Introduction to Seaborn

Setup

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

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

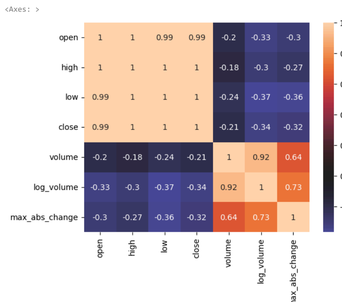
time	mag	magType	place	tsunami	parsed_place
2018-09-28 10:02:43.480	7.5	mww	78km N of Palu, Indonesia	1	Indonesia

stripplot()

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( # stripping plots
  x='magType',
  y='mag',
  hue='tsunami',
  data=quakes.query('parsed_place == "Indonesia"')
)
```

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

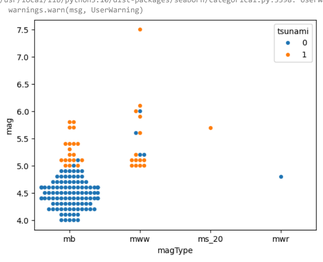


swarmplot()

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( # bee swarm plotting
  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)
```



Correlations and Heatmaps

heatmap()

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

```
sns.heatmap( # heatmapping when the color coding seems to rely on hot and cold
  fb.sort_index().assign(
  log_volume=np.log(fb.volume),
  max_abs_change=fb.high - fb.low
).corr(),
  annot=True, center=0
)
```

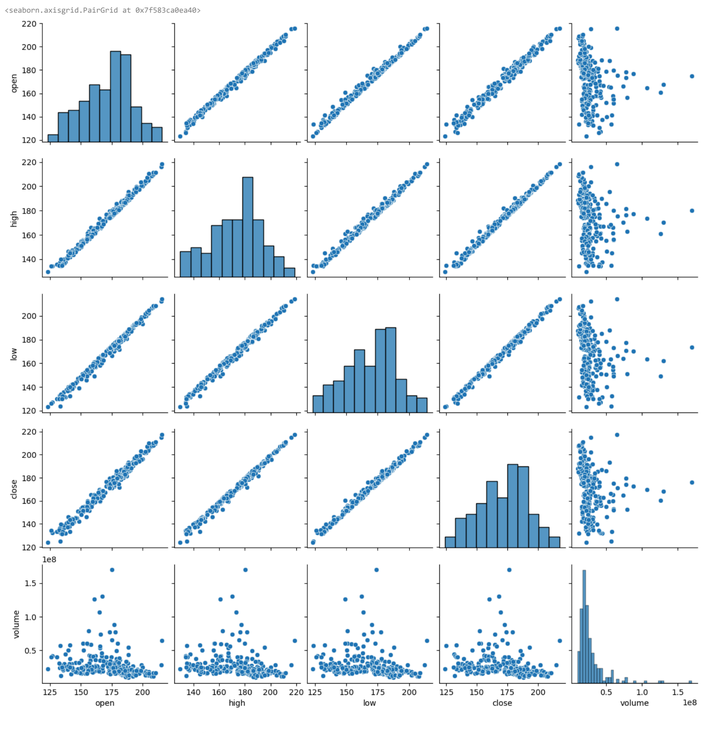
```
<Axes: >
```


pairplot()

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

```
sns.pairplot(fb) # pairplot() is the scatter matrix of seaborn
```

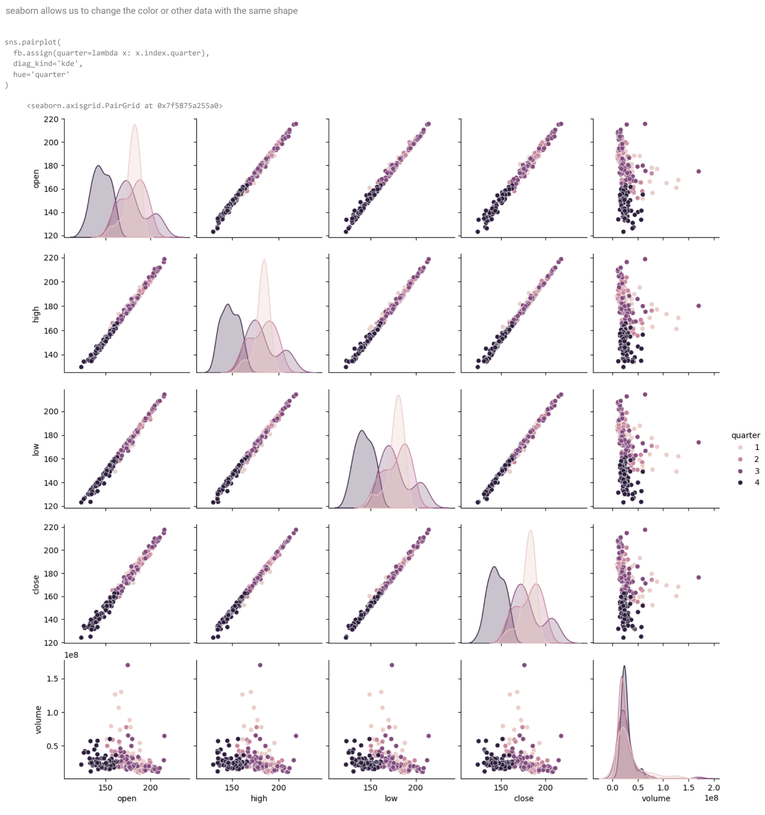
```
<seaborn.axisgrid.PairGrid at 0x7f583ca0ea40>
```



seaborn allows us to change the color 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 0x7f5875a255a0>
```

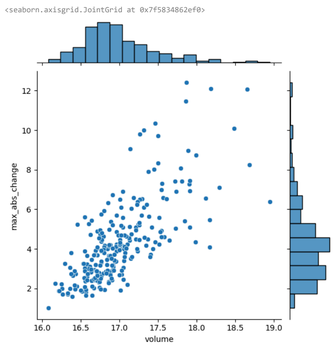


jointplot()

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 0x7f5834862ef0>
```



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:

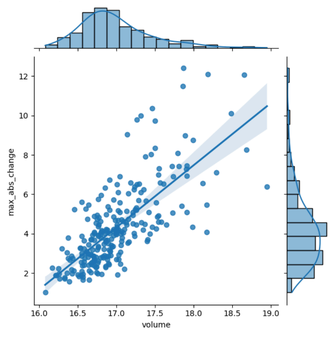
```
sns.jointplot(
  x='volume',
  y='max_abs_change',
  kind='hex',
  data=fb.assign(
  volume=np.log(fb.volume),
  max_abs_change=fb.high - fb.low
  )
)
```

```
<seaborn.axisgrid.JointGrid at 0x7f5835090c40>
```


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
  )
)
```

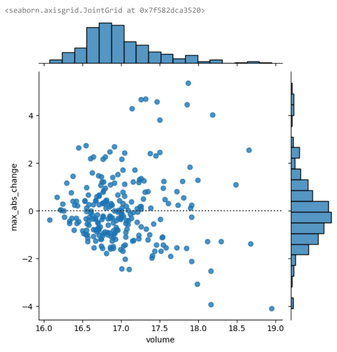
```
<seaborn.axisgrid.JointGrid at 0x7f582df1ac80>
```



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
  )
)
```

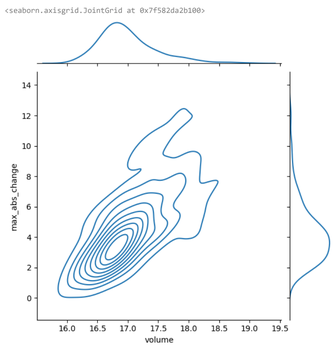
```
<seaborn.axisgrid.JointGrid at 0x7f582dca3520>
```



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 0x7f582da2b100>
```



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:]
```

```
import itertools # to use permutation and combination
```

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
```

```
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)
```

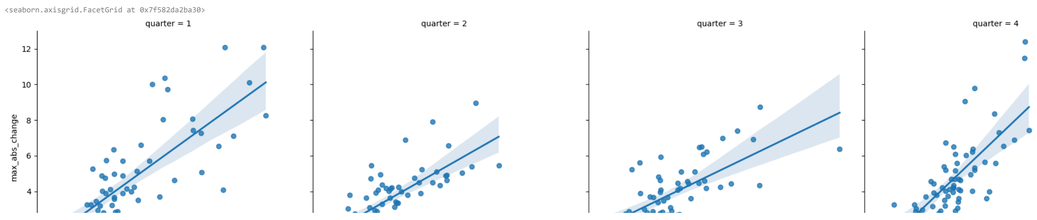
```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-27-ae2d095ec697> in <cell line: 1>()
----> 1 from reg_resid_plot import reg_resid_plots
      2 reg_resid_plots(fb_reg_data)

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(
  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 0x7f582da2ba30>
```



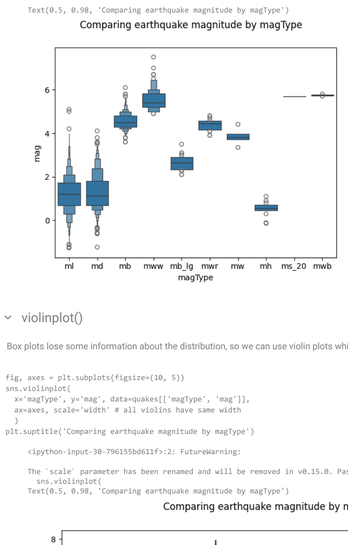
Distributions

boxenplot()

box plot that shows additional quantiles