CPE311 Computational Thinking with Python

Name:

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 # used to create visualizations
import numpy as np # used to
import seaborn as sns
import pandas as pd
%matplotlib inline
# used to not always type show()
fb = pd.read_csv('fb_stock_prices_2018.csv', index_col='date', parse_dates=True
)
quakes = pd.read_csv('earthquakes.csv')
```

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")
# this converts the time column to a datatime value
# then sets the time column as the index and uses data indexing
# to take the '2018-09-28' index in the dataframe
# and the rows in those collected index
# that has Indonesia in the 'parsed_place' column
# tsunami with a 1 value
# and a 7.5 value in mag
# all those must be met
```

Out[2]:

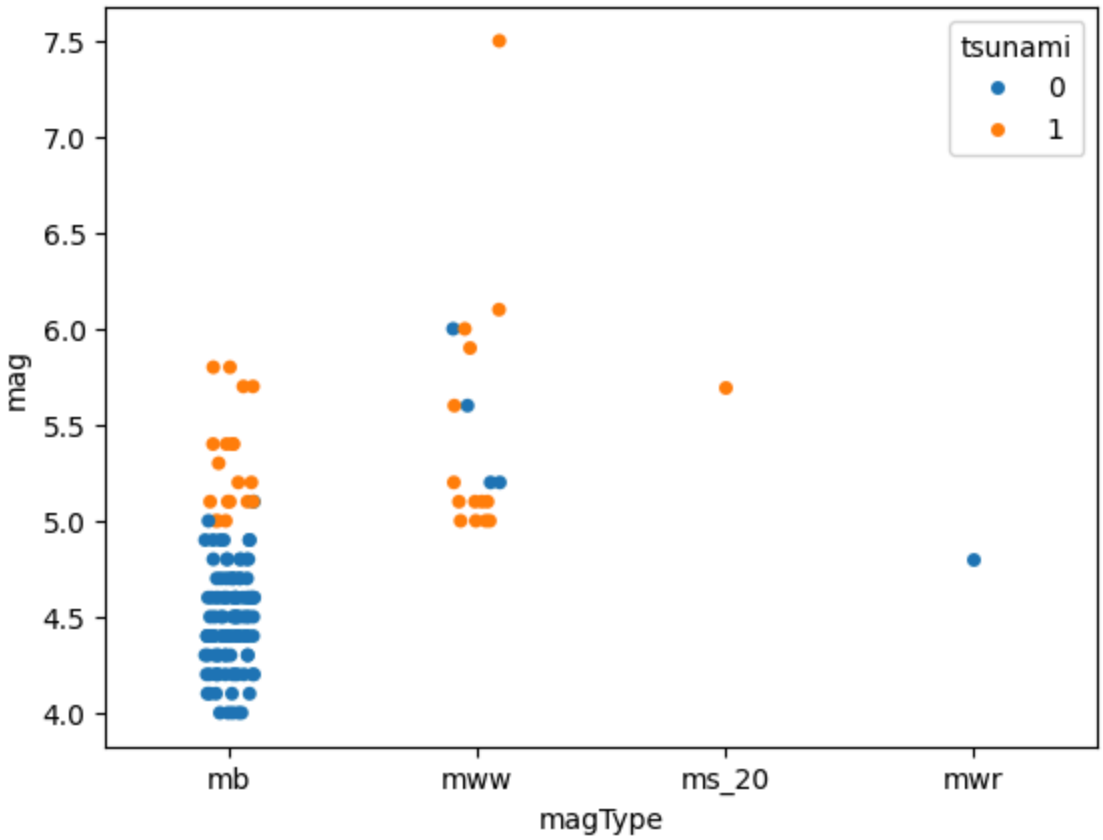
time	mag	magType	place	tsunami	parsed_place
2018-09-28 10:02:43.480	7.5	mww	78km N of Palu, Indonesia	1	Indonesia

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"')
)
# this create a stripplot
# using the magType as the x axis
# the tsunami column is the indicator for
# the the color of the plot entries
# and the data from the quakes data with the
# parsed_place entries containing Indonesia
plt.show()
```



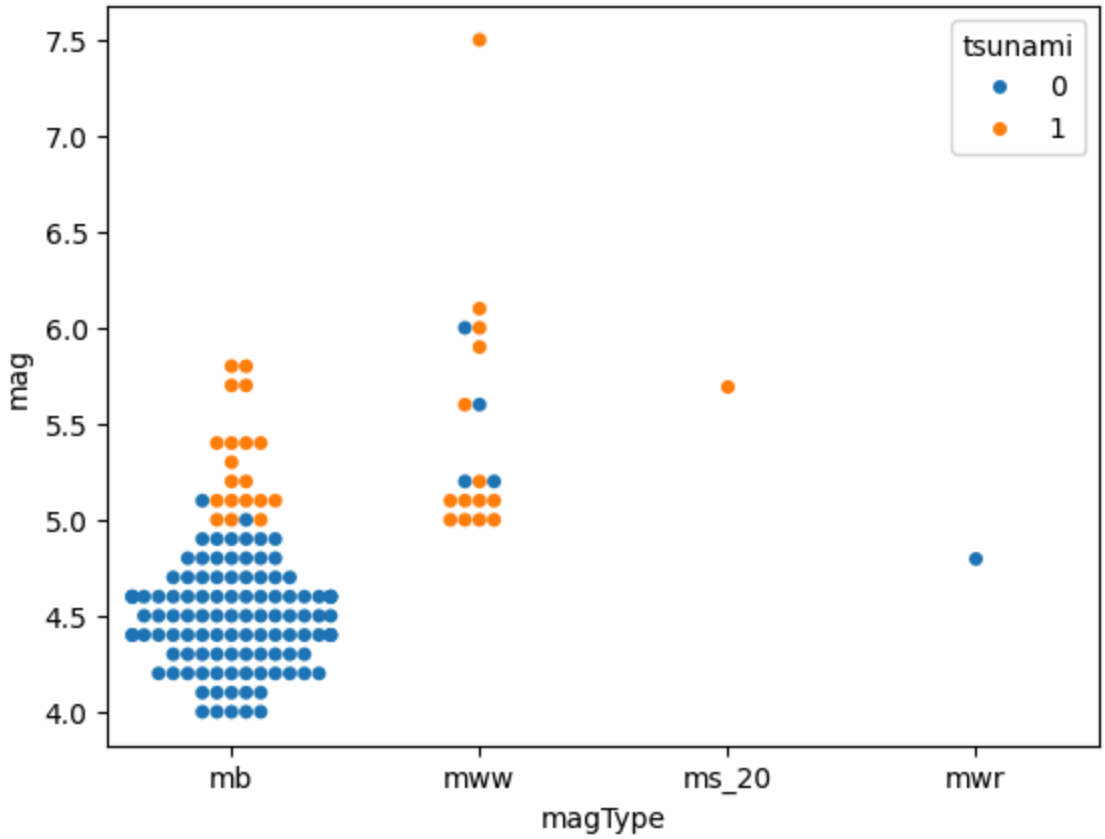
swarmplot()

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"')
)
# this create a swarmplot
# using the magType as the x axis
# the tsunami column is the indicator for
# the the color of the plot entries
# and the data from the quakes data with the
# parsed_place entries containing 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)
```



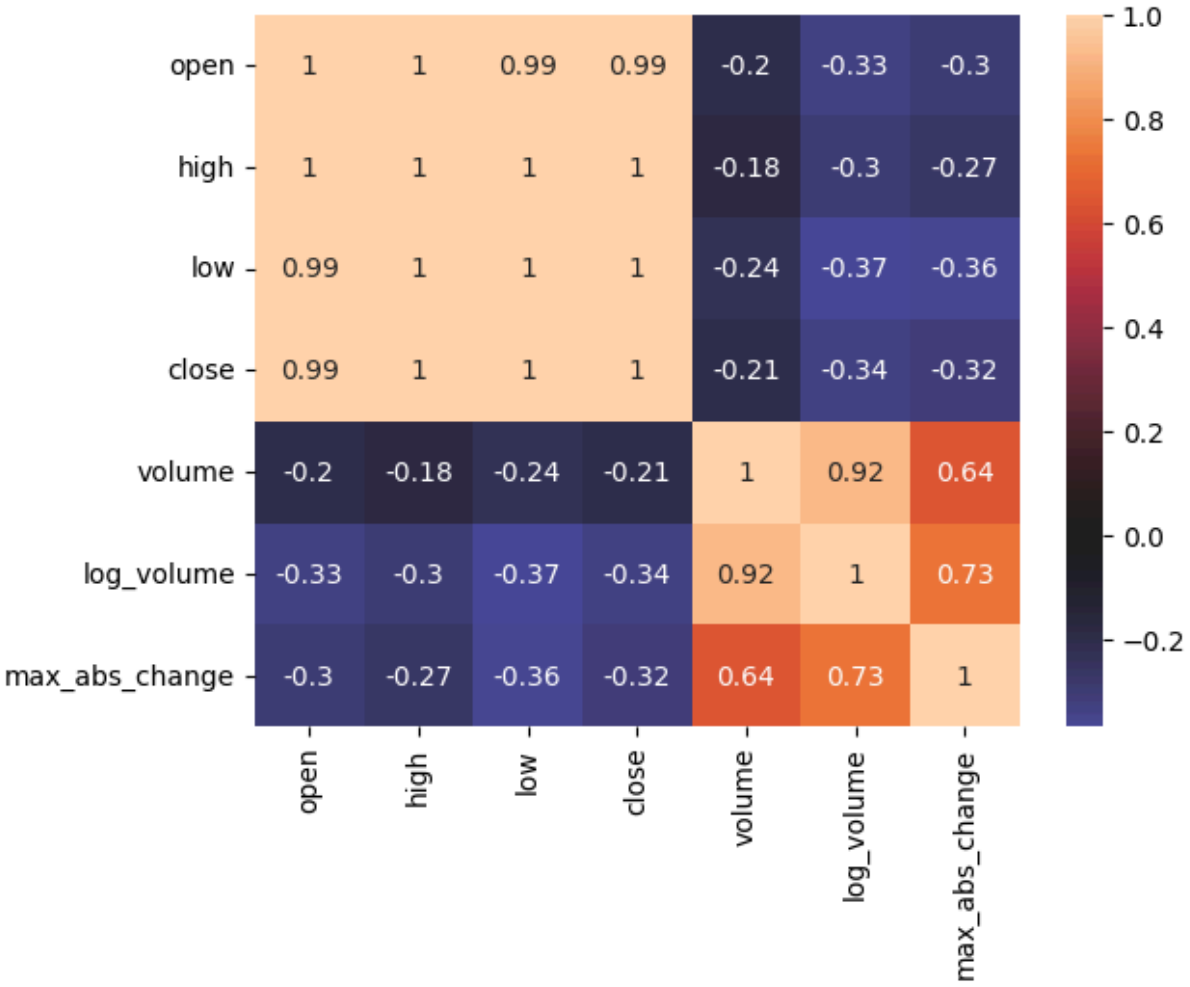
Correlations and Heatmaps

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
```

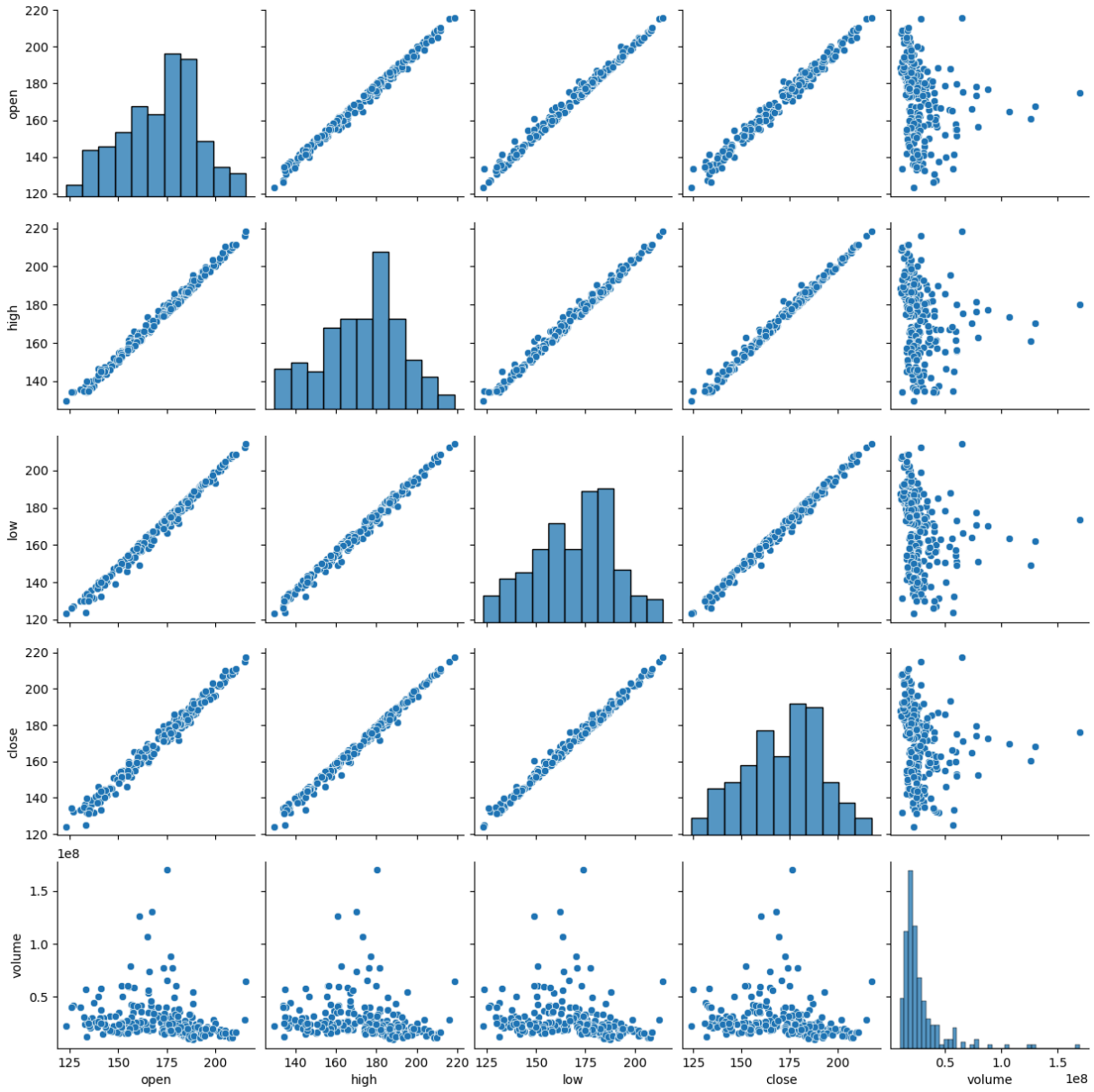
```
)

