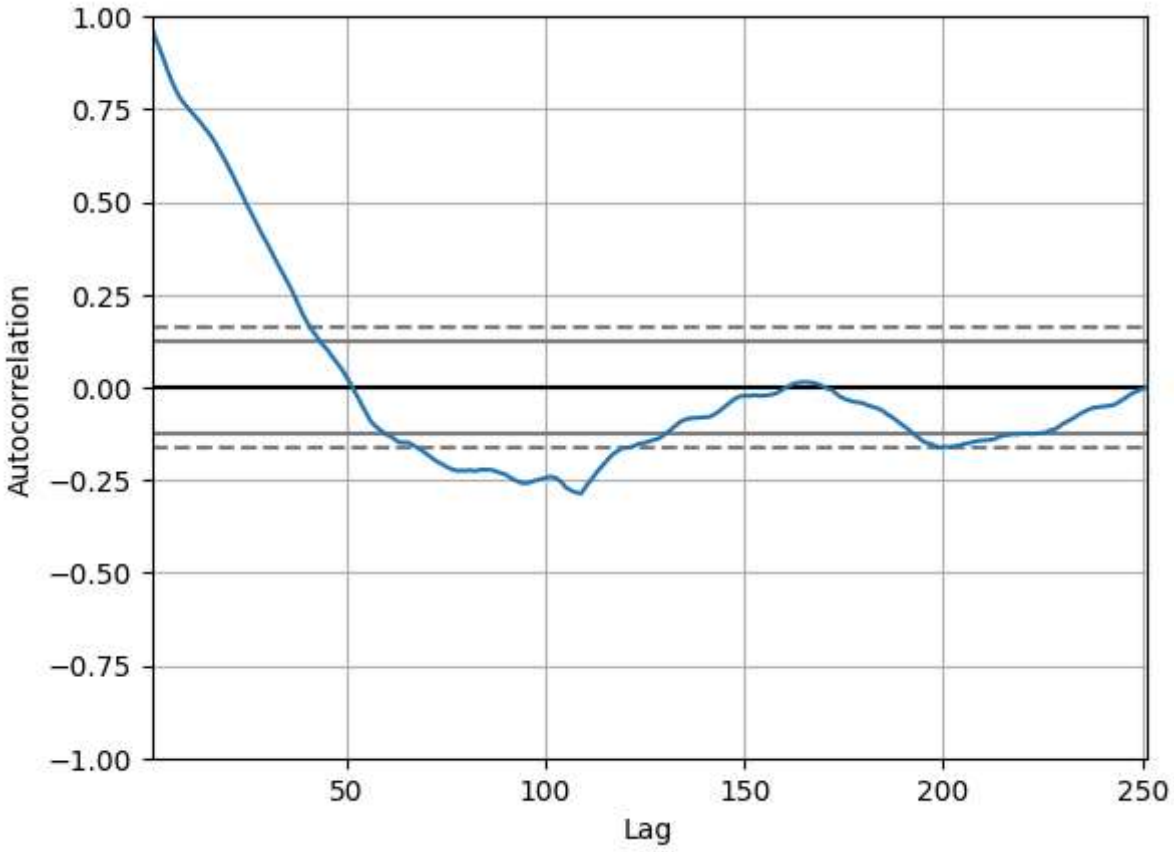
We can use the autocorrelation plot to see if this relationship may be meaningful or just noise. Random data will not have any significant autocorrelation (it stays within the bounds below):

In [74]:
```
from pandas.plotting import autocorrelation_plot
np.random.seed(0)
autocorrelation_plot(pd.Series(np.random.random(size=200)))
plt.show()
```



