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

9.4 Introduction to Seaborn

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

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

Categorical Data

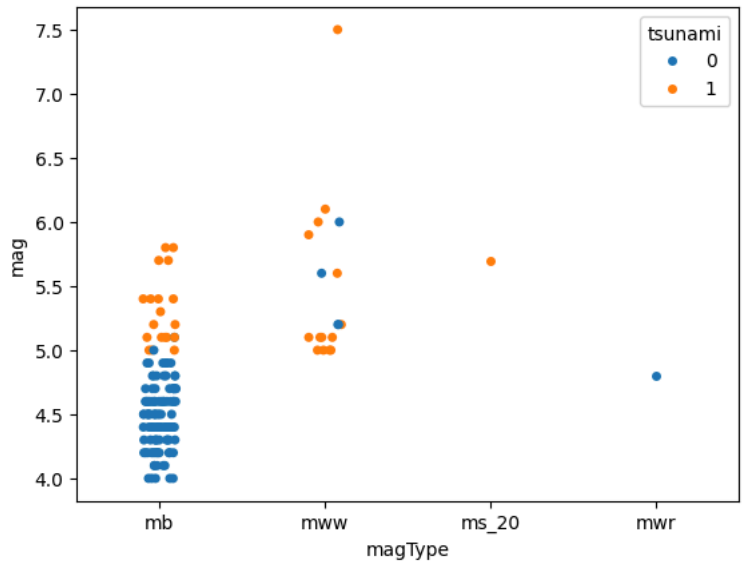
```
quakes.assign(
    time=lambda x: pd.to_datetime(x.time, unit='ms')
    ).set_index('time').loc['2018-09-28'].query(
        "parsed_place == 'Indonesia' and tsunami == 1 and mag == 7.5"
        )
```

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()

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

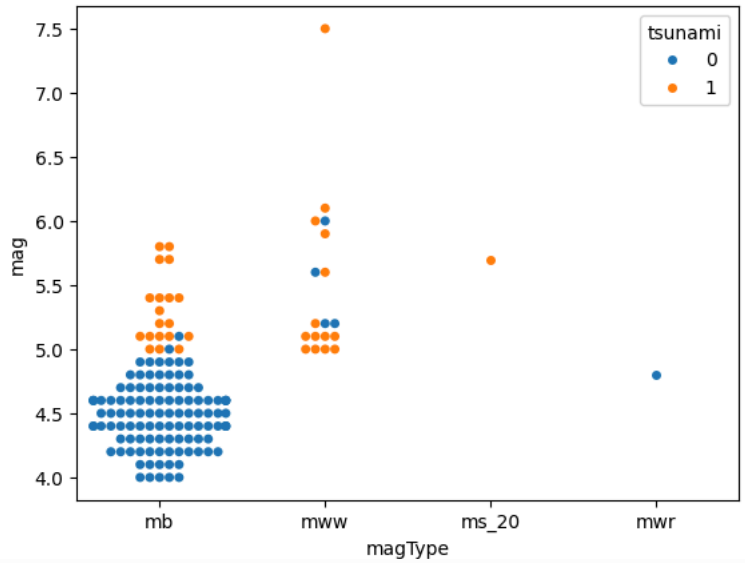
```
<Axes: xlabel='magType', ylabel='mag'>
```



swarmplot()

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

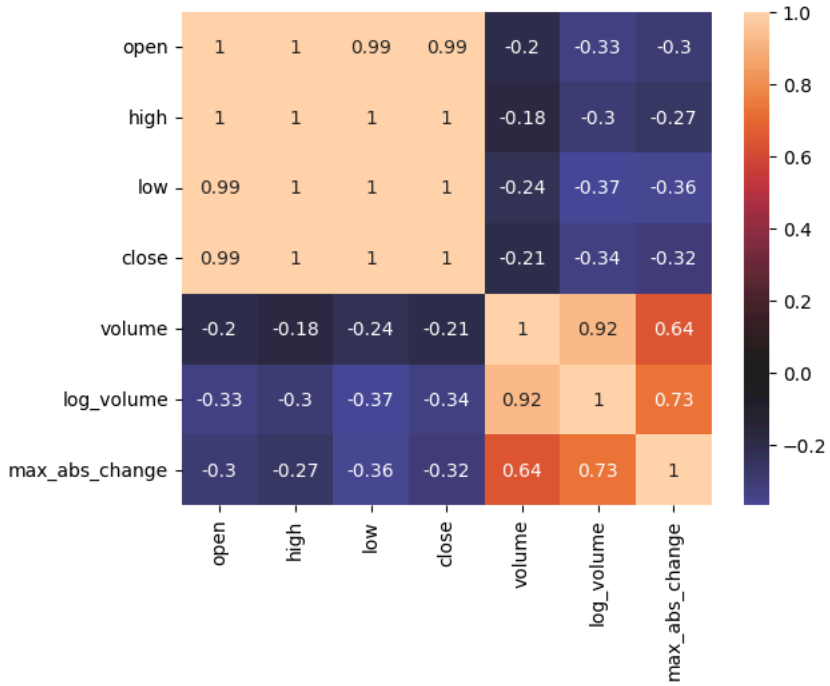
```
<Axes: xlabel='magType', ylabel='mag'>
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3398: UserWarning: 10.2%
  warnings.warn(msg, UserWarning)
```



heatmap()

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

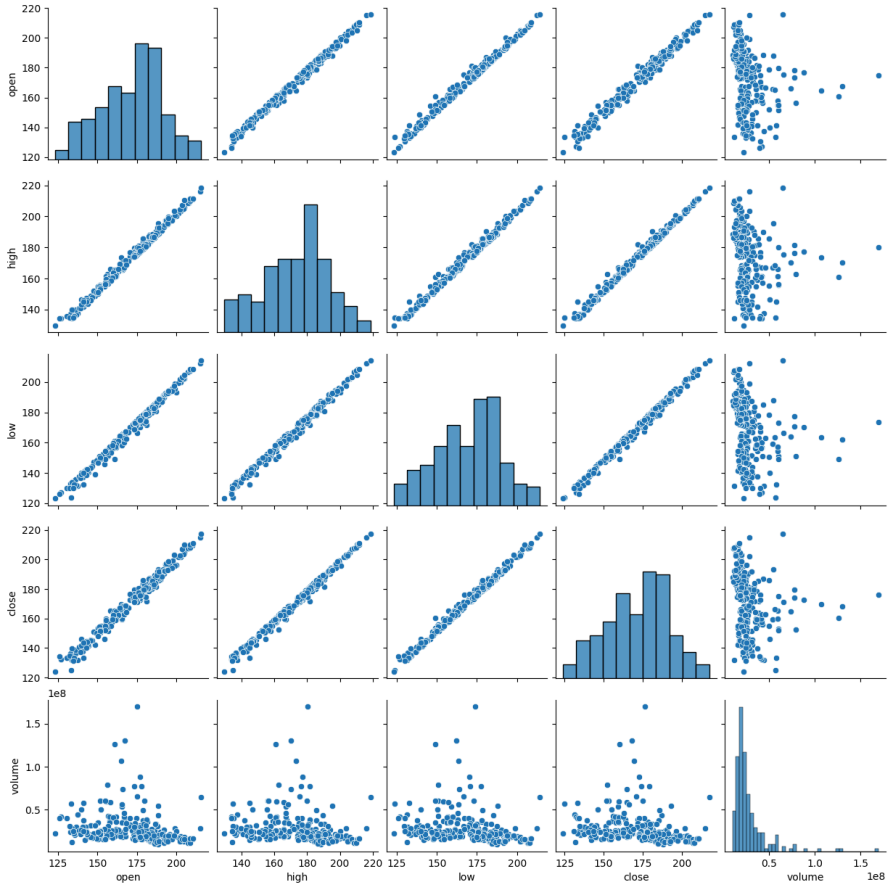
```
<Axes: >
```



pairplot()

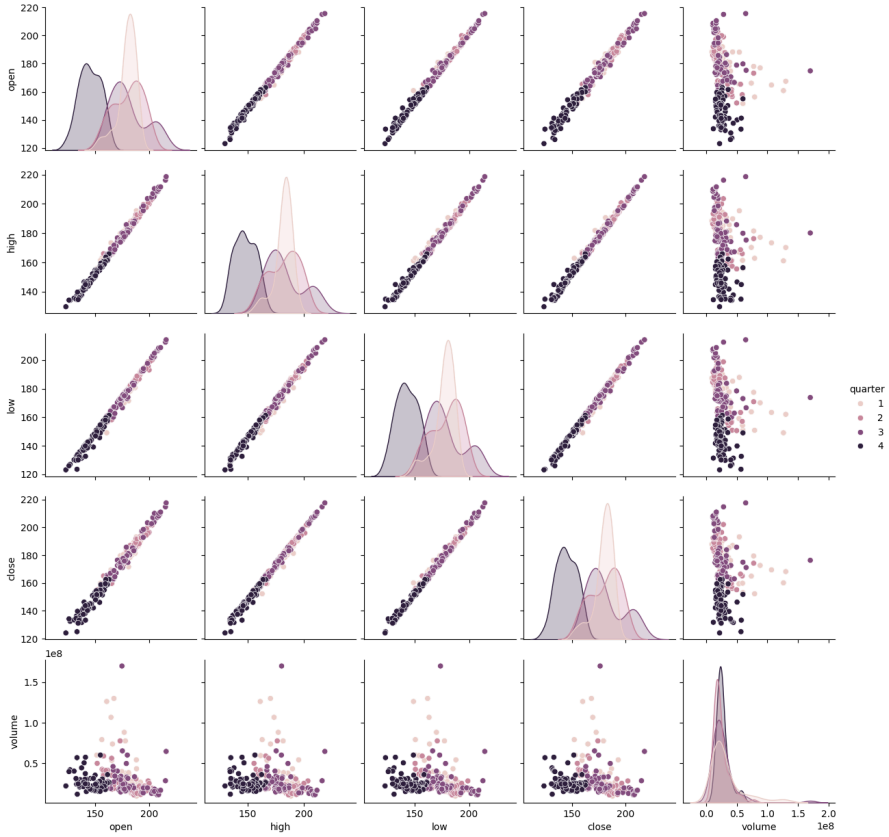
```
1 sns.pairplot(fb)
```

```
<seaborn.axisgrid.PairGrid at 0x7b295b6eb070>
```




```
1 sns.pairplot(
2     fb.assign(quarter=lambda x: x.index.quarter),
3     diag_kind='kde',
4     hue='quarter'
5     )
```

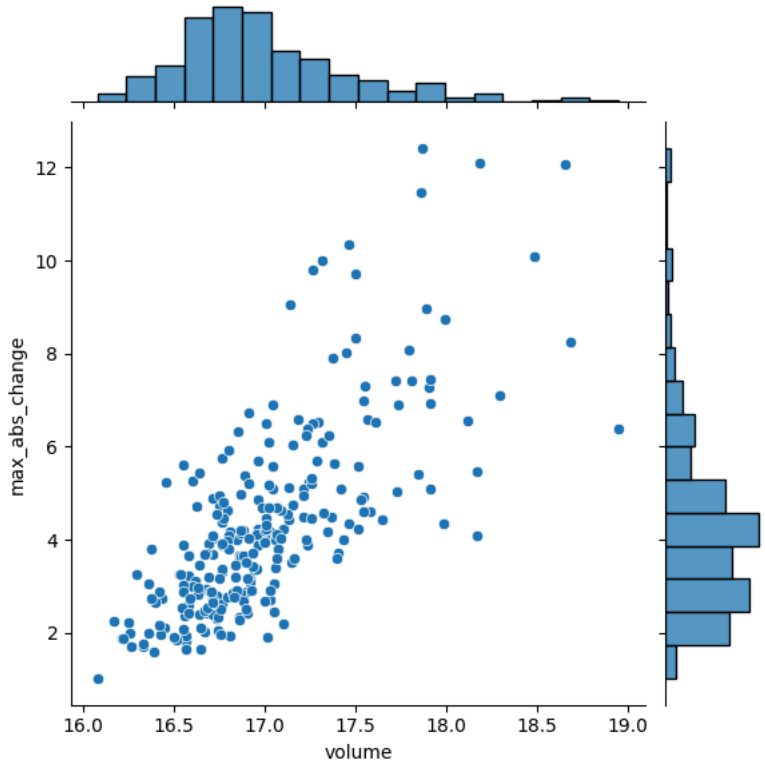
```
<seaborn.axisgrid.PairGrid at 0x7b295a5c2290>
```



jointplot()

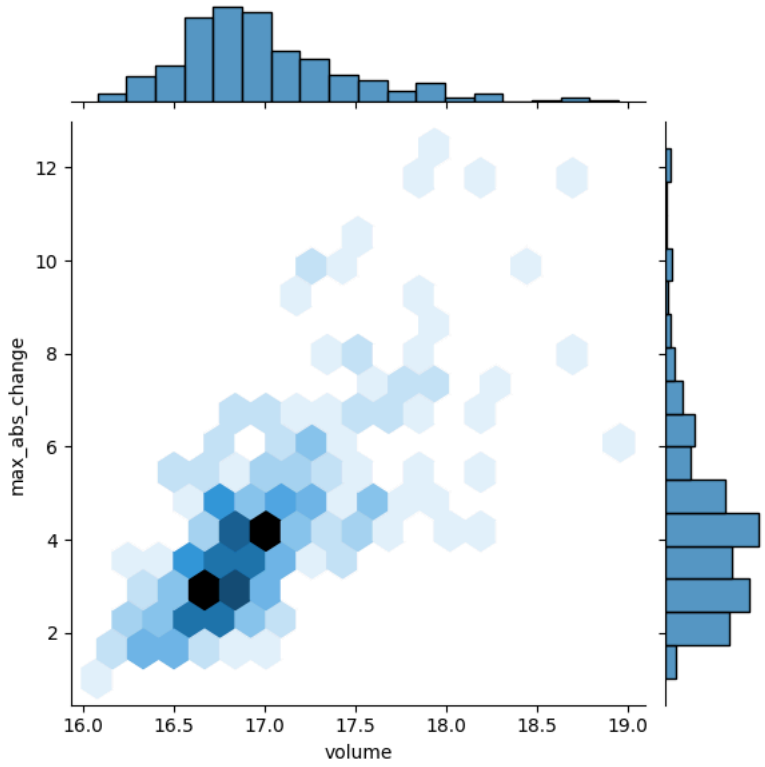
```
sns.jointplot(
    x='volume',
    y='max_abs_change',
    data=fb.assign(
        volume=np.log(fb.volume),
        max_abs_change=fb.high - fb.low
        )
    )
```

```
<seaborn.axisgrid.JointGrid at 0x7b295aedcd30>
```



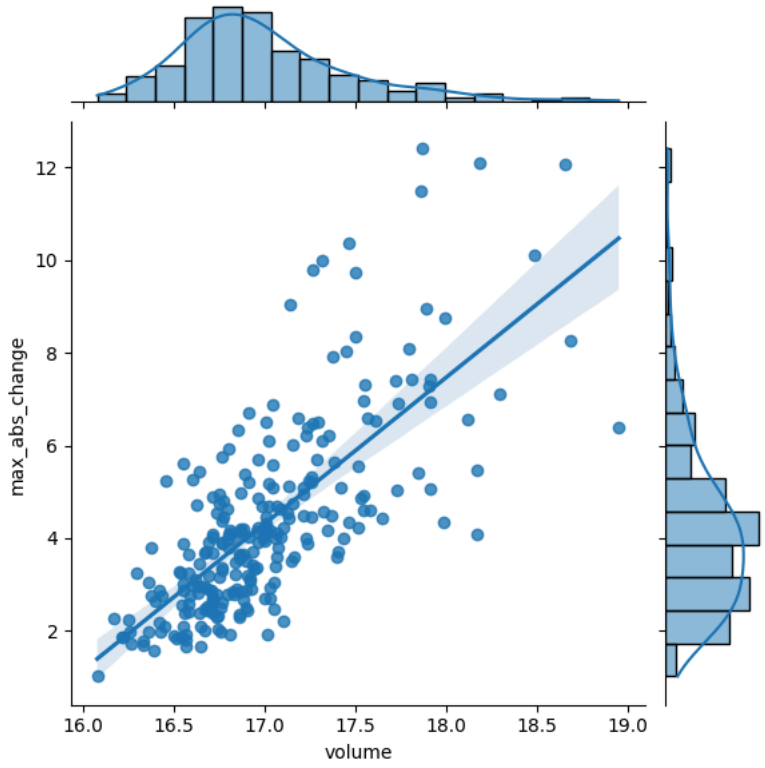

```
sns.jointplot(
    x='volume',
    y='max_abs_change',
    kind='hex',
    data=fb.assign(
        volume=np.log(fb.volume),
        max_abs_change=fb.high - fb.low
        )
    )
```

```
<seaborn.axisgrid.JointGrid at 0x7b29510a6290>
```



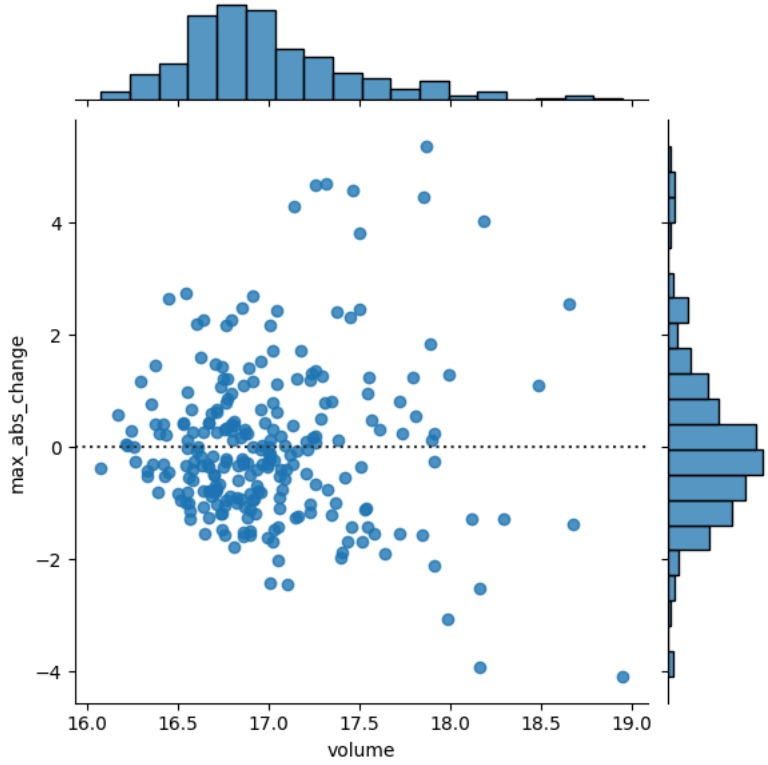

```
sns.jointplot(
    x='volume',
    y='max_abs_change',
    kind='reg',
    data=fb.assign(
        volume=np.log(fb.volume),
        max_abs_change=fb.high - fb.low
        )
    )
```

```
<seaborn.axisgrid.JointGrid at 0x7b2950e14b50>
```



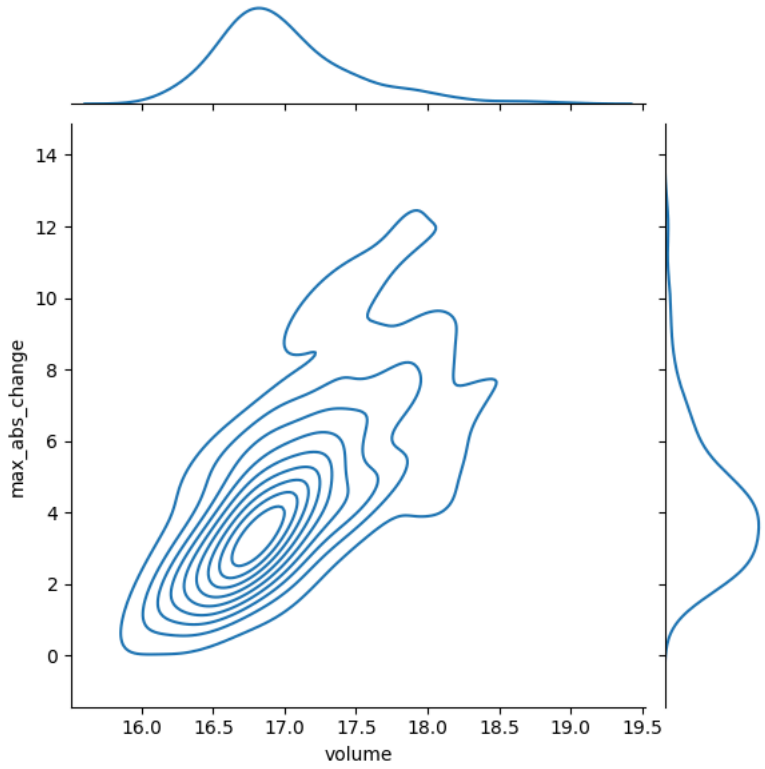
```
sns.jointplot(
    x='volume',
    y='max_abs_change',
    kind='resid',
    data=fb.assign(
        volume=np.log(fb.volume),
        max_abs_change=fb.high - fb.low
        )
    )
```

```
<seaborn.axisgrid.JointGrid at 0x7b2950fb4cd0>
```




```
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 0x7b29513f02b0>
```



Regression Plots

```
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')
for i in iterator:
  print(f'-->{i}')
```

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

```
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}')
```

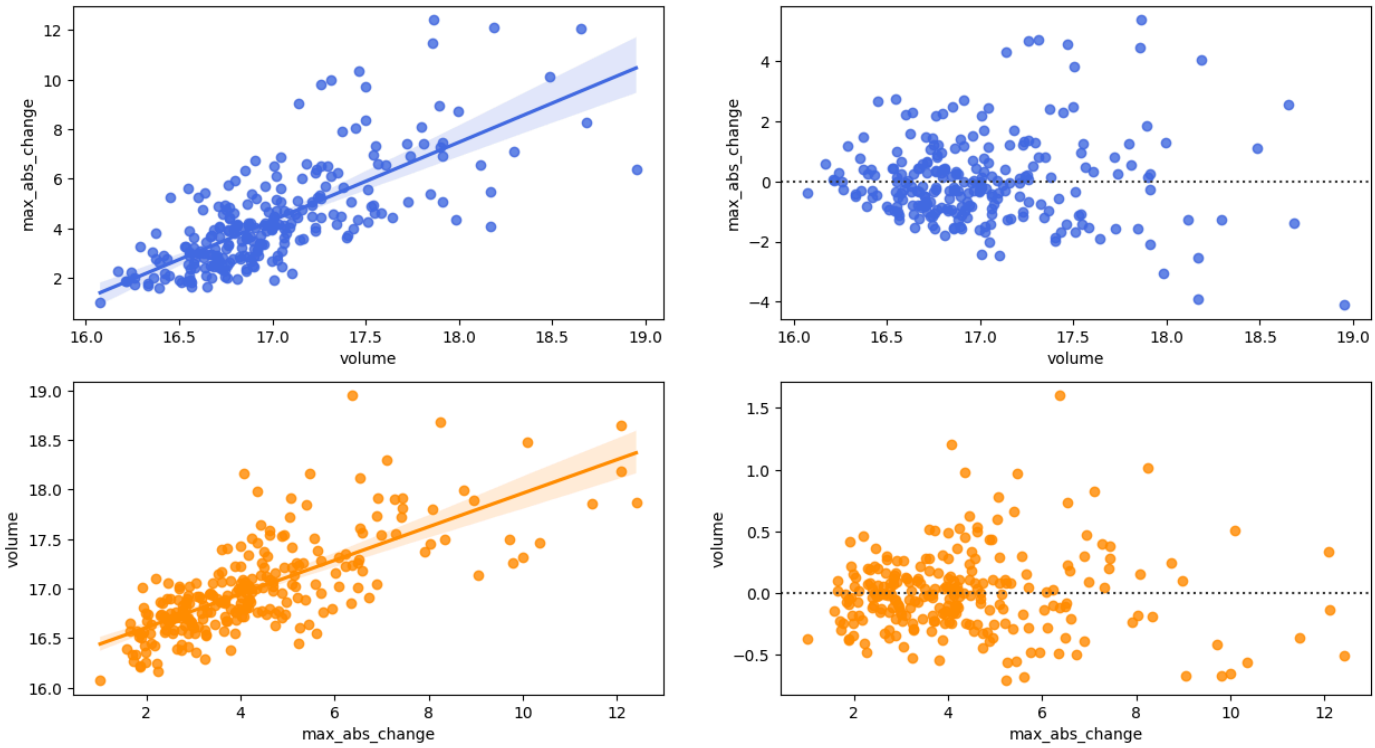
```
-->I'm an iterable
This prints again because it's an iterable:
-->I'm an iterable
```

```
from reg_resid_plot import reg_resid_plots
reg_resid_plots(fb_reg_data)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-16-ae2d095ec697> in <cell 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'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

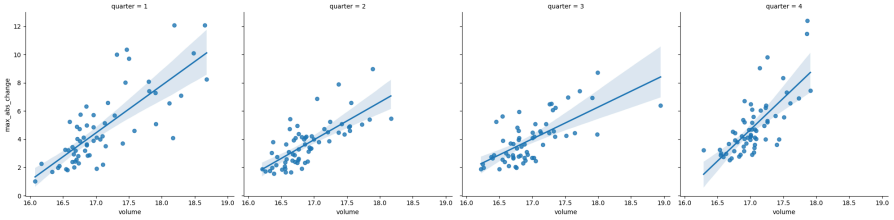
OPEN EXAMPLES

```python
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
reg_resid_plots(fb_reg_data)
```




```
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'
    )

```

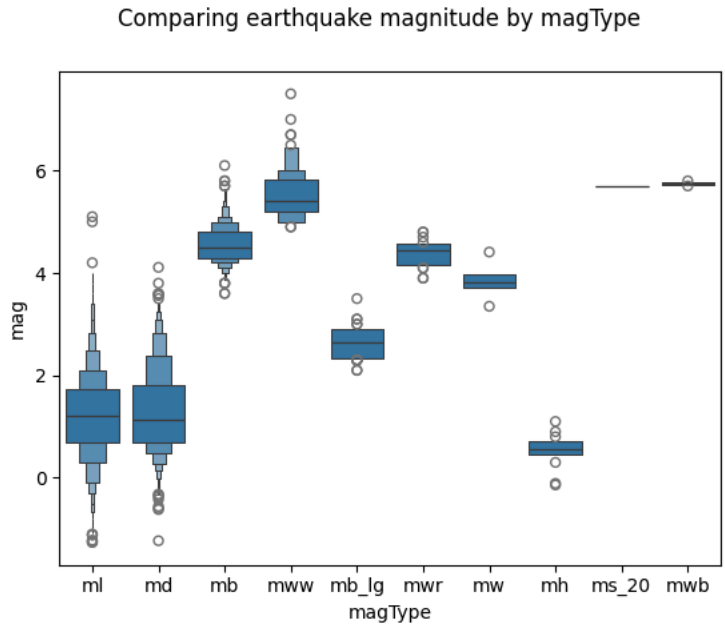
```
<seaborn.axisgrid.FacetGrid at 0x7b295033f9d0>
```



boxenplot()

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

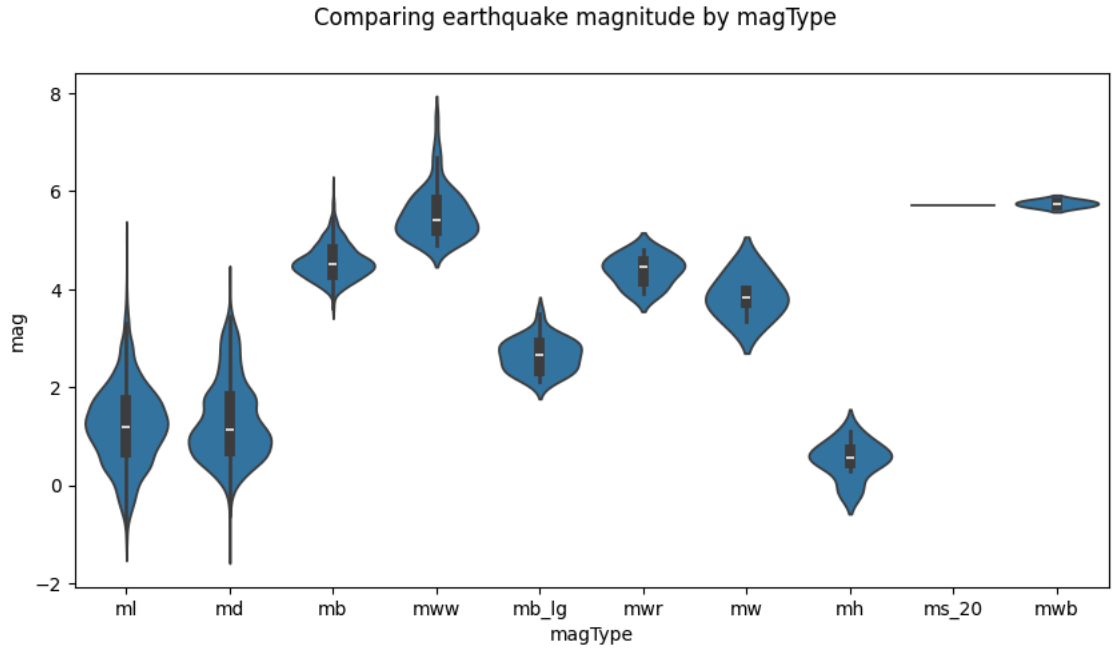
```
Text(0.5, 0.98, 'Comparing earthquake magnitude by magType')
```



violinplot()

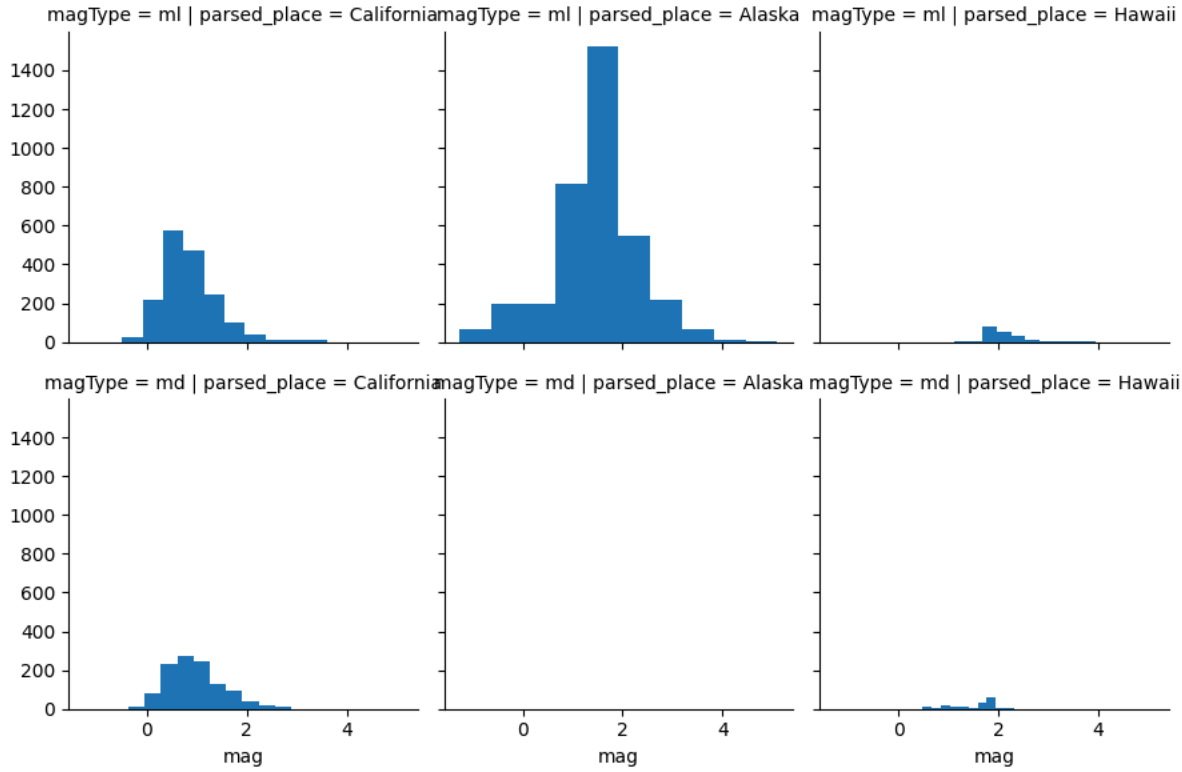
```
1 fig, axes = plt.subplots(figsize=(10, 5))
2 sns.violinplot(x='magType', y='mag', data=quakes[['magType', 'mag']],
3                ax=axes, scale='width') # all violins have same width
4
5 plt.suptitle('Comparing earthquake magnitude by magType')
```

```
<ipython-input-22-a8ac06ad4d9e>: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(x='magType', y='mag', data=quakes[['magType', 'mag']],
Text(0.5, 0.98, 'Comparing earthquake magnitude by magType')
```



Faceting

```
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')
```



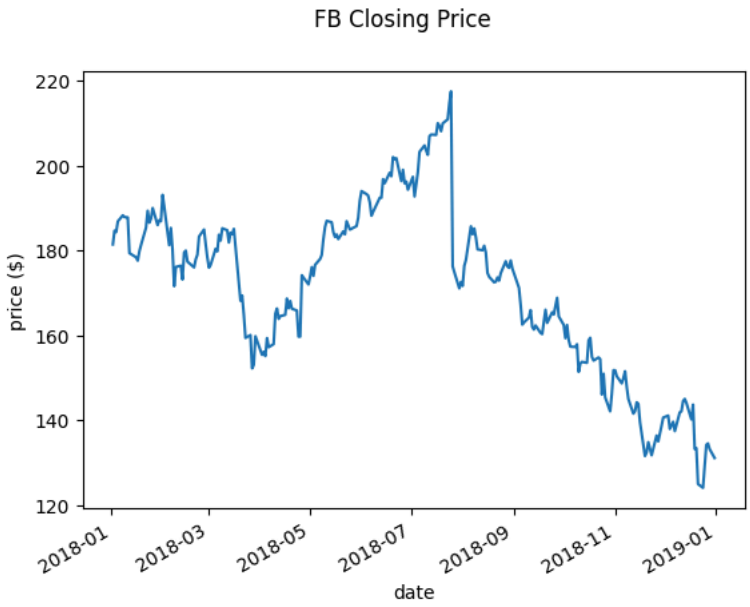
9.5 Formatting Plots

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

Titles and Axis Labels

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

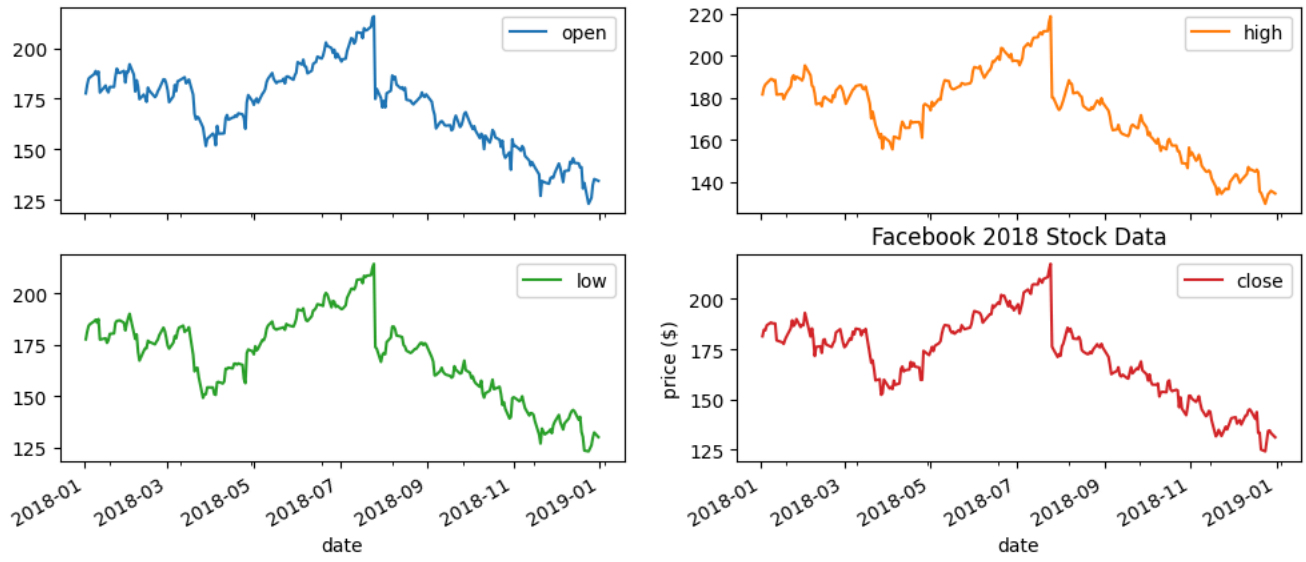
```
Text(0, 0.5, 'price ($)')
```



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

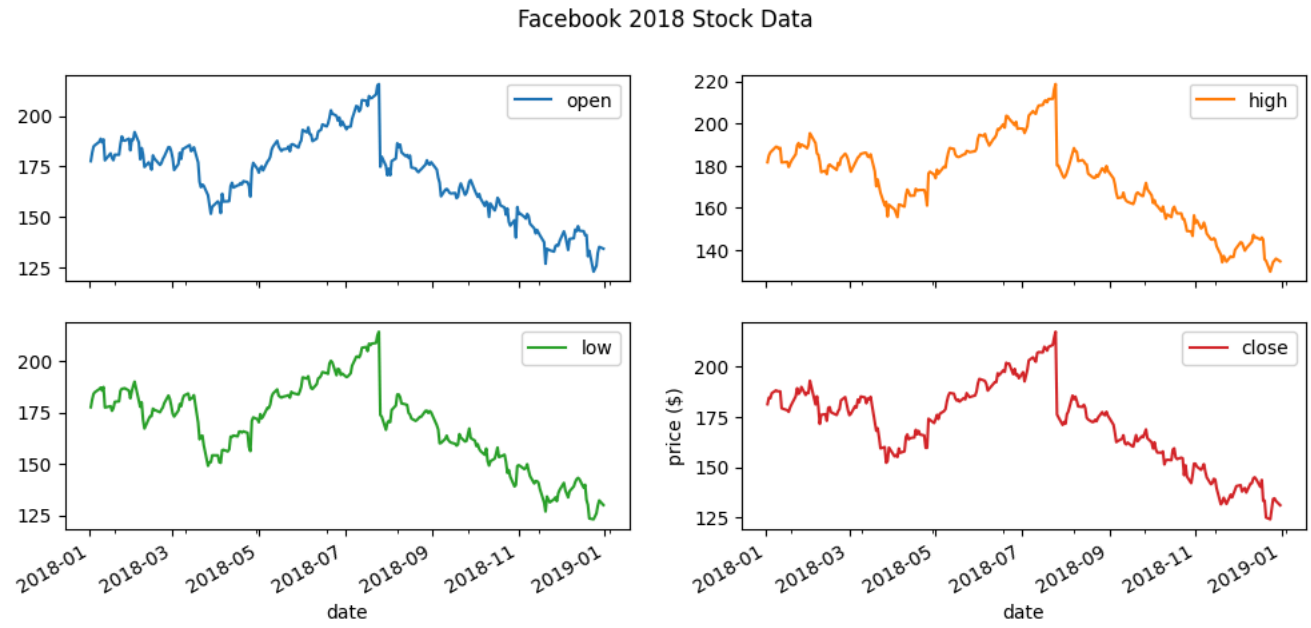
```
1 fb.iloc[:,:4].plot(subplots=True, layout=(2, 2), figsize=(12, 5))
2 plt.title('Facebook 2018 Stock Data')
3 plt.xlabel('date')
4 plt.ylabel('price ($)')
```

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




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

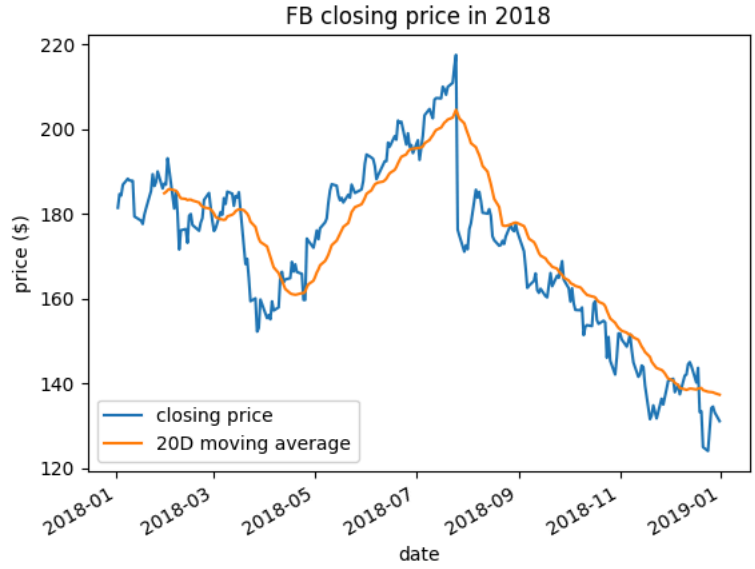
```
Text(0, 0.5, 'price ($)')
```



Legends

```
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 ($)')
```

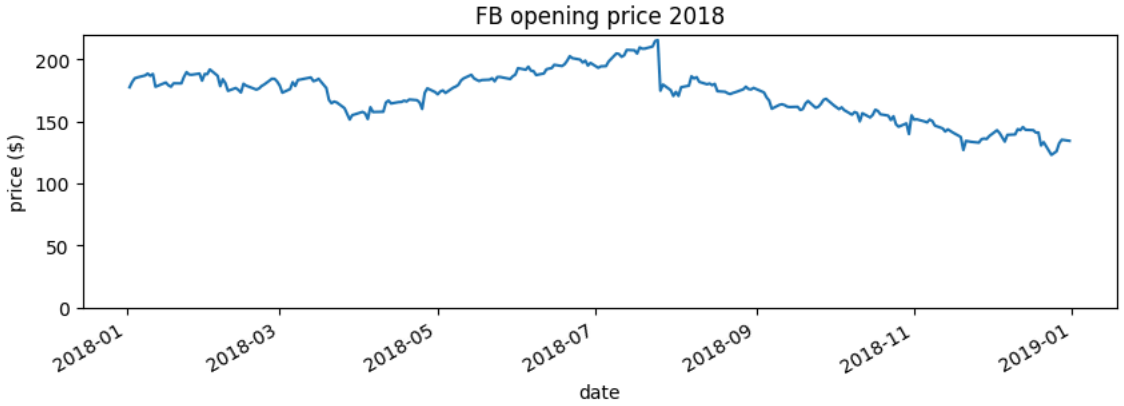
```
Text(0, 0.5, 'price ($)')
```



Formatting Axes

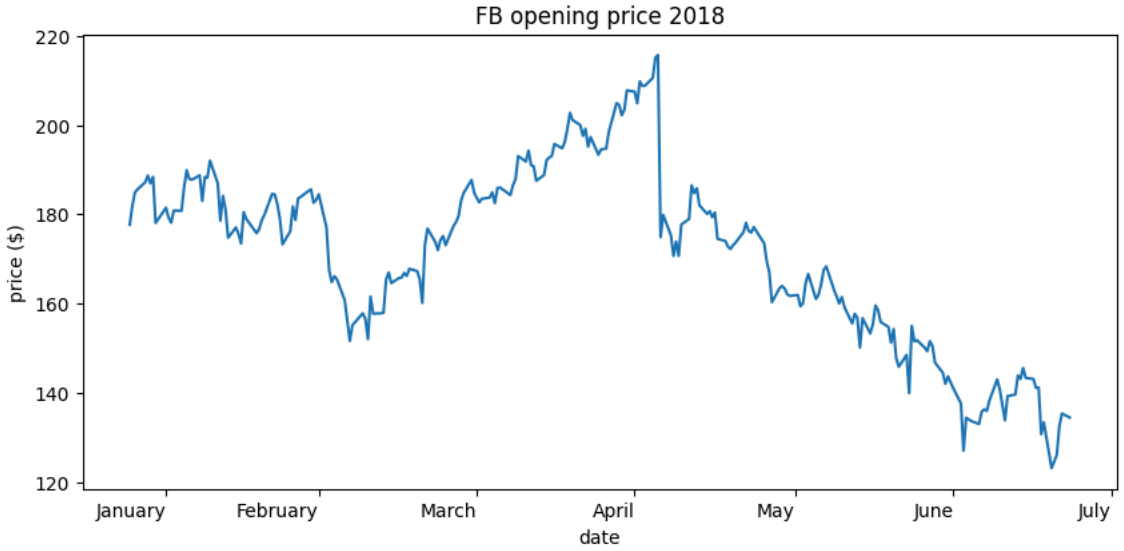
```
fb.open.plot(figsize=(10, 3), title='FB opening price 2018')
plt.ylim(0, None)
plt.ylabel('price ($)')
```

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



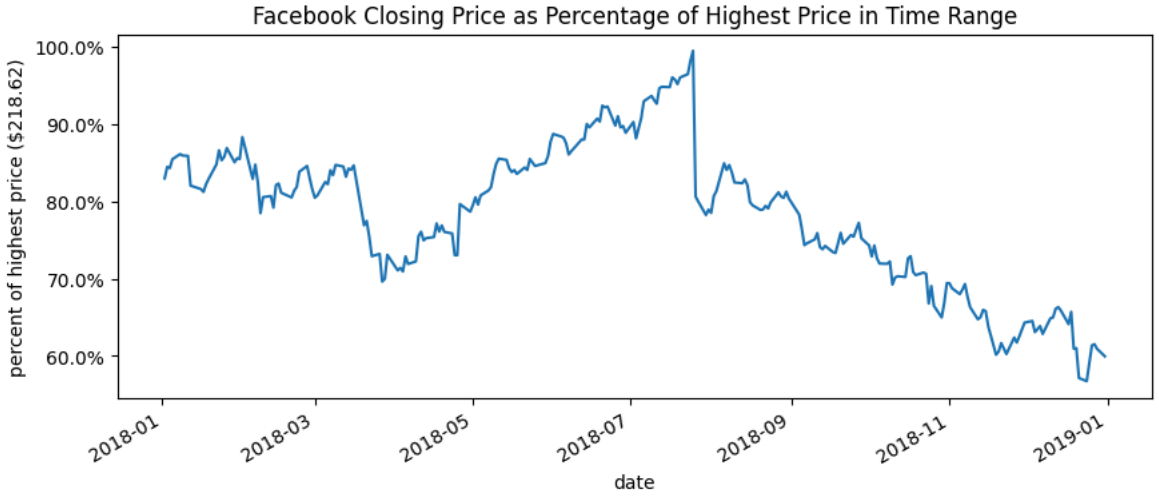

```
1 import calendar
2
3 fb.open.plot(figsize=(10, 5), rot=0, title='FB opening price 2018')
4 locs, labels = plt.xticks()
5 plt.xticks(locs + 15 , calendar.month_name[1:8])
6 plt.ylabel('price ($)')
```

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



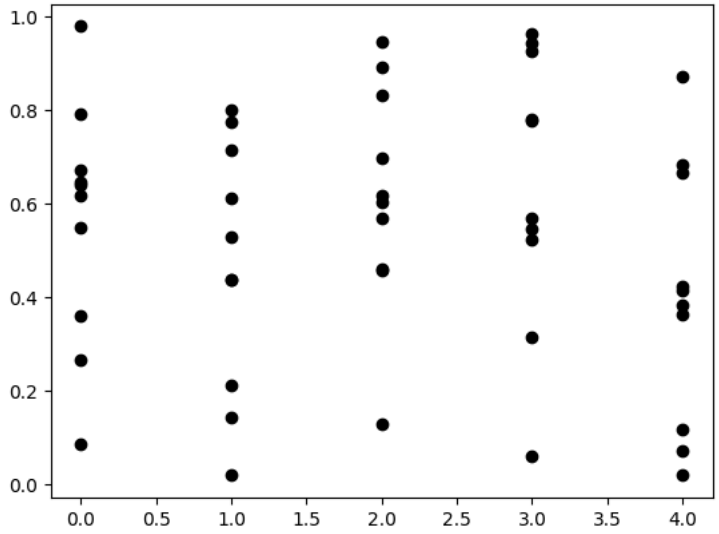

```
1 import matplotlib.ticker as ticker
2
3 ax = fb.close.plot(figsize=(10, 4),
4                    title='Facebook Closing Price as Percentage of Highest Price in Time Range')
5
6 ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=fb.high.max()))
7
8 ax.set_yticks([fb.high.max()*pct for pct in np.linspace(0.6, 1, num=5)]) # show round percentages only (60%, 80%, etc.)
9 ax.set_ylabel(f'percent of highest price (${fb.high.max()})')
```

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

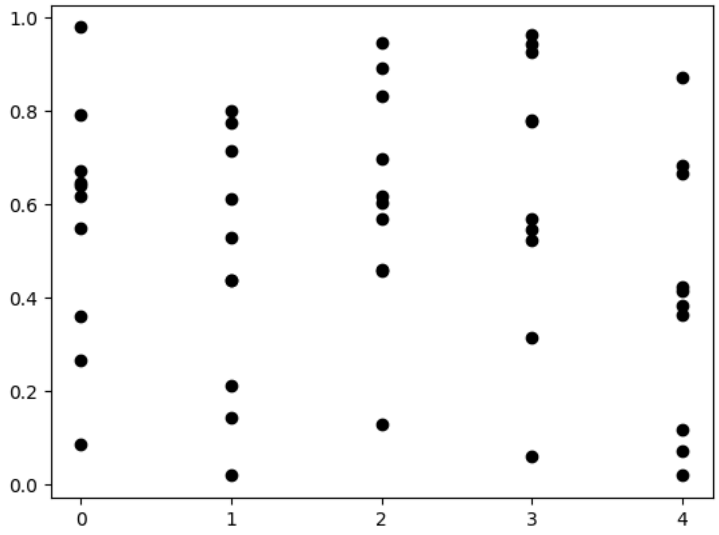



```
1 fig, ax = plt.subplots(1, 1)
2 np.random.seed(0)
3 ax.plot(np.tile(np.arange(0, 5), 10), np.random.rand(50), 'ko')
```

```
[<matplotlib.lines.Line2D at 0x7b294cf3f7f0>]
```




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



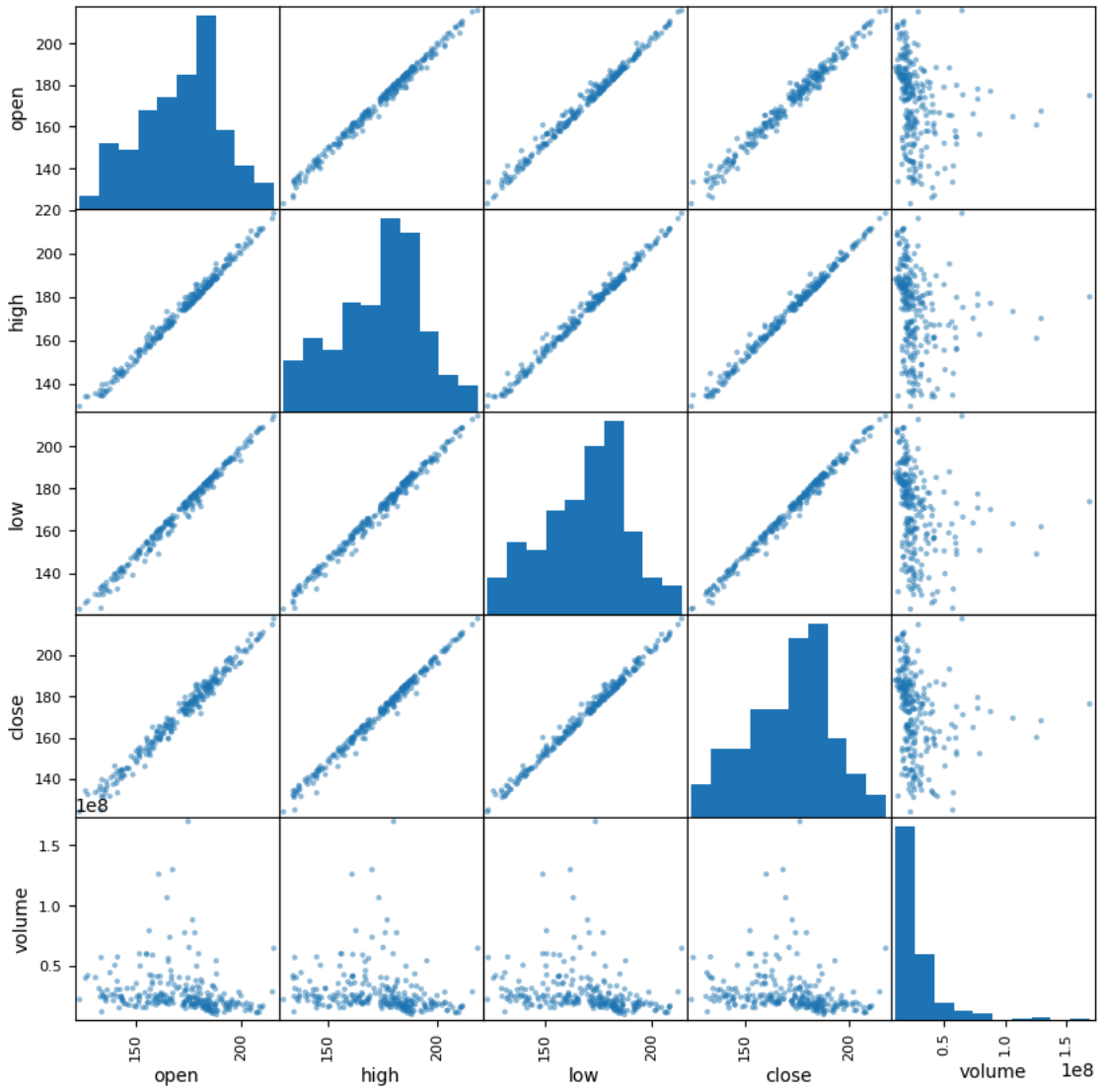
9.6 Customizing Visualizations

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

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

Scatter Matrix

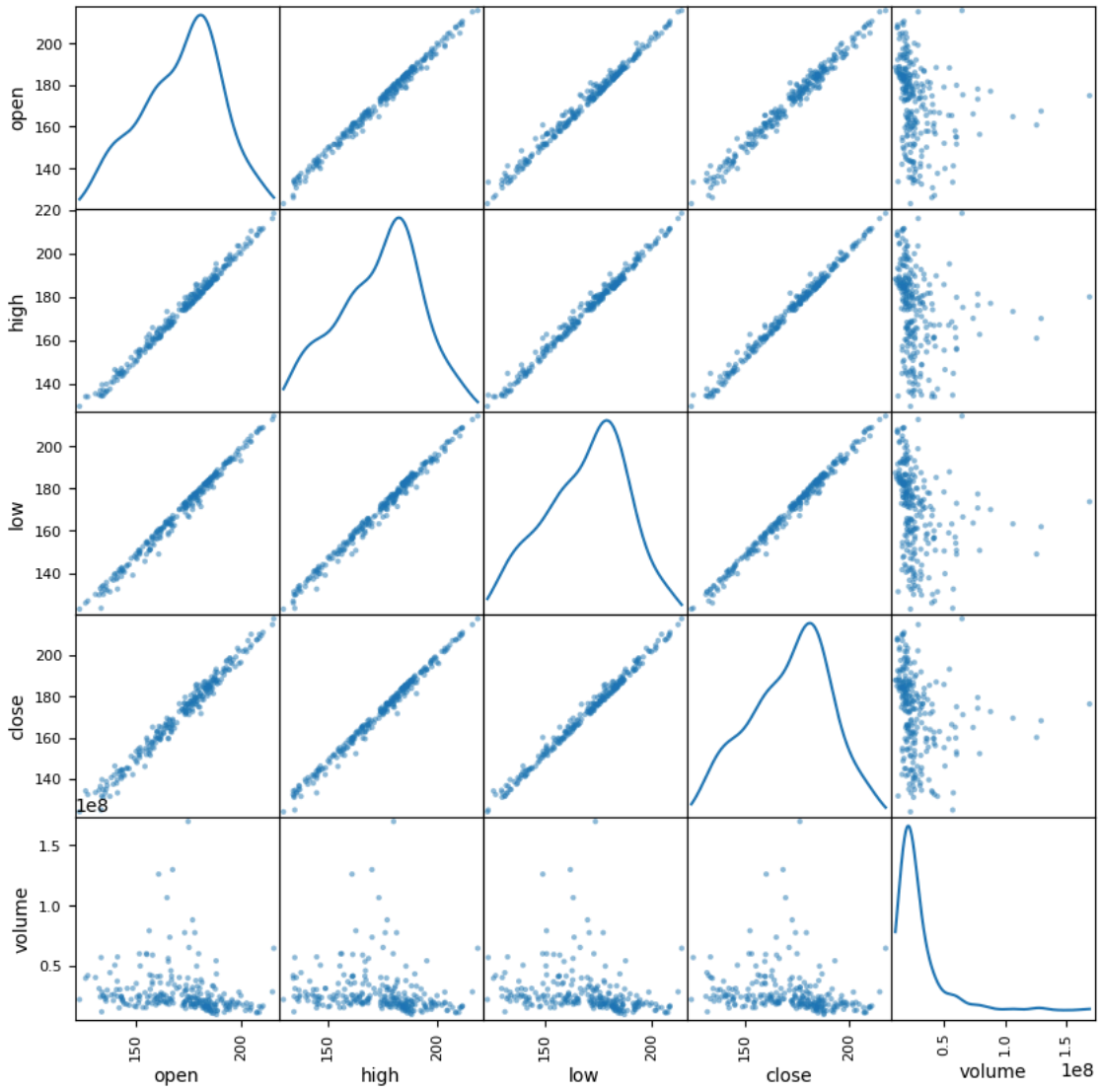
```
from pandas.plotting import scatter_matrix
scatter_matrix(fb, figsize=(10, 10))
```

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




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

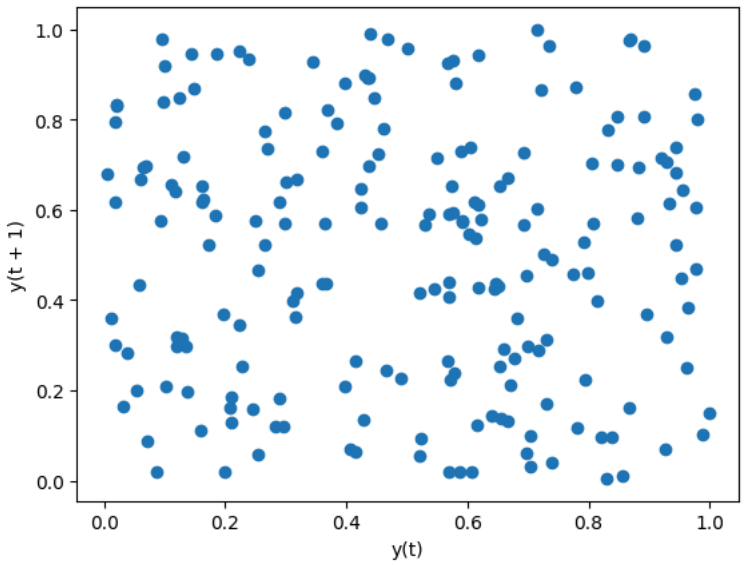
```
array([[<Axes: xlabel='open', ylabel='open'>,
        <Axes: xlabel='high', ylabel='open'>,
        <Axes: xlabel='low', ylabel='open'>,
        <Axes: xlabel='close', ylabel='open'>,
        <Axes: xlabel='volume', ylabel='open'>],
       [<Axes: xlabel='open', ylabel='high'>,
        <Axes: xlabel='high',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



Lag Plot

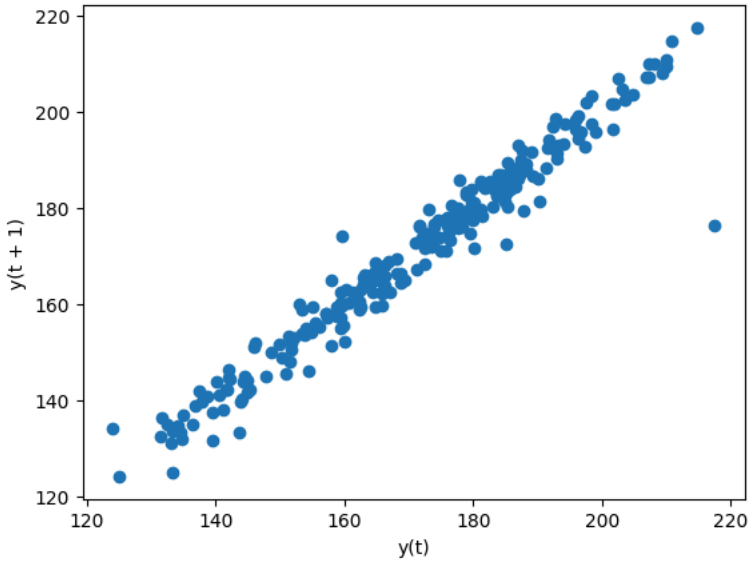
```
1 from pandas.plotting import lag_plot
2 np.random.seed(0) # make this repeatable
3 lag_plot(pd.Series(np.random.random(size=200)))
```

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



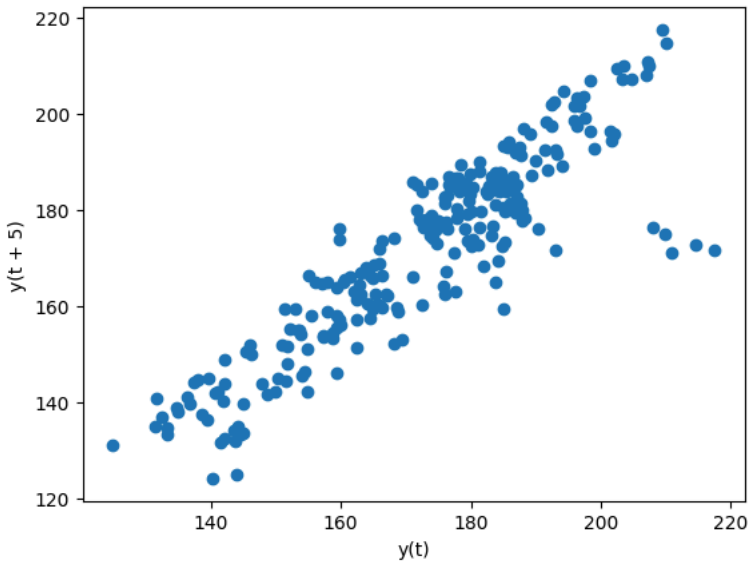

```
1 lag_plot(fb.close)
```

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




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

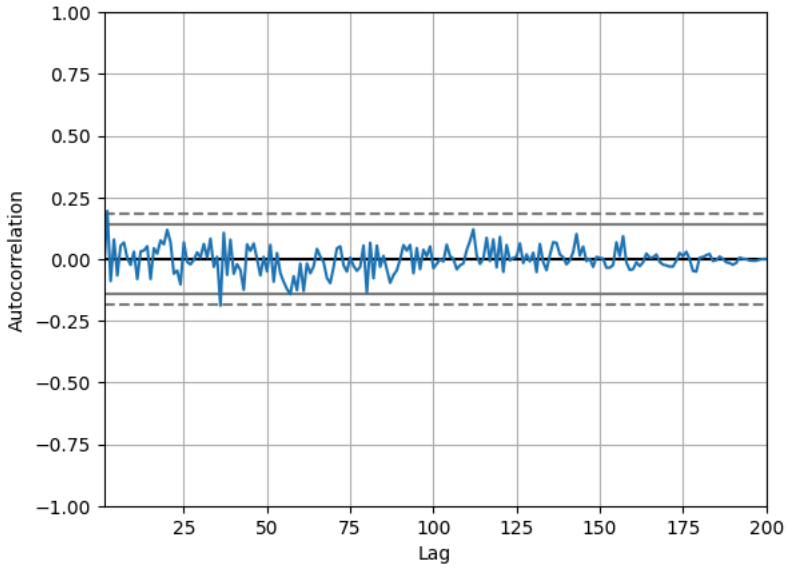
```
<Axes: xlabel='y(t)', ylabel='y(t + 5)'>
```



Autocorrelation Plots

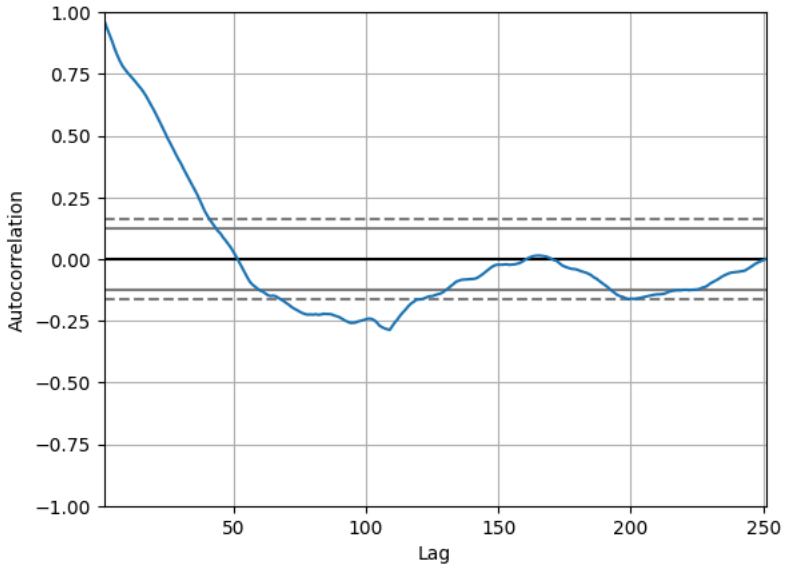
```
1 from pandas.plotting import autocorrelation_plot
2 np.random.seed(0) # make this repeatable
3 autocorrelation_plot(pd.Series(np.random.random(size=200)))
```

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



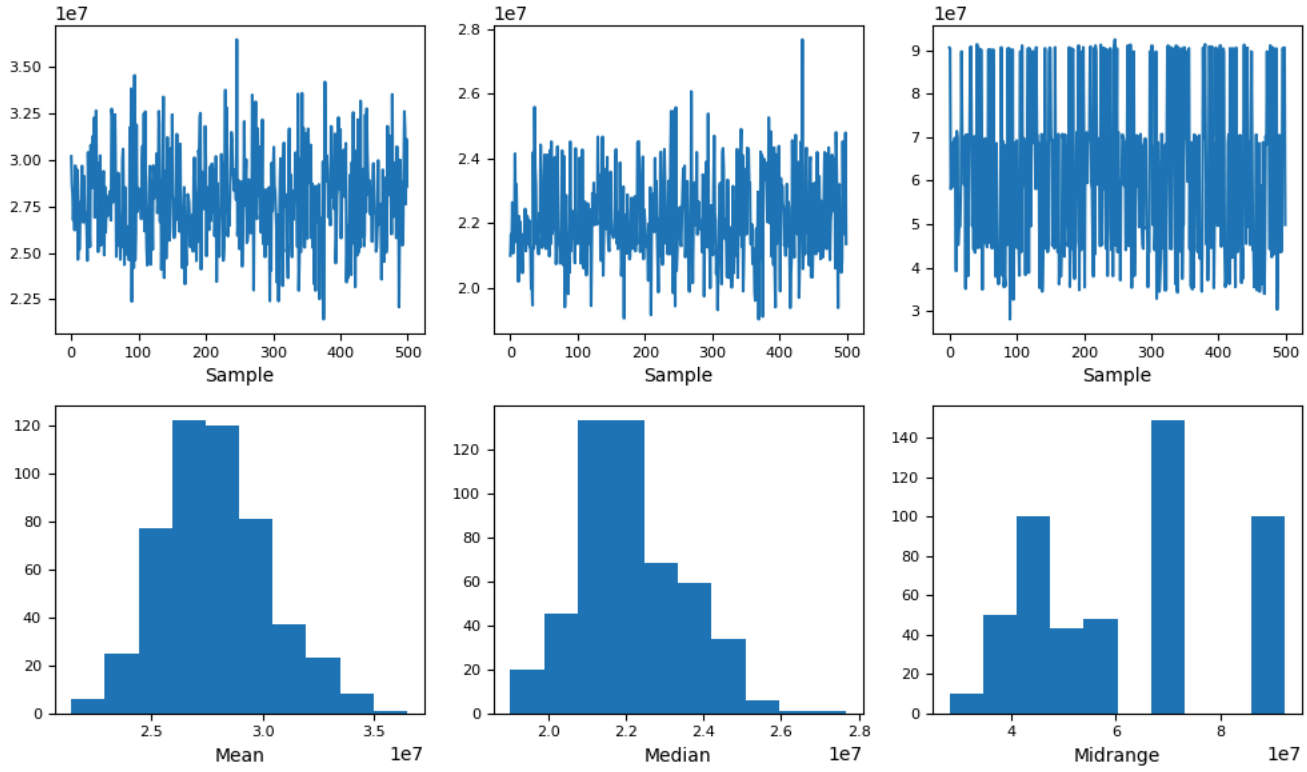

```
1 autocorrelation_plot(fb.close)
```

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



Bootstrap Plot

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



Data Analysis:

Provide comments on output from the procedures.

- Analyzing the output of the procedures about Seaborn, Plots, and Visualzations made me understand the logic behind the various functions and methods in this module.
- From the output, I can see that seaborn and its functions provides colorful and informative plots even though the codes used were significantly shorter than the ones used before in the other libaries. Lots of other plots are also presented, which will be a great help to me in the future if I need to personally cuztomize the visual data to fit the theme or topic at hand.

Supplementary Activity:

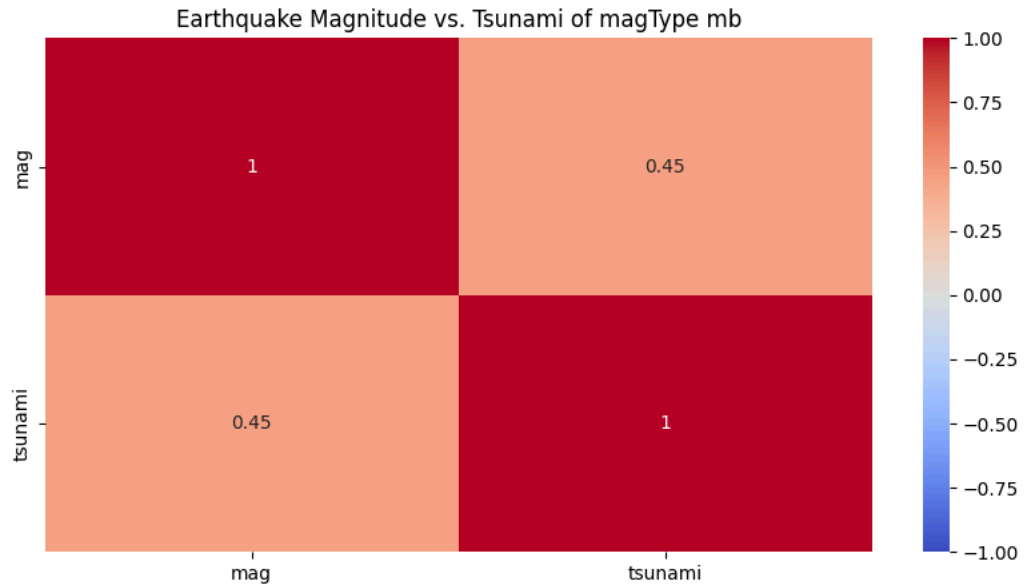
Using the CSV files provided and what we have learned so far in this module, complete the following exercises:

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.

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

```
#1. Using seaborn, create a heatmap to visualize the correlation coefficients between earthquake magnitude and whether there was a tsunar

earthquakes = pd.read_csv('/content/data/earthquakes.csv')
magType_mb_quake = earthquakes[earthquakes['magType'] == 'mb']
correlation_matrix = magType_mb_quake[['mag', 'tsunami']].corr()

plt.figure(figsize=(10, 5))
sns.heatmap(correlation_matrix,
            annot=True,
            cmap='coolwarm',
            vmin=-1,
            vmax=1)
plt.title('Earthquake Magnitude vs. Tsunami of magType mb')
```

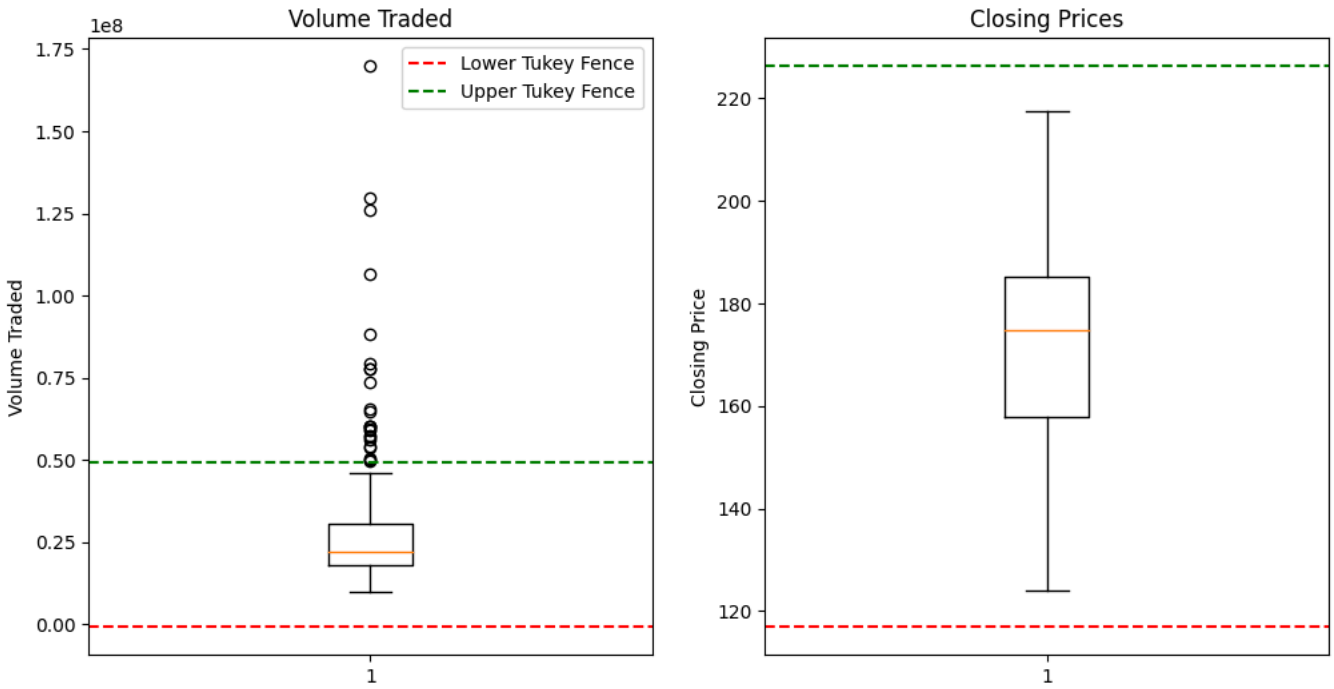
```
Text(0.5, 1.0, 'Earthquake Magnitude vs. Tsunami of magType mb')
```




```
#Create a box plot of Facebook volume traded and closing prices, and draw reference lines for the bounds of a Tukey fence with a multipl
#Creating the Bounds:
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

Q1_close = fb['close'].quantile(0.25)
Q3_close = fb['close'].quantile(0.75)
IQR_close = Q3_close - Q1_close
lower_bound_close = Q1_close - 1.5 * IQR_close
upper_bound_close = Q3_close + 1.5 * IQR_close
```

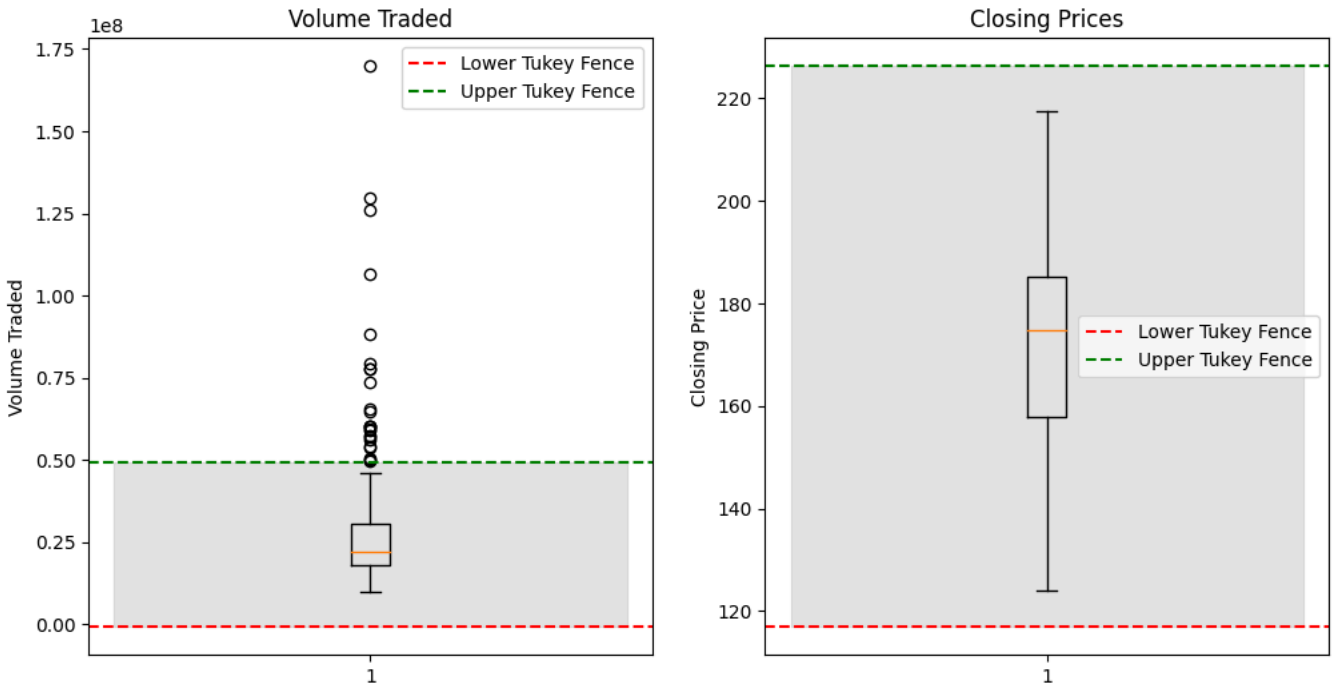
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

# Box plot for the volume traded
axes[0].boxplot(fb['volume'])
axes[0].set_title('Volume Traded')
axes[0].axhline(lower_bound_volume, color='red', linestyle='--', label='Lower Tukey Fence')
axes[0].axhline(upper_bound_volume, color='green', linestyle='--', label='Upper Tukey Fence')
axes[0].set_ylabel('Volume Traded')
axes[0].legend()

# Box plot for the closing prices
axes[1].boxplot(fb['close'])
axes[1].set_title('Closing Prices')
axes[1].axhline(lower_bound_close, color='red', linestyle='--', label='Lower Tukey Fence')
axes[1].axhline(upper_bound_close, color='green', linestyle='--', label='Upper Tukey Fence')
axes[1].set_ylabel('Closing Price')
```

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



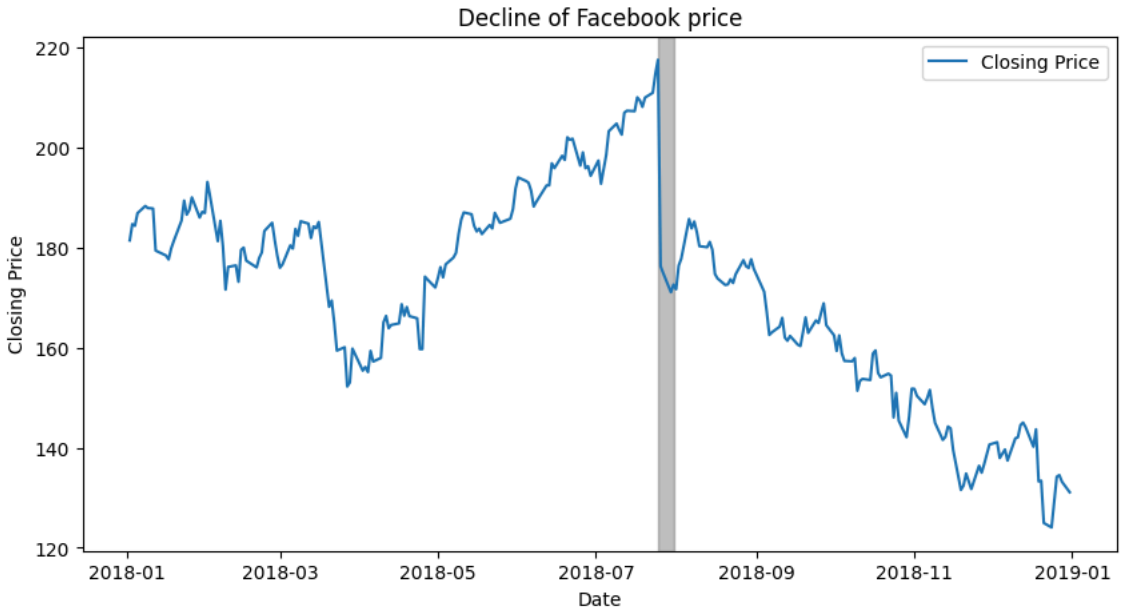

```
#Fill in the area between the bounds in the plot from exercise #2.
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

#Filled Box plot for the volume traded
axes[0].boxplot(fb['volume'])
axes[0].set_title('Volume Traded')
axes[0].axhline(lower_bound_volume, color='red', linestyle='--', label='Lower Tukey Fence')
axes[0].axhline(upper_bound_volume, color='green', linestyle='--', label='Upper Tukey Fence')
axes[0].fill_between([0, 2], lower_bound_volume, upper_bound_volume, color='gray', alpha=0.2)
axes[0].set_ylabel('Volume Traded')
axes[0].legend()

#FIlled Box plot for the closing prices
axes[1].boxplot(fb['close'])
axes[1].set_title('Closing Prices')
axes[1].axhline(lower_bound_close, color='red', linestyle='--', label='Lower Tukey Fence')
axes[1].axhline(upper_bound_close, color='green', linestyle='--', label='Upper Tukey Fence')
axes[1].fill_between([0, 2], lower_bound_close, upper_bound_close, color='gray', alpha=0.2)
axes[1].set_ylabel('Closing Price')
axes[1].legend()
```

```
<matplotlib.legend.Legend at 0x7b29a5055330>
```



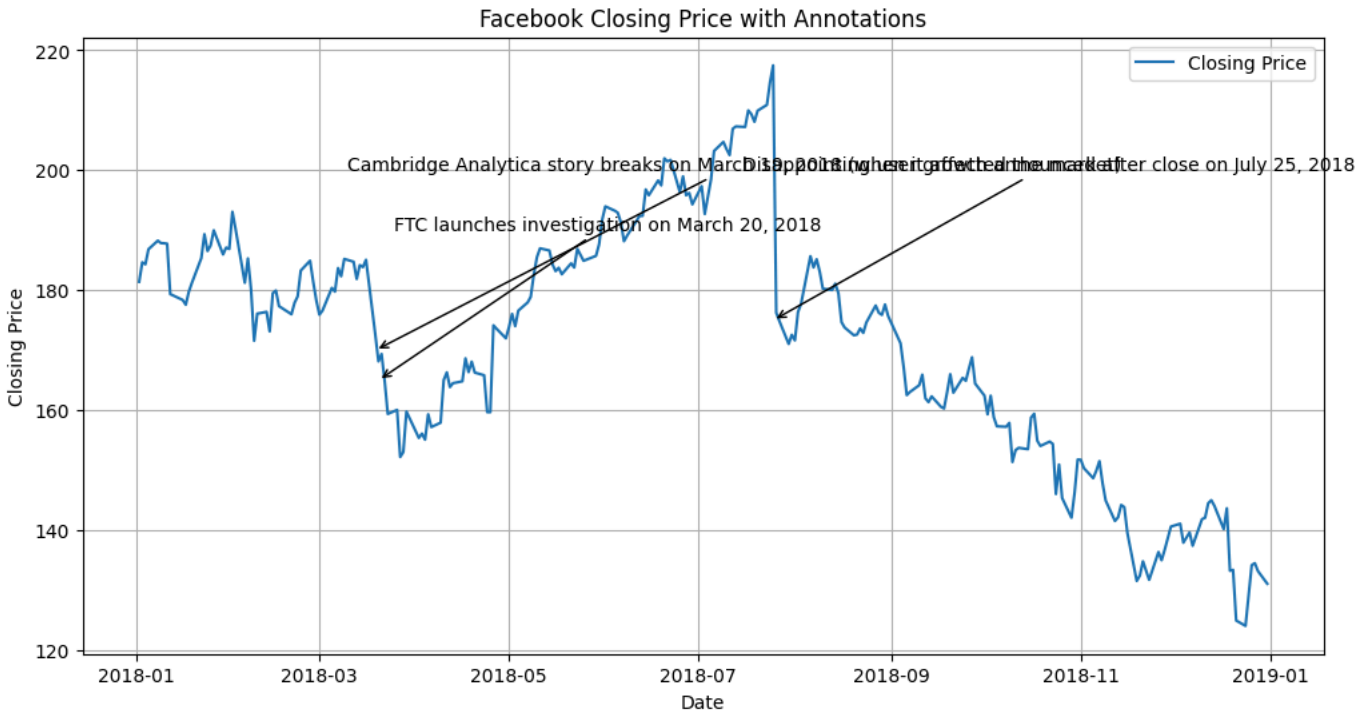

```
#Use axvspan() to shade a rectangle from '2018-07-25' to '2018-07-31', which marks the large decline in Facebook price on a line plot of
fb = pd.read_csv('/content/data/fb_stock_prices_2018.csv')
fb['date'] = pd.to_datetime(fb['date'])

plt.figure(figsize=(10, 5))
plt.plot(fb['date'], fb['close'], label='Closing Price')
plt.axvspan('2018-07-25', '2018-07-31', color='gray', alpha=0.5)
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title('Decline of Facebook price')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b29a50c3ac0>
```




```
#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

fb = pd.read_csv('/content/data/fb_stock_prices_2018.csv')
fb['date'] = pd.to_datetime(fb['date'])

plt.figure(figsize=(10, 6))
```

```
plt.figure(figsize=(12, 6))
plt.plot(fb['date'], fb['close'], label='Closing Price')

plt.annotate('Disappointing user growth announced after close on July 25, 2018',
             xy=(pd.Timestamp('2018-07-25'), 175),
             xytext=(pd.Timestamp('2018-07-15'), 200),
             arrowprops=dict(facecolor='black', arrowstyle='->'),
             fontsize=10
            )

plt.annotate('Cambridge Analytica story breaks on March 19, 2018 (when it affected the market)',
             xy=(pd.Timestamp('2018-03-19'), 170),
             xytext=(pd.Timestamp('2018-03-10'), 200),
             arrowprops=dict(facecolor='black', arrowstyle='->'),
             fontsize=10
            )

plt.annotate('FTC launches investigation on March 20, 2018',
             xy=(pd.Timestamp('2018-03-20'), 165),
             xytext=(pd.Timestamp('2018-03-25'), 190),
             arrowprops=dict(facecolor='black', arrowstyle='->'),
             fontsize=10
            )

plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title('Facebook Closing Price with Annotations')
plt.legend()
plt.grid(True)
```




```
#Modify the reg_resid_plots() function to use a matplotlib colormap instead of cycling between two colors. Remember, for this use case,
```

Summary/Conclusion Provide a summary of your learnings and the conclusions for this activity.

- I therefore conclude that these activities allowed me to have a deeper udnerstanding on the new and various data visualization techniques using Seaborn and the recent ones, the matplotlib and pandas. These, of course, include the different plots and their cuztomization.