# this create a heatmap
# it first sorts the index of the fb dataframe
# then assigning new columns
# the log_volume column is the natural logarithm
# of the values in the volume column
# and the max_abs_change
# is the difference of the values in the high column
# and the low column of the fb dataframe
plt.show()
```



pairplot()

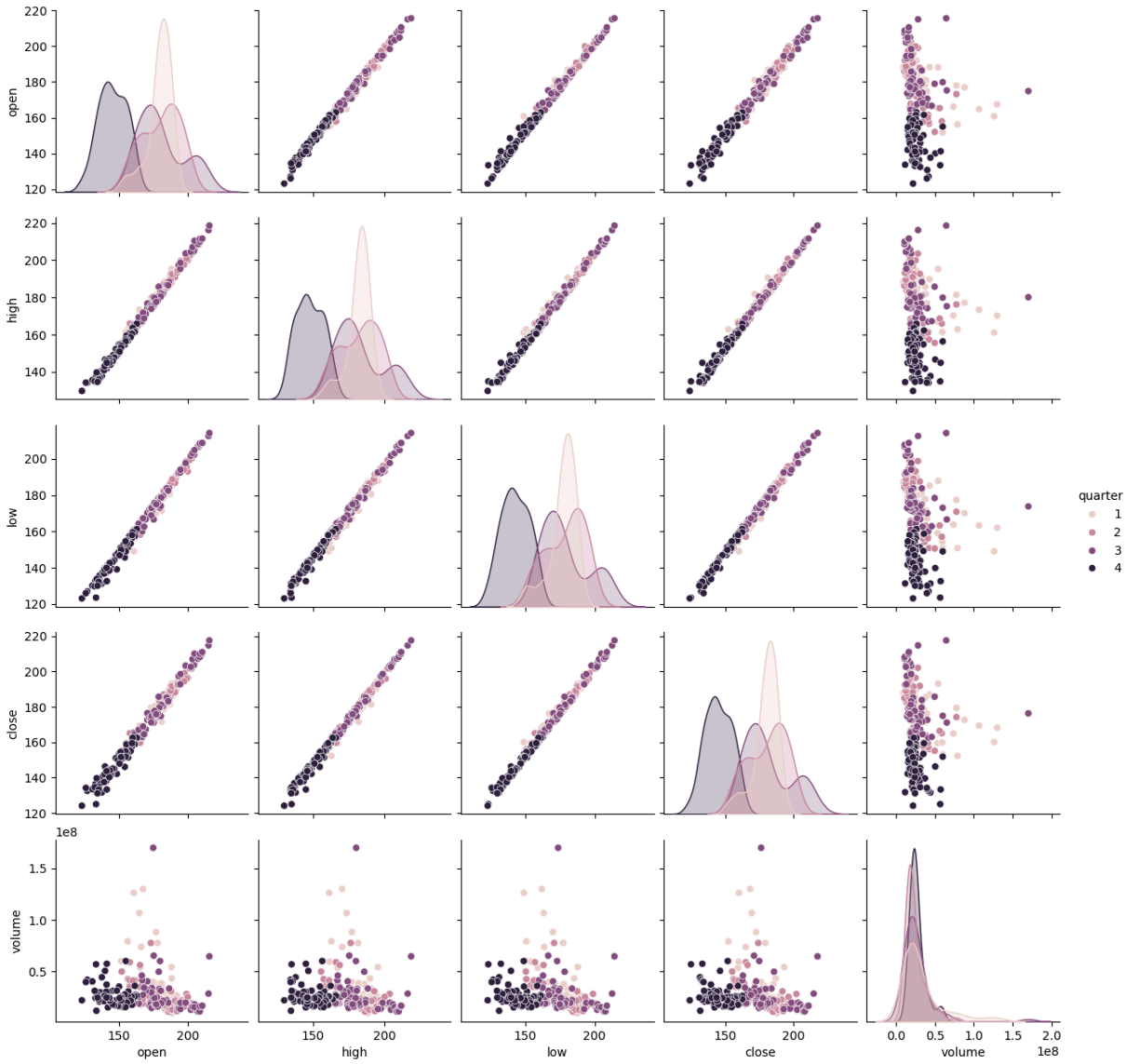
The pair plot is seaborn's answer to the scatter matrix we saw in the pandas subplotting notebook:

In [17]:
```
sns.pairplot(fb)
# creates plots relationships
# from each column with numerical values
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'
)
#this creates a new column in the the fb dataframe
# that groups each entries by datetime
# then create a pairplot but in a form
# of KDE and uses a new colour

plt.show()
```

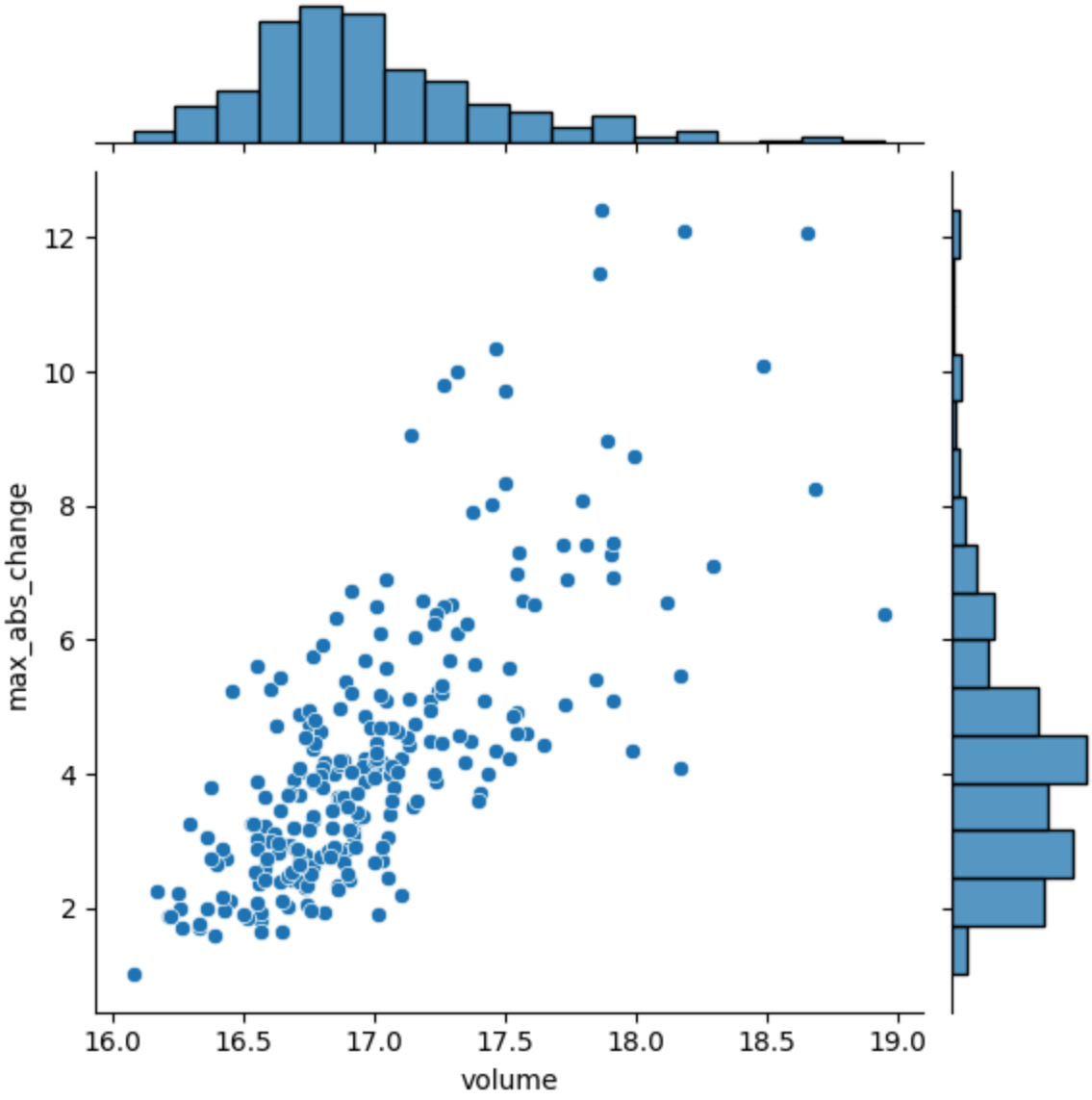


jointplot()

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
 )
)

# this creates a joinplot
# the volume as the x value
```

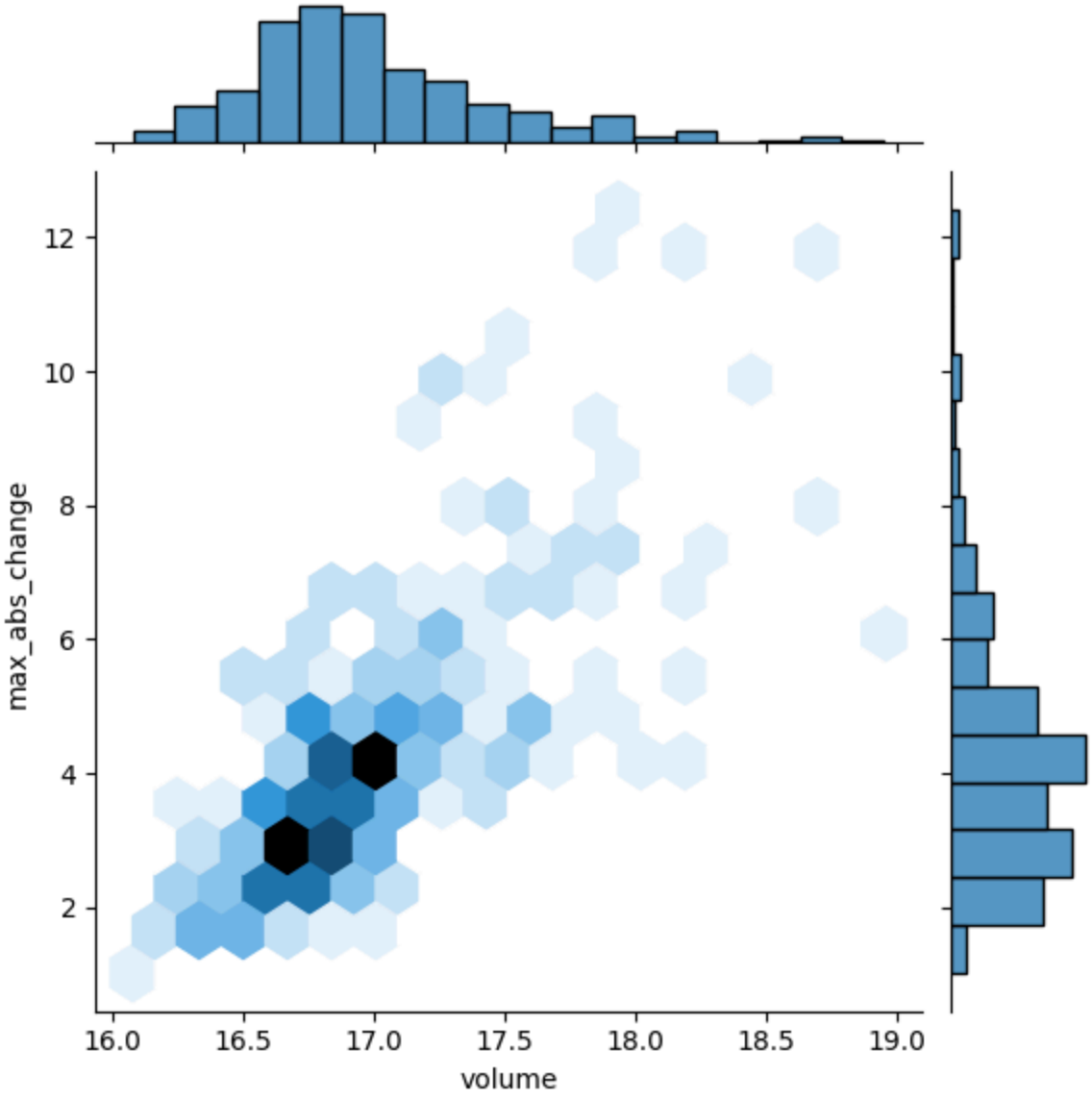
```
# it changes temporarily the volume column
# with the logarithm of its values
# and a temporary column which is the difference of the high and low columns
# as its data as the y axis value

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
 )
)
```

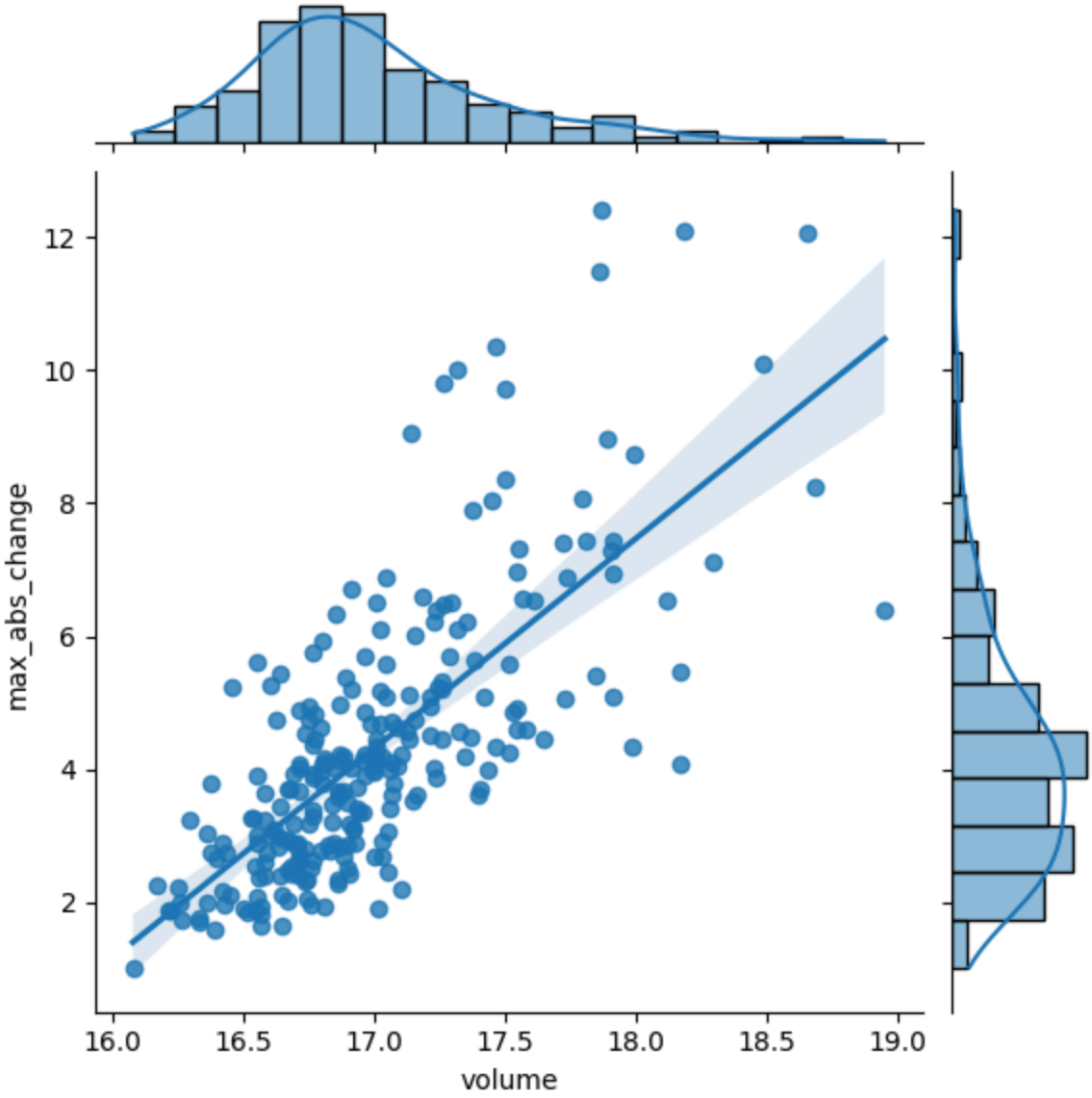
```
# this does the same as the one above it
# but it instead uses a hex kind
# it shows the plot points as hexagons
# the darker the hexagon the higher the value

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
 )
)

# this is the same as the code made two cells above
```

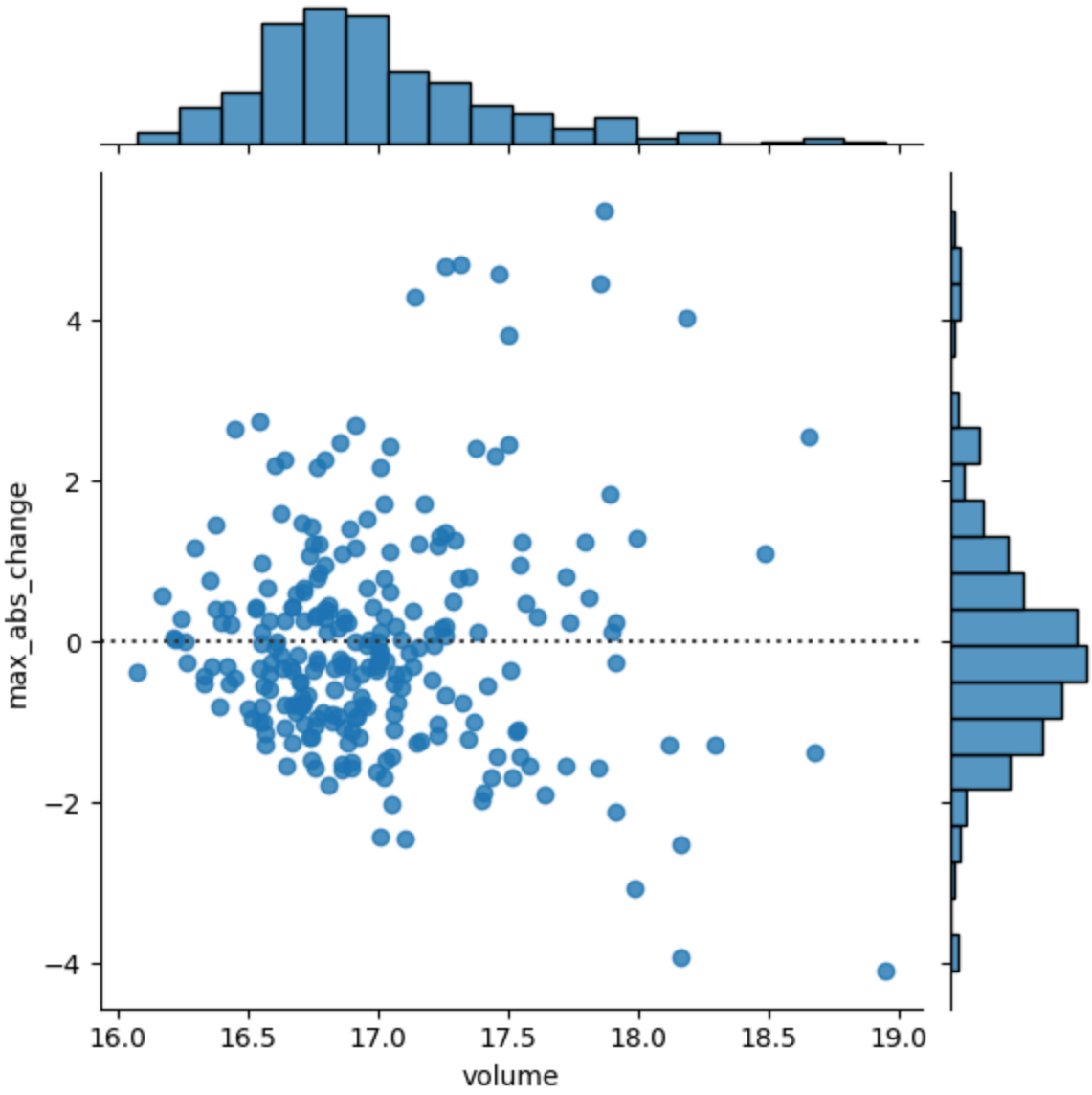
```
# the  kind='reg', just puts a regression line in the graph


plt.show()
```