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

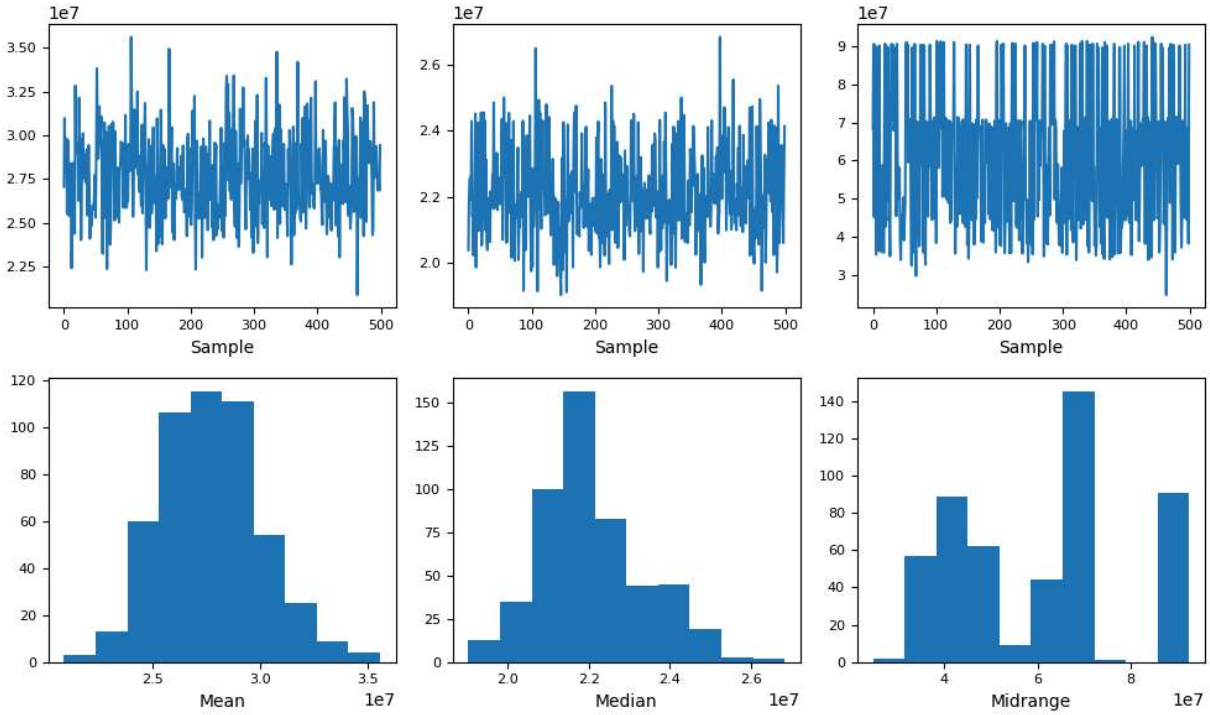
In [75]:
```
autocorrelation_plot(fb.close)
plt.show()
```



Bootstrap plot

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

In [77]:
```
from pandas.plotting import bootstrap_plot
fig = bootstrap_plot(fb.volume, fig=plt.figure(figsize=(10, 6)))
plt.show()
```



Data Analysis:

Provide comments on output from the procedures

Supplementary Activity:

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

Preprocessing

In [8]:
```
import pandas as pd
fb = pd.read_csv('fb_stock_prices_2018.csv') # fb stock prices 2018

earth = pd.read_csv('earthquakes-1.csv') # earth quakes-1
```

In [11]:
```
fb.head() # checking values
```

Out[11]:

	date	open	high	low	close	volume
0	2018-01-02	177.68	181.58	177.5500	181.42	18151903
1	2018-01-03	181.88	184.78	181.3300	184.67	16886563
2	2018-01-04	184.90	186.21	184.0996	184.33	13880896
3	2018-01-05	185.59	186.90	184.9300	186.85	13574535
4	2018-01-08	187.20	188.90	186.3300	188.28	17994726

In [13]:
```
fb.info() # checking for null and data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 251 entries, 0 to 250
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   date    251 non-null    object 
 1   open    251 non-null    float64
 2   high    251 non-null    float64
 3   low     251 non-null    float64
 4   close   251 non-null    float64
 5   volume  251 non-null    int64  
dtypes: float64(4), int64(1), object(1)
memory usage: 11.9+ KB
```

In [19]:
```
fb['date'] = pd.to_datetime(fb['date']) # pd to datetime for changing data types
```

In [22]:
```
fb.dtypes # checking
```

Out[22]:
```
date      datetime64[ns]
open             float64
high             float64
low              float64
close            float64
volume             int64
dtype: object
```

In [25]:
```
earth.head() # checking values
```

Out[25]:

	mag	magType	time	place	tsunami	parsed_place
0	1.35	ml	1539475168010	9km NE of Aguanga, CA	0	California
1	1.29	ml	1539475129610	9km NE of Aguanga, CA	0	California
2	3.42	ml	1539475062610	8km NE of Aguanga, CA	0	California
3	0.44	ml	1539474978070	9km NE of Aguanga, CA	0	California
4	2.16	md	1539474716050	10km NW of Avenal, CA	0	California

