Submitted by: Angelo Luis C. Cu

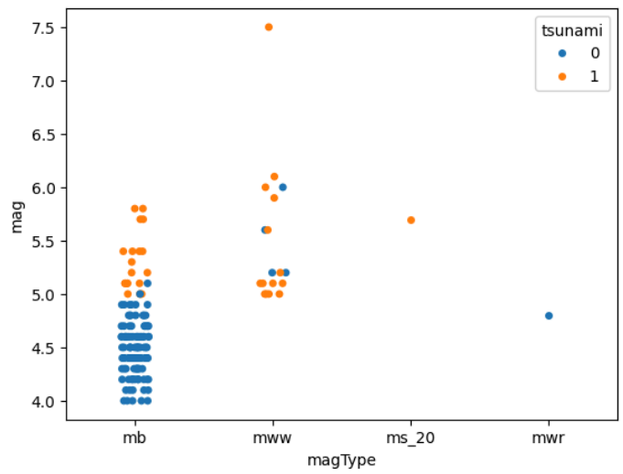
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
fb = pd.read_csv(
'data/fb_stock_prices_2018.csv', index_col='date', parse_dates=True
)
quakes = pd.read_csv('data/earthquakes.csv')
```

```
quakes.assign(
  time=lambda x: pd.to_datetime(x.time, unit='ms') # adds time
).set_index('time').loc['2018-09-28'].query( # gets the earthquake that happened in Indonesia
  "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

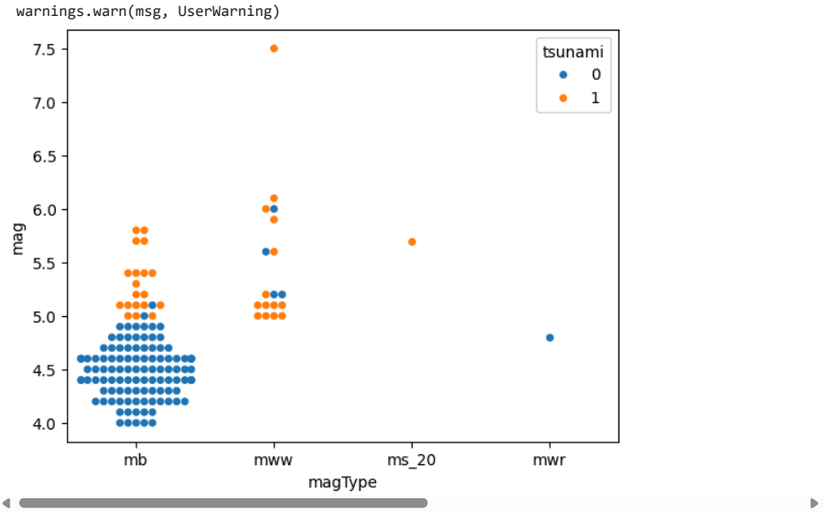
```
sns.stripplot( # creates a strip plot via Seaborn
  x='magType',
  y='mag',
  hue='tsunami',
  data=quakes.query('parsed_place == "Indonesia"') # gets all earthquakes recorded in Indonesia
) # data is hard to see as there are overlaps
```

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



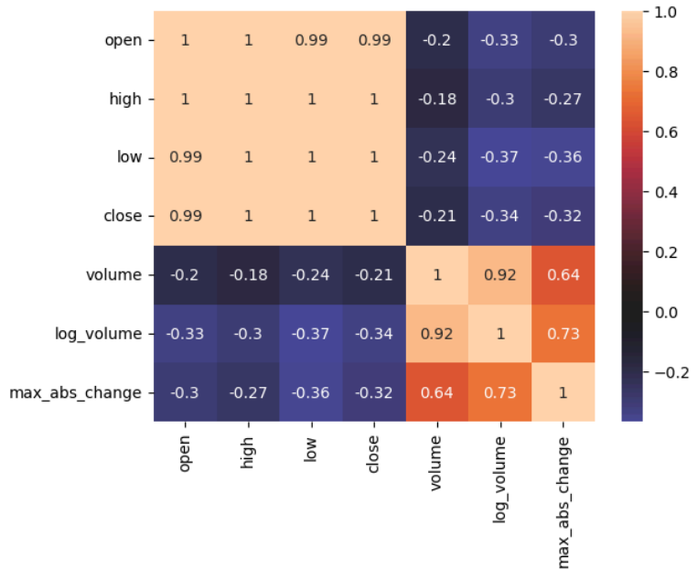

```
sns.swarmplot( # creates a swarm plot, which prevents overlap
  x='magType',
  y='mag',
  hue='tsunami',
  data=quakes.query('parsed_place == "Indonesia"')
)
```

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



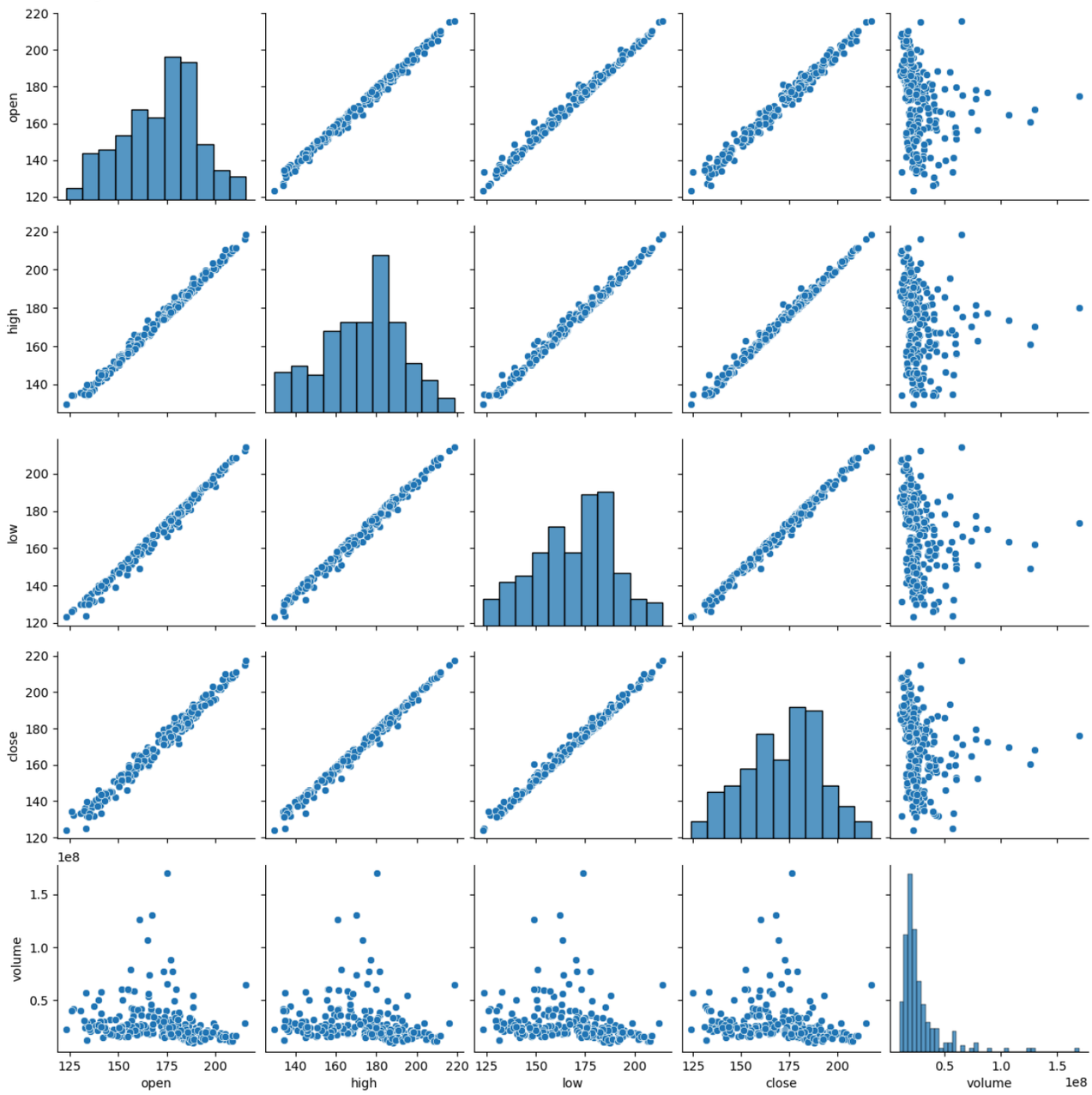

```
sns.heatmap( # creates a heatmap
  fb.sort_index().assign( # from facebook data
  log_volume=np.log(fb.volume), # gets the logarithmic volume
  max_abs_change=fb.high - fb.low # and the change between high and low
  ).corr(), # for correlation
  annot=True, center=0
)
```

```
<Axes: >
```



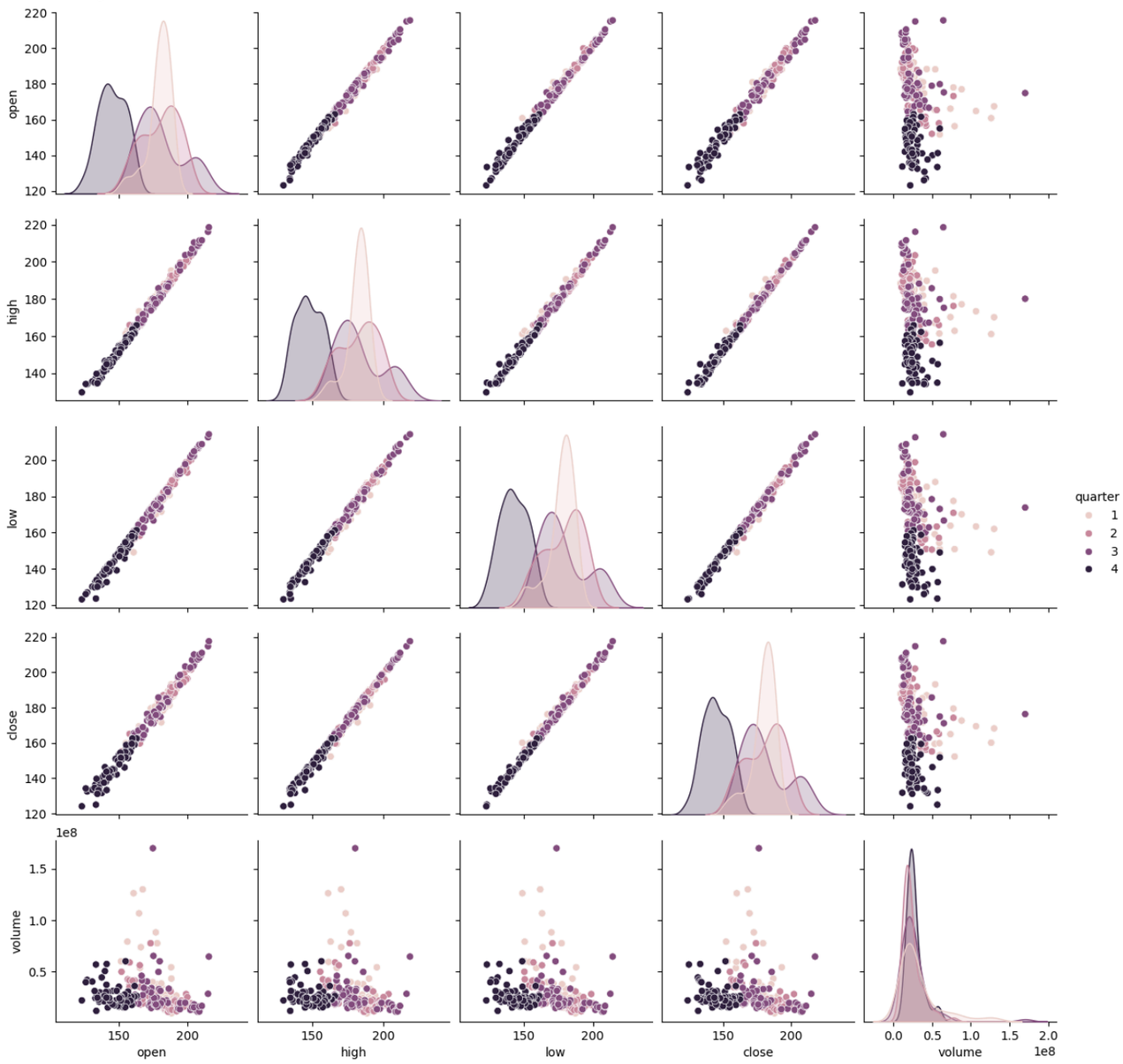

```
sns.pairplot(fb) # equivalent to pandas scatter matrix
```

```
<seaborn.axisgrid.PairGrid at 0x7de1d5759270>
```



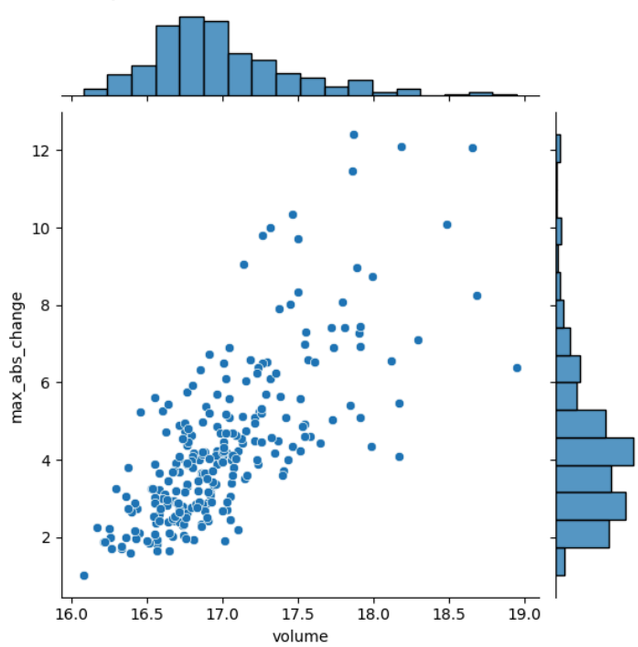

```
sns.pairplot(
  fb.assign(quarter=lambda x: x.index.quarter), # per quarter data
  diag_kind='kde', # with kernel density estimation
  hue='quarter'
)
```

```
<seaborn.axisgrid.PairGrid at 0x7de1d5734d00>
```



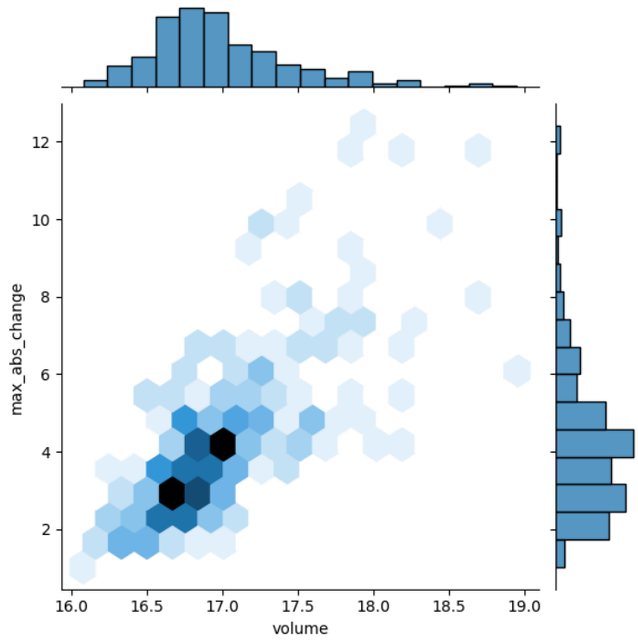

```
sns.jointplot( # creates a joint plot
  x='volume',
  y='max_abs_change',
  data=fb.assign( # gets logarithmic volume and high and low change
    volume=np.log(fb.volume),
    max_abs_change=fb.high - fb.low
  )
) # a scatter plot with the histogram or kde at the side (histogram is default)
```

```
<seaborn.axisgrid.JointGrid at 0x7de1d7a8faf0>
```



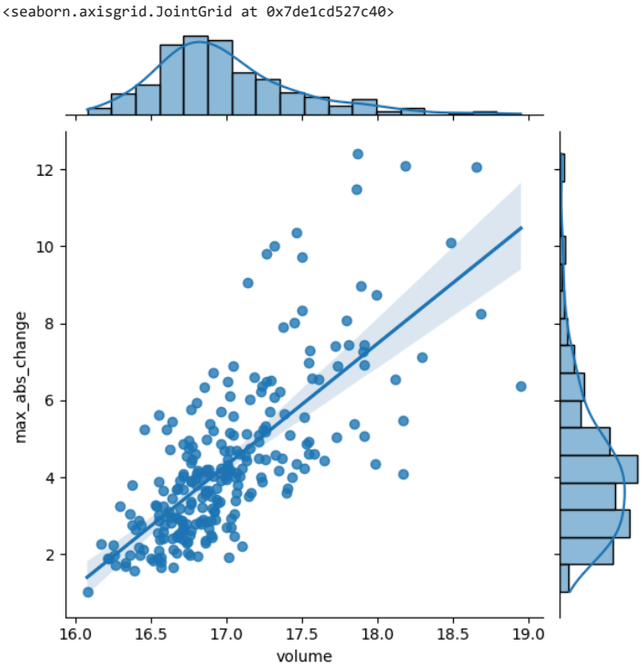
```
sns.jointplot(
  x='volume',
  y='max_abs_change',
  kind='hex', # hex plot instead of scatter plot
  data=fb.assign(
    volume=np.log(fb.volume),
    max_abs_change=fb.high - fb.low
  )
)
```

```
<seaborn.axisgrid.JointGrid at 0x7de1cd2ecaf0>
```



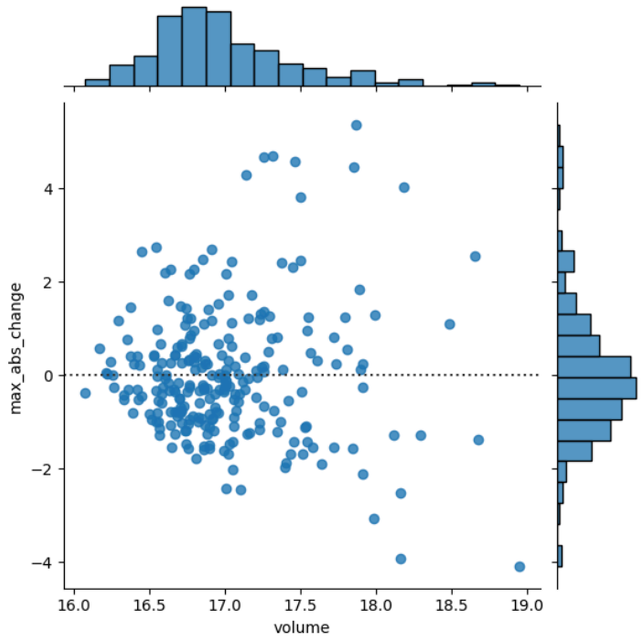

```
sns.jointplot(
  x='volume',
  y='max_abs_change',
  kind='reg', # regression line
  data=fb.assign(
    volume=np.log(fb.volume),
    max_abs_change=fb.high - fb.low
  )
)
```

```
<seaborn.axisgrid.JointGrid at 0x7de1cd527c40>
```



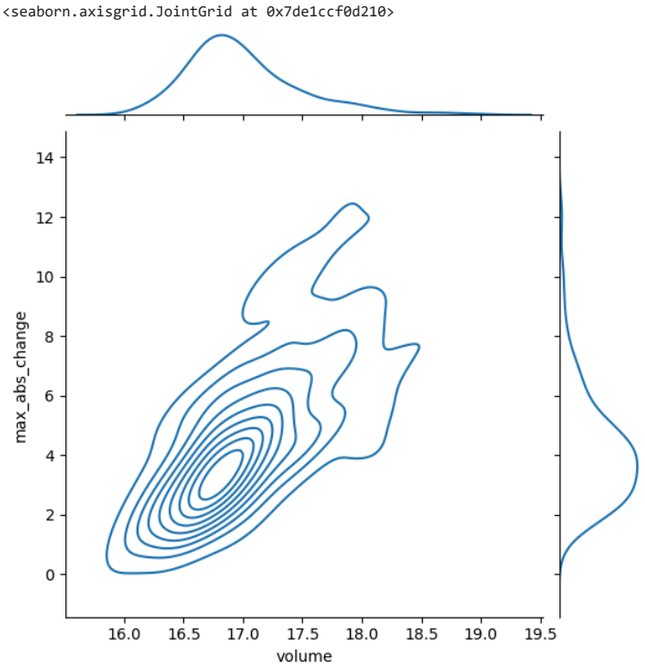

```
sns.jointplot(
  x='volume',
  y='max_abs_change',
  kind='resid', # residual plot from regression
  data=fb.assign(
    volume=np.log(fb.volume),
    max_abs_change=fb.high - fb.low
  )
)
```

```
<seaborn.axisgrid.JointGrid at 0x7de1cd0ebc40>
```




```
sns.jointplot(
  x='volume',
  y='max_abs_change',
  kind='kde', # contour plot with KDE in side instead of histogram
  data=fb.assign(
    volume=np.log(fb.volume),
    max_abs_change=fb.high - fb.low
  )
)
```

```
<seaborn.axisgrid.JointGrid at 0x7de1ccf0d210>
```




```
fb_reg_data = fb.assign( # creates a separate dataframe
  volume=np.log(fb.volume), # with only the volume and absolute change as data
  max_abs_change=fb.high - fb.low
).iloc[:,-2:]
```

```
import itertools # using itertools to get permutations (and combinations)
```

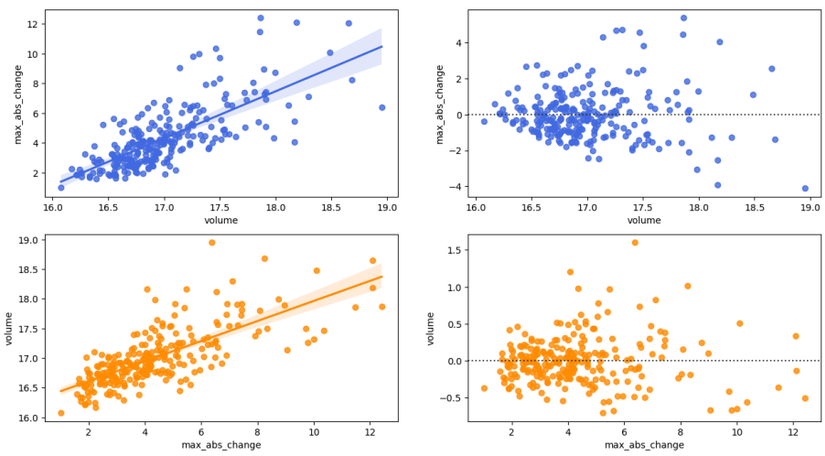
```
iterator = itertools.repeat("I'm an iterator", 1) # repeats the string only once
for i in iterator:
  print(f'-->{i}')
print('This printed once because the iterator has been exhausted')
for i in iterator: # doesn't print anymore as the iterator is done
  print(f'-->{i}')
```

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

```
iterable = list(itertools.repeat("I'm an iterable", 1)) # turns the iterator to a list
for i in iterable:
  print(f'-->{i}')
print('This prints again because it\'s an iterable:')
for i in iterable:
  print(f'-->{i}')
```

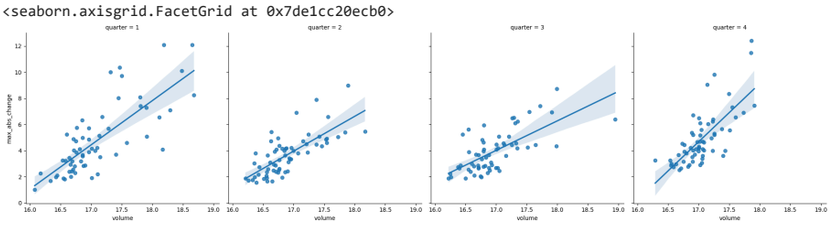
```
    -->I'm an iterable
    This prints again because it's an iterable:
    -->I'm an iterable
```

```
from reg_resid_plot import reg_resid_plots # upload the reg_resid_plot.py for this to run
reg_resid_plots(fb_reg_data) # result is the regression and residual per column of the dataframe
```



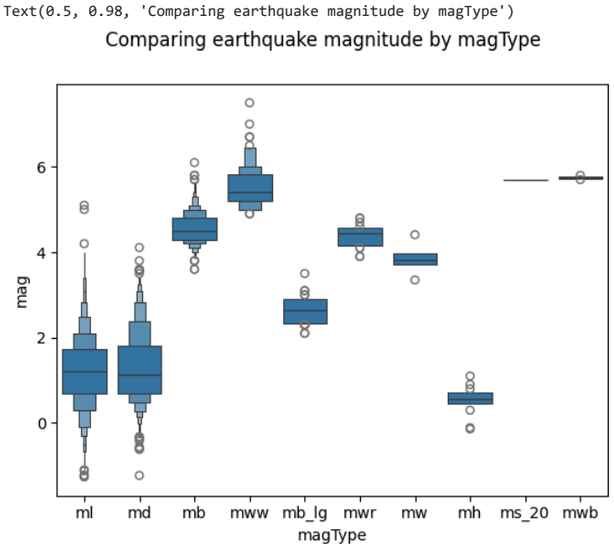

```
sns.lmplot( # regression plot but across subsets
  x='volume',
  y='max_abs_change',
  data=fb.assign(
    volume=np.log(fb.volume),
    max_abs_change=fb.high - fb.low,
    quarter=lambda x: x.index.quarter # creates a new column quarter
  ),
  col='quarter' # col specifies the subset to perfrom regression on
)
```

```
<seaborn.axisgrid.FacetGrid at 0x7de1cc20ecb0>
```




```
sns.boxenplot( # creates a box plot
  x='magType', y='mag', data=quakes[['magType', 'mag']]
) # but with additional quantiles
plt.suptitle('Comparing earthquake magnitude by magType')
```

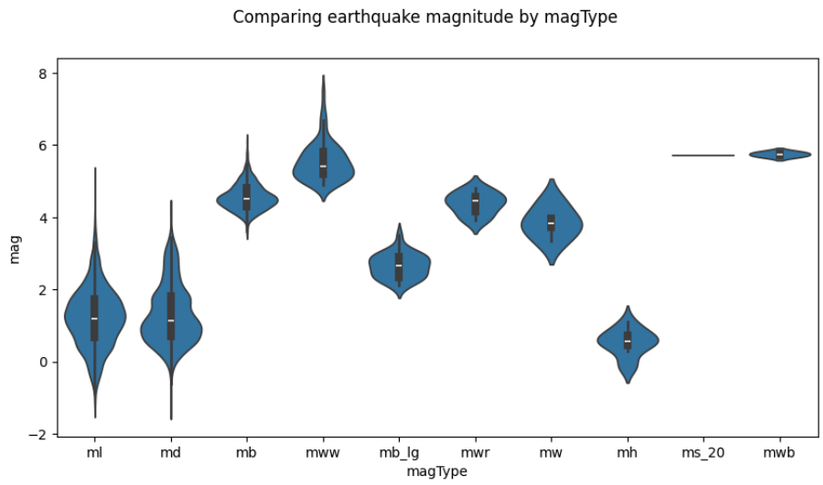
```
Text(0.5, 0.98, 'Comparing earthquake magnitude by magType')
```



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

```
<ipython-input-22-eefbbd076ecb>:2: FutureWarning:

The `scale` parameter has been renamed and will be removed in v0.15.0. Pass `density_norm='width'
  sns.violinplot( # creates a violin plot
Text(0.5, 0.98, 'Comparing earthquake magnitude by magType')
```




```
g = sns.FacetGrid( # we can also create subplots across subsets by faceting
                  # which specifies the layout of the subplots
  quakes[
    (quakes.parsed_place.isin([ # gets data from the locations below
      'California', 'Alaska', 'Hawaii'
      ]))\
      & (quakes.magType.isin(['ml', 'md'])) # and only these magType
  ],
  row='magType',
  col='parsed_place'
)
g = g.map(plt.hist, 'mag') # creates a histogram for the facet grid
```