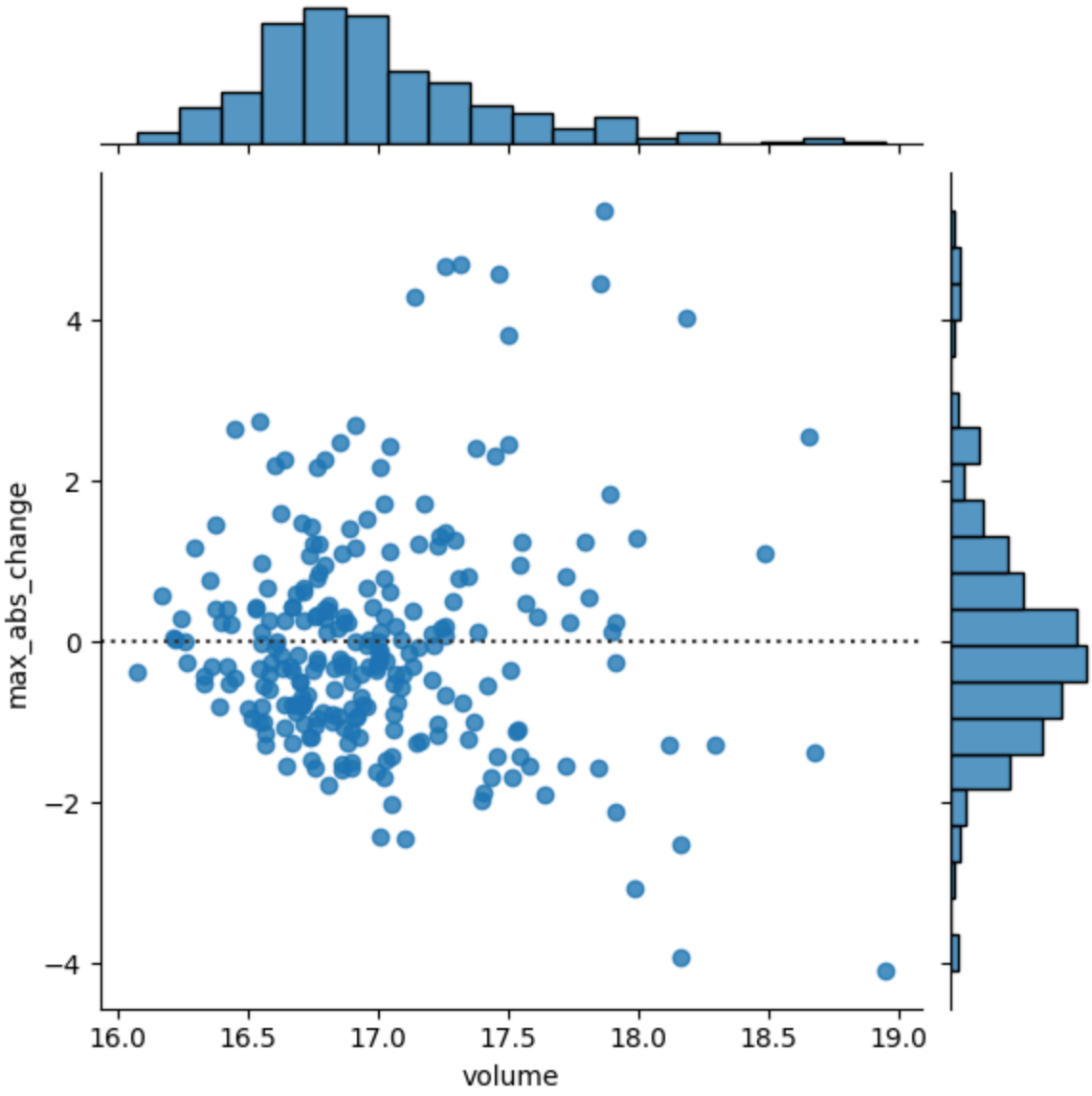


If we pass kind='resid' , we get the residuals from the aforementioned regression:

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
 )
)

# this creates a residual plot for the graph
#
```

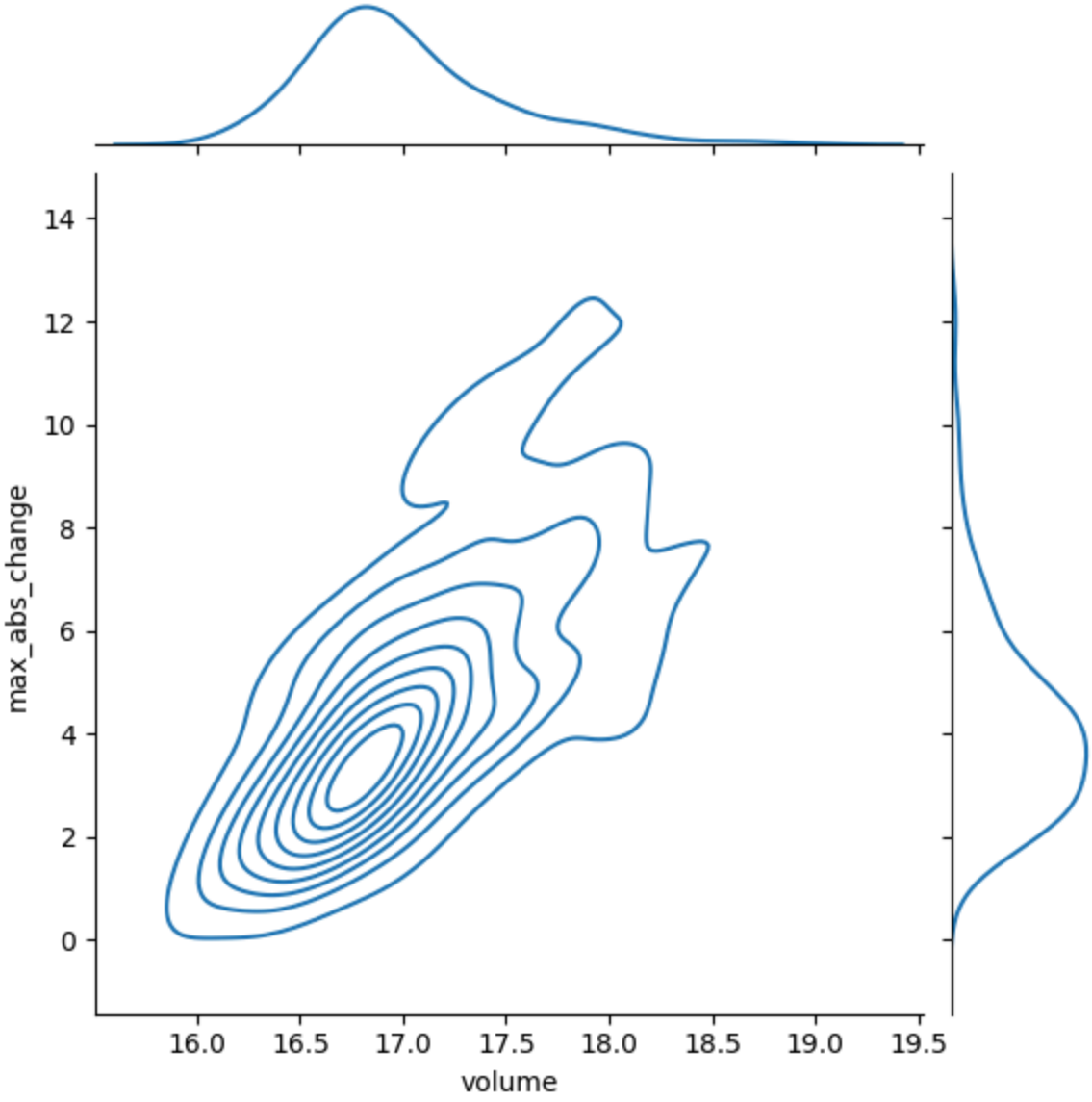
```
plt.show()
```





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
 )
)

#this shows the graph in a kde format


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:]

#this creates a new dataframe with the the fb dataframe
# but changed the volume with the logarithm of its value
# and a new column that contains the difference of the high and low columns
```

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
```

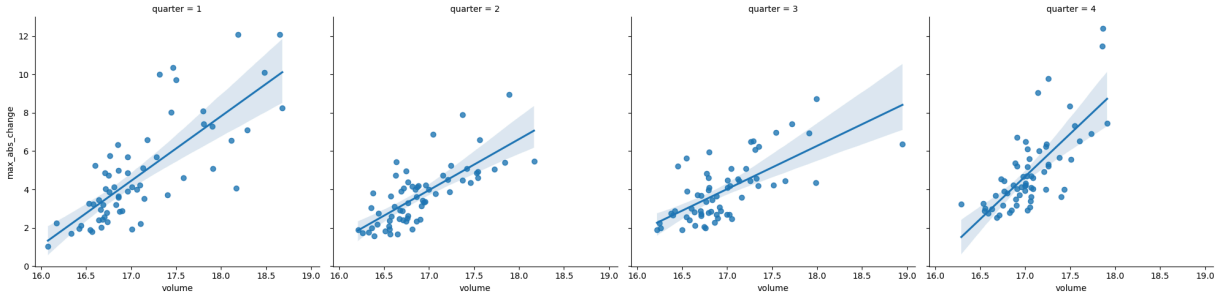
The reg_resid_plots() function from the reg_resid_plot.py module in this folder uses regplot() and residplot() from seaborn along with itertools to plot the regression and residuals side-by-side:

In [216…
```
from reg_resid_plot import reg_resid_plots
reg_resid_plots(fb_reg_data)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[216], line 1
----> 1 from reg_resid_plot import reg_resid_plots
      2 reg_resid_plots(fb_reg_data)

ModuleNotFoundError: No module named 'reg_resid_plot'
```

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()
```



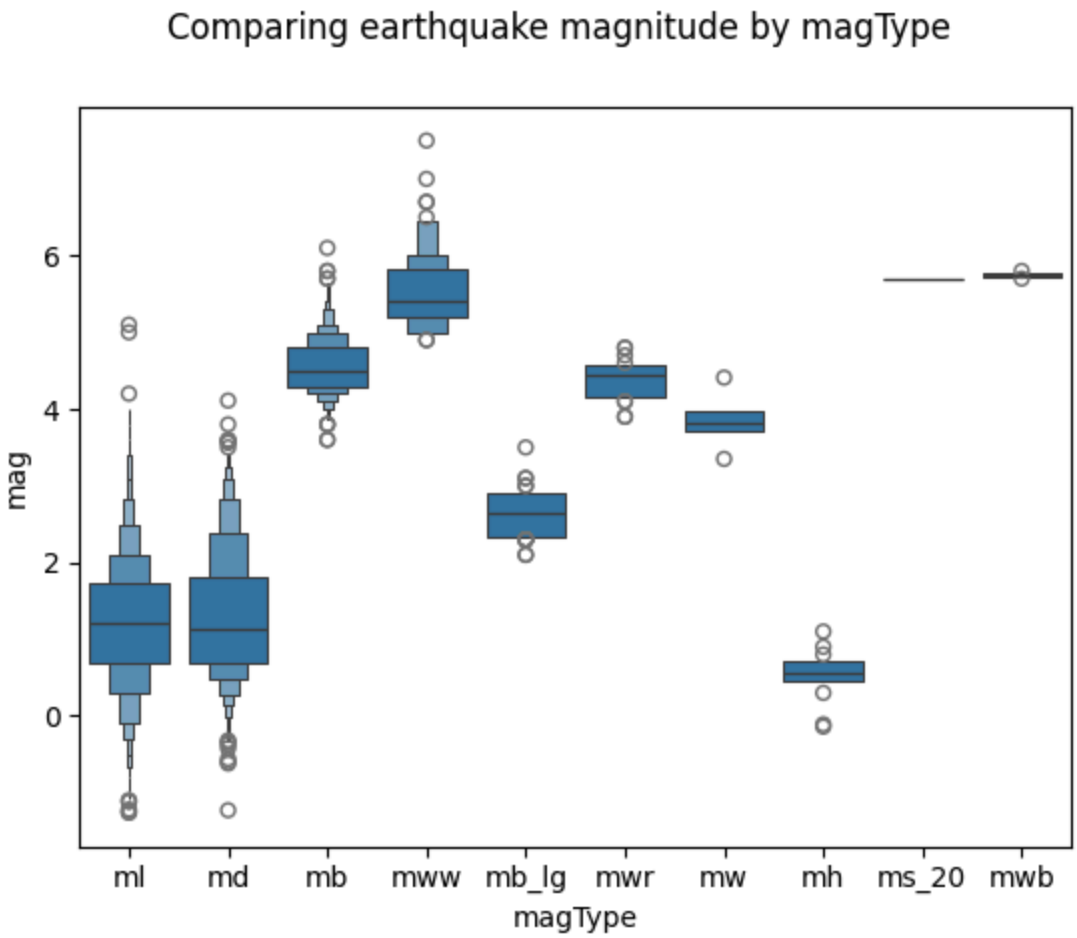
Distributions

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')
# this create a boxen plot
# this take the magType and mag columns
# as x and y axis respectively
# then uses those columns again
# for the data to be taken

plt.show()
```

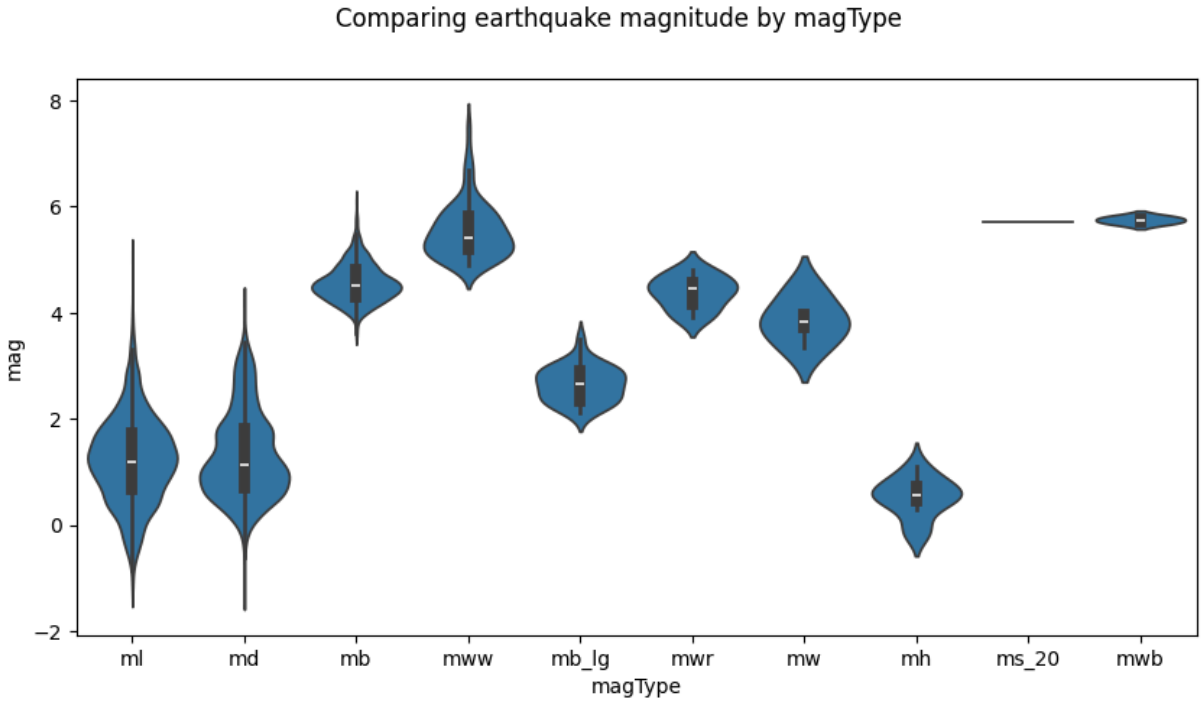


violinplot()

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')
# this creates a matplotlib figure that has a 10x5 size
# this creates a violin plot
# and uses  magType and mag columns
# as x and y axis respectively
# then uses those columns again
# for the data to be taken
# it scales the violins to be the same width with each other

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(
```