In [28]:
```
earth.info() # checking for null and data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9332 entries, 0 to 9331
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mag           9331 non-null   float64
 1   magType       9331 non-null   object 
 2   time          9332 non-null   int64  
 3   place         9332 non-null   object 
 4   tsunami       9332 non-null   int64  
 5   parsed_place  9332 non-null   object 
dtypes: float64(1), int64(2), object(3)
memory usage: 437.6+ KB
```

In [31]:
```
earth[earth.isnull().any(axis=1)] # finding null value row
```

Out[31]:

	mag	magType	time	place	tsunami	parsed_place
6404	NaN	NaN	1537906325240	13km NW of Parkfield, CA	0	California

In [34]:
```
earth.dropna(inplace =True) # dropping the single nan value
```

In [37]:
```
earth.isnull().sum() # checking
```

Out[37]:
```
mag             0
magType         0
time            0
place           0
tsunami         0
parsed_place    0
dtype: int64
```

Supplementary Activity:

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

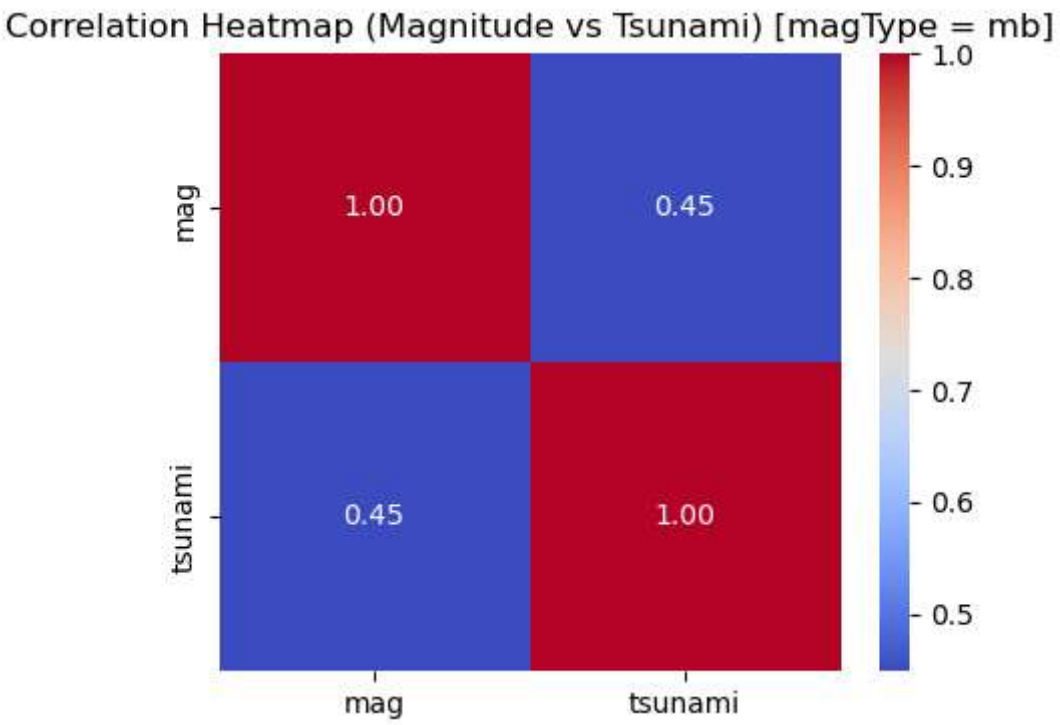
1. Using seaborn, create a heatmap to visualize the correlation coefficients between earthquake magnitude and whether there was a tsunami with the magType of mb.
2. Create a box plot of Facebook volume traded and closing prices, and draw reference lines for the bounds of a Tukey fence with a multiplier of 1.5. The bounds will be at Q1 - 1.5 * IQR and Q3 + 1.5 * IQR. Be sure to use the quantile() method on the data to make this easier. (Pick whichever orientation you prefer for the plot, but make sure to use subplots.)
3. Fill in the area between the bounds in the plot from exercise #2.
4. Use axvspan() to shade a rectangle from '2018-07-25' to '2018-07-31', which marks the large decline in Facebook price on a line plot of the closing price.
5. Using the Facebook stock price data, annotate the following three events on a line plot of the closing price:

- Disappointing user growth announced after close on July 25, 2018
- Cambridge Analytica story breaks on March 19, 2018 (when it affected the market)
- FTC launches investigation on March 20, 2018

6. Modify the reg_resid_plots() function to use a matplotlib colormap instead of cycling between two colors. Remember, for this use case, we should pick a qualitative colormap or make our own.

1. Using seaborn, create a heatmap to visualize the correlation coefficients between earthquake magnitude and whether there was a tsunami with the magType of mb.

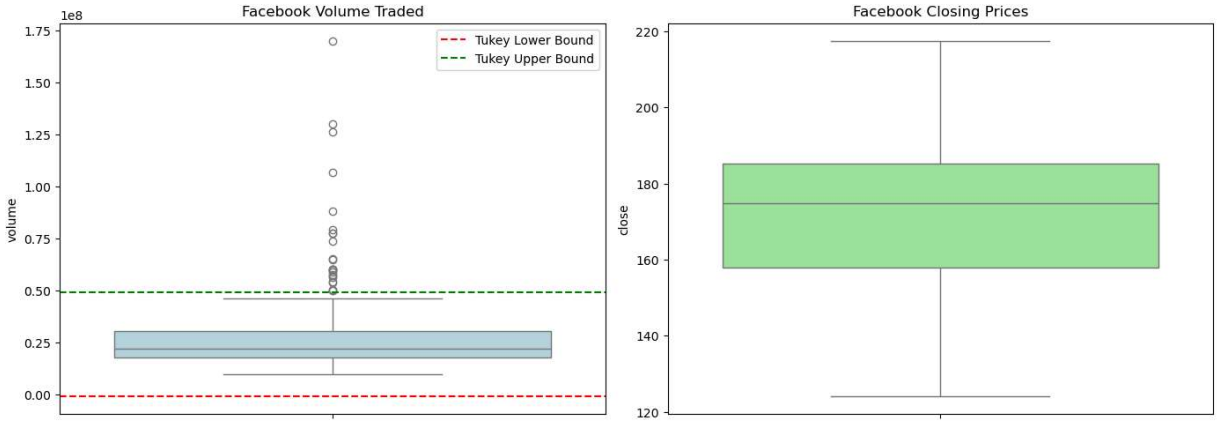
In [57]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# finding the data with magtype of mb
mb_quakes = earth[earth['magType'] == 'mb']

# relevant columns
correlation_data = mb_quakes[['mag', 'tsunami']]

# correlation matrix
corr = correlation_data.corr()

# Create heatmap
plt.figure(figsize=(5, 4))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap (Magnitude vs Tsunami) [magType = mb]')
plt.show()
```



2. Create a box plot of Facebook volume traded and closing prices, and draw reference lines for the bounds of a Tukey fence with a multiplier of 1.5. The bounds will be at Q1 - 1.5 * IQR and Q3 + 1.5 * IQR. Be sure to use the quantile() method on the data to make this easier. (Pick whichever orientation you prefer for the plot, but make sure to use subplots.)

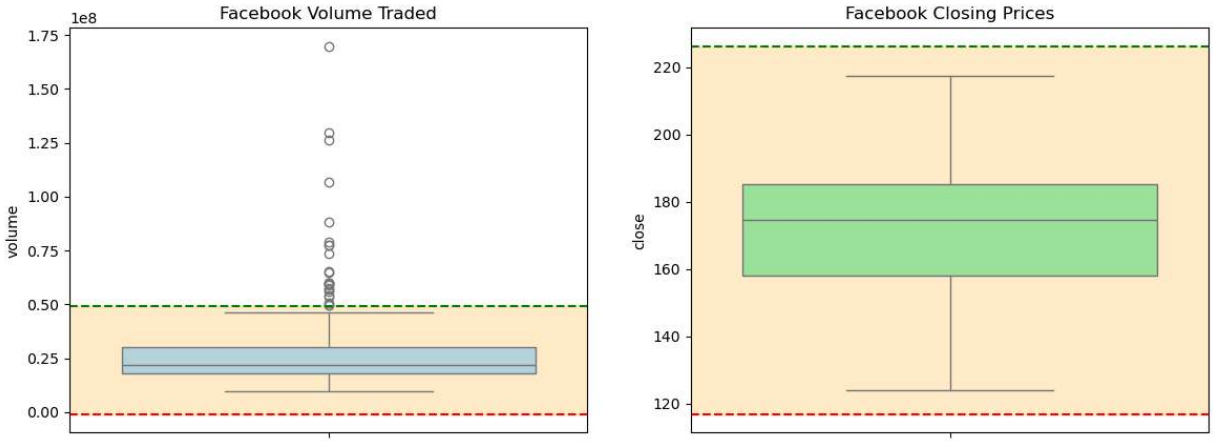
In [94]:
```
#  calculate Tukey bounds
def getTukeyBounds(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return lower, upper

# get bounds
vol_lower, vol_upper = getTukeyBounds(fb['volume'])
close_lower, close_upper = getTukeyBounds(fb['close'])

# creating subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# box plot for volume
sns.boxplot(y=fb['volume'], ax=axes[0], color='lightblue')
axes[0].set_title('Facebook Volume Traded')
axes[0].axhline(vol_lower, color='red', linestyle='--', label='Tukey Lower Bound')
axes[0].axhline(vol_upper, color='green', linestyle='--', label='Tukey Upper Bound'
axes[0].legend()

# box plot for closing price
sns.boxplot(y=fb['close'], ax=axes[1], color='lightgreen')
```

```
axes[1].set_title('Facebook Closing Prices')
plt.tight_layout()
plt.show()
```



3. Fill in the area between the bounds in the plot from exercise #2.

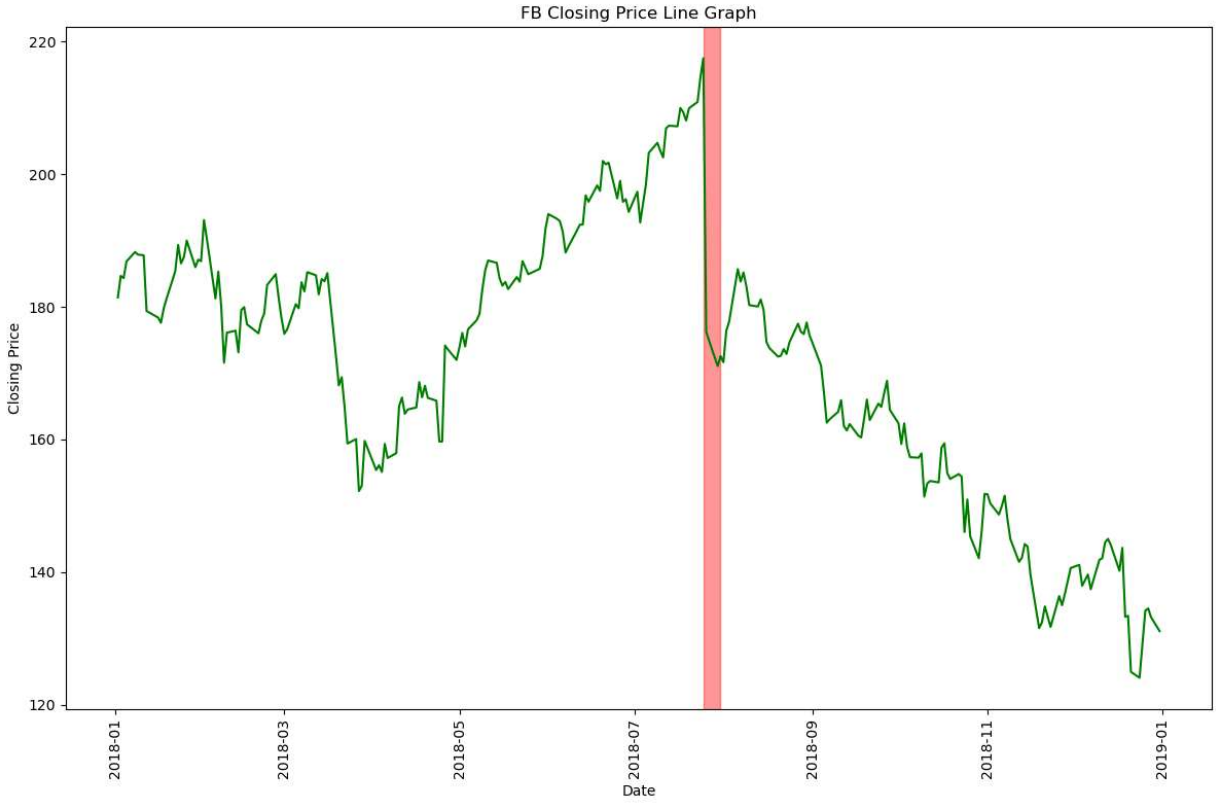
In [92]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def getTukeyBounds(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return lower, upper

# get Tukey bounds
vol_lower, vol_upper = getTukeyBounds(fb['volume'])
close_lower, close_upper = getTukeyBounds(fb['close'])

# creating subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# box plot for volume
sns.boxplot(y=fb['volume'], ax=axes[0], color='lightblue')
axes[0].set_title('Facebook Volume Traded')
axes[0].axhline(vol_lower, color='red', linestyle='--')
axes[0].axhline(vol_upper, color='green', linestyle='--')
axes[0].axhspan(vol_lower, vol_upper, color='orange', alpha=0.2) # set color and al

# box plot for closing price
sns.boxplot(y=fb['close'], ax=axes[1], color='lightgreen')
axes[1].set_title('Facebook Closing Prices')
axes[1].axhline(close_lower, color='red', linestyle='--')
axes[1].axhline(close_upper, color='green', linestyle='--')
axes[1].axhspan(close_lower, close_upper, color='orange', alpha=0.2)# set color and
plt.show() # showing plot
```



4. Use axvspan() to shade a rectangle from '2018-07-25' to '2018-07-31', which marks the large decline in Facebook price on a line plot of the closing price.

In [122…

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

# getting the closing price
g4 = fb['close']

# create plot
fig, ax = plt.subplots(figsize=(12, 8))

# creating the lineplot for data
sns.lineplot(x=g4.index, y=g4.values, ax=ax, color='green')

# highlighting the drop of value based on the
# setting min and max bounderies to find the drop
ax.axvspan(pd.to_datetime('2018-07-25'), pd.to_datetime('2018-07-31'), color='red',

# laberl and show plot
plt.title('FB Closing Price Line Graph')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



5. Using the Facebook stock price data, annotate the following three events on a line plot of the closing price:

- Disappointing user growth announced after close on July 25, 2018
- Cambridge Analytica story breaks on March 19, 2018 (when it affected the market)
- FTC launches investigation on March 20, 2018

In [131…

```
# to anottate we need to set a dataframe containing theses events then
# merge to find which of the date aligns with the annotation

sample = {
    'date':['2018-07-25', '2018-03-19', '2018-03-20'],
    'event':['Disappointing user growth announced after close.', 'Cambridge Analyti
}


df = pd.DataFrame(sample)

df.head()
```

Out[131…

	date	event
0	2018-07-25	Disappointing user growth announced after close.
1	2018-03-19	Cambridge Analytica story
2	2018-03-20	FTC investigation

In [135…
```
df['date'] = pd.to_datetime(df['date']) # change data type for easy use
df.dtypes
```

Out[135…
```
date     datetime64[ns]
event            object
dtype: object
```

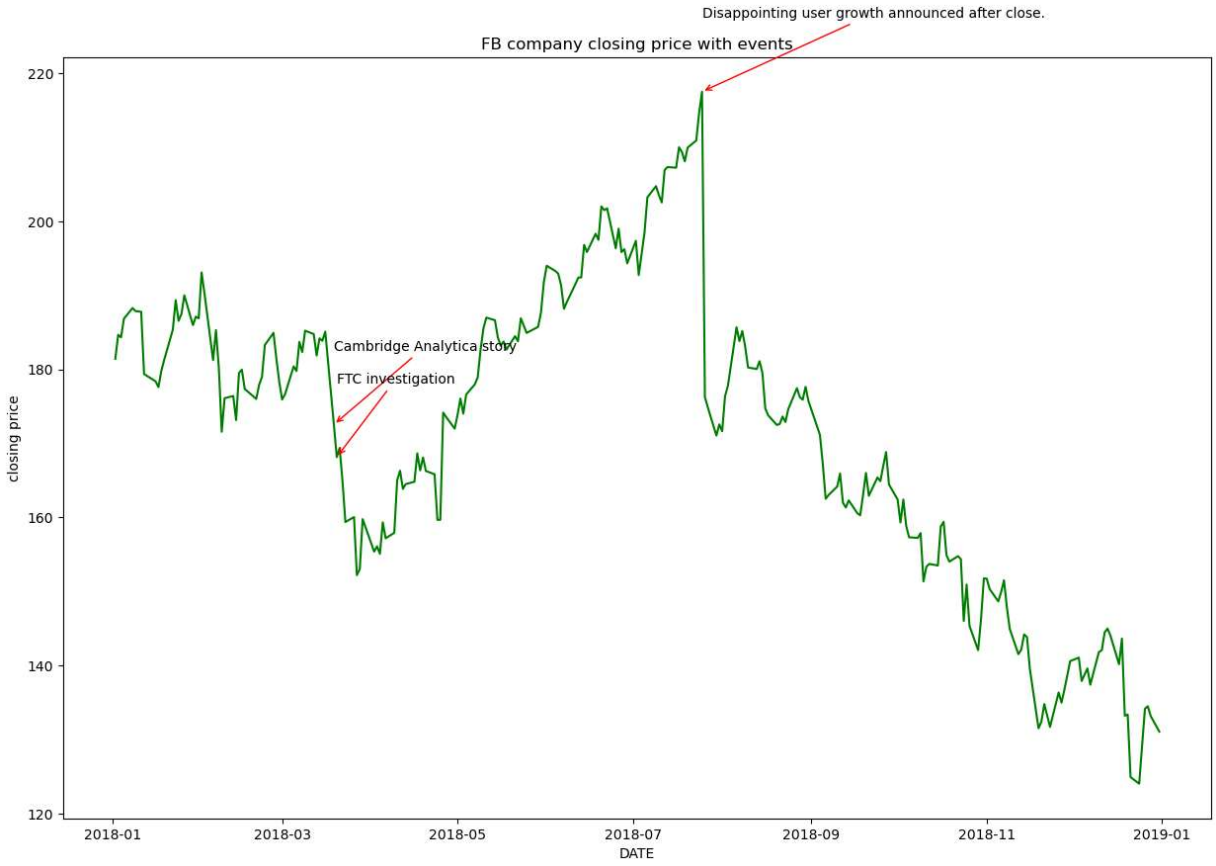
In [137…
```
# we need to set the date colum a our index
df.set_index('date',inplace = True)
```

In [141…
```
merger = pd.merge(df, fb, on = 'date',how = 'outer')

# we can see in the output that we successfull merge the 2 dataframes
merger = merger.sort_index()
```

In [144…
```
closing = merger['close'] # get data for annotate
# only get what is important for annotate
annotate = merger['event']
annotate.dropna(inplace= True)

annotate # checking
```

Out[144…
```
date
2018-03-19                          Cambridge Analytica story
2018-03-20                                  FTC investigation
2018-07-25    Disappointing user growth announced after close.
Name: event, dtype: object
```

In [147…
```
# using a subplot for the line graph then annotate
fig, ax = plt.subplots(figsize = (15,10))
sns.lineplot(closing, color = 'green') # tracing the data of closing

# iterating through the data then annotate on the correct one
for date, event in annotate.items(): # items() will get the date index and the valu
    ax.annotate(event, # the event
                xy= (date, closing.loc[date]), # indicator
                xytext = (date, closing.loc[date]+ 10), # labeling of annotate
                arrowprops = dict(arrowstyle='->', color='red') # arrow style of th
               )
plt.title('FB company closing price with events')
plt.xlabel('DATE')
plt.ylabel('closing price')
plt.show() # plotting
```



6. Modify the reg_resid_plots() function to use a matplotlib colormap instead of cycling between two colors. Remember, for this use case, we should pick a qualitative colormap or make our own.

In [ ]:

Summary/Conclusion:

After following the procedure I was able to learn about the different capabilities of seaborn in visulization of data. I learned the different plotting of graphs using seaborn library. It was hard learning about seaborn for me because it is my first time using it and it provided more in depth customization but you have to set your data and arguements right. Even though it was hard, I was still able to implement my learnings in the supplementary activities. I need to practice more on using this library and understand how different customizations affect presentation of data.

In [ ]: