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
- earthquakes-1.csv

Procedures:

- 9.4 Introduction to Seaborn
- 9.5 Formatting Plots
- 9.6 Customizing Visualizations

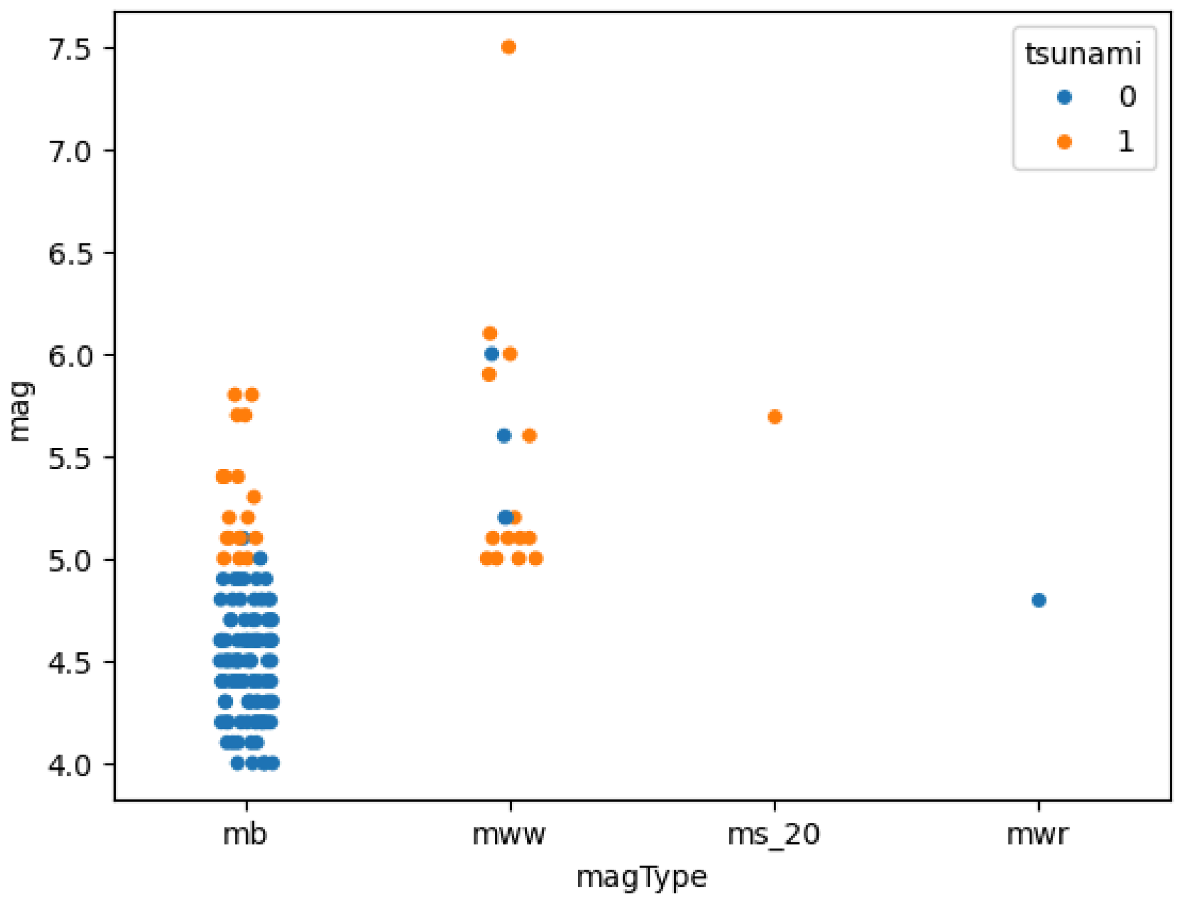
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
fb = pd.read_csv('/content/fb_stock_prices_2018.csv', index_col='date', parse_dates=True
)
quakes = pd.read_csv('/content/earthquakes.csv')
```

```
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"
)
```

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

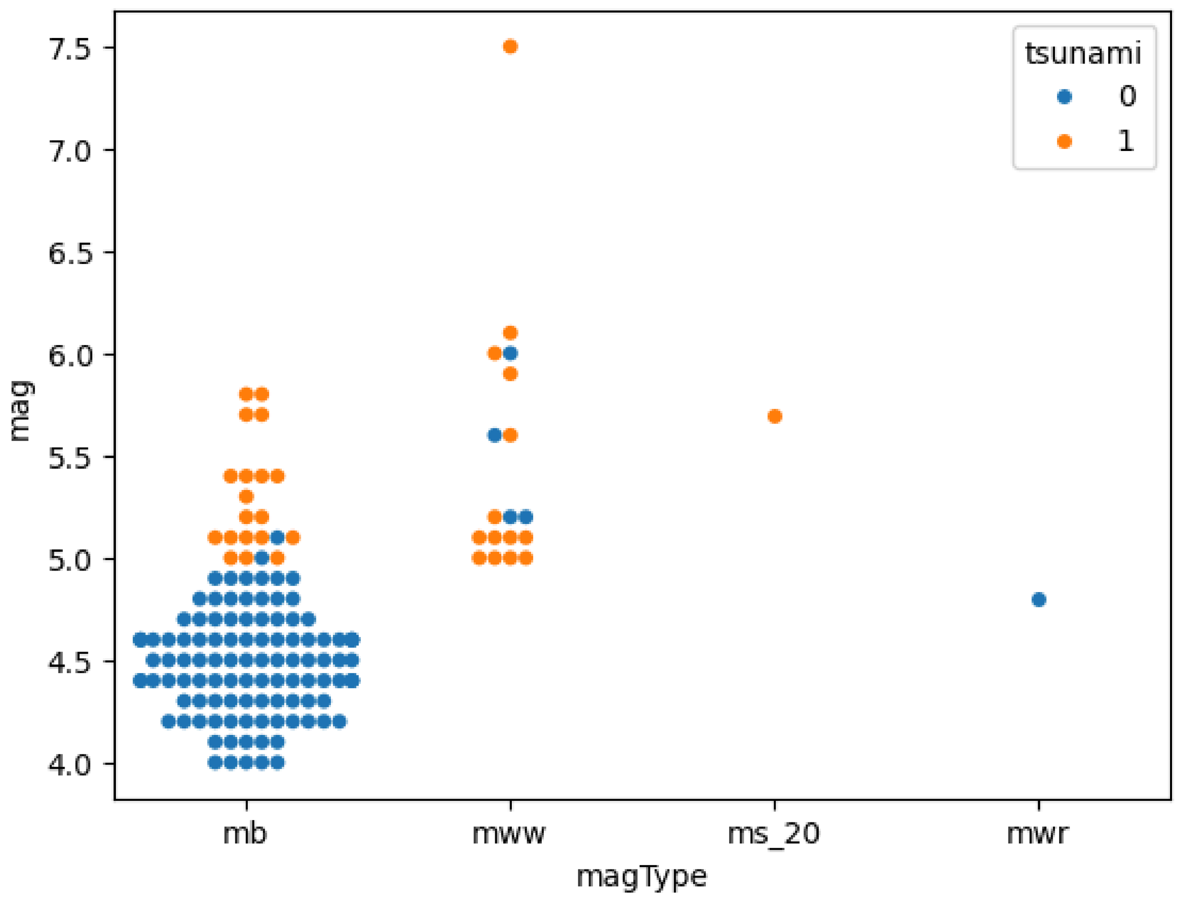
```
sns.stripplot(
x='magType',
y='mag',
hue='tsunami',
data=quakes.query('parsed_place == "Indonesia"')
)
```

```
<Axes: xlabel='magType', ylabel='mag'>
```



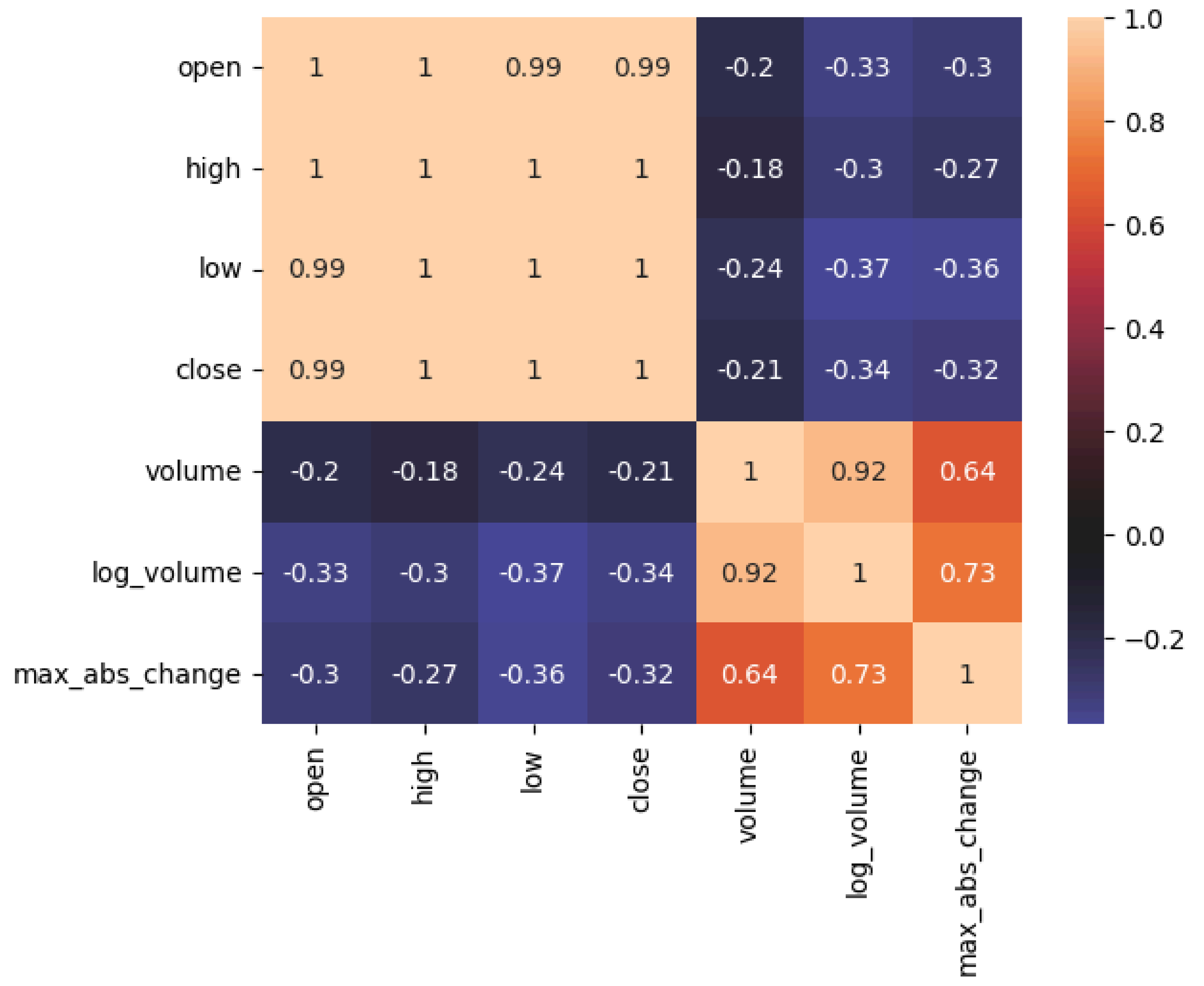

```
sns.swarmplot(
x='magType',
y='mag',
hue='tsunami',
data=quakes.query('parsed_place == "Indonesia"')
)
```

```
<Axes: xlabel='magType', ylabel='mag'>
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3398: 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)
```



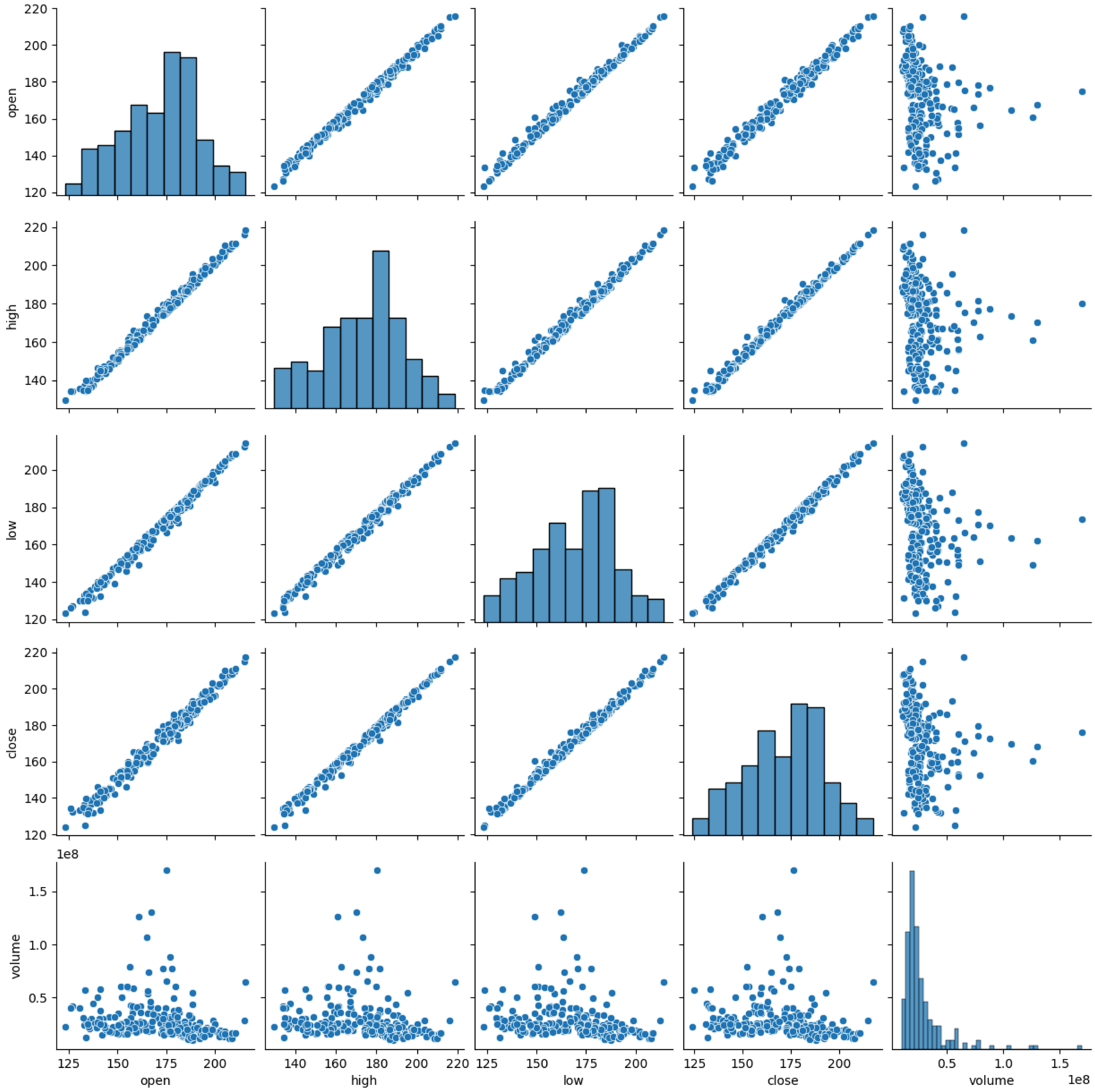

```
sns.heatmap(
fb.sort_index().assign(
log_volume=np.log(fb.volume),
max_abs_change=fb.high - fb.low
).corr(),

annot=True, center=0
)
```

```
<Axes: >
```



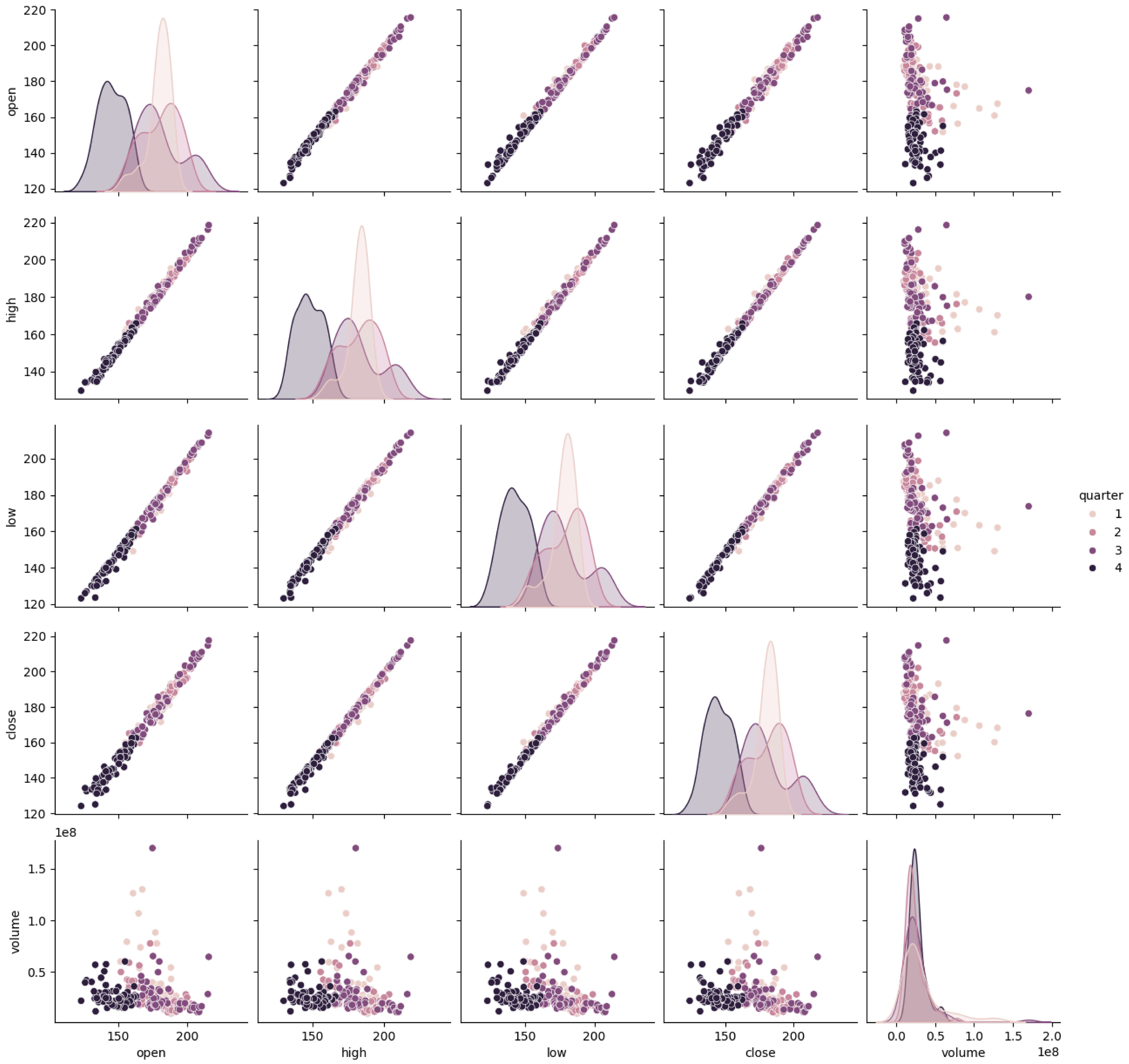

```
sns.pairplot(fb)
```

```
<seaborn.axisgrid.PairGrid at 0x79ab93ed3c40>
```



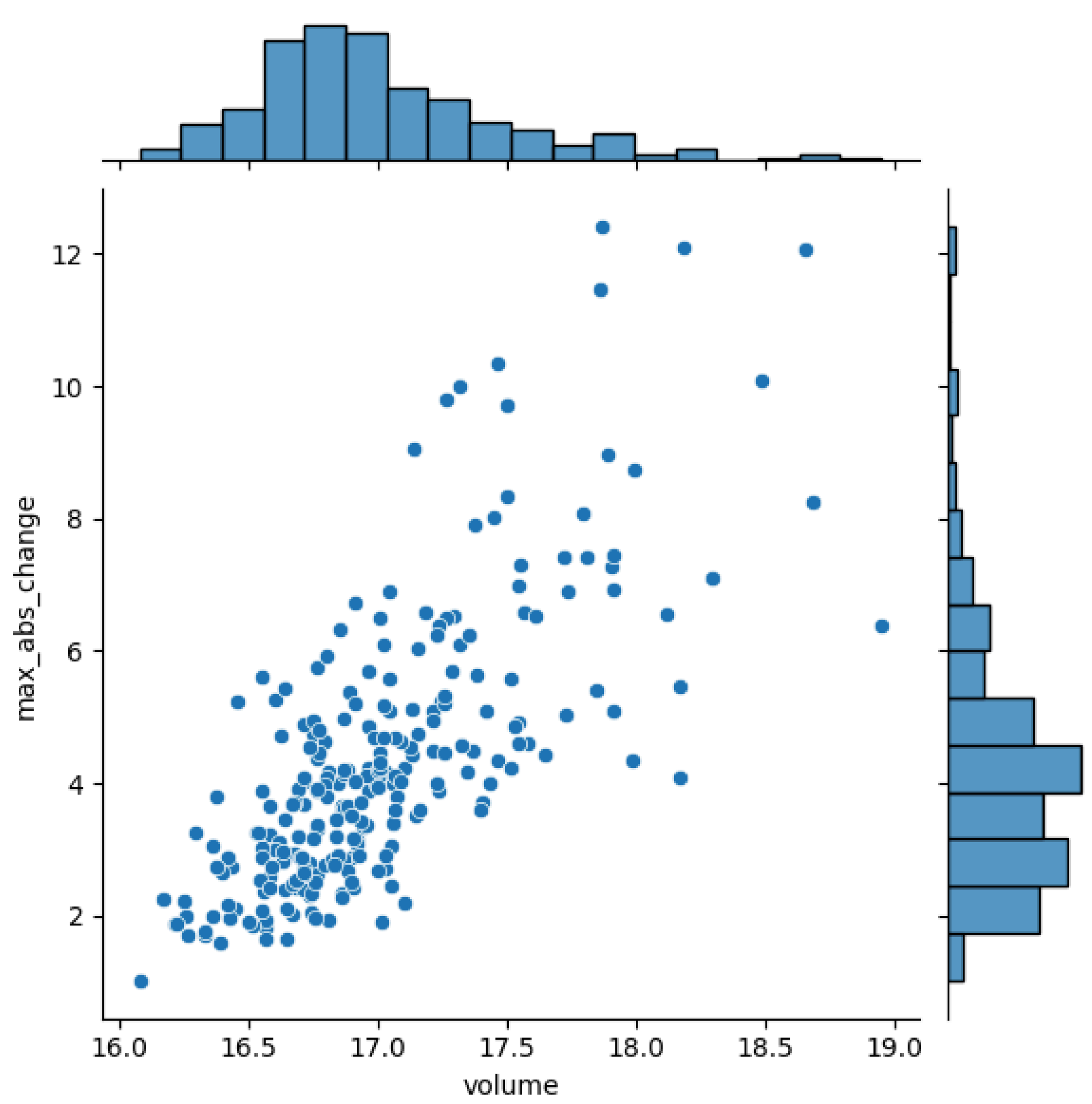

```
sns.pairplot(
fb.assign(quarter=lambda x: x.index.quarter),
diag_kind='kde',
hue='quarter'
)
```

<seaborn.axisgrid.PairGrid at 0x79ab94a4c340>



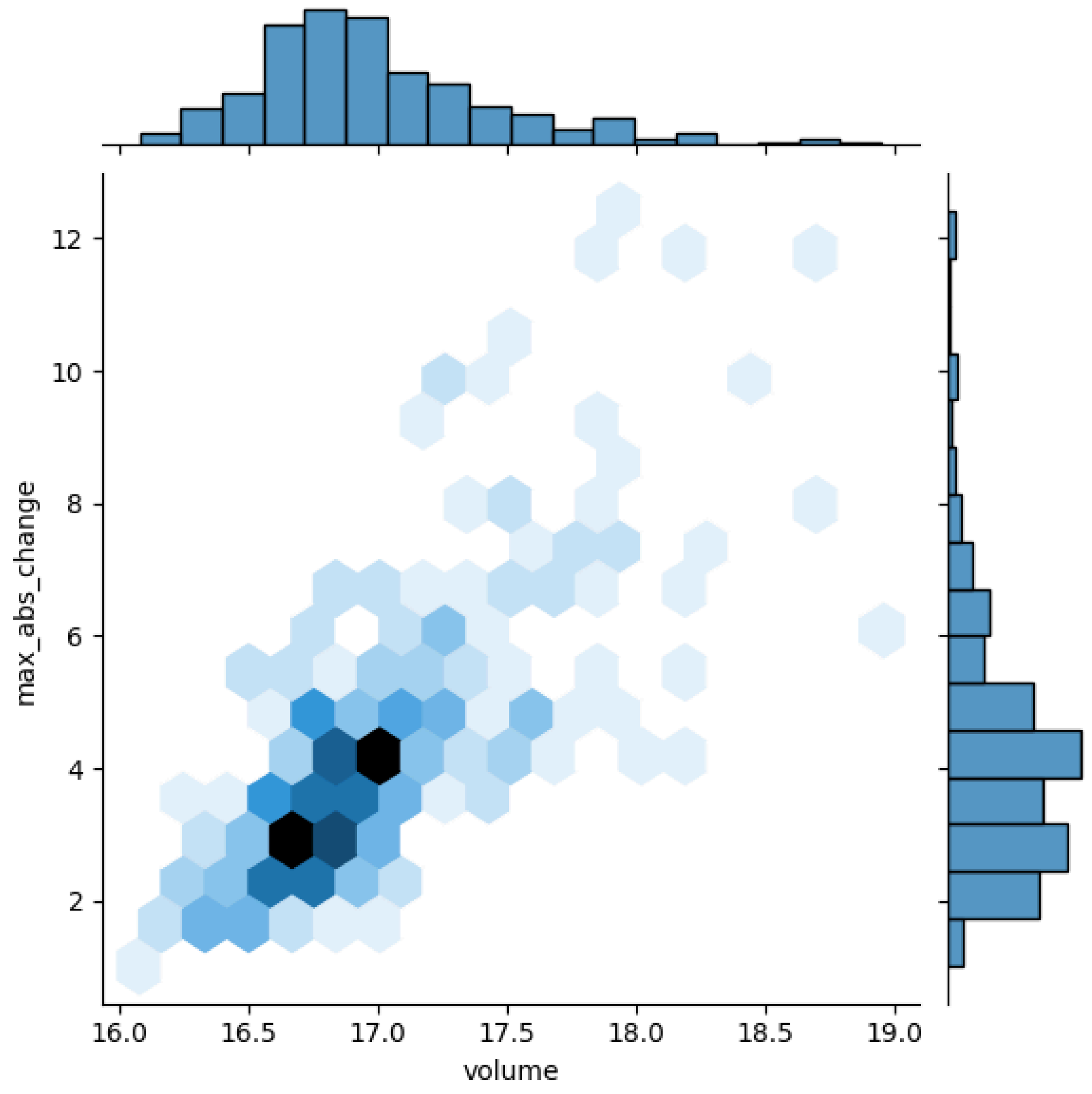

```
sns.jointplot(
x='volume',
y='max_abs_change',
data=fb.assign(
volume=np.log(fb.volume),
max_abs_change=fb.high - fb.low
)
)
```

<seaborn.axisgrid.JointGrid at 0x79ab92f2ded0>



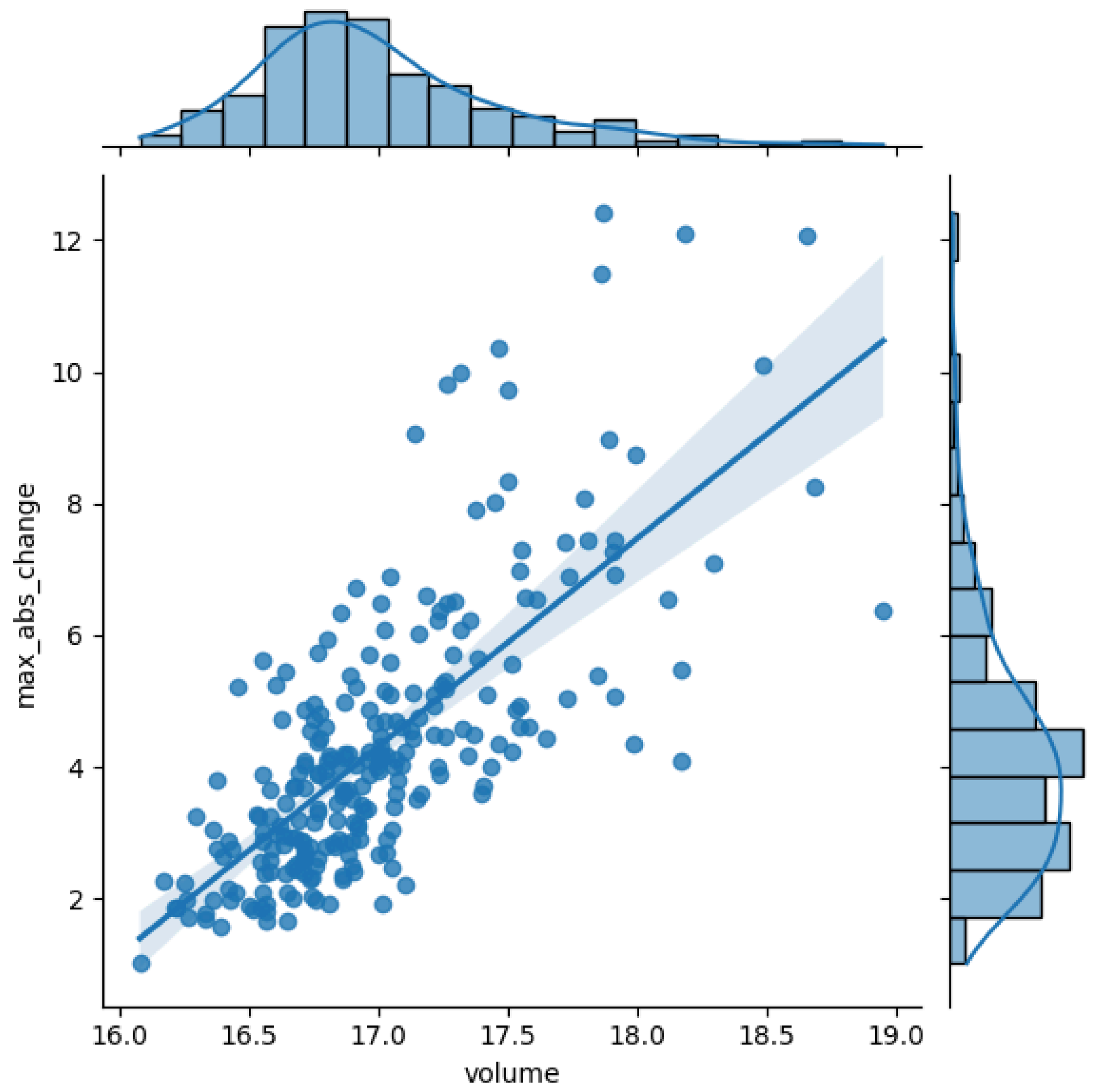

```
sns.jointplot(
x='volume',
y='max_abs_change',
kind='hex',
data=fb.assign(
volume=np.log(fb.volume),
max_abs_change=fb.high - fb.low
)
)
```

<seaborn.axisgrid.JointGrid at 0x79ab9341d810>



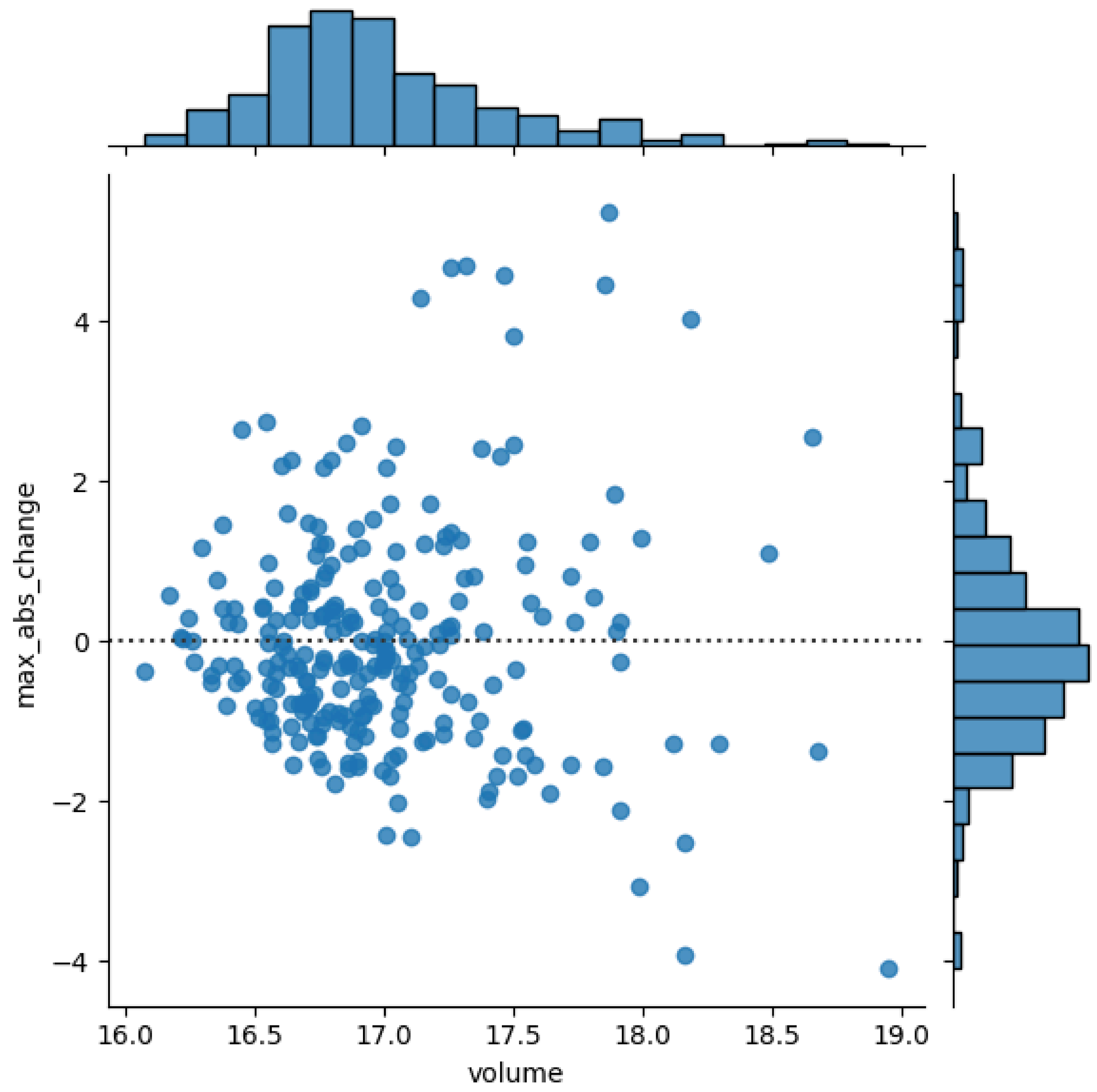

```
sns.jointplot(
x='volume',
y='max_abs_change',
kind='reg',
data=fb.assign(
volume=np.log(fb.volume),
max_abs_change=fb.high - fb.low
)
)
```

<seaborn.axisgrid.JointGrid at 0x79ab93d30c40>



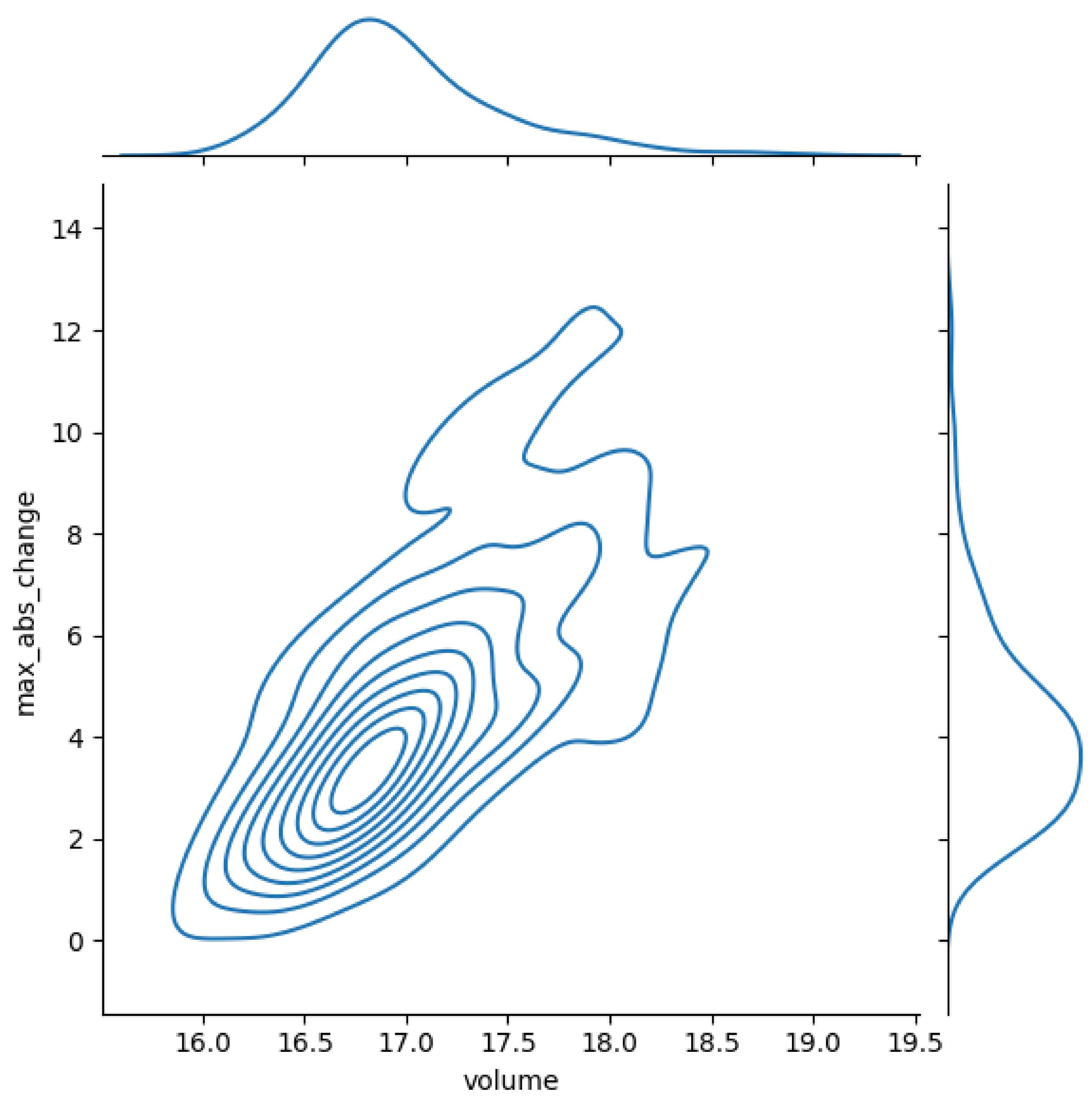

```
sns.jointplot(
x='volume',
y='max_abs_change',
kind='resid',
data=fb.assign(
volume=np.log(fb.volume),
max_abs_change=fb.high - fb.low
)
)
```

<seaborn.axisgrid.JointGrid at 0x79ab92800970>



```
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
)
)
```

```
<seaborn.axisgrid.JointGrid at 0x79ab92802830>
```




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

```
import itertools

iterator = itertools.repeat("I'm an iterator", 1)
for i in iterator:
    print(f'-->{i}')
print('This printed once because the iterator has been exhausted')

# Reset the iterator
iterator = itertools.repeat("I'm an iterator", 1)
for i in iterator:
    print(f'-->{i}')
```

```
-->I'm an iterator
This printed once because the iterator has been exhausted
-->I'm an iterator
```

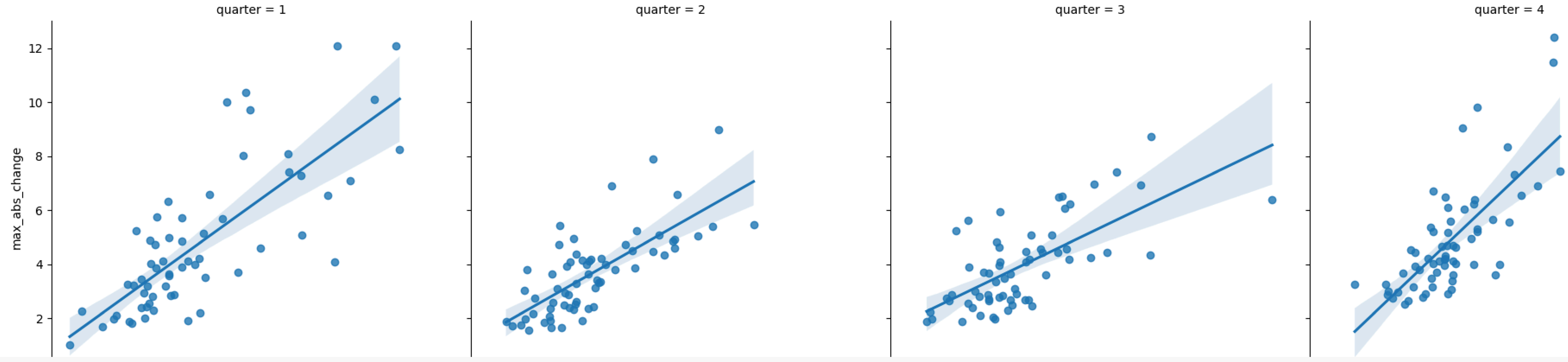
```
import itertools

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
```

```
import itertools

import matplotlib.pyplot as plt
import seaborn as sns

def reg_resid_plots(data):
    """
    Using seaborn, plot the regression and residuals
    plots side-by-side for every permutation of 2 columns
    in the data.

    Parameters:
        - data: A pandas DataFrame

    Returns:
        A matplotlib Figure object.
    """
    num_cols = data.shape[1]
    permutation_count = num_cols * (num_cols - 1)

    fig, ax = plt.subplots(permutation_count, 2, figsize=(15, 8))

    for (x, y), axes, color in zip(
        itertools.permutations(data.columns, 2),
        ax,
        itertools.cycle(['royalblue', 'darkorange'])
    ):
        for subplot, func in zip(axes, (sns.regplot, sns.residplot)):
            func(x=x, y=y, data=data, ax=subplot, color=color)
    plt.close()
    return fig
```

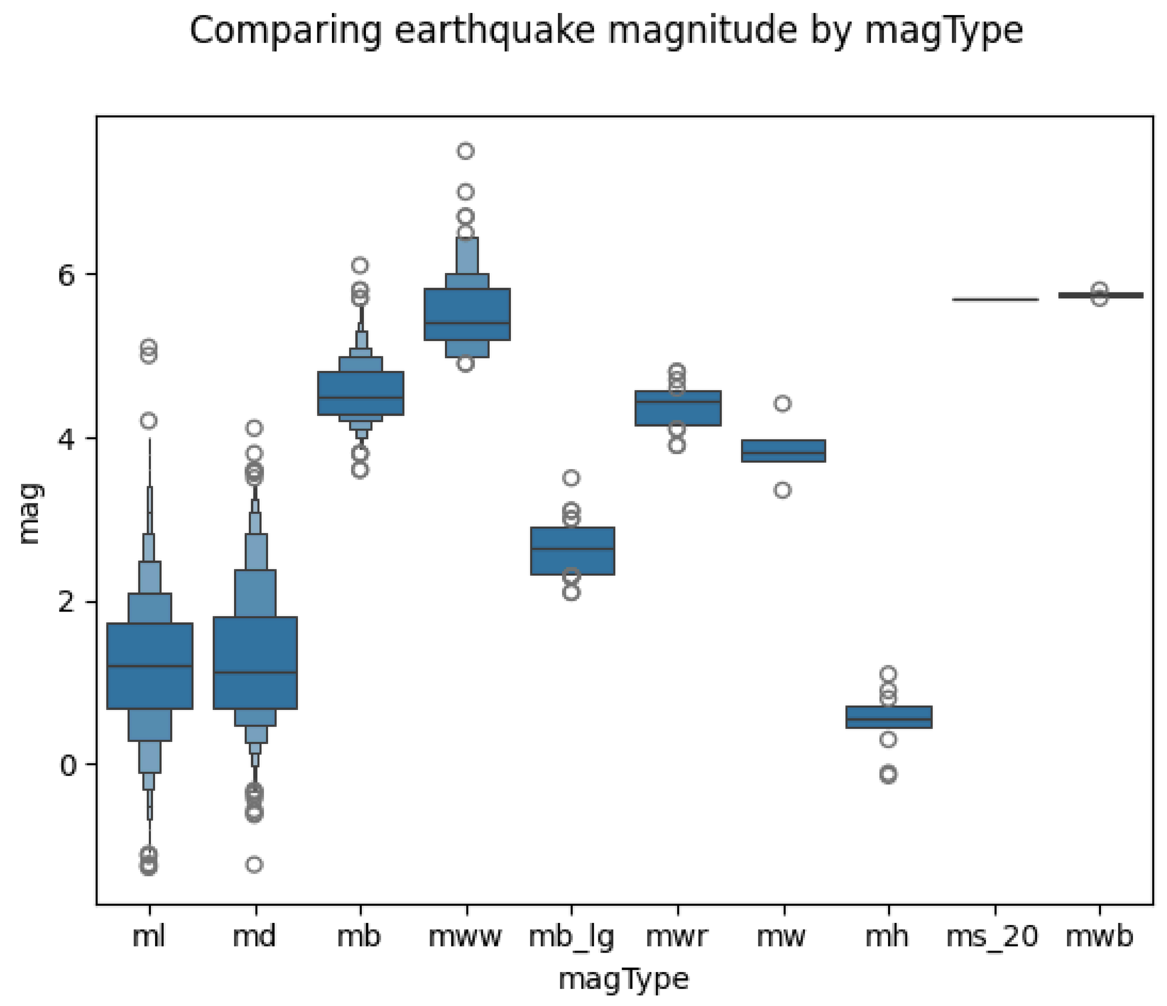
```
sns.lmplot(
x='volume',
y='max_abs_change',
data=fb.assign(
volume=np.log(fb.volume),
max_abs_change=fb.high - fb.low,
quarter=lambda x: x.index.quarter
),
col='quarter'
)
```

```
<seaborn.axisgrid.FacetGrid at 0x79ab926db730>
```



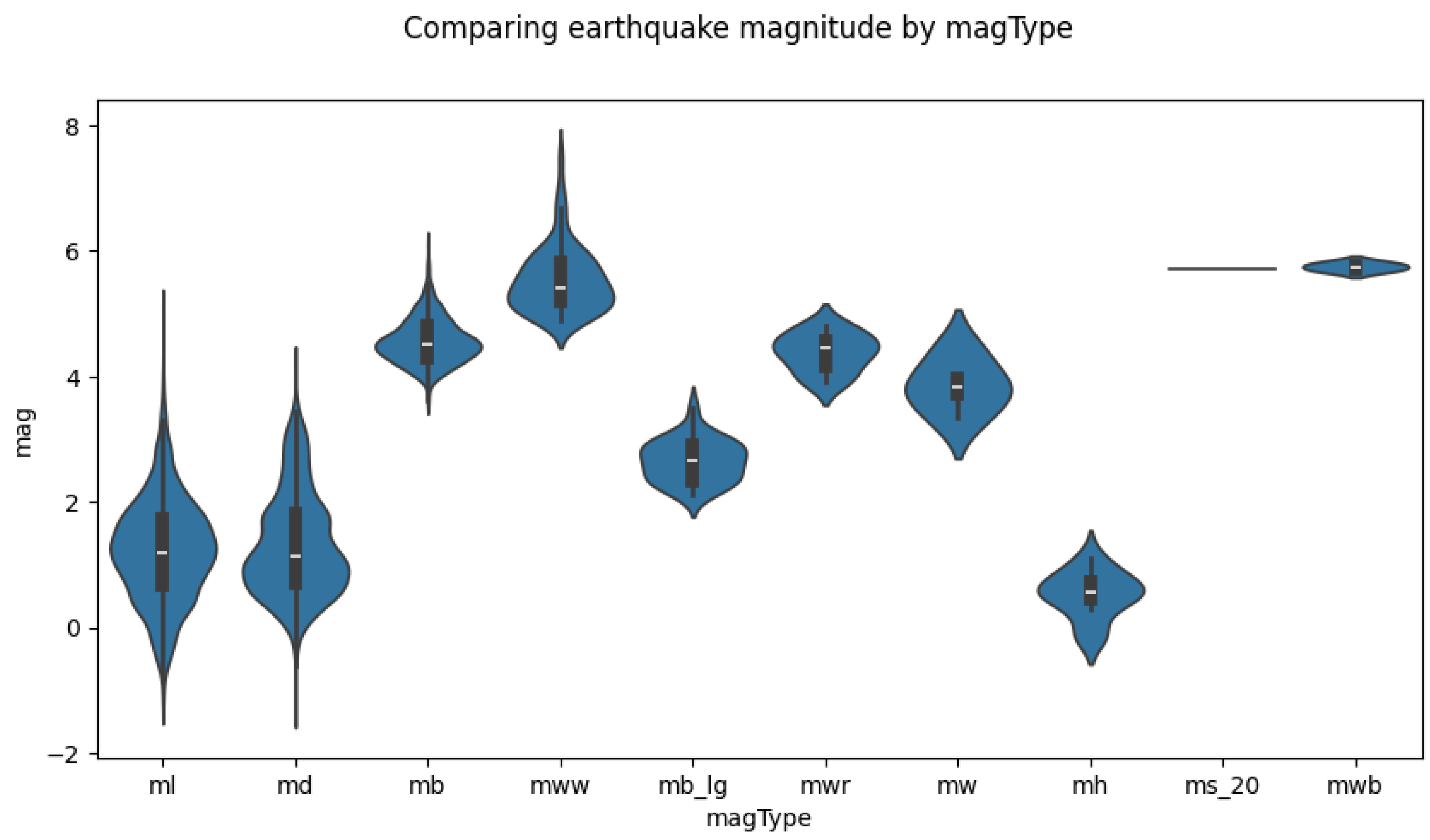

```
sns.boxenplot(
x='magType', y='mag', data=quakes[['magType', 'mag']]
)
plt.suptitle('Comparing earthquake magnitude by magType')
```

```
Text(0.5, 0.98, 'Comparing earthquake magnitude by magType')
```

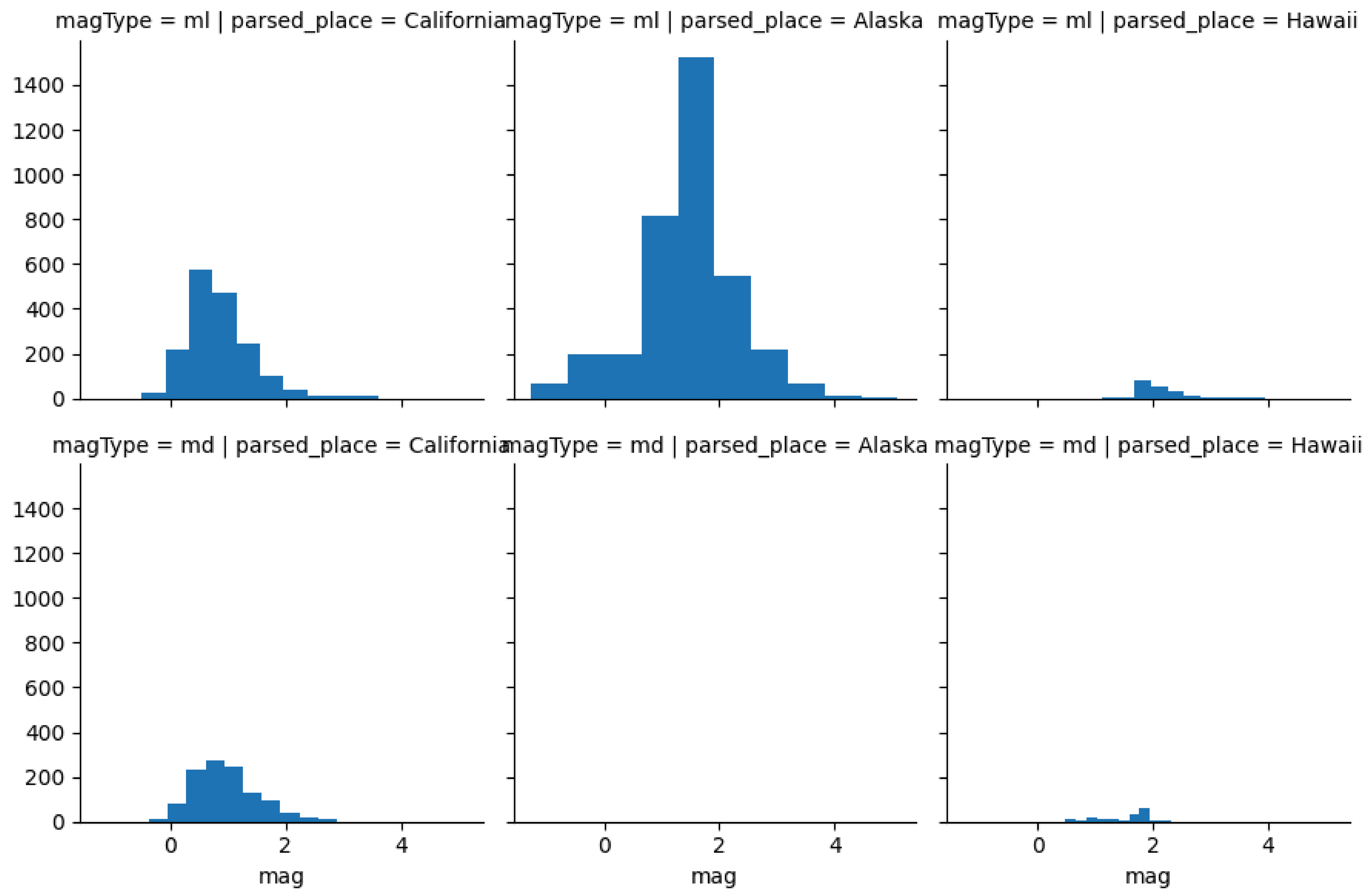



```
fig, axes = plt.subplots(figsize=(10, 5))
sns.violinplot(
x='magType', y='mag', data=quakes[['magType', 'mag']],
ax=axes, scale='width' # all violins have same width
)
plt.suptitle('Comparing earthquake magnitude by magType')
```

```
<ipython-input-95-4950d579cebb>:2: FutureWarning:

The `scale` parameter has been renamed and will be removed in v0.15.0. Pass `density_norm='width'` for the same effect.
  sns.violinplot(
Text(0.5, 0.98, 'Comparing earthquake magnitude by magType')
```



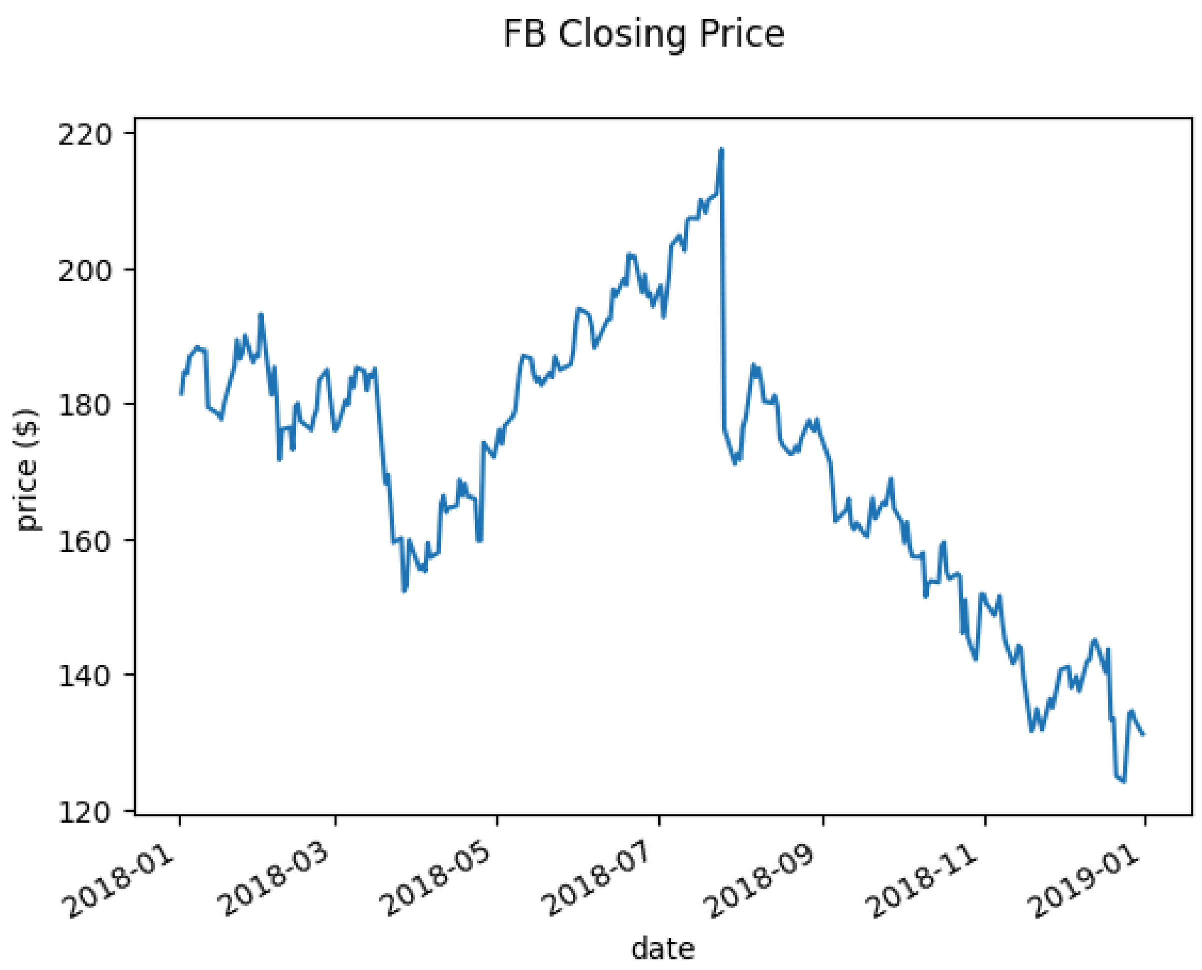

```
g = sns.FacetGrid(
quakes[
(quakes.parsed_place.isin([
'California', 'Alaska', 'Hawaii'
]))\
& (quakes.magType.isin(['ml', 'md']))
],
row='magType',
col='parsed_place'
)
g = g.map(plt.hist, 'mag')
```




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

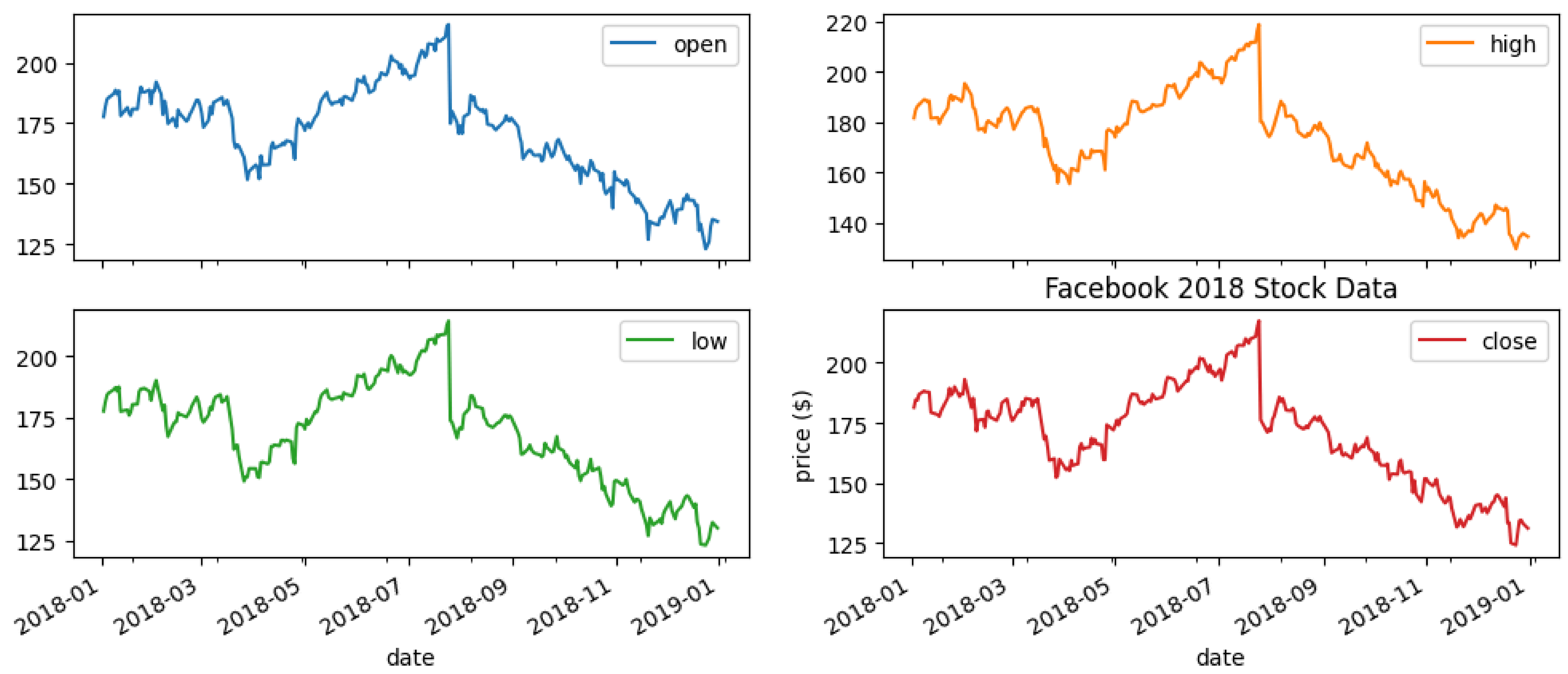
```
fb.close.plot()
plt.suptitle('FB Closing Price')
plt.xlabel('date')
plt.ylabel('price ($)')
```

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



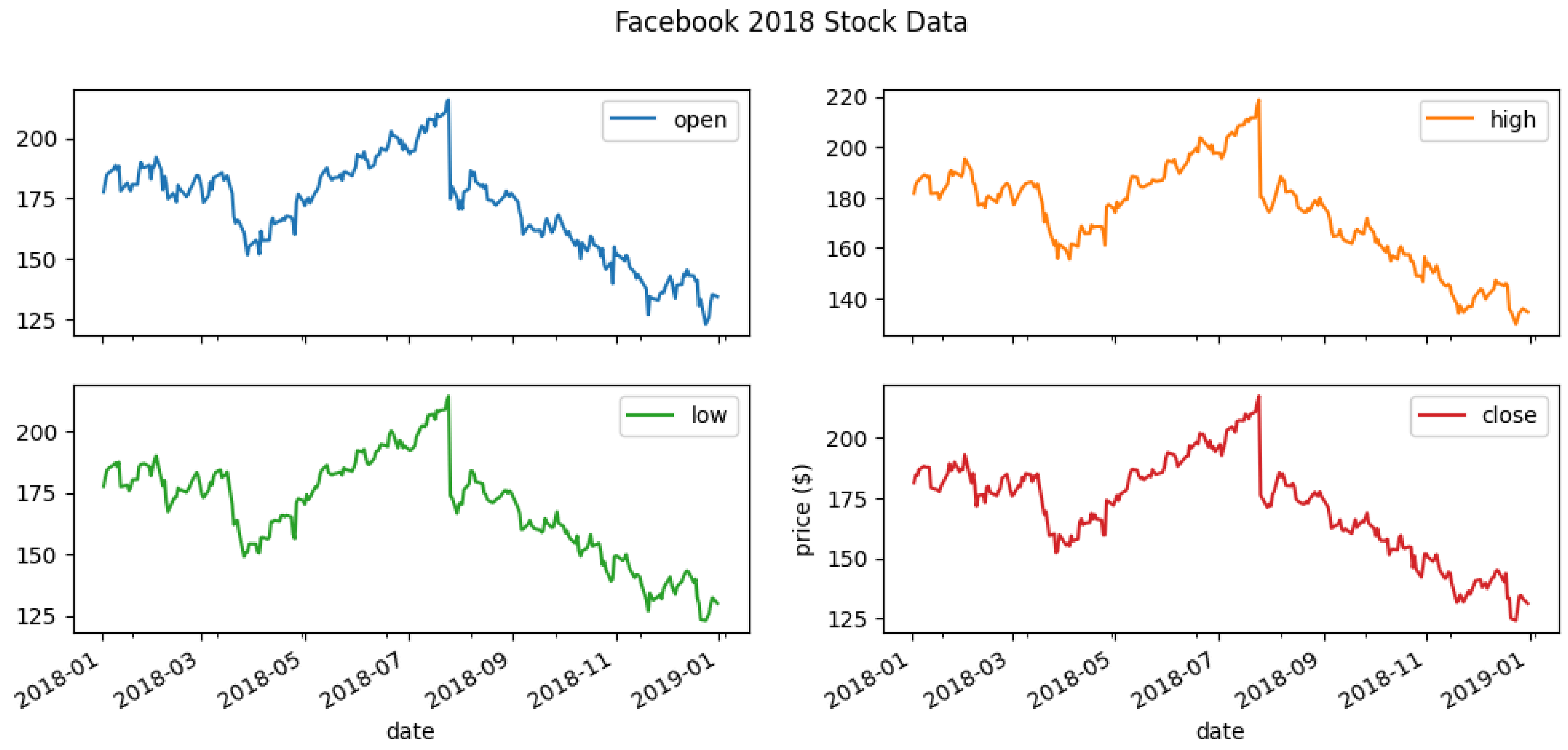

```
fb.iloc[:,:4].plot(subplots=True, layout=(2, 2), figsize=(12, 5))
plt.title('Facebook 2018 Stock Data')
plt.xlabel('date')
plt.ylabel('price ($)')
```

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



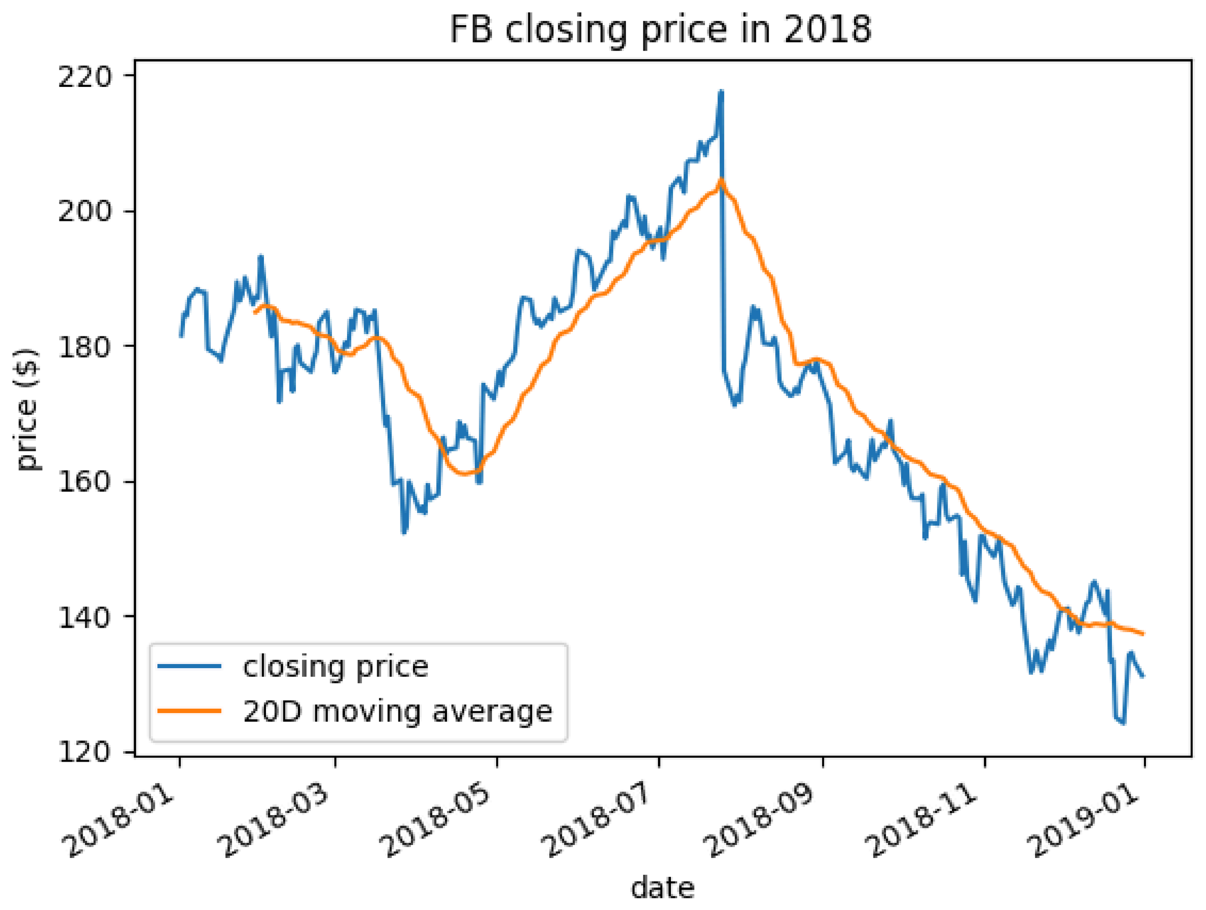

```
fb.iloc[:,:4].plot(subplots=True, layout=(2, 2), figsize=(12, 5))
plt.suptitle('Facebook 2018 Stock Data')
plt.xlabel('date')
plt.ylabel('price ($)')
```

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



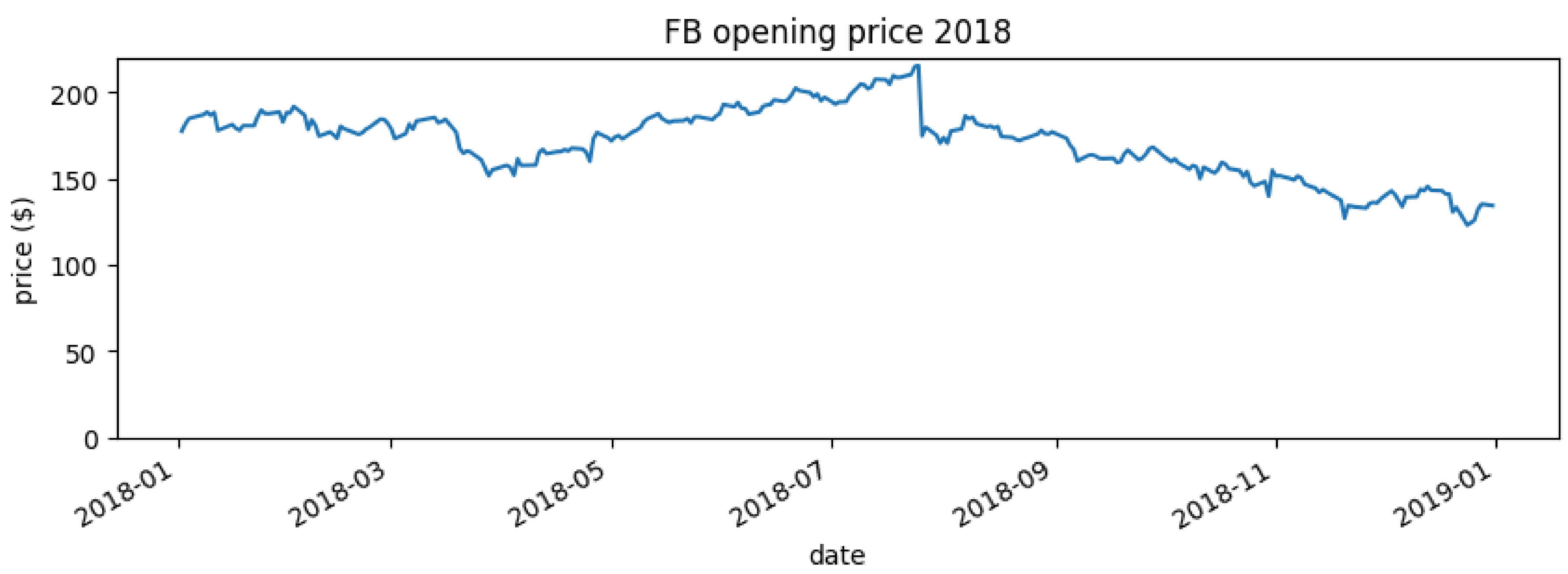

```
fb.assign(
ma=lambda x: x.close.rolling(20).mean()
).plot(
y=['close', 'ma'],
title='FB closing price in 2018',
label=['closing price', '20D moving average']
)
plt.legend(loc='lower left')
plt.ylabel('price ($)')
```

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

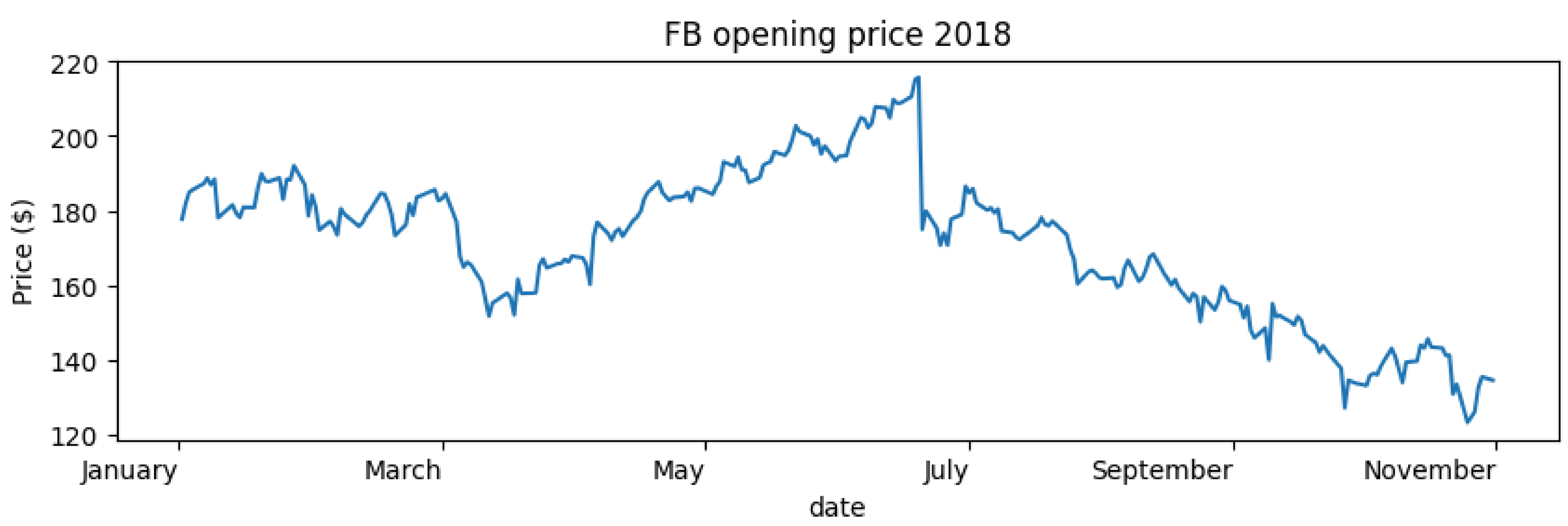



```
fb.open.plot(figsize=(10, 3), title='FB opening price 2018')
plt.ylim(0, None)
plt.ylabel('price ($)')
```

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

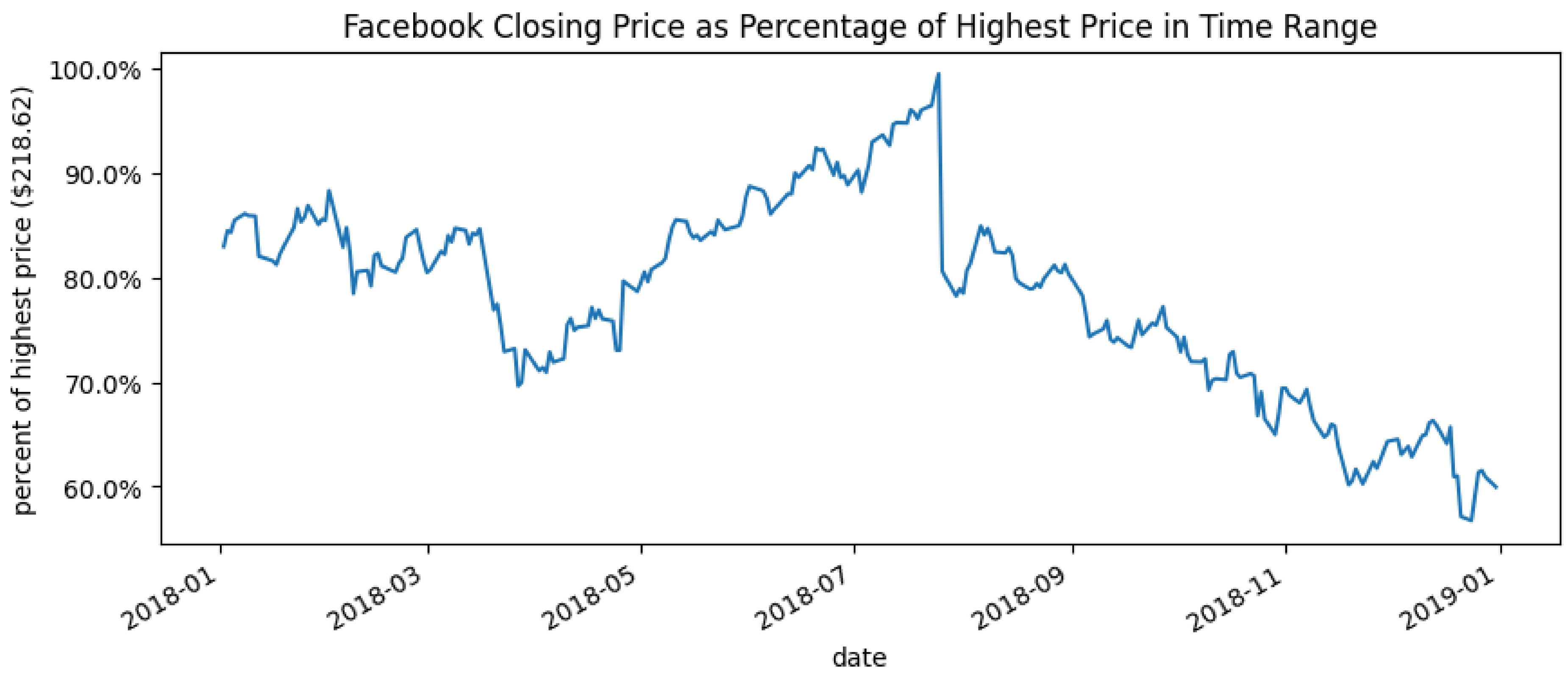



```
import matplotlib.pyplot as plt
import calendar
import pandas as pd  # Assuming you're using pandas for your data

# Plot the opening price with the initial setup
fb.open.plot(figsize=(10, 3), rot=0, title='FB opening price 2018')

# Get the current locations and labels (this will be modified)
locs, labels = plt.xticks()

month_labels = [calendar.month_name[i] for i in range(1, 13, 2)]

# Then, calculate new locations. Assume you want them evenly spaced, based on the plotted data range
new_locs = np.linspace(start=locs[0], stop=locs[-1], num=len(month_labels))

# Set the new locations and labels
plt.xticks(ticks=new_locs, labels=month_labels)

plt.ylabel('Price ($)')
plt.show()
```



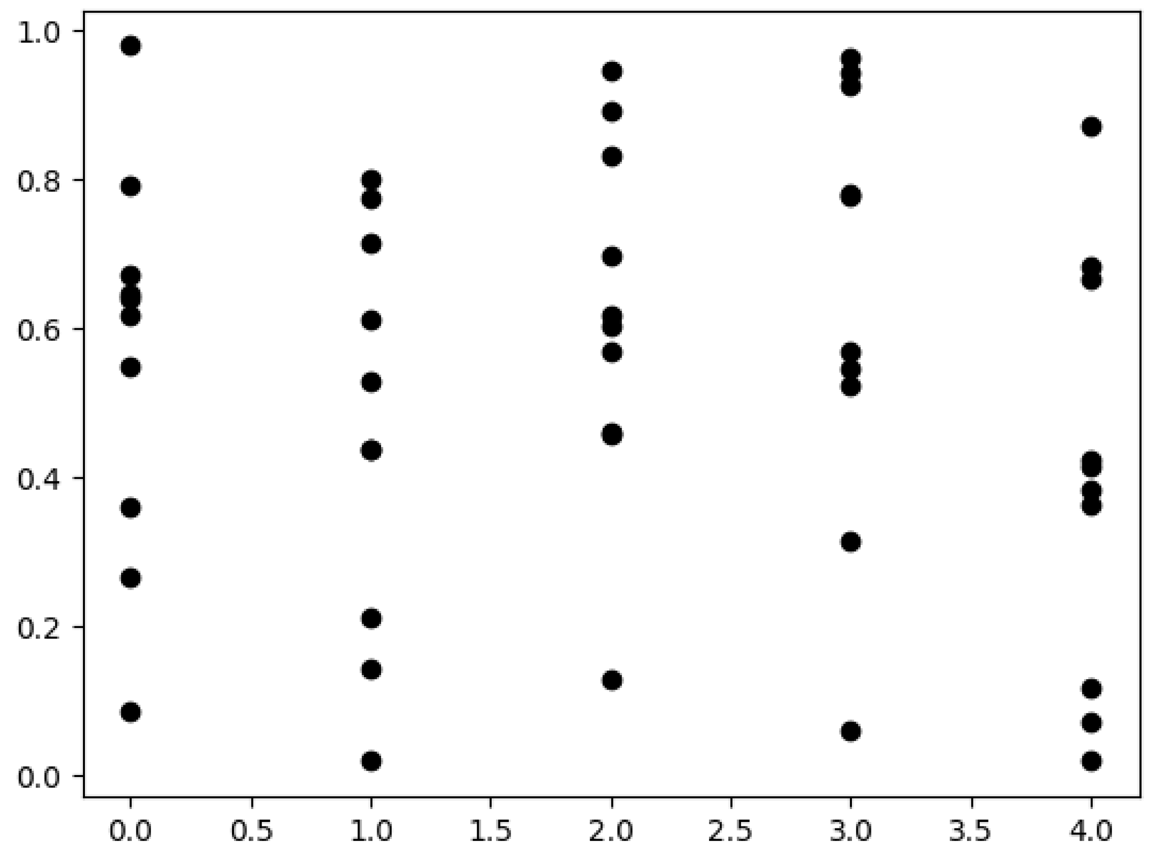
```
import matplotlib.ticker as ticker
ax = fb.close.plot(
figsize=(10, 4),
title='Facebook Closing Price as Percentage of Highest Price in Time Range'
)
ax.yaxis.set_major_formatter(
ticker.PercentFormatter(xmax=fb.high.max())
)
ax.set_yticks([
fb.high.max()*pct for pct in np.linspace(0.6, 1, num=5)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

```
Text(0, 0.5, 'percent of highest price ($218.62)')
```

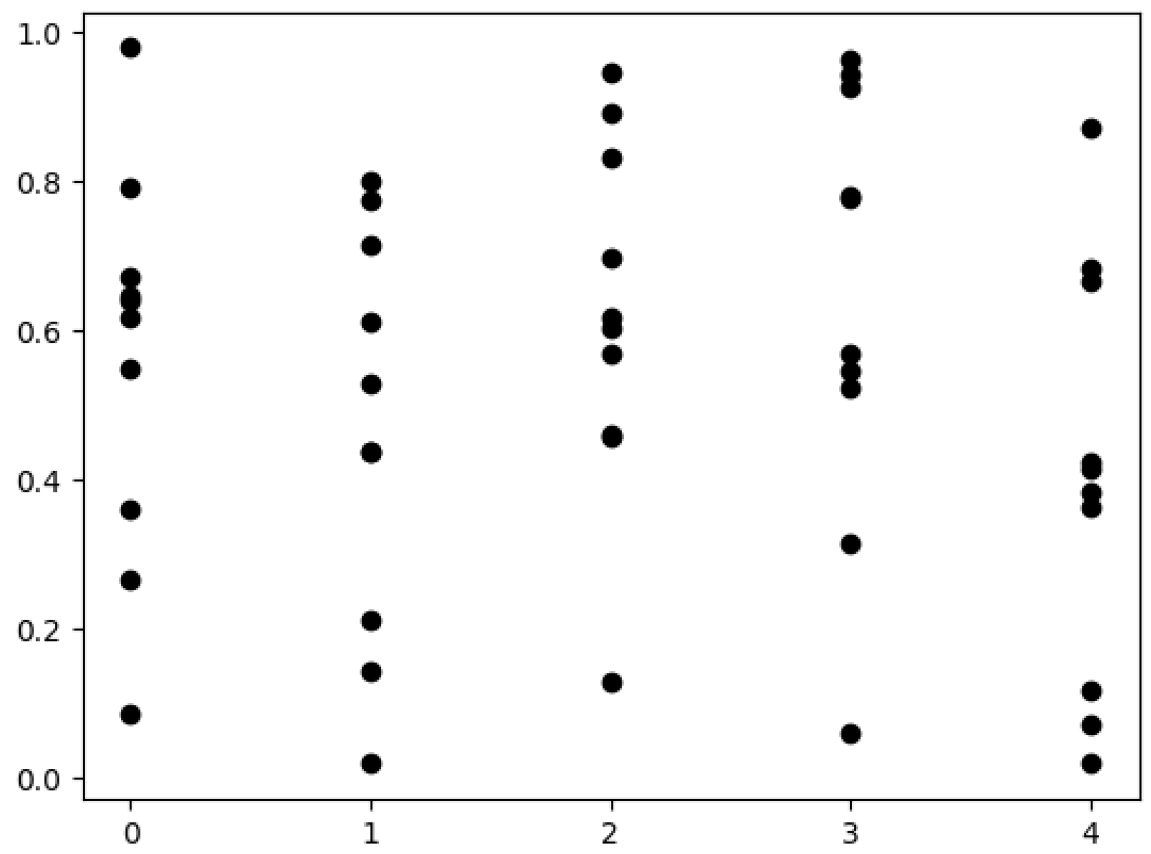



```
fig, ax = plt.subplots(1, 1)
np.random.seed(0)
ax.plot(np.tile(np.arange(0, 5), 10), np.random.rand(50), 'ko')
```

```
[<matplotlib.lines.Line2D at 0x79ab91720670>]
```



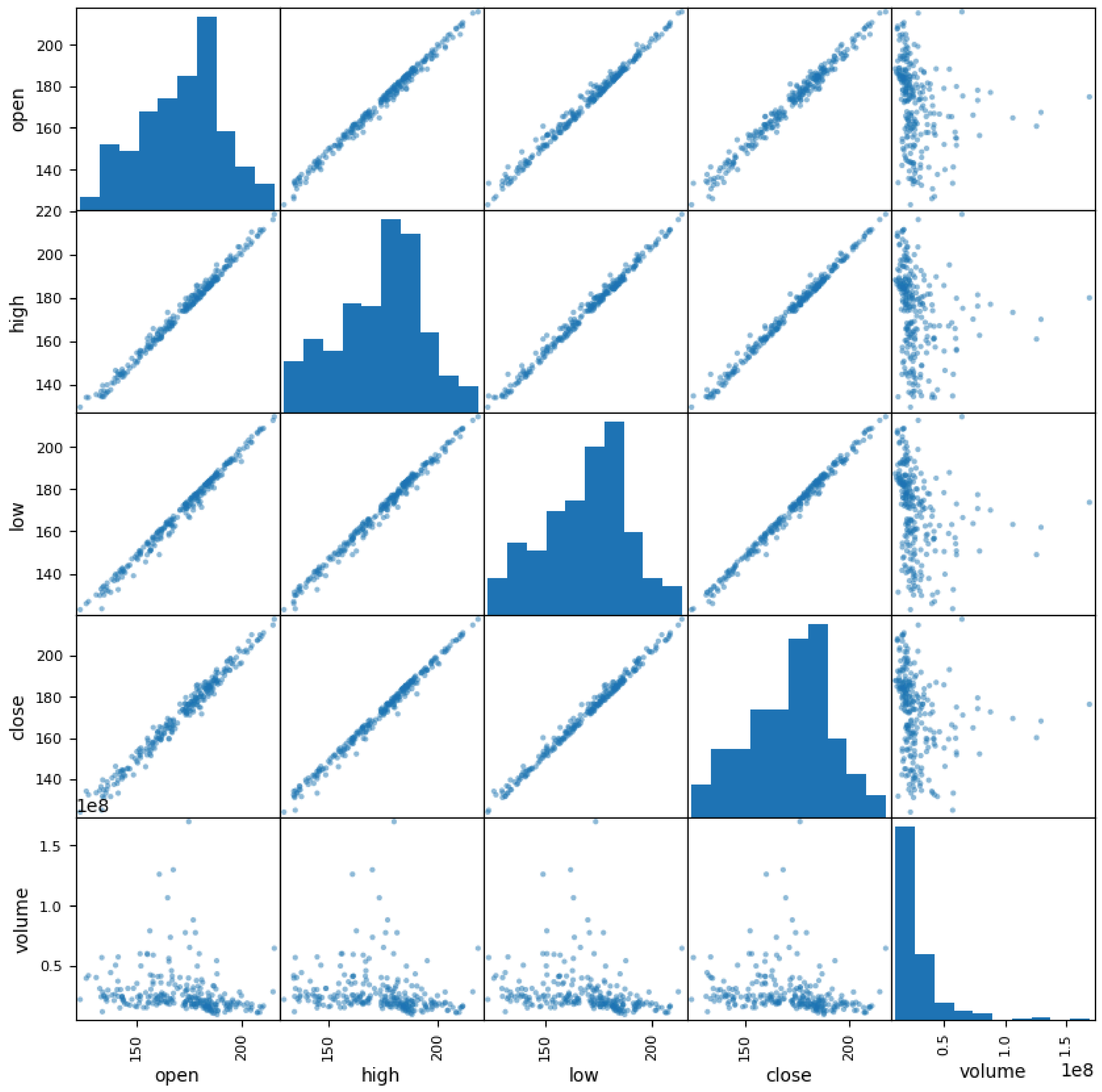

```
fig, ax = plt.subplots(1, 1)
np.random.seed(0)
ax.plot(np.tile(np.arange(0, 5), 10), np.random.rand(50), 'ko')
ax.get_xaxis().set_major_locator(
ticker.MultipleLocator(base=1)
)
```




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

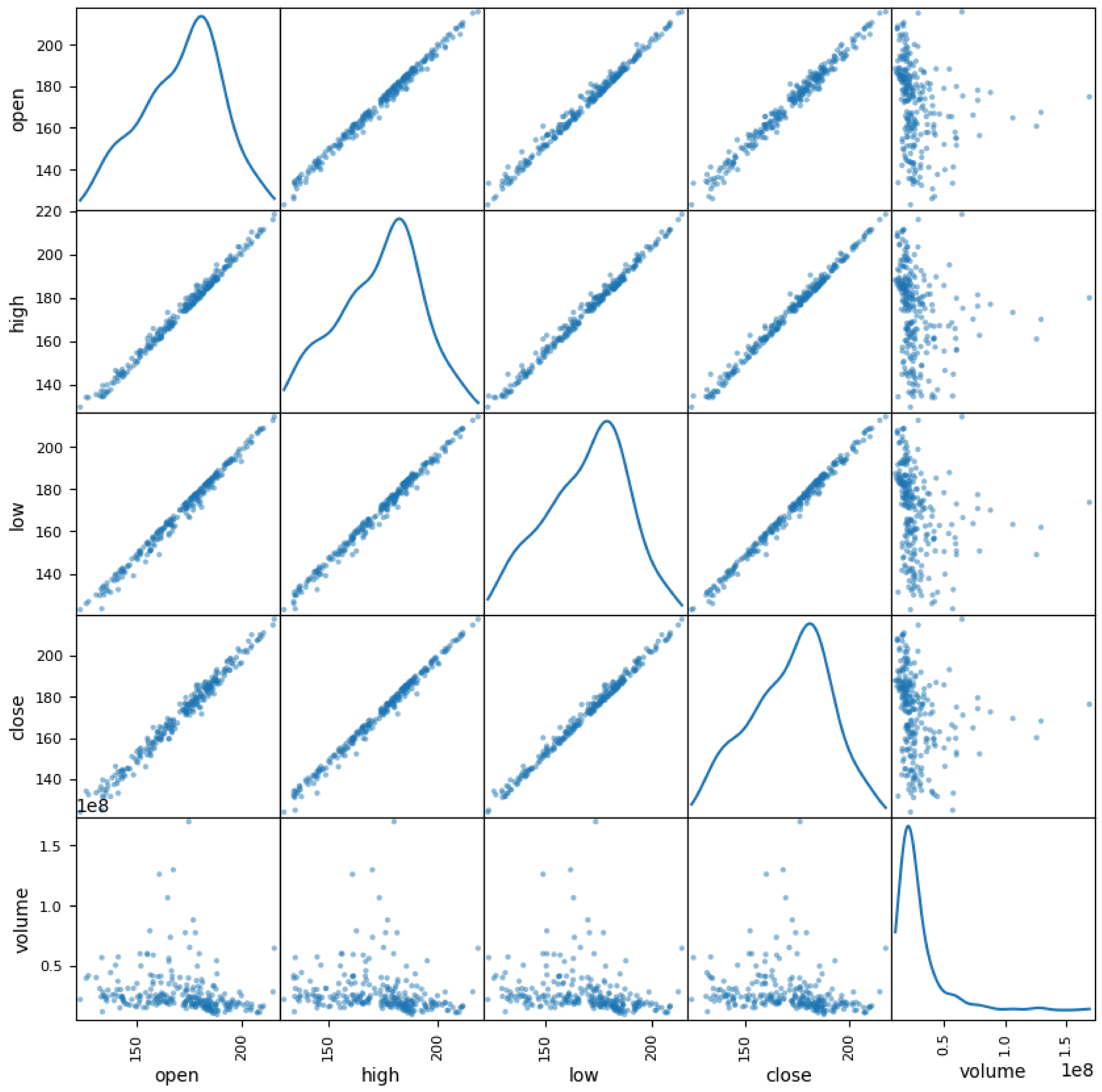
```
from pandas.plotting import scatter_matrix
scatter_matrix(fb, figsize=(10, 10))
```

```
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)
```



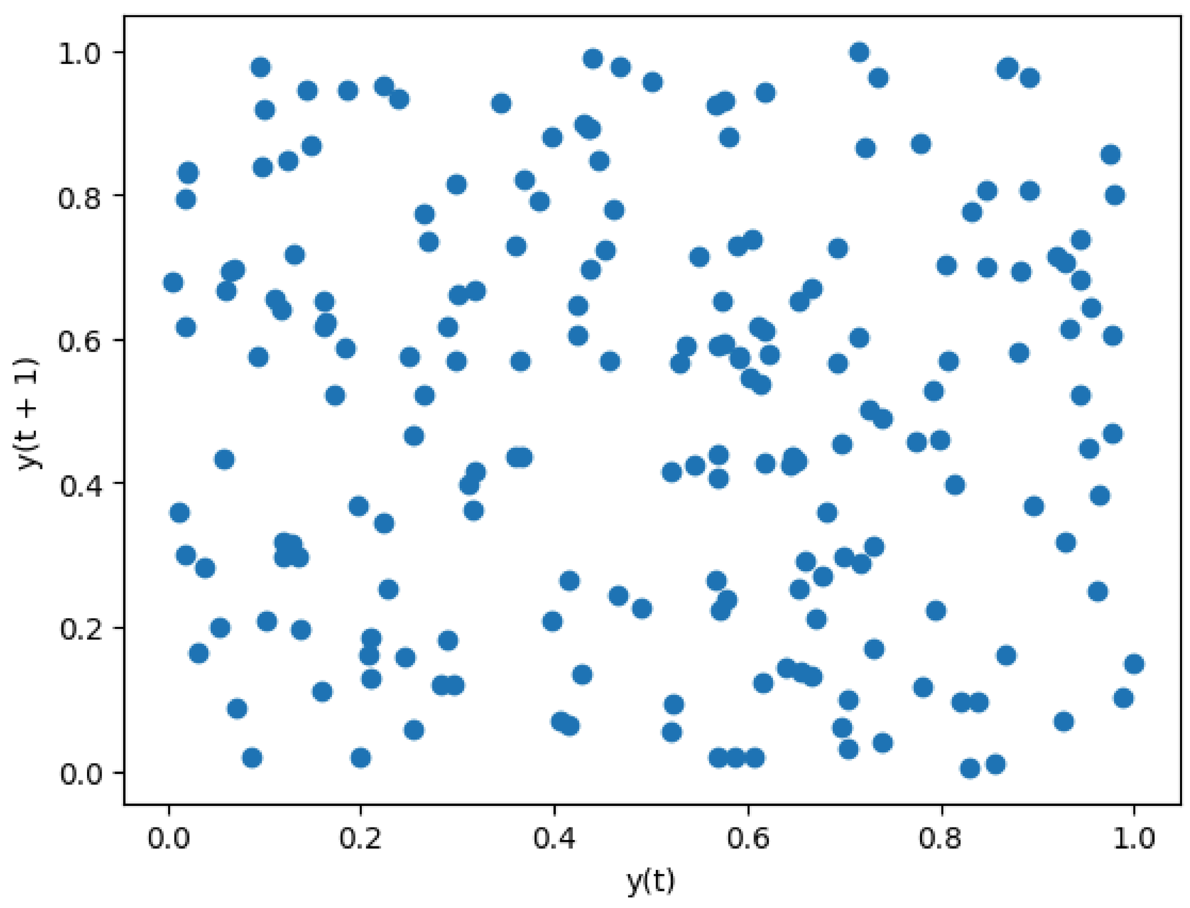

```
scatter_matrix(fb, figsize=(10, 10), diagonal='kde')
```

```
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)
```



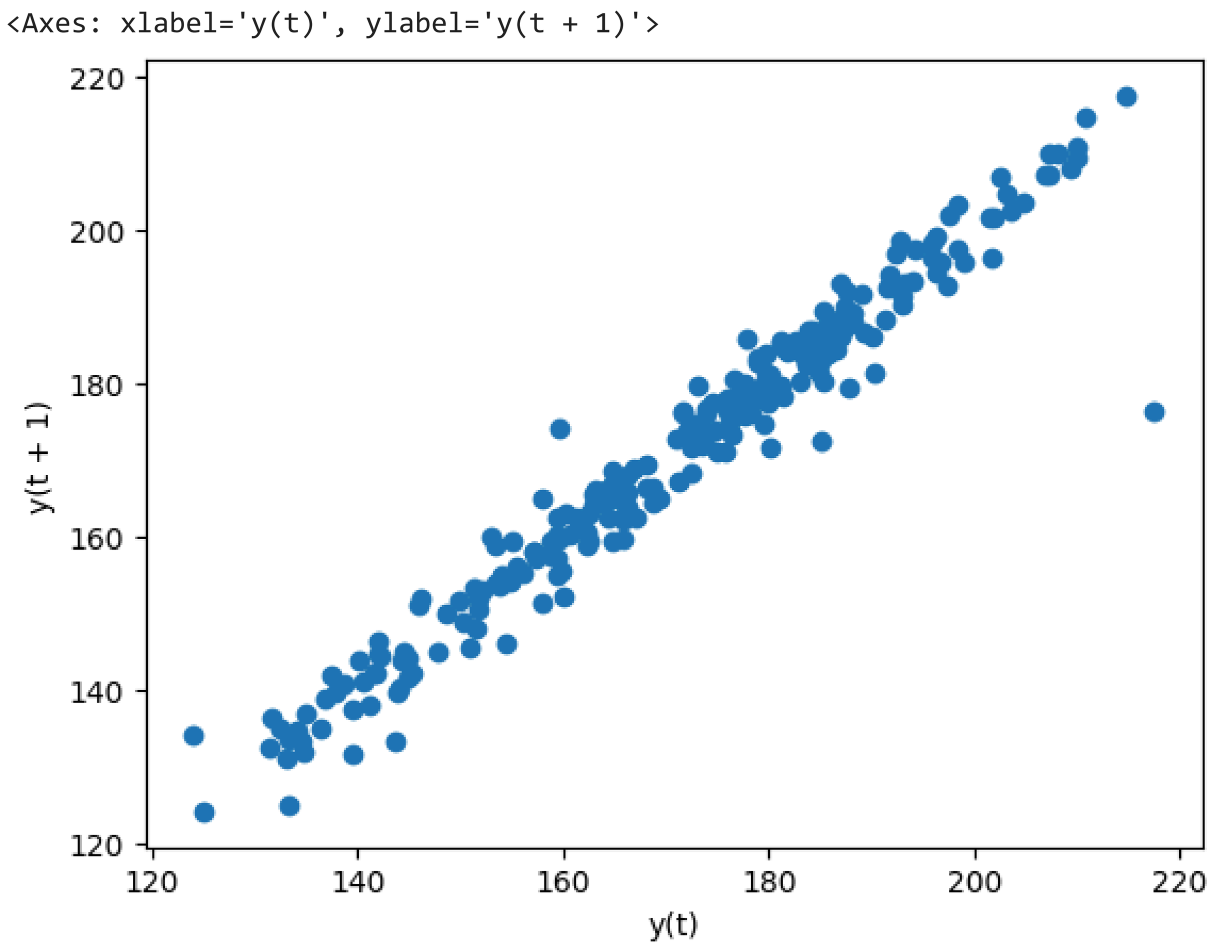

```
from pandas.plotting import lag_plot
np.random.seed(0) # make this repeatable
lag_plot(pd.Series(np.random.random(size=200)))
```

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



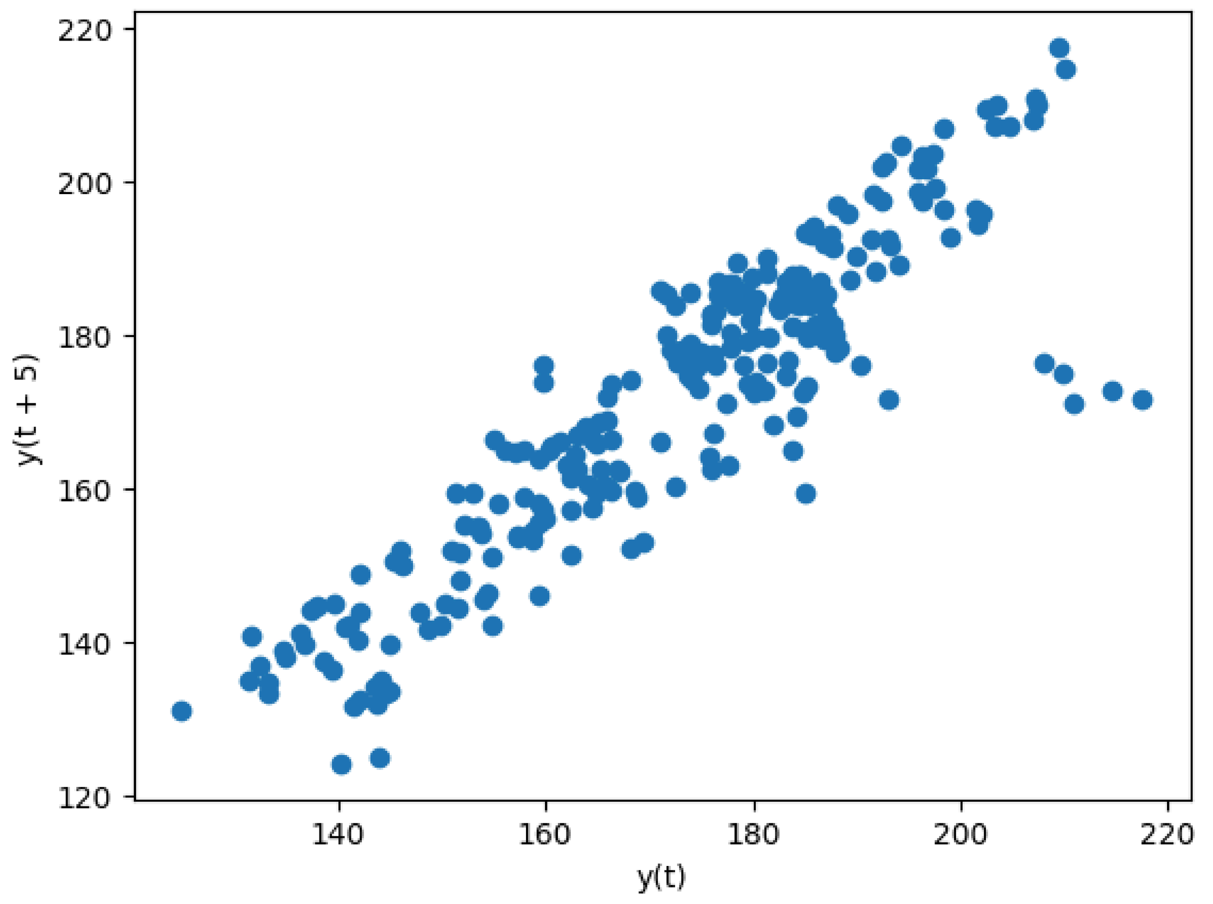

```
lag_plot(fb.close)
```

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



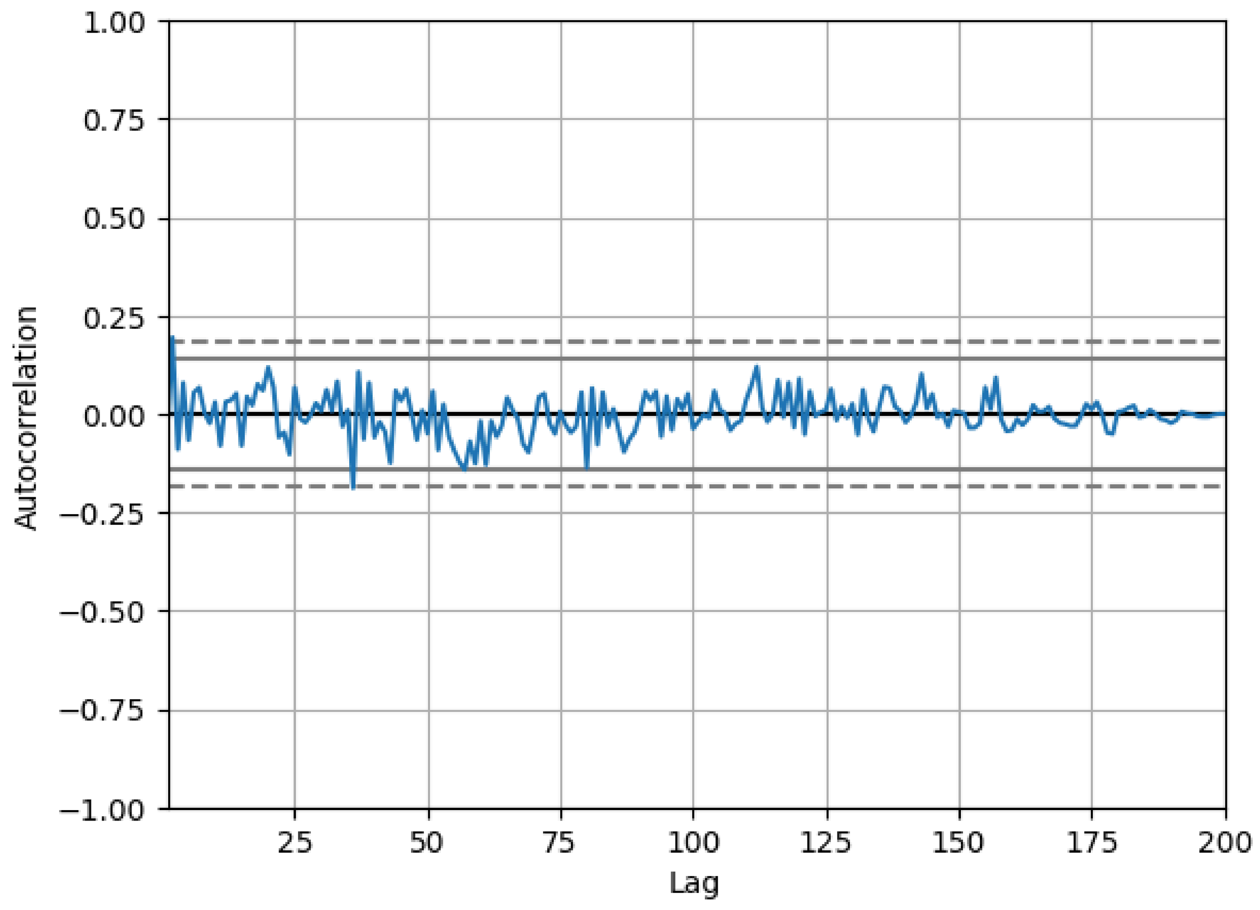

```
lag_plot(fb.close, lag=5)
```

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



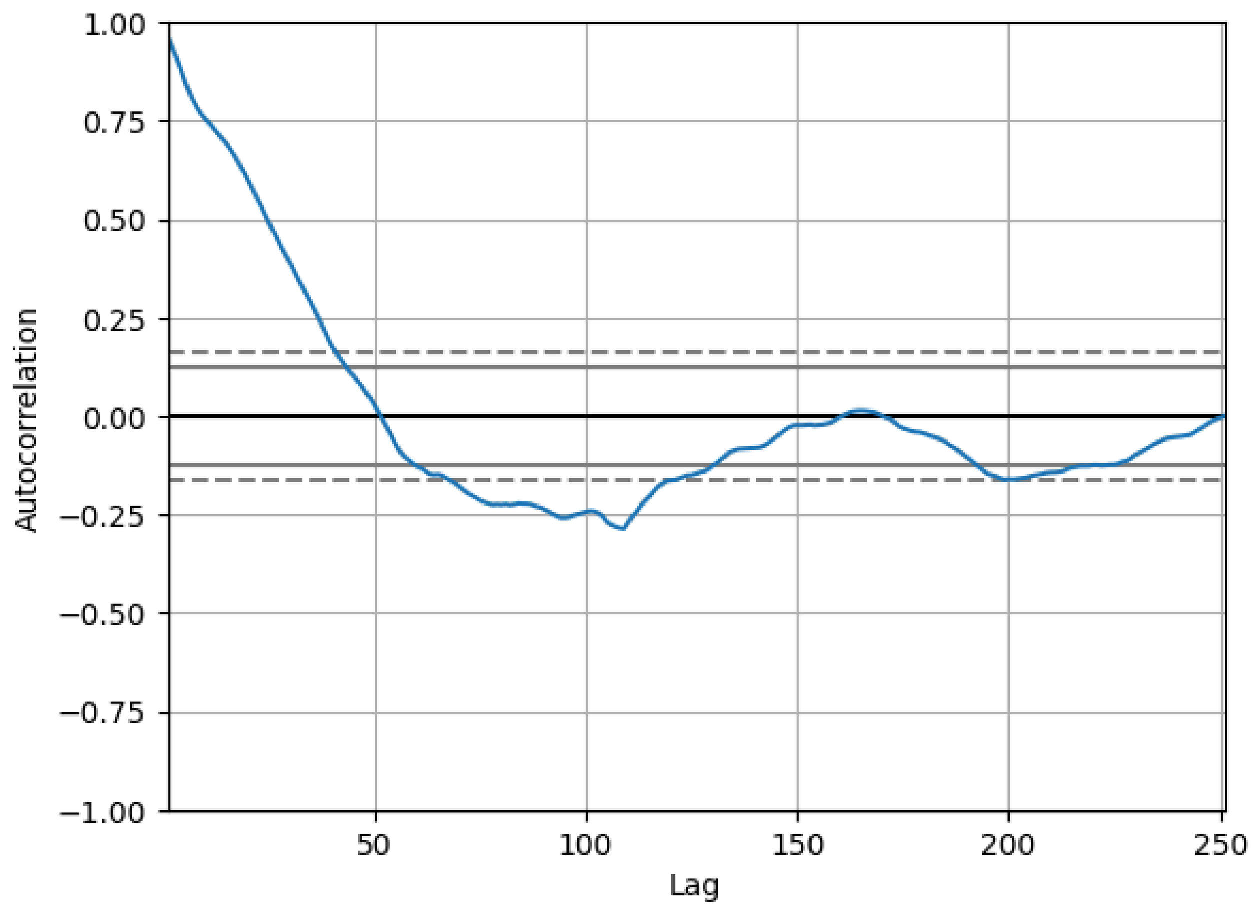

```
from pandas.plotting import autocorrelation_plot
np.random.seed(0) # make this repeatable
autocorrelation_plot(pd.Series(np.random.random(size=200)))
```

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

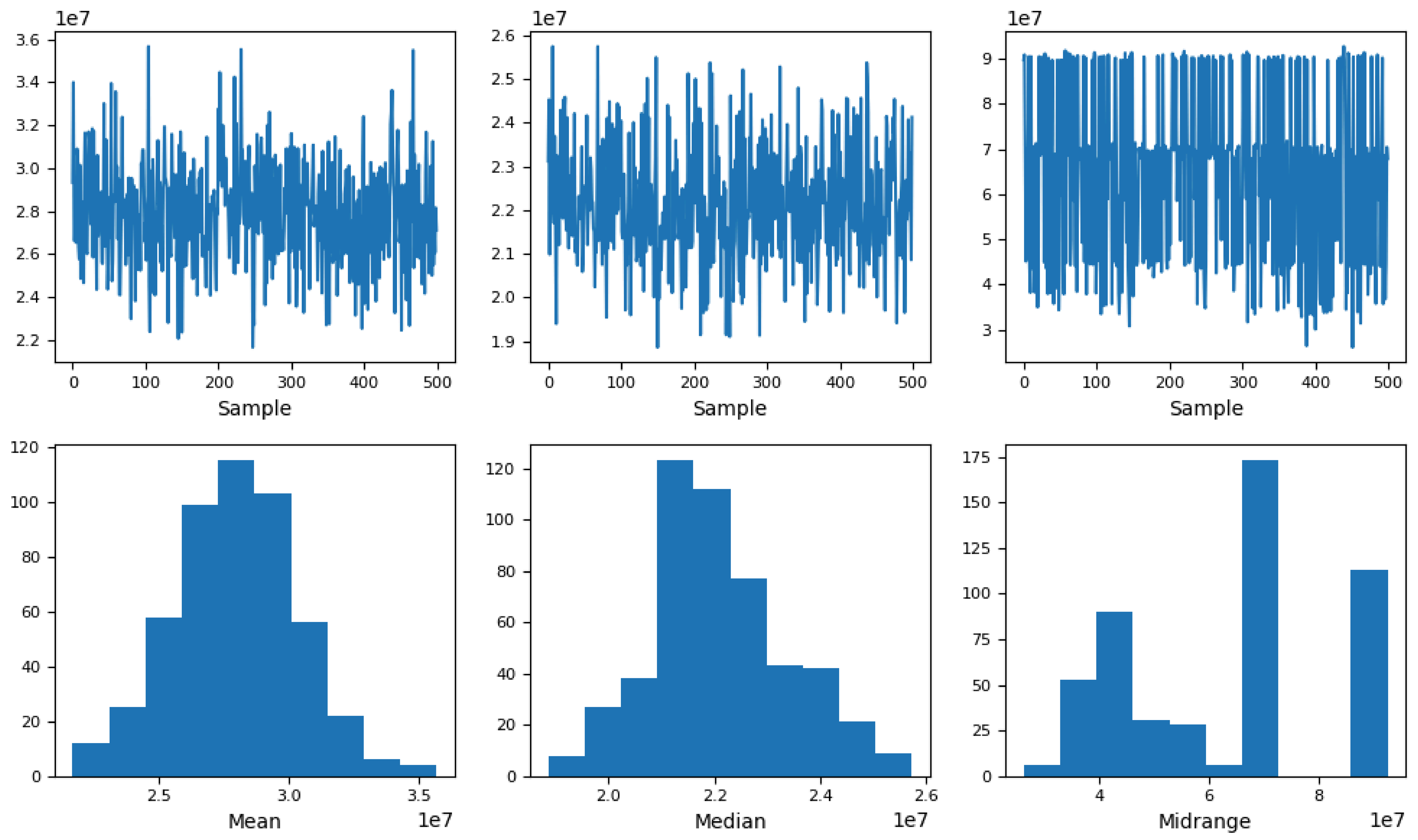



```
autocorrelation_plot(fb.close)
```

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




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



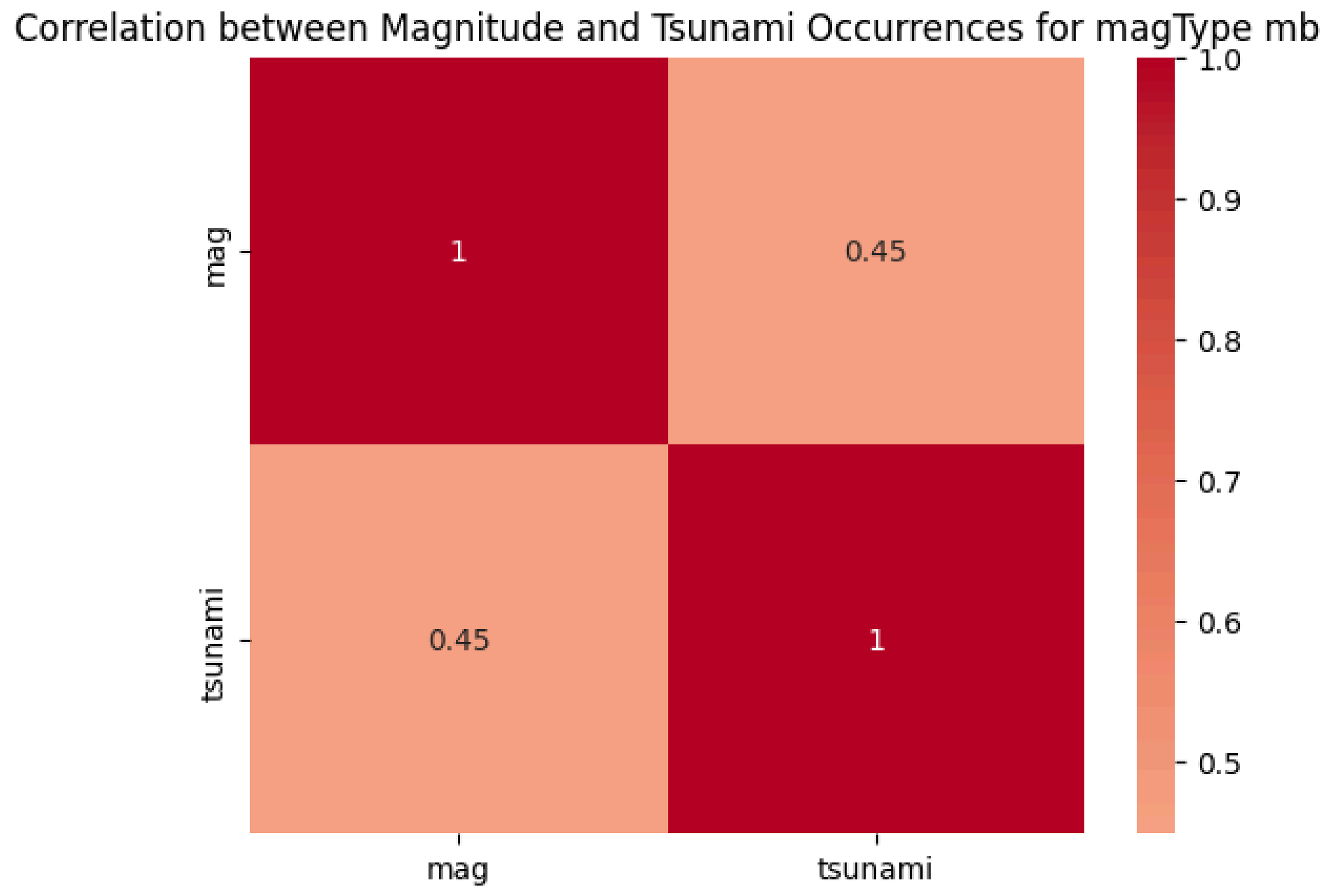
Data Analysis:

The heatmap analysis revealing the correlation between earthquake magnitudes (with a specific focus on those categorized under magType 'mb') and the occurrences of tsunamis likely showcases a nuanced relationship between these natural phenomena. A demonstrated low to moderate positive correlation not only aligns with intuitive expectations—that larger earthquakes have a higher propensity to trigger tsunamis—but also highlights the complex interplay of factors beyond mere magnitude, such as geological conditions and proximity to water bodies, in tsunami generation. In parallel, a box plot detailing Facebook's traded volume and closing prices, supplemented by Tukey fences, provides a granular look into the stock's trading behavior. This visualization method is particularly adept at spotlighting the central tendencies and variabilities within the data, with the identification of outliers potentially signaling periods of heightened volatility or market sensitivity to external events. Such insights are invaluable for investors and researchers alike, offering a clearer understanding of both natural disaster dynamics and financial market reactions.

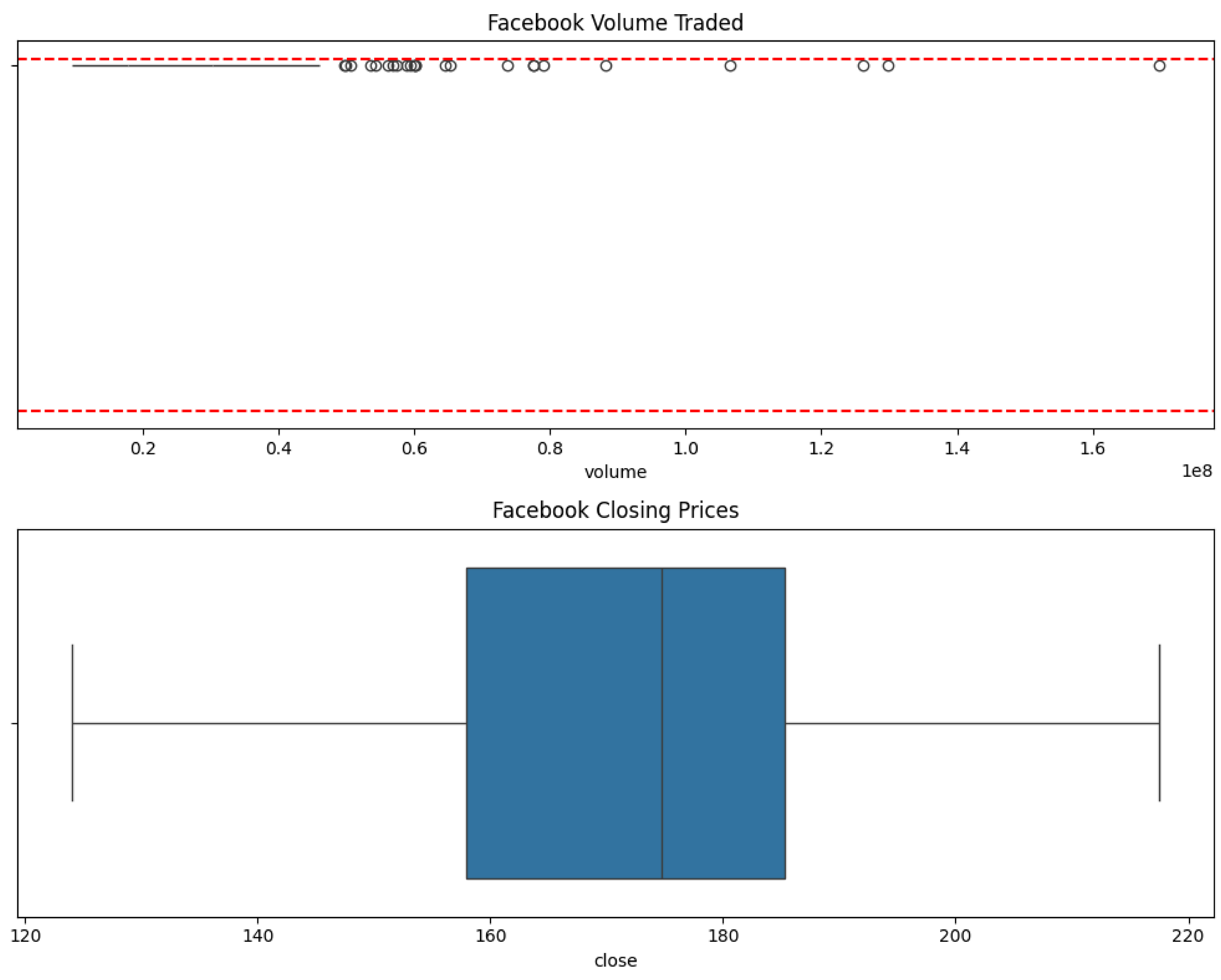
Supplementary Activity:

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

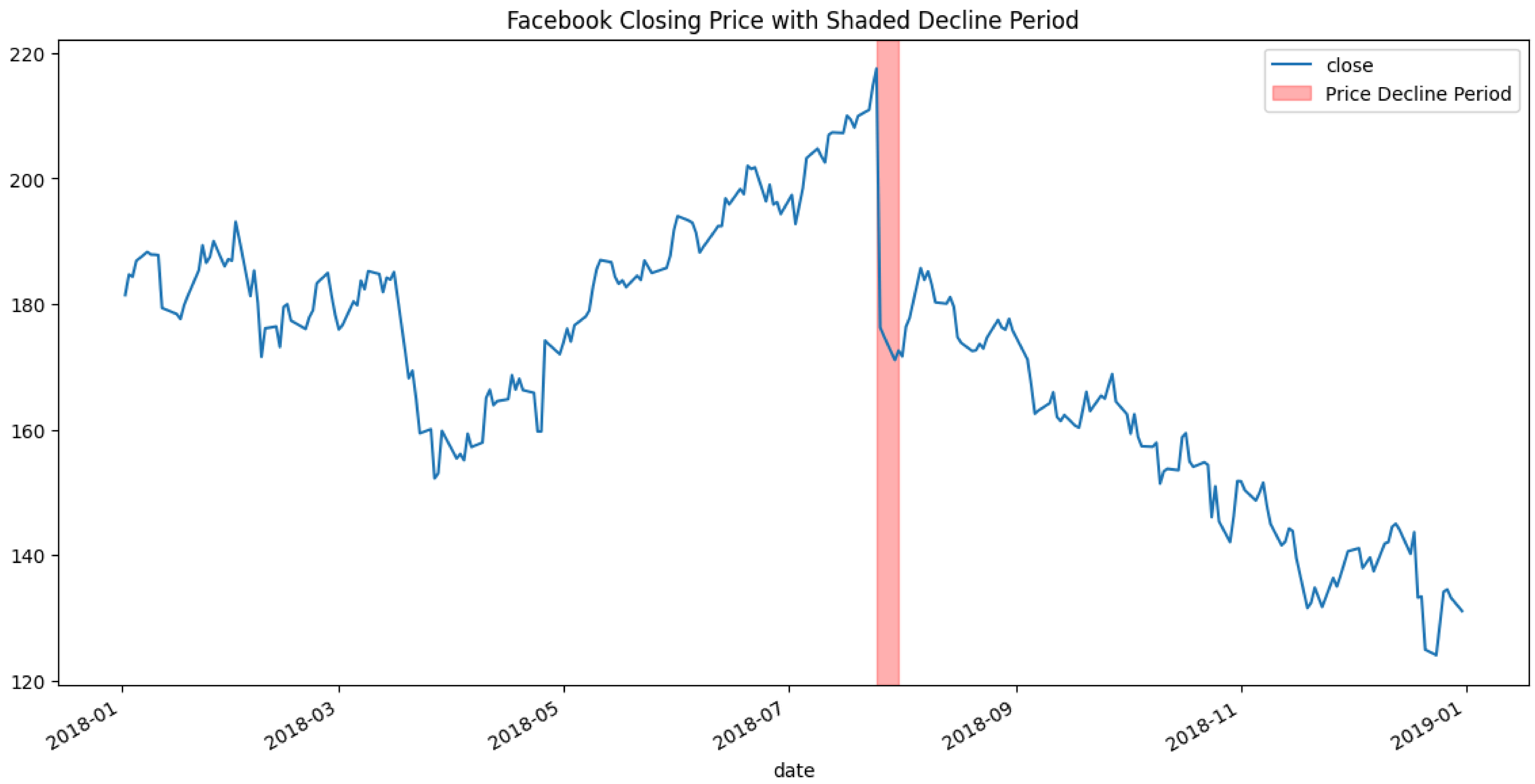
```
import seaborn as sns
import matplotlib.pyplot as plt

# Filter earthquakes with magType 'mb'
mb_quakes = quakes[quakes['magType'] == 'mb']

# Calculate correlation matrix
corr_matrix = mb_quakes[['mag', 'tsunami']].corr()

# Create heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation between Magnitude and Tsunami Occurrences for magType mb')
plt.show()
```



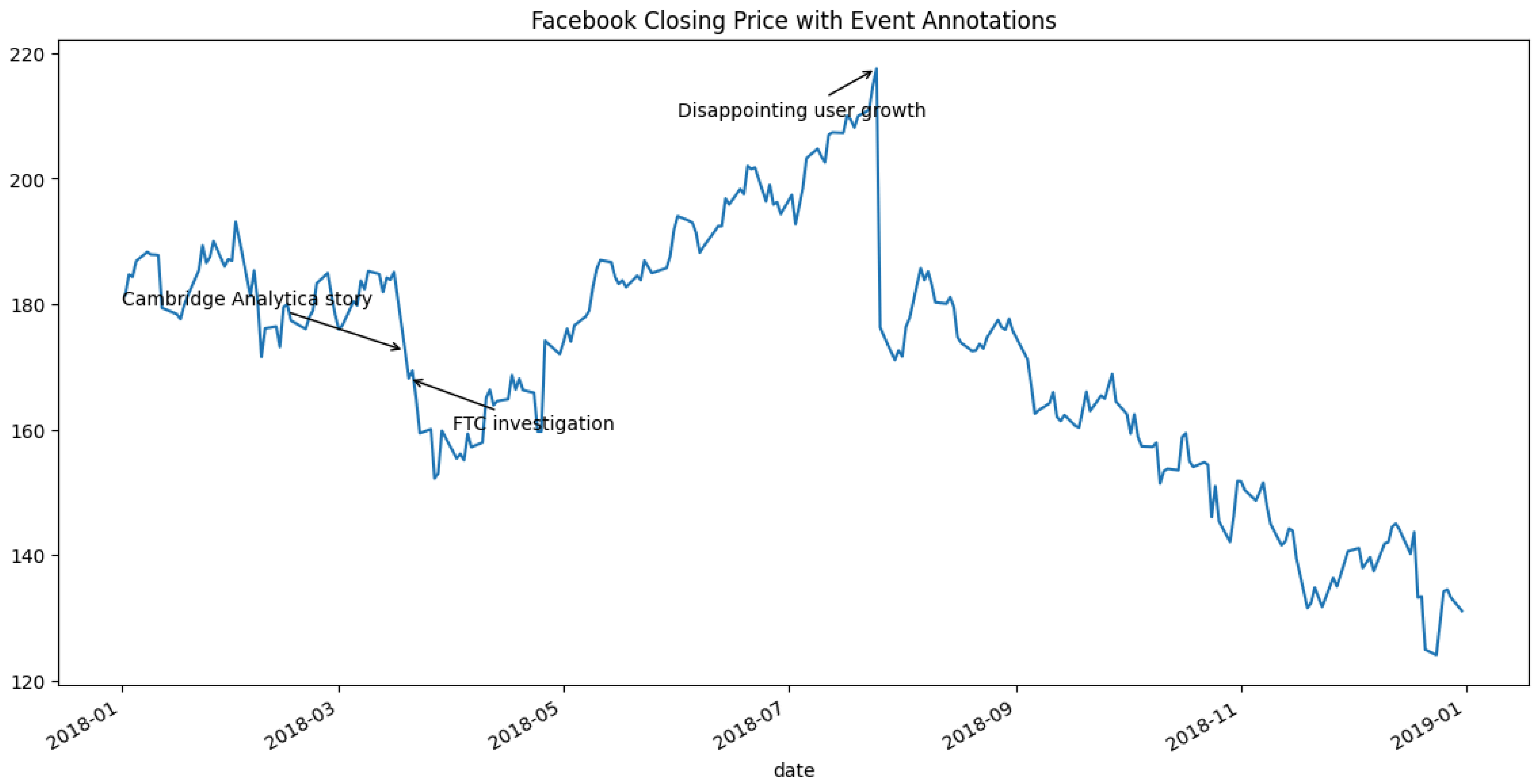

```
fig, ax = plt.subplots(2, 1, figsize=(10, 8))

# Box plot for volume traded
sns.boxplot(x=fb['volume'], ax=ax[0])
ax[0].set_title('Facebook Volume Traded')

# Calculate IQR for volume traded
Q1_volume = fb['volume'].quantile(0.25)
Q3_volume = fb['volume'].quantile(0.75)
IQR_volume = Q3_volume - Q1_volume
lower_bound_volume = Q1_volume - 1.5 * IQR_volume
upper_bound_volume = Q3_volume + 1.5 * IQR_volume

# Drawing reference lines for volume
ax[0].axhline(y=lower_bound_volume, color='red', linestyle='--')
ax[0].axhline(y=upper_bound_volume, color='red', linestyle='--')

# Box plot for closing prices
sns.boxplot(x=fb['close'], ax=ax[1])
ax[1].set_title('Facebook Closing Prices')

# Similar calculations and reference lines for closing prices are omitted for brevity

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




```
fb['close'].plot(figsize=(14, 7))
plt.axvspan('2018-07-25', '2018-07-31', color='red', alpha=0.3, label='Price Decline Period')
plt.legend()
plt.title('Facebook Closing Price with Shaded Decline Period')
plt.show()
```




```
fb['close'].plot(figsize=(14, 7))
plt.annotate('Disappointing user growth', xy=('2018-07-25', fb.loc['2018-07-25', 'close']), xytext=('2018-06', 210), arrowprops=dict(arrowstyle='->'))
plt.annotate('Cambridge Analytica story', xy=('2018-03-19', fb.loc['2018-03-19', 'close']), xytext=('2018-01', 180), arrowprops=dict(arrowstyle='->'))
plt.annotate('FTC investigation', xy=('2018-03-20', fb.loc['2018-03-20', 'close']), xytext=('2018-04', 160), arrowprops=dict(arrowstyle='->'))
plt.title('Facebook Closing Price with Event Annotations')
plt.show()
```




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

def reg_resid_plots(data, x_vars, y_var, colormap='tab10'):
    """
    Plots regression and residual plots for pairs of columns in a DataFrame.

    Parameters:
    - data: DataFrame containing the data
    - x_vars: list of column names to be used as independent variables
    - y_var: column name to be used as dependent variable
    - colormap: name of the matplotlib colormap to use
    """

    # Get the colormap
    cmap = plt.cm.get_cmap(colormap, len(x_vars))

    # Setup the matplotlib figure
    fig, axs = plt.subplots(len(x_vars), 2, figsize=(10, 5 * len(x_vars)))

    for i, x_var in enumerate(x_vars):
        # Set the color for current plot
        color = cmap(i)
```