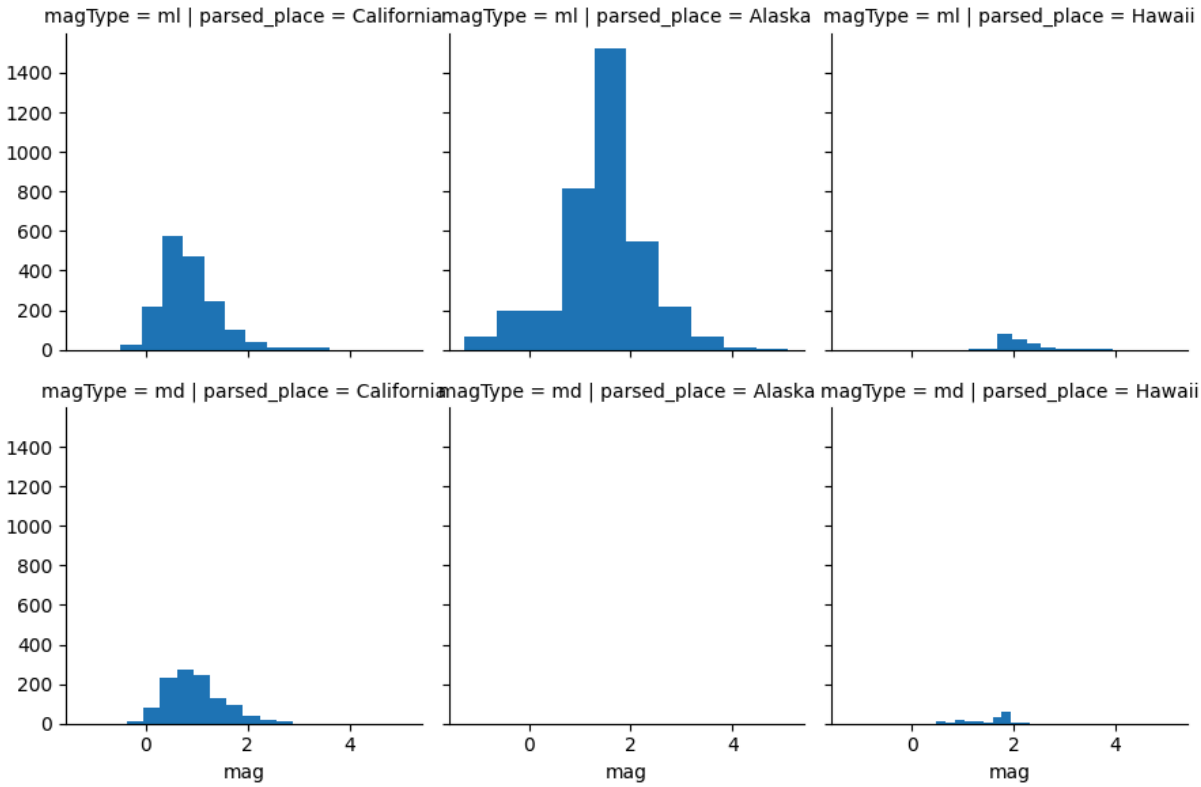
Faceting

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

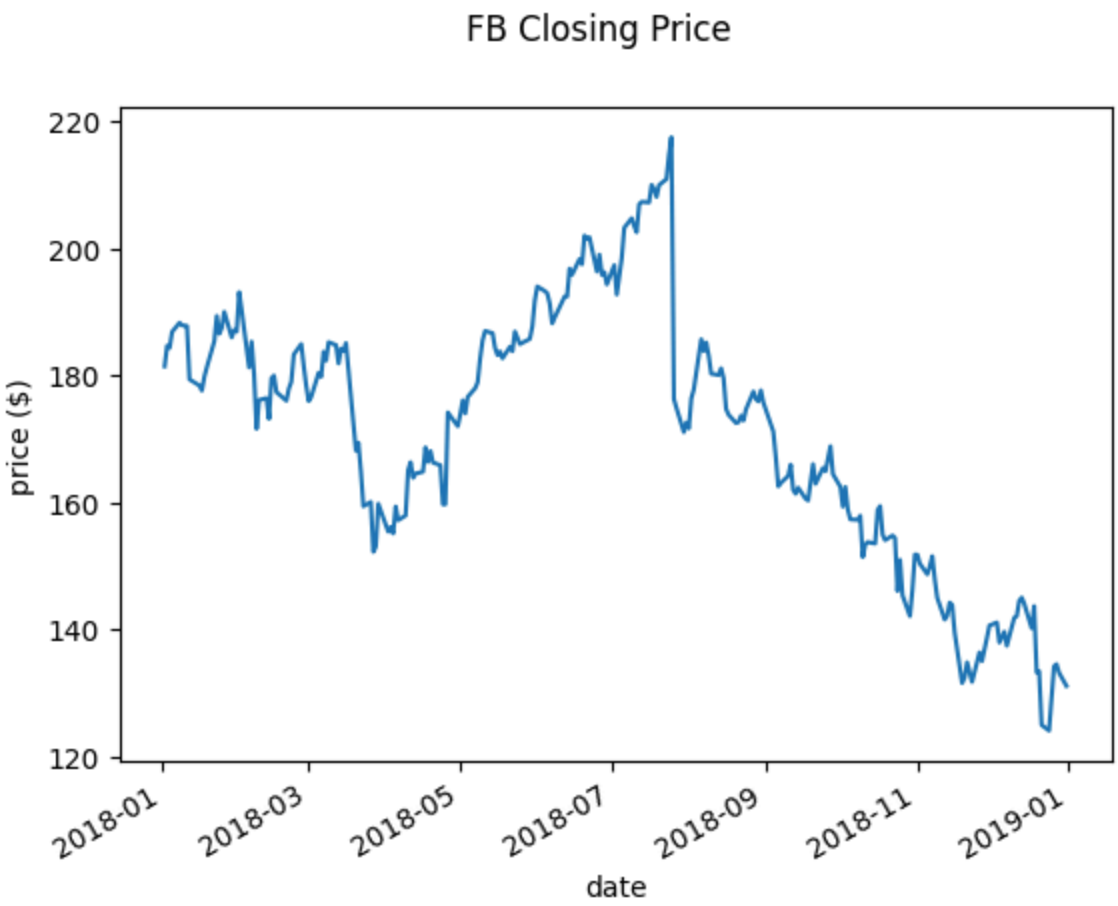
plt.suptitle() adds a title to plots and subplots

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

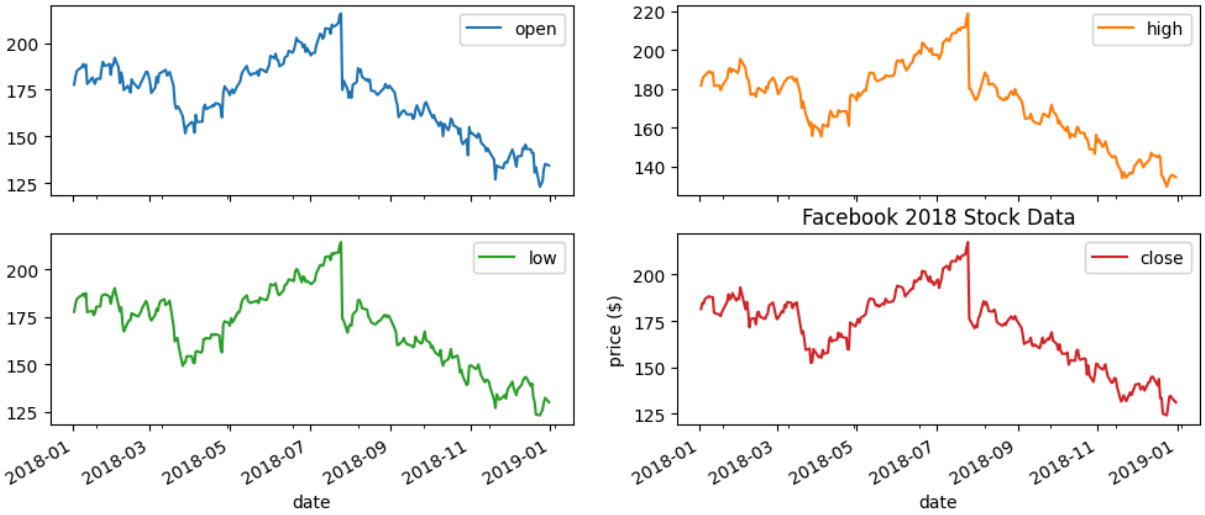
In [50]:
```
fb.close.plot()
plt.suptitle('FB Closing Price')
plt.xlabel('date')
plt.ylabel('price ($)')

# this creates a plot for the close column
# of the fb database
# then places a name for the plot
# the x and y axis

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 ($)')
# this picks the first 4 columns
# then creates a subplot for those columns
# with a 2x2 layout, 2 rows and 2 columns as seen below
# and sets the size of the each plot
# as a 12,5
plt.show()
```

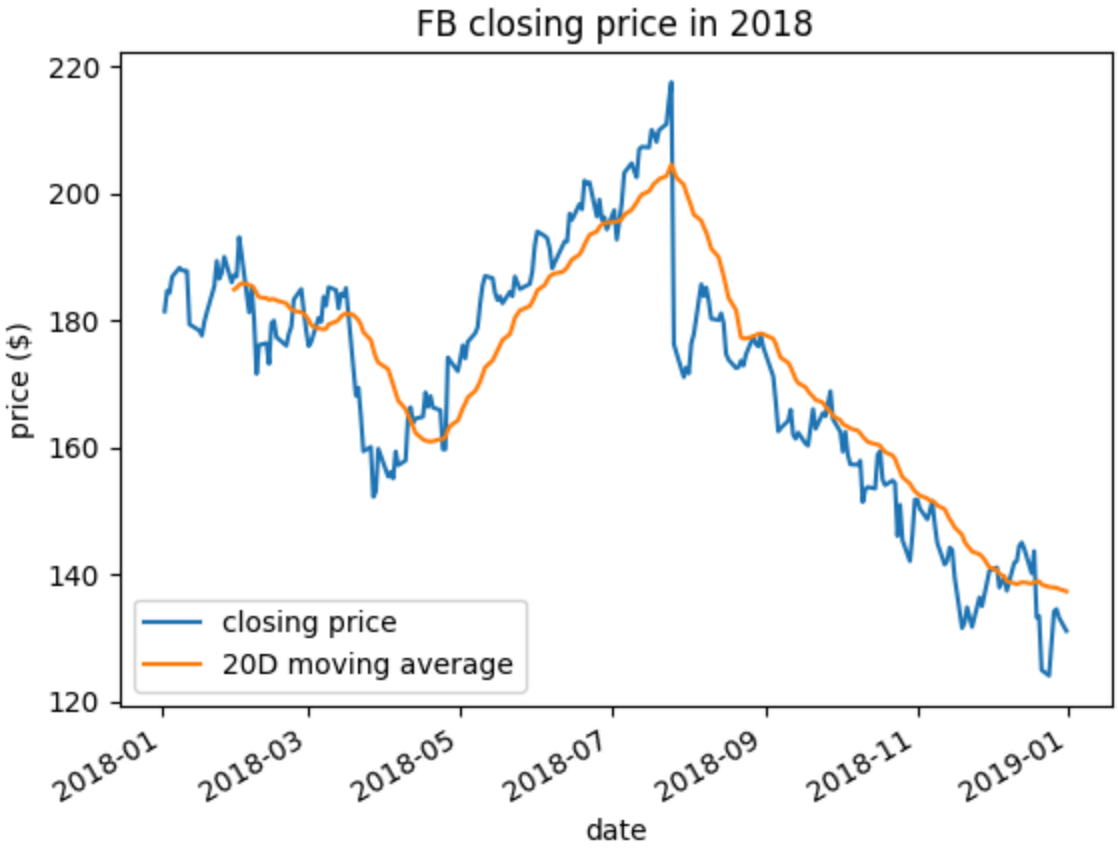


Legends

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 ($)')

# this creates a ma column to the dataframe
# that creates a rolling window of 20 days of the close column
# and takes the mean of it

# then takes the values of close and teh created ma
```

```
# columns for the plot

plt.show()
```

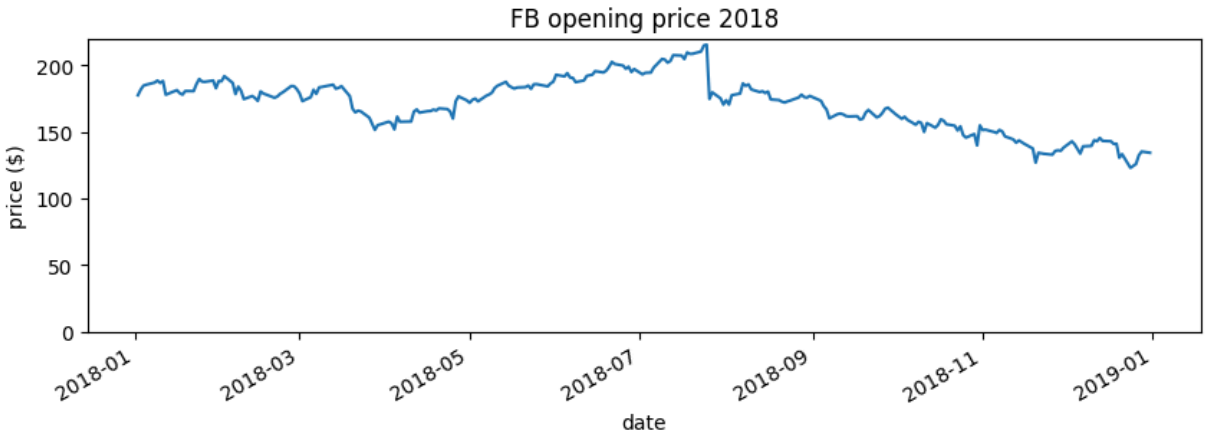


Formatting Axes

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.

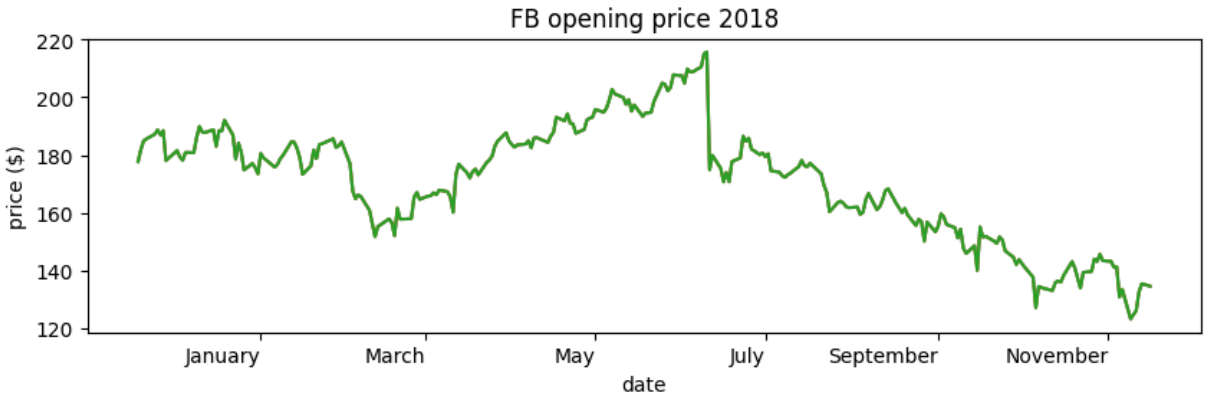
In [58]:
```
fb.open.plot(figsize=(10, 3), title='FB opening price 2018')
# this creates a plot using the open column of the fb dataframe
# with a size of 10x3
plt.ylim(0, None)
# this set the lower limit of the y axis to 0
# and upper limit to none
plt.ylabel('price ($)')
# this sets the name for the y axis

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()
```

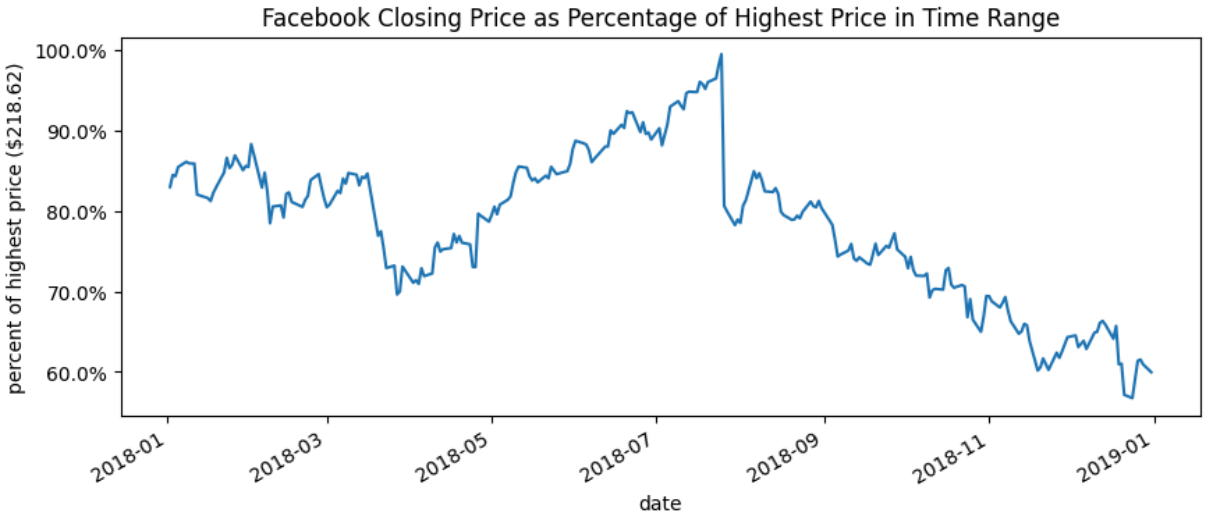


PercentFormatter

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),
```

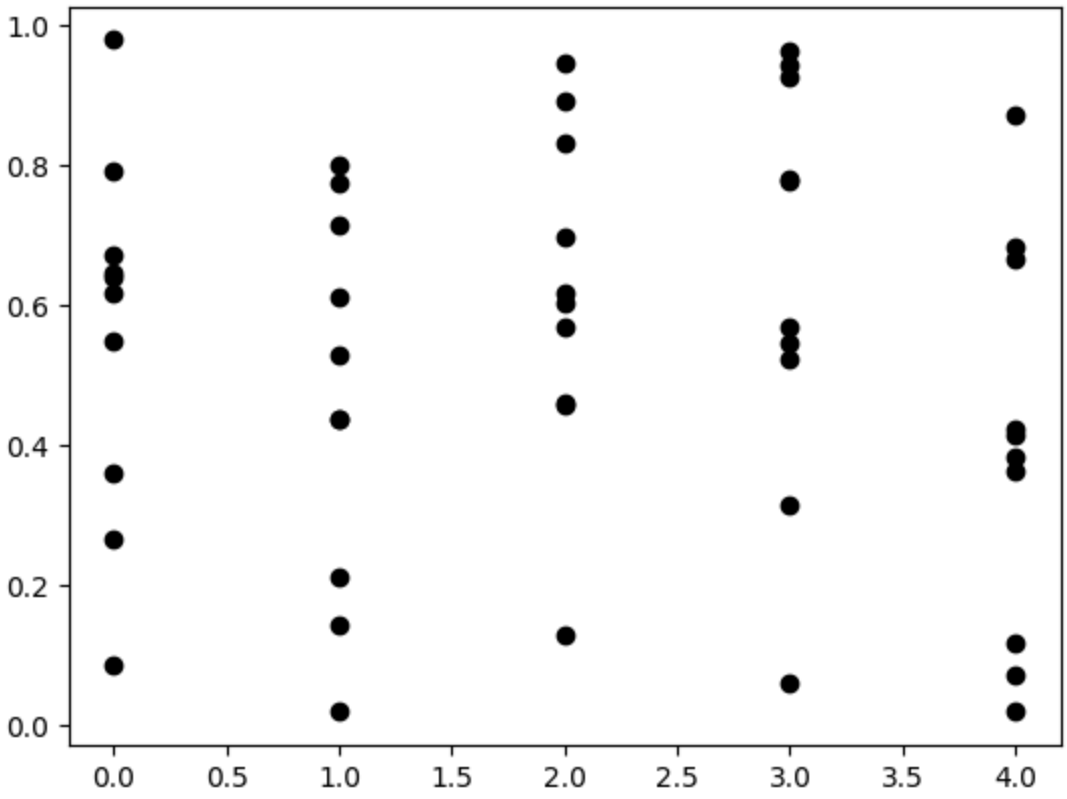
```
 title='Facebook Closing Price as Percentage of Highest Price in Time Range'
)
# this creates a plot for the close column and places it in a variable


ax.yaxis.set_major_formatter(
 ticker.PercentFormatter(xmax=fb.high.max())
)
#

ax.set_yticks([
 fb.high.max()*pct for pct in np.linspace(0.6, 1, num=5)
]) # show round percentages only (60%, 80%, etc.)
ax.set_ylabel(f'percent of highest price (${fb.high.max()})')

plt.show()
```

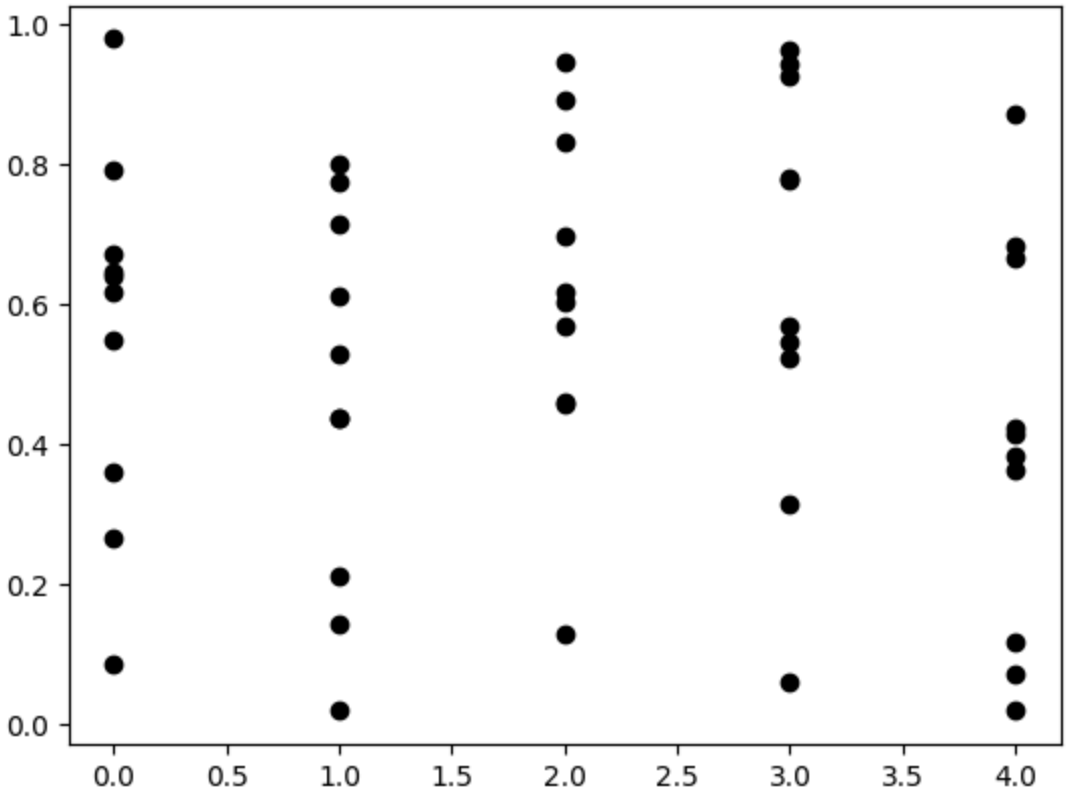


MultipleLocator

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

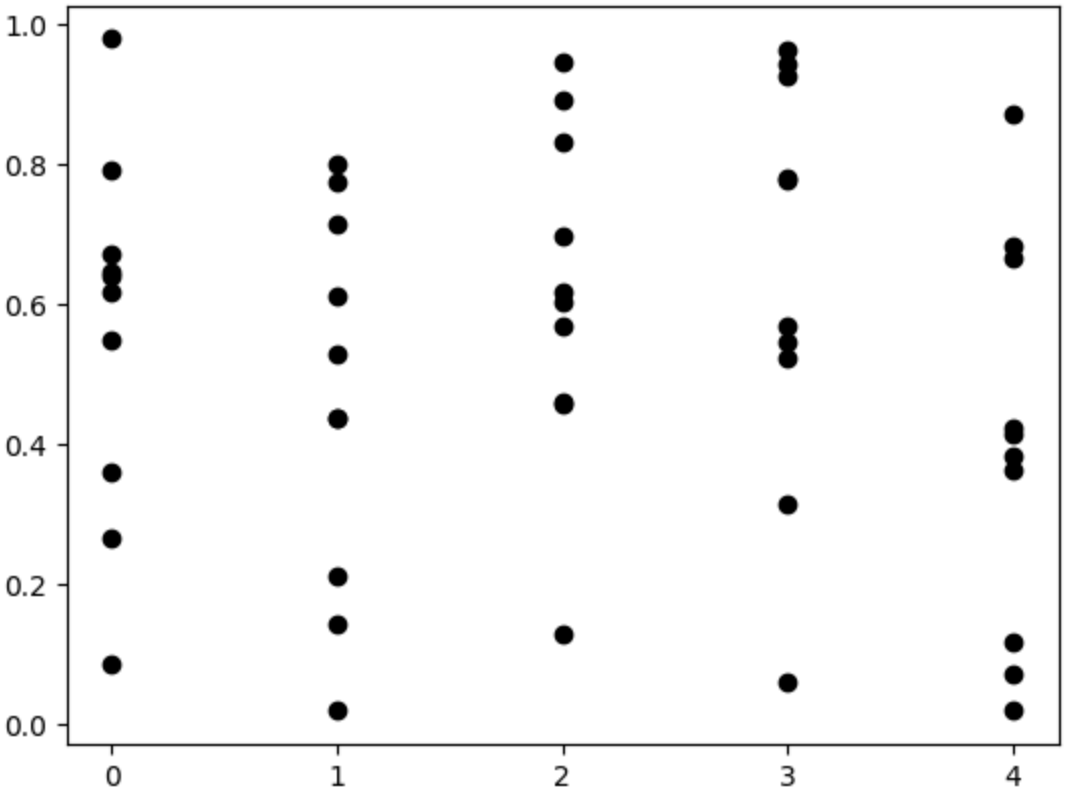
In [64]:
```
fig, ax = plt.subplots(1, 1)
np.random.seed(0)
ax.plot(np.tile(np.arange(0, 5), 10), np.random.rand(50), 'ko')

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.

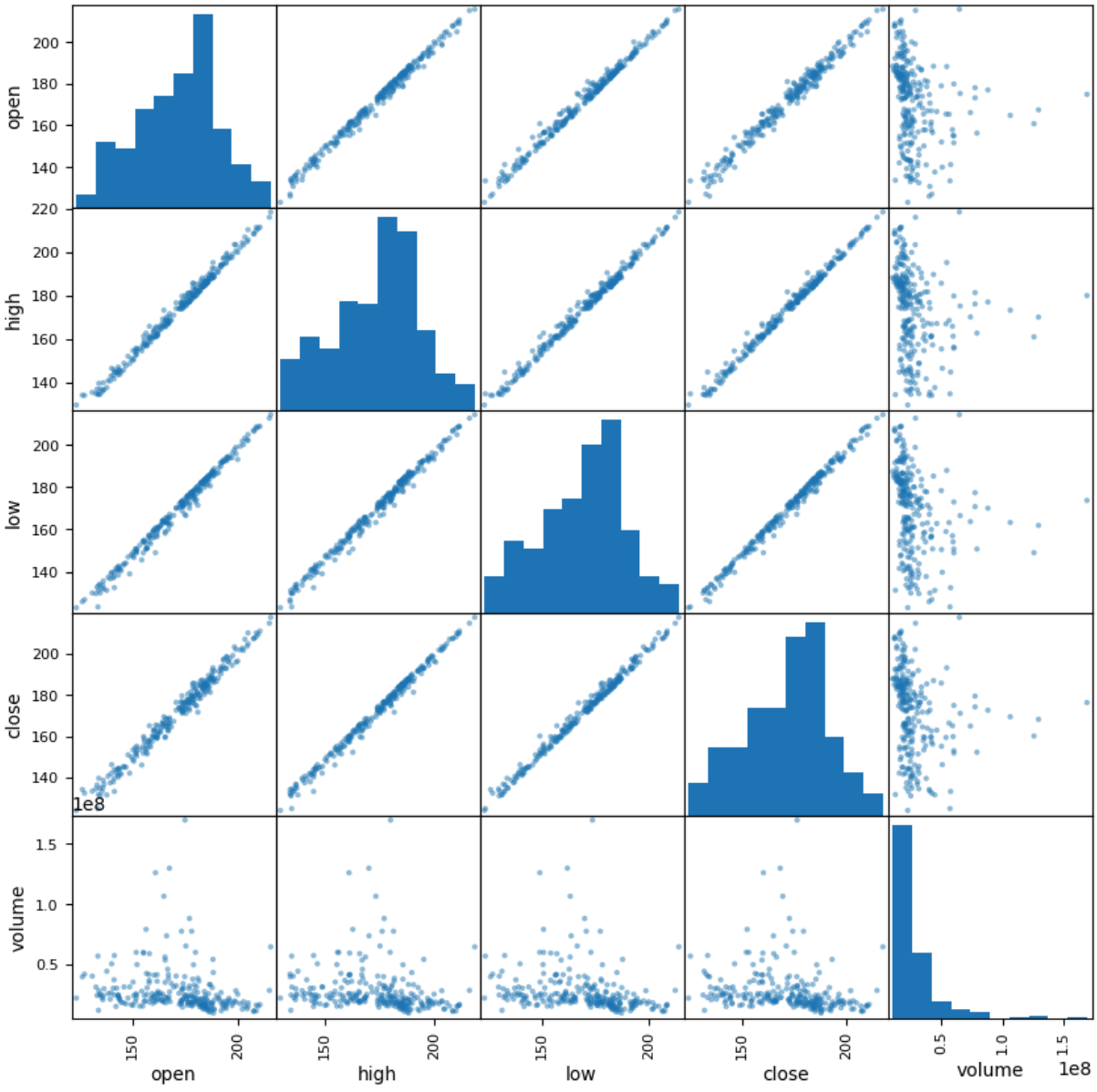
About the Data

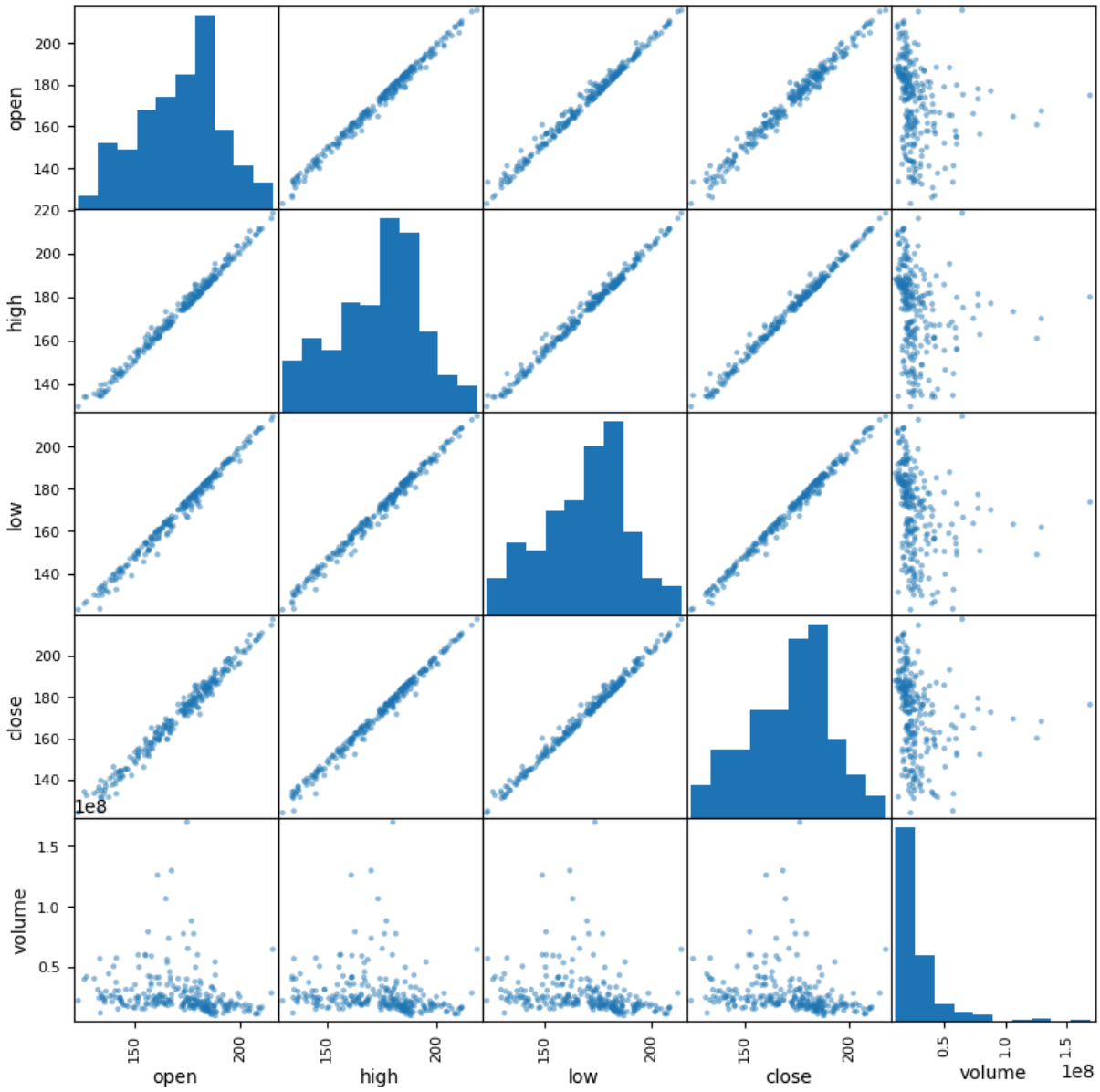
In this notebook, we will be working with Facebook's stock price throughout 2018 (obtained using the stock_analysis package).

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
)
```

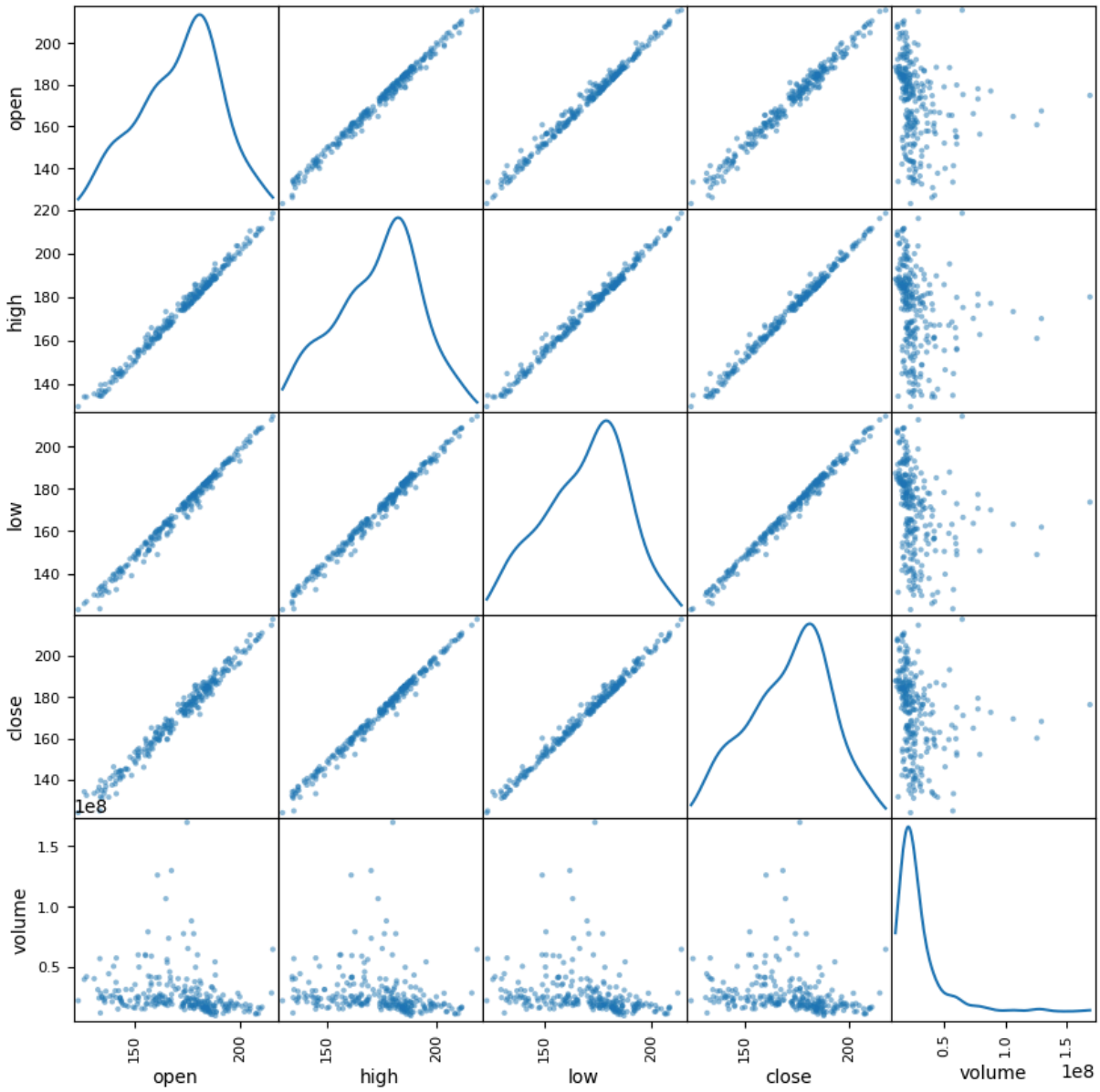
In [68]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(fb, figsize=(10, 10))

plt.show()
```





Changing the diagonal from histograms to KDE:

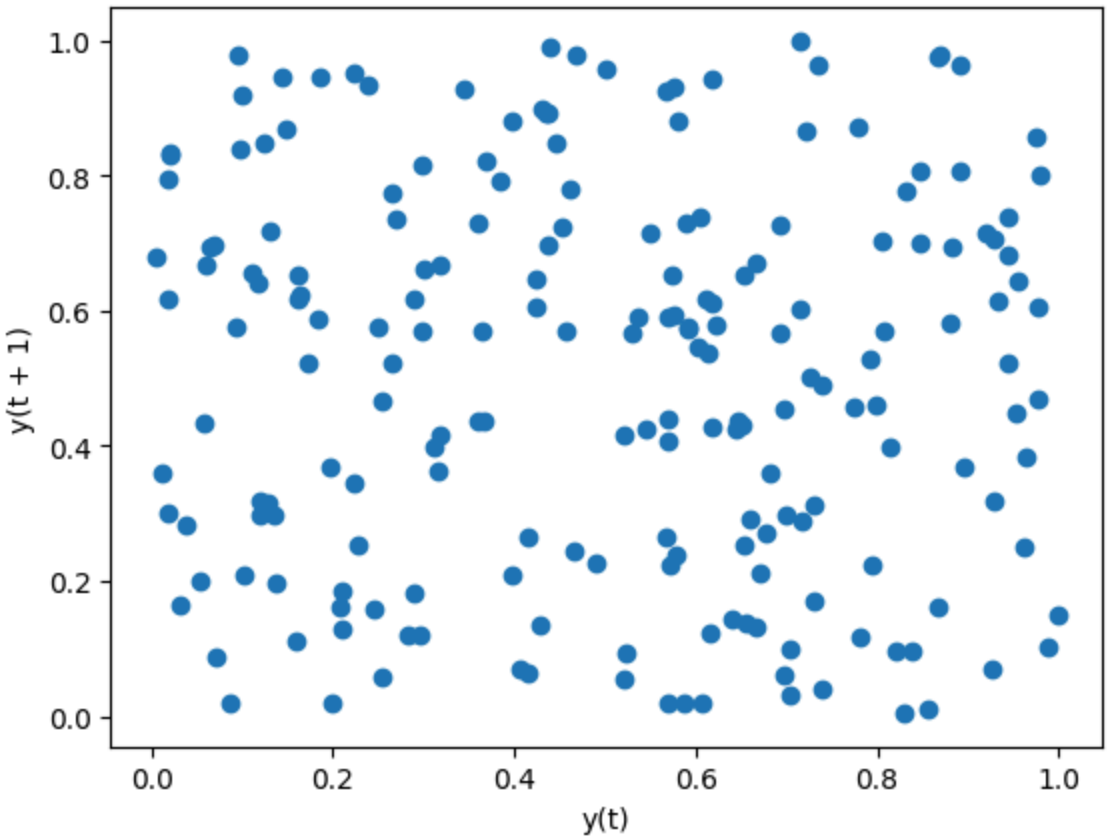
In [69]:
```
scatter_matrix(fb, figsize=(10, 10), diagonal='kde')

plt.show()
```



Lag plot

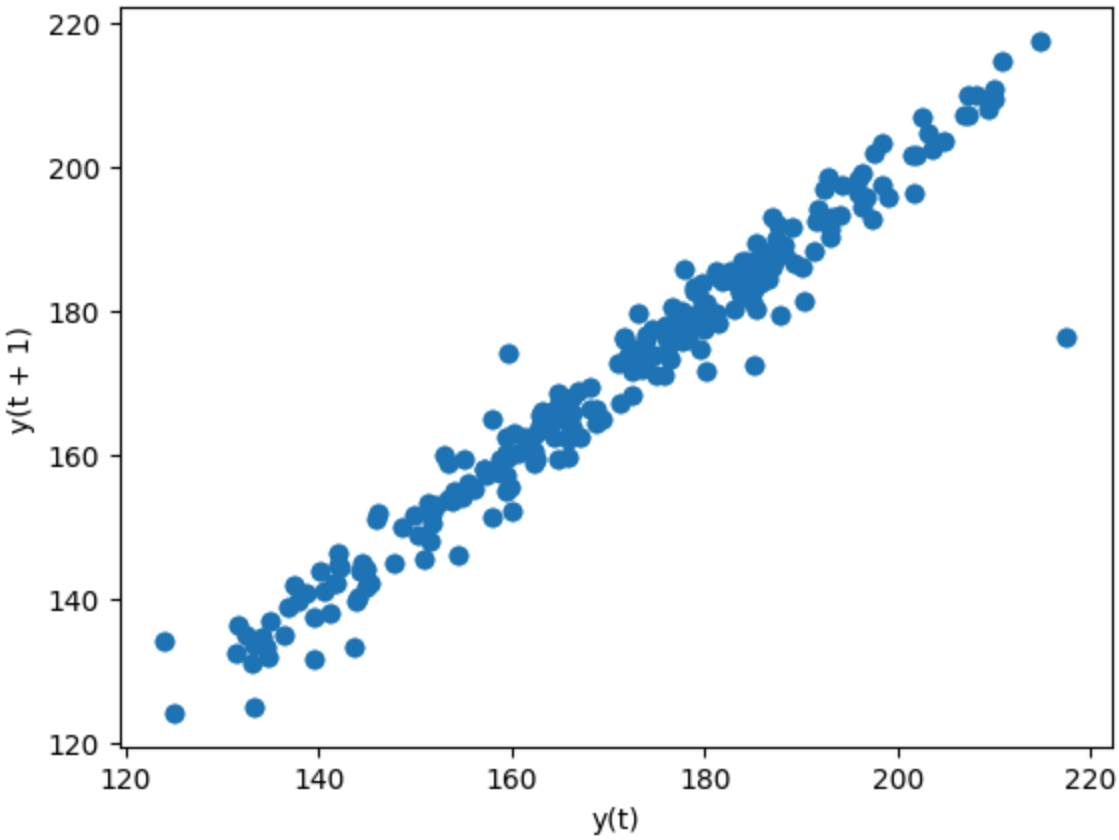
Lag plots let us see how the variable correlations with past observations of itself. Random data has no pattern:

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

plt.show()
```



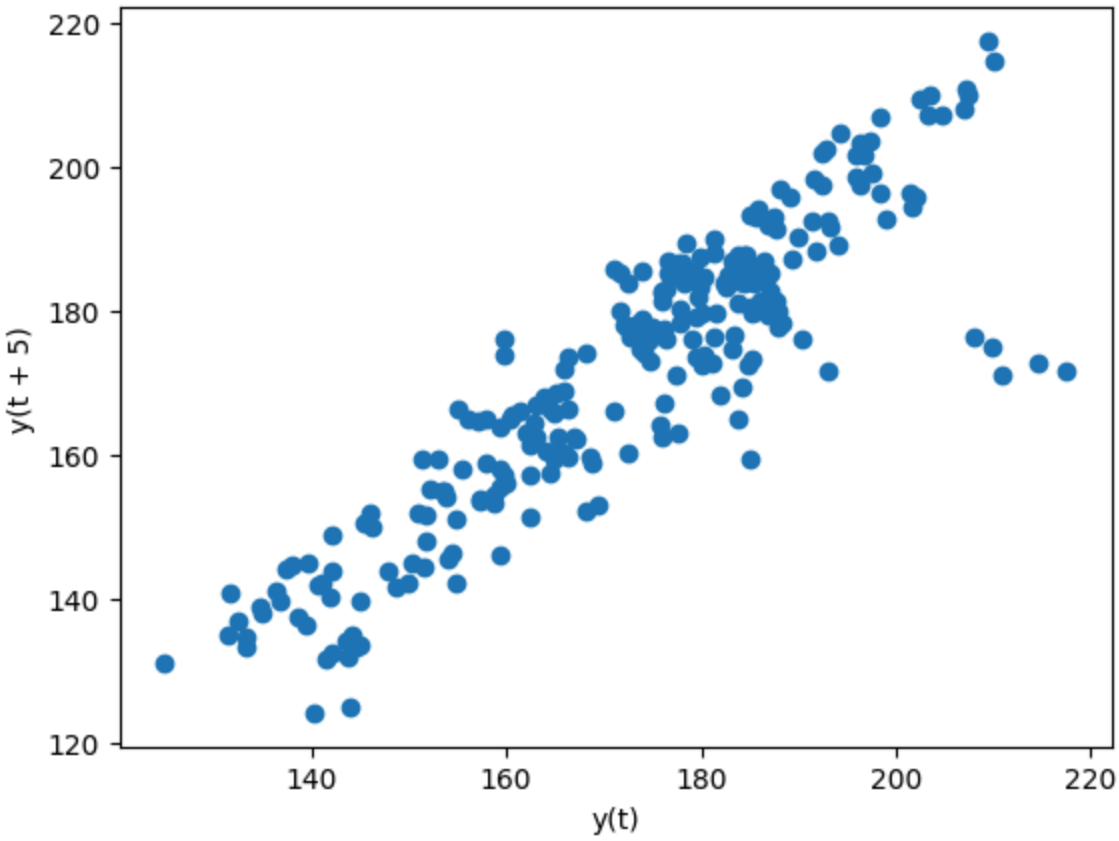
Data with some level of correlation to itself (autocorrelation) may have patterns. Stock prices are highly auto-correlated:

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):

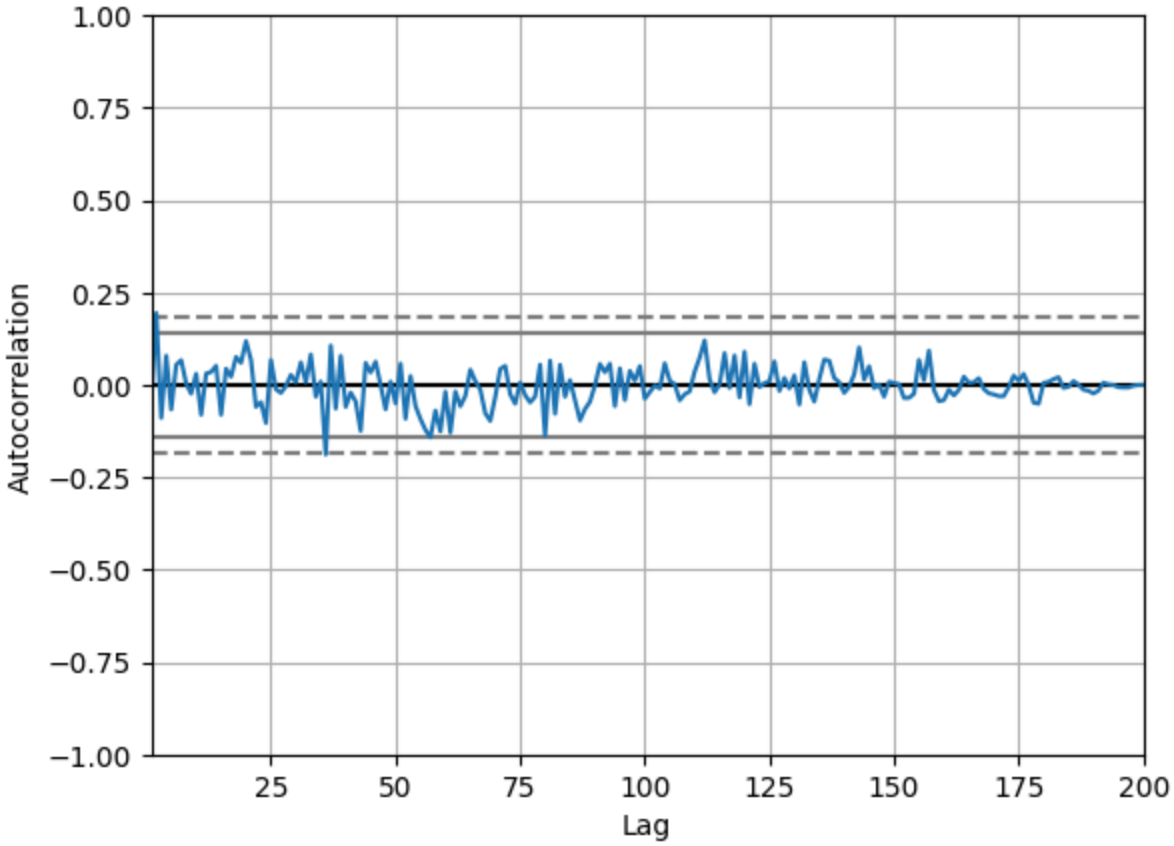
In [73]:
```
lag_plot(fb.close, lag=5)

plt.show()
```



Autocorrelation plots

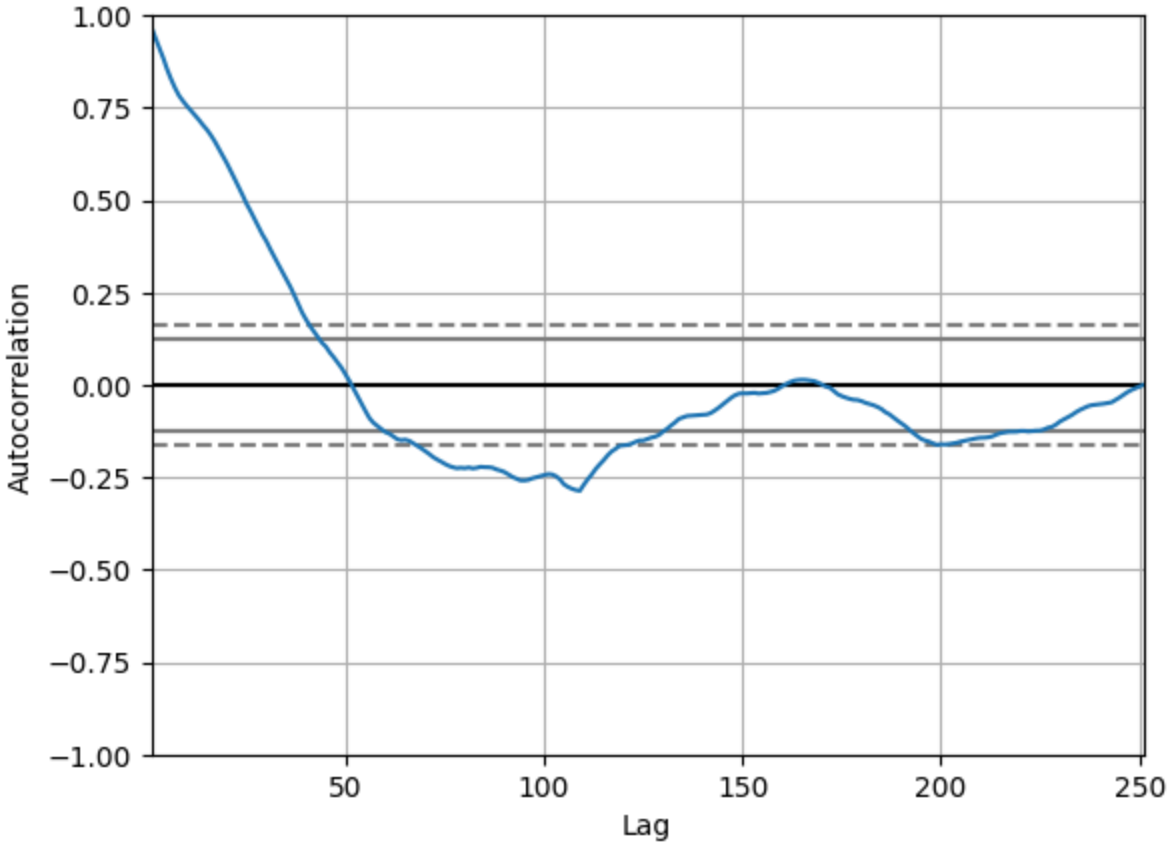
We can use the autocorrelation plot to see if this relationship may be meaningful or just noise. Random data will not have any significant autocorrelation (it stays within the bounds below):

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



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

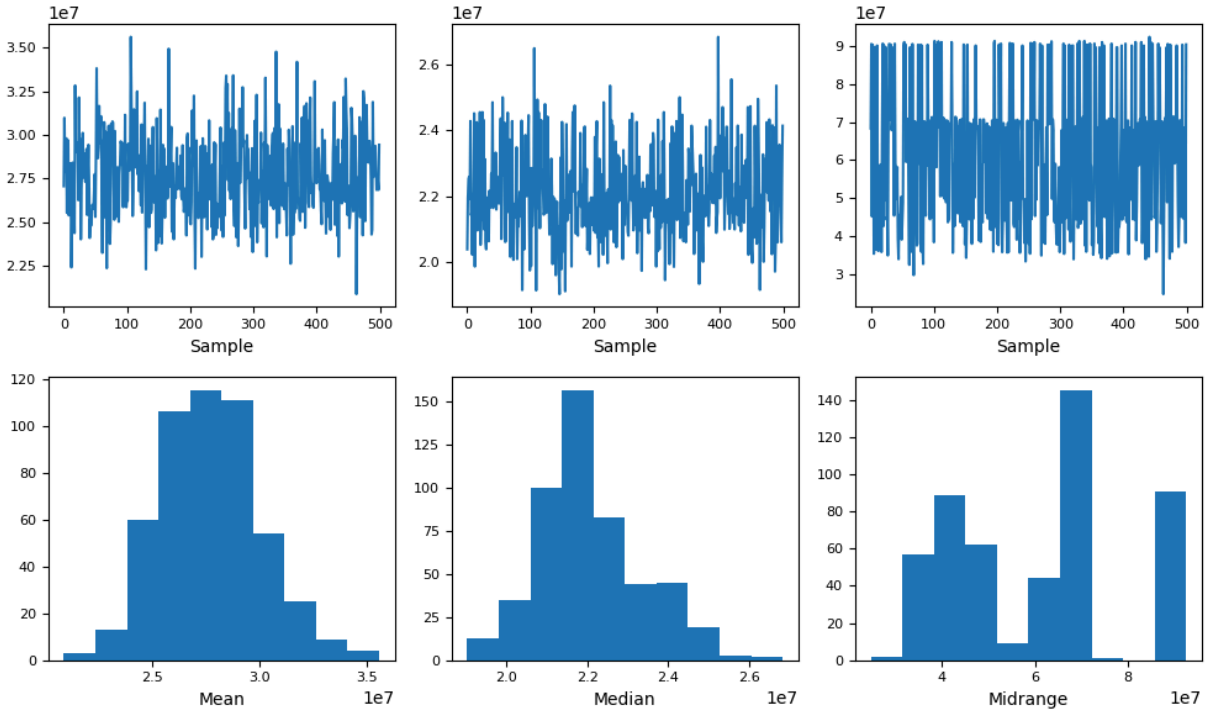
In [75]:
```
autocorrelation_plot(fb.close)
plt.show()
```



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:

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 [217…

```
import matplotlib.pyplot as plt # library for data visualization
import seaborn as sns # library for data visualization
import pandas as pd # library to read files and to dataframes

df = pd.read_csv('earthquakes-1.csv')
df.head(1)
```

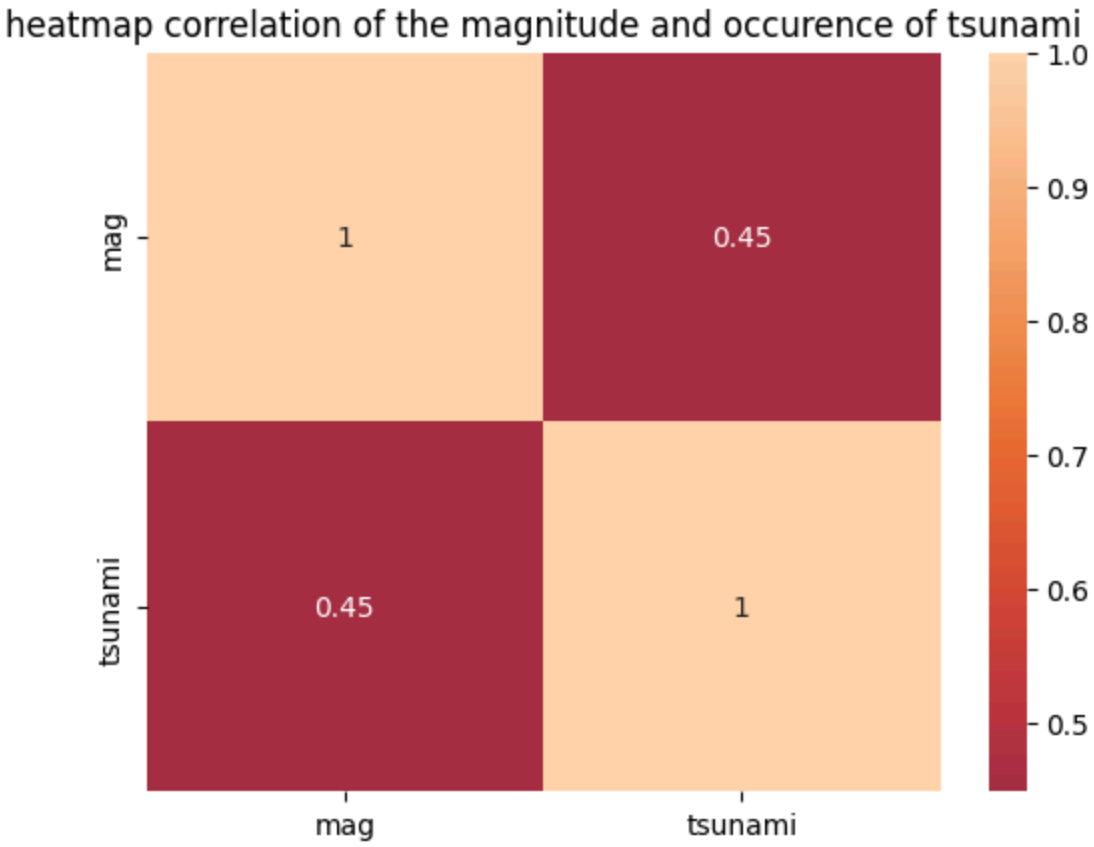
Out[217…

	mag	magType	time	place	tsunami	parsed_place
0	1.35	ml	1539475168010	9km NE of Aguanga, CA	0	California

In [220…

```
a = df[(df['magType'] == 'mb')]
# creates a dataframe from the drated dataframe
#but filters it with the magType columns that has
# mb in the entries
```

```
sns.heatmap(a[['mag','tsunami']]
# calls out the mag and tsunami columns
# to be used in the heatmap
.corr(),annot=True, center=0)
# the .corr is placed to show correlation
plt.title('heatmap correlation of the magnitude and occurence of tsunami')
plt.show()
```



In [ ]:

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 [219…
```
fb = pd.read_csv('fb_stock_prices_2018.csv') # this read the needed data
fb.head(1)
```

Out[219…

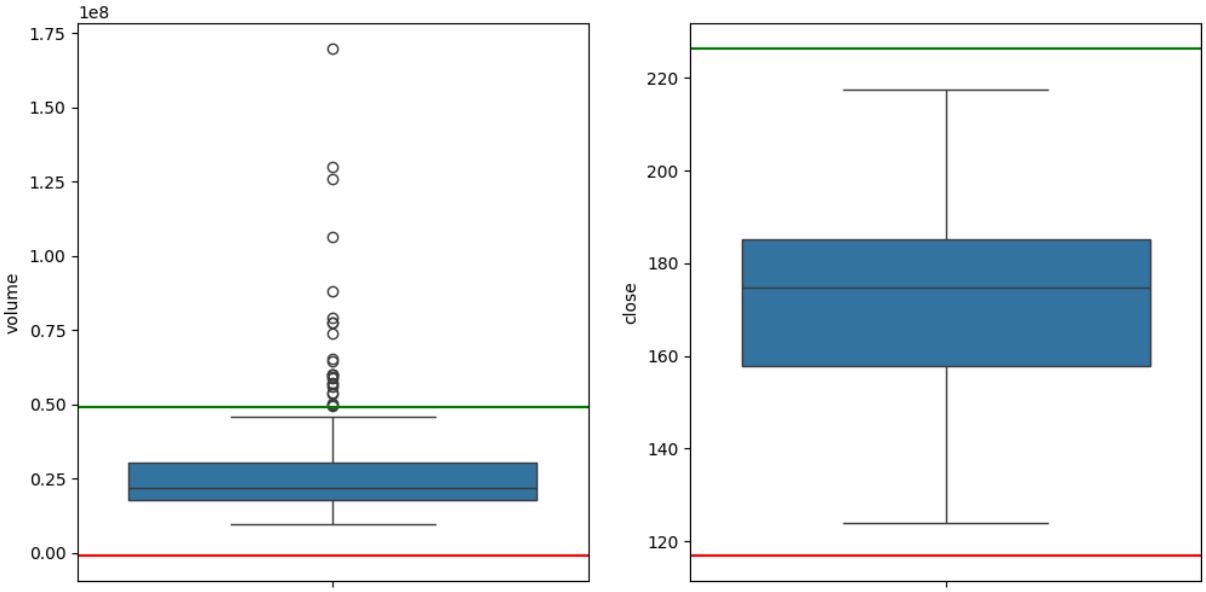
	date	open	high	low	close	volume
0	2018-01-02	177.68	181.58	177.55	181.42	18151903

In [144…
```
# using quantile() to take the quartile of the volum and close values
#0.25 is the first quartile
#0.75 is the third quartile
```

```
# the IQR is equal to Q3 - Q1
# then using the formula usaed for the bounds
Q1_volume = fb['volume'].quantile(0.25)
Q3_volume = fb['volume'].quantile(0.75)
IQR_volume = Q3_volume - Q1_volume

Q1_close = fb['close'].quantile(0.25)
Q3_close = fb['close'].quantile(0.75)
IQR_close = Q3_close - Q1_close

lower_bound_volume = Q1_volume - 1.5 * IQR_volume
upper_bound_volume = Q3_volume + 1.5 * IQR_volume

lower_bound_close = Q1_close - 1.5 * IQR_close
upper_bound_close = Q3_close + 1.5 * IQR_close
```

In [145…

```
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

sns.boxplot(fb['volume'], ax=axes[0])
axes[0].axhline(lower_bound_volume, color = 'red')
axes[0].axhline(upper_bound_volume, color = 'green')

sns.boxplot(fb['close'], ax=axes[1])
axes[1].axhline(lower_bound_close, color = 'red')
axes[1].axhline(upper_bound_close, color = 'green')
plt.title('Box plot for facebook volume and closing prices with reference lines')
plt.show()
```



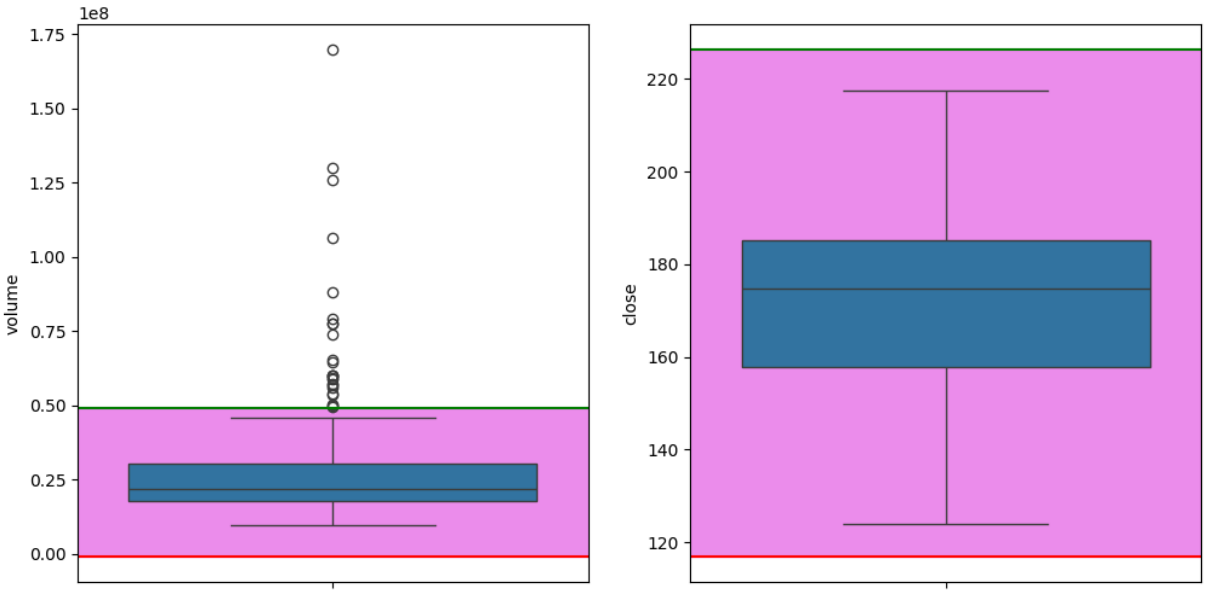
In [ ]:

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

In [146…

```
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Volume boxplot with bounds and filled area
sns.boxplot(fb['volume'], ax=axes[0])
```

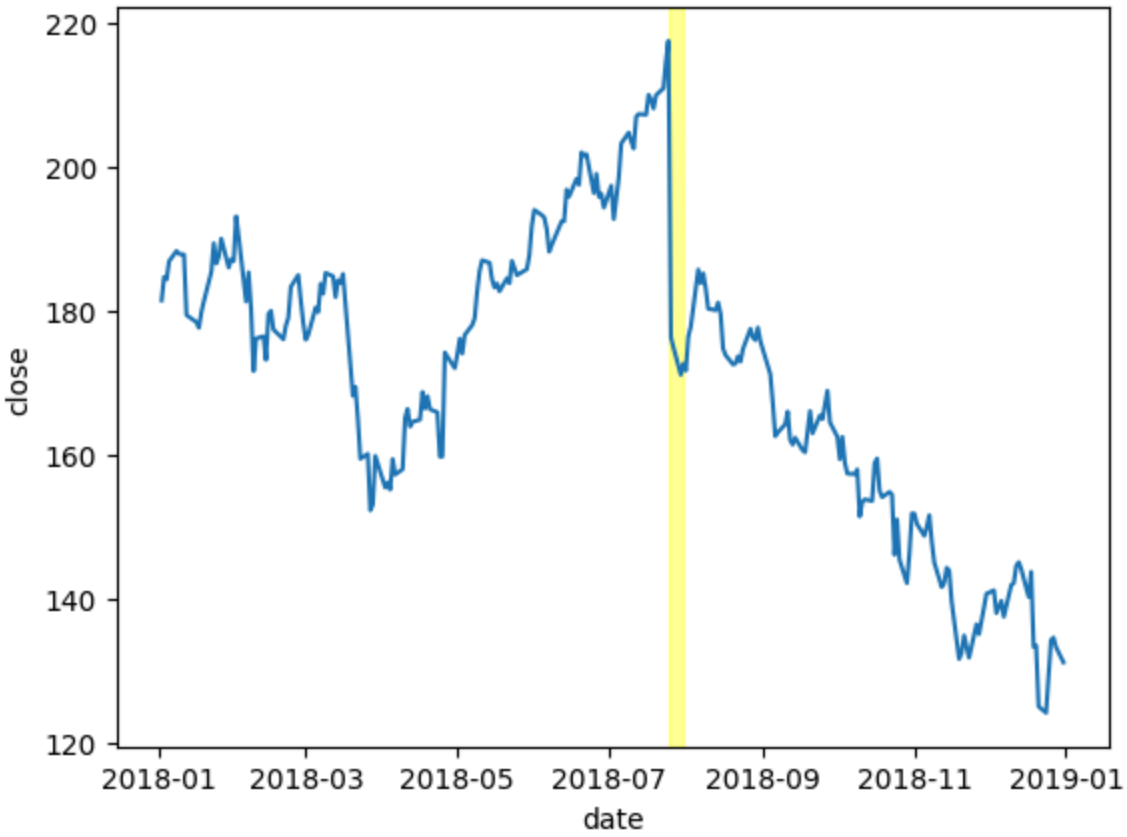
```
axes[0].axhline(lower_bound_volume, color='red')
axes[0].axhline(upper_bound_volume, color='green')
axes[0].axhspan(lower_bound_volume, upper_bound_volume, color = 'violet', alpha= 0.

sns.boxplot(fb['close'], ax=axes[1])
axes[1].axhline(lower_bound_close, color='red')
axes[1].axhline(upper_bound_close, color='green')
axes[1].axhspan(lower_bound_close, upper_bound_close, color = 'violet', alpha= 0.9)

plt.show()
```



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 [152…

```
fb['date'] = pd.to_datetime(fb['date'])
sns.lineplot(x='date', y='close', data=fb)
ax = plt.gca()
ax.axvspan('2018-07-25','2018-07-31', color = 'yellow', alpha= 0.4)
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 [208…

```
df3 = {'date':['2018-07-25', '2018-03-19', '2018-03-20'],'event':['Disappointing us
}
df3 = pd.DataFrame(df3)

df3.head()
```

Out[208…

	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 [209…

```
df3['date'] = pd.to_datetime(df3['date'])
df3.dtypes
```

```
df3.set_index('date',inplace = True)
```

In [210…
```
merge_df = fb.merge(df3, on = 'date',how = 'outer')
```

In [211…
```
merge_df = merge_df.sort_index()
```

In [212…
```
merge_df.set_index('date',inplace = True)

merge_df.loc['2018-03-19':'2018-03-20']
```

Out[212…

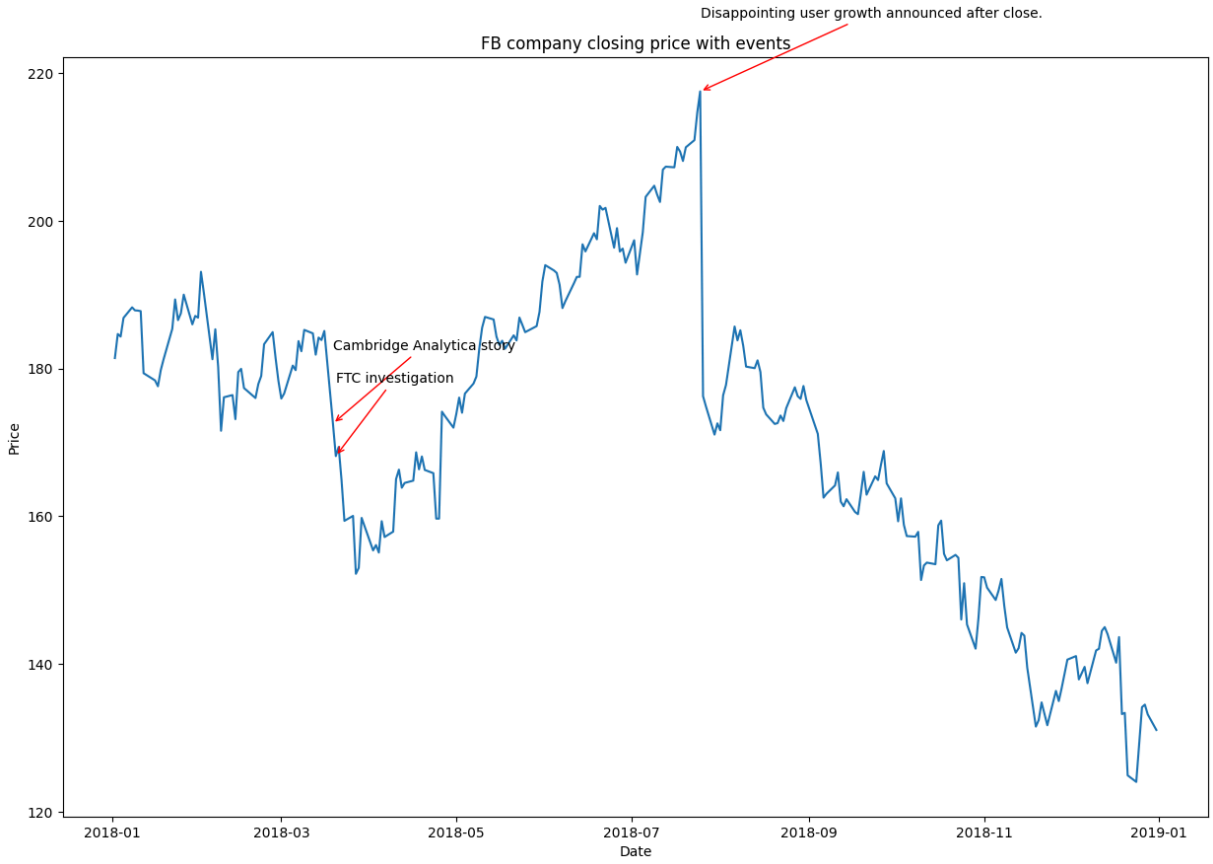
date	open	high	low	close	volume	event
2018-03-19	177.01	177.17	170.06	172.56	88140060	Cambridge Analytica story
2018-03-20	167.47	170.20	161.95	168.15	129851768	FTC investigation

In [213…
```
close_price = merge_df['close']
```

In [214…
```
annotate = merge_df['event']
annotate.dropna(inplace= True)

annotate
```

Out[214…
```
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 [215…
```
fig, ax = plt.subplots(figsize = (15,10))
sns.lineplot(close_price)
for date, event in annotate.items(): # using the items() we during our iterartion w
    ax.annotate(event, #here we insert the event
                xy= (date, close_price.loc[date]), # for the xy part this will be t
                xytext = (date, close_price.loc[date]+ 10), #for the xytext this wi
                arrowprops = dict(arrowstyle='->', color='red') # this part refers
               )
plt.title('FB company closing price with events')
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```

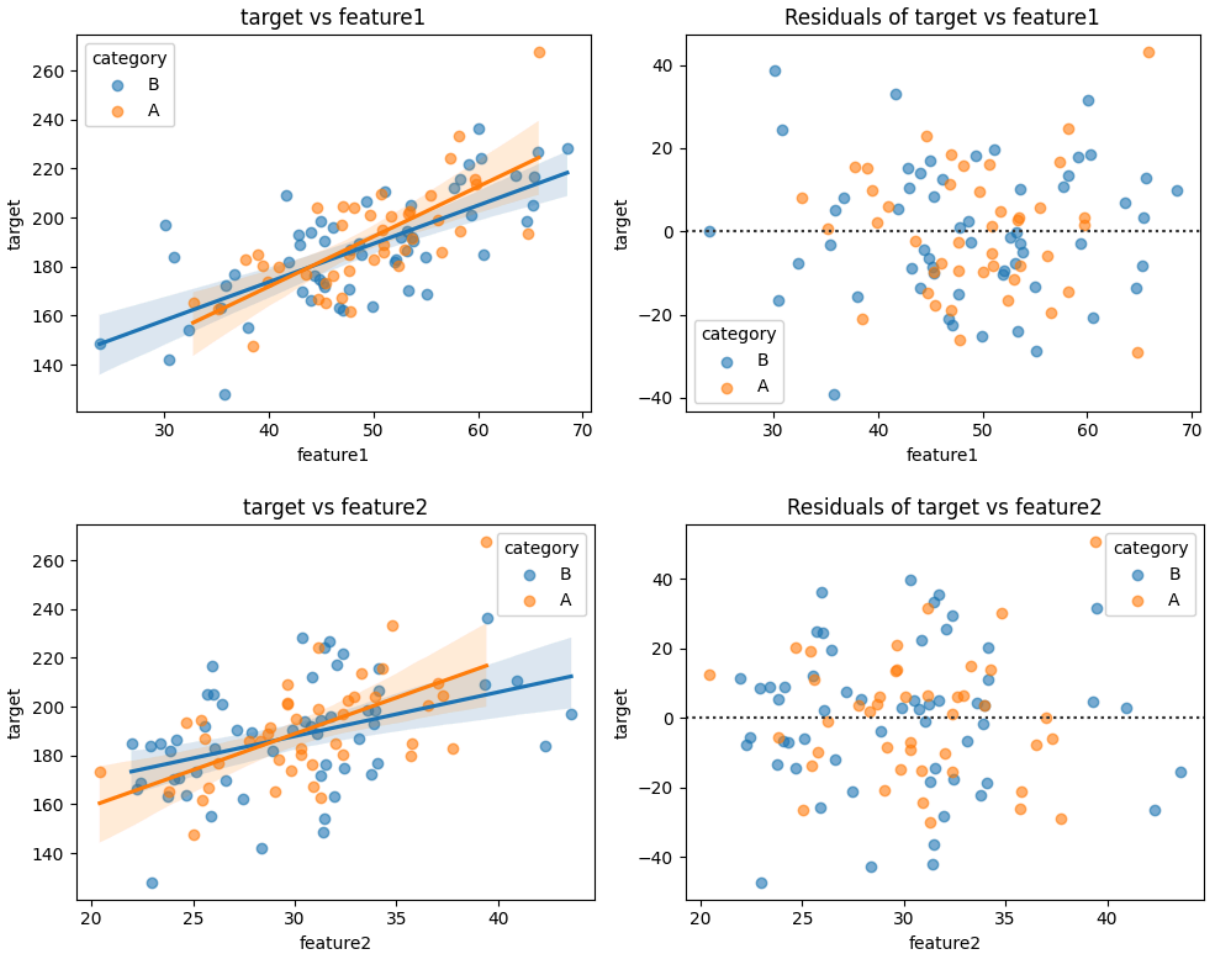


6. Modify the reg_resid_plots() function to use a matplotlib colormap instead of cycling between two colors. Remember, for this use case, we should pick a qualitative colormap or make our own

In [267…

```
def simple_reg_resid(data, x_vars, y_var, hue=None):
    for x in x_vars:
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
        if hue:
            for level in data[hue].unique():
                subset = data[data[hue] == level]
                sns.regplot(data=subset, x=x, y=y_var, ax=ax1, label=level, scatter
                sns.residplot(data=subset, x=x, y=y_var, ax=ax2, label=level, scatt
            ax1.legend(title=hue)
            ax2.legend(title=hue)
        else:
            sns.regplot(data=data, x=x, y=y_var, ax=ax1, scatter_kws={'alpha': 0.6}
            sns.residplot(data=data, x=x, y=y_var, ax=ax2, scatter_kws={'alpha': 0.
        ax1.set_title(f'{y_var} vs {x}')
        ax2.set_title(f'Residuals of {y_var} vs {x}')
        plt.tight_layout()
        plt.show()

# Sample data
np.random.seed(42)
n = 100
df = pd.DataFrame({
    'feature1': np.random.normal(50, 10, n),
```

```
    'feature2': np.random.normal(30, 5, n),
    'category': np.random.choice(['A', 'B'], n)
})
df['target'] = 2 * df['feature1'] + 3 * df['feature2'] + np.random.normal(0, 10, n)

# Run it
simple_reg_resid(df, ['feature1', 'feature2'], 'target', hue='category')
```



Summary/Conclusion:

> Provide a summary of your learnings and the conclusion for this activity.

This activity is has given me a lot of information about visualizing data, seaborn and matplotlib is very useful in data visualizing, it is somehow easy to understand, but in the supplementary, it is hard, because I misinterpreted some of the instructions, but I was still able to do what was needed. I need more practice about this, because I am having a hard time doing what needs to be done

In [ ]: