Introduction to Seaborn

Marquez, Keith Leigh Zhen R.

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) *italicized text* Earthquake data from September 18, 2018 - October 13, 2018 (obtained from the US Geological Survey (USGS) using the USGS API

Setup

```
%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')
```

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.

```
# Convert the time column to datetime format using ms unit
quakes = quakes.assign(
    time=lambda x: pd.to_datetime(x.time, unit='ms')
    )

# Set the time column as the index
quakes = quakes.set_index('time')

# Filter the data for earthquakes occurring on September 28, 2018, in Indonesia with a magnitude of 7.5 and causing a tsunami
indonesia_tsunami_eq = quakes.loc['2018-09-28'].query(
    "parsed_place == 'Indonesia' and tsunami == 1 and mag == 7.5"
)
```

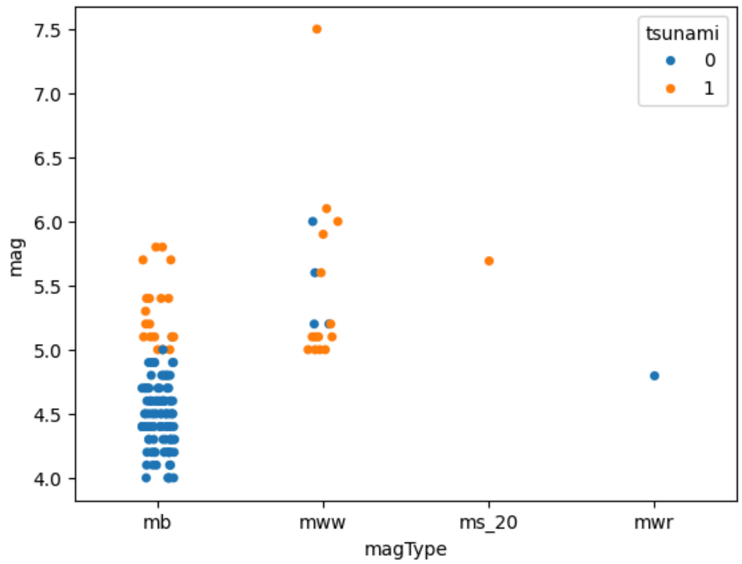
	mag	magType	place	tsunami	parsed_place
time					
2018-09-28 10:02:43.480	7.5	mww	78km N of Palu, Indonesia	1	Indonesia

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 was; however, it isn't too easy to see density of the points due to overlap:

```
sns.stripplot( # Create a stripplot to visualize earthquake magnitude by magnitude type, colored by tsunami occurrence, for earthquakes in In
    x='magType',
    y='mag',
    hue='tsunami',
    data=quakes.query('parsed_place == "Indonesia"')
)
```

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

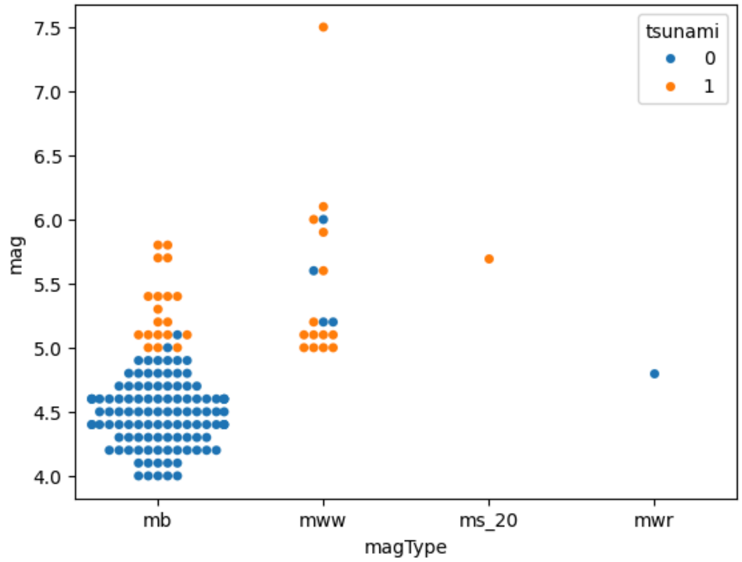


swarmplot()

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 :

```
sns.swarmplot( # Create a swarmplot to visualize earthquake magnitude by magnitude type, colored by tsunami occurrence, for earthquakes in I
    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
  warnings.warn(msg, UserWarning)
```



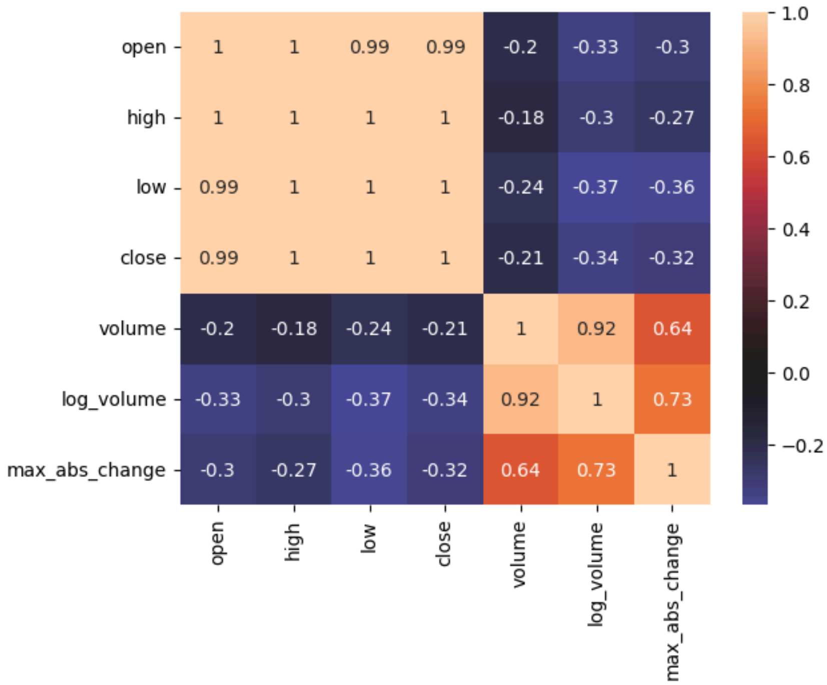
Correlations and Heatmaps

heatmap()

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

```
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()

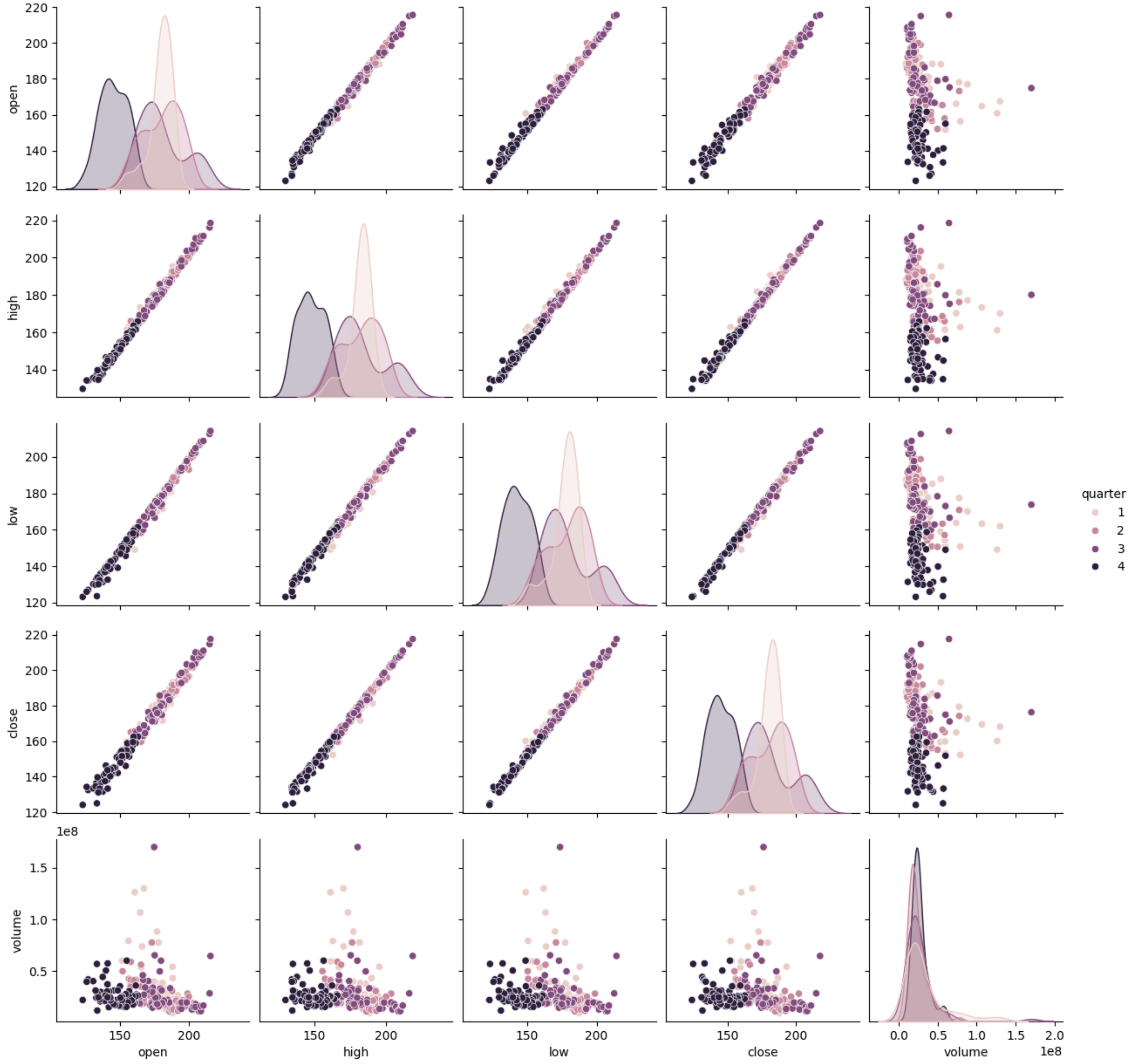
The pair plot is seaborn's answer to the scatter matrix we saw in the pandas subplotting notebook:

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

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):

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

```
<seaborn.axisgrid.PairGrid at 0x7bf61af471c0>
```

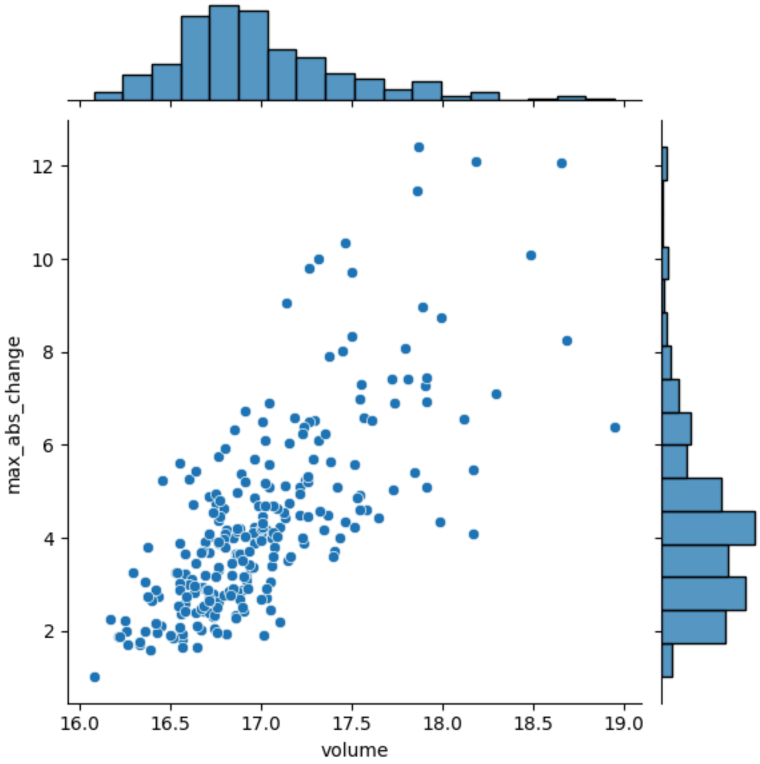


jointplot()

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:

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

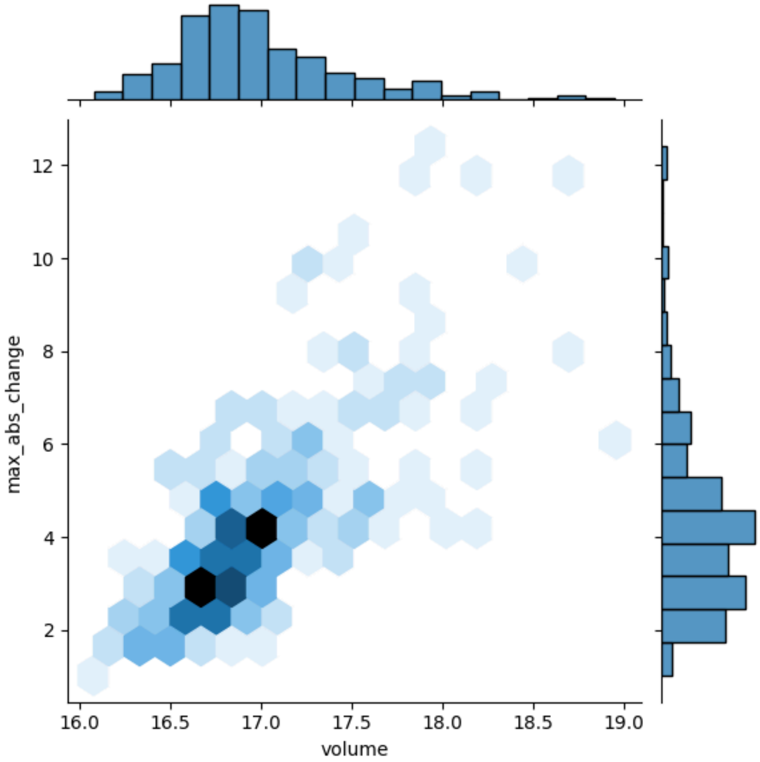
```
<seaborn.axisgrid.JointGrid at 0x7bf6110850f0>
```



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:

```
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
        )
)
```

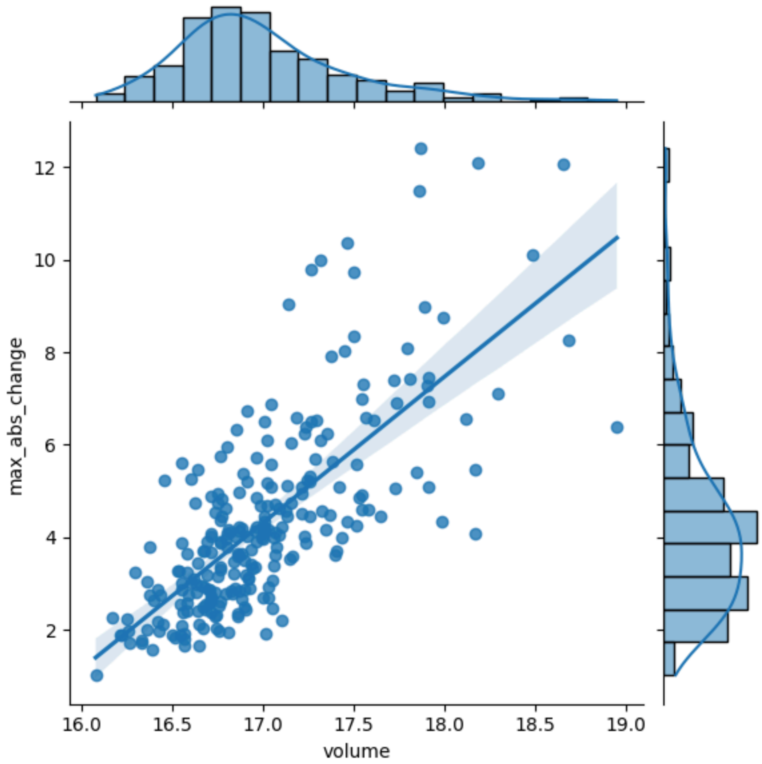
```
<seaborn.axisgrid.JointGrid at 0x7bf61151a230>
```



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

```
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
        )
    )
```

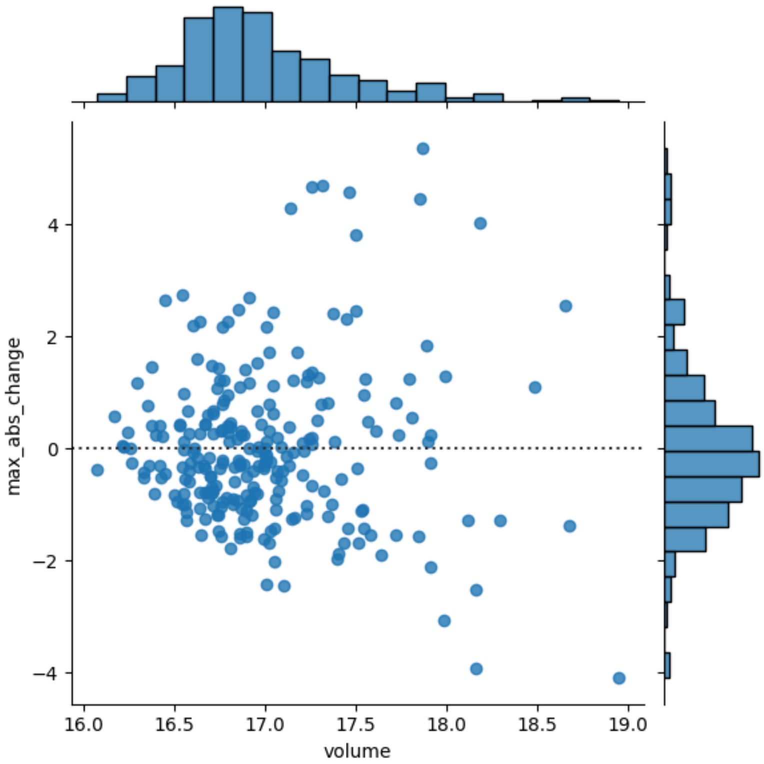
```
<seaborn.axisgrid.JointGrid at 0x7bf61139cd60>
```



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

```
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
        )
)
```

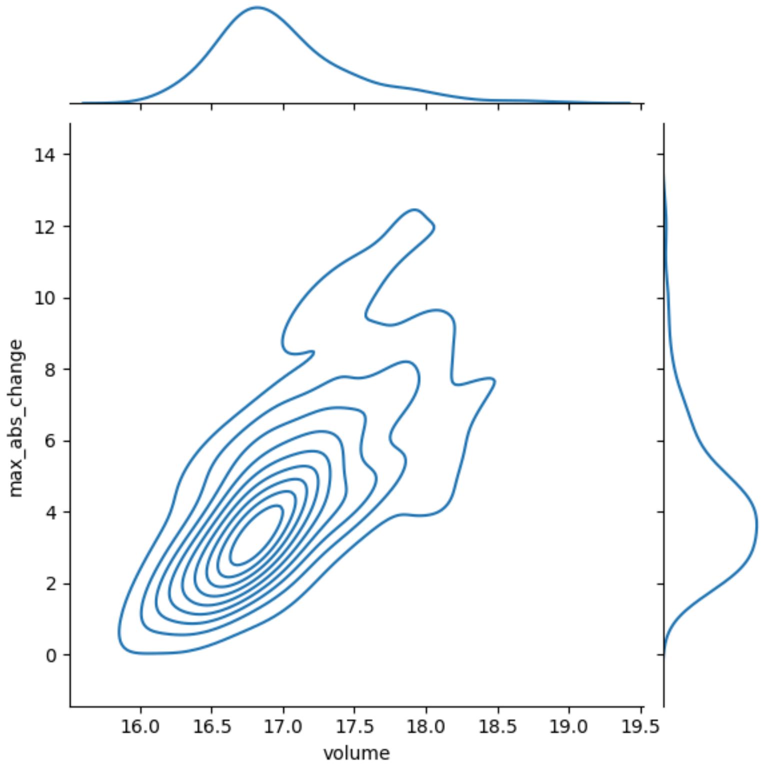
```
<seaborn.axisgrid.JointGrid at 0x7bf61134b6d0>
```



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

```
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 0x7bf611331630>
```



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:

```
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:

```
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:

```
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
```

Iterables are objects that can be iterated over. When entering a loop, an iterator is made from the iterable to handle the iteration. Iterators are iterables, but not all iterables are iterators. A list is an iterable. If we turn that iterator into an iterable (a list in this case), the second loop runs:

```
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:

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

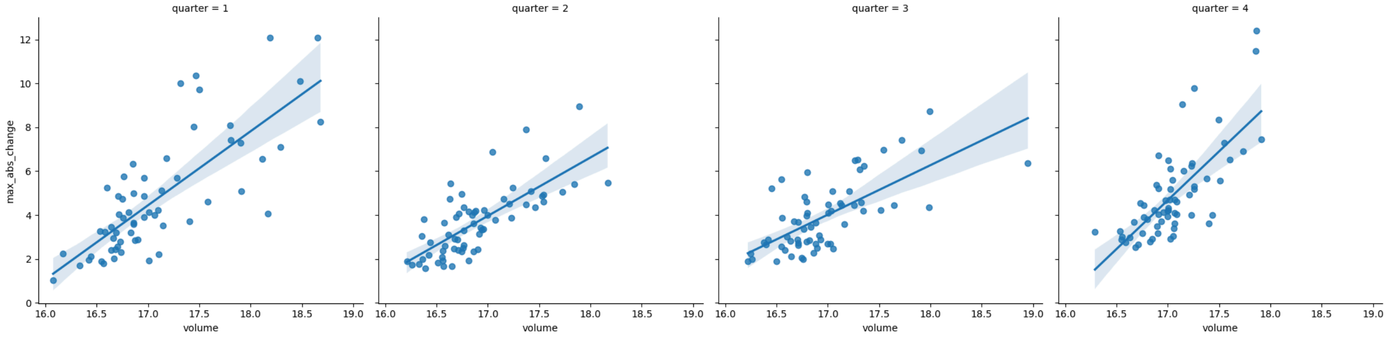
```
    ---------------------------------------------------------------------------
    ModuleNotFoundError                       Traceback (most recent call last)
    <ipython-input-16-ae2d095ec697> in <cell line: 1>()
    ----> 1 from reg_resid_plot import reg_resid_plots
          2 reg_resid_plots(fb_reg_data)

    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

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:

```
sns.lmplot( # Create a lmplot to visualize the relationship between log volume and maximum absolute change, with quarters as columns
    x='volume',
    y='max_abs_change',
    data=fb.assign( # Assign log volume, maximum absolute change, and quarter to fb DataFrame
        volume=np.log(fb.volume),
        max_abs_change=fb.high - fb.low,
        quarter=lambda x: x.index.quarter
        ),
    col='quarter'
)
```

```
    <seaborn.axisgrid.FacetGrid at 0x7bf611980700>
```



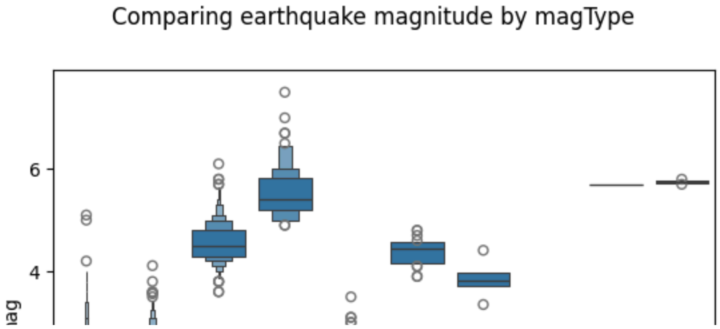
Distributions

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()

```
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')
```


Comparing earthquake magnitude by magType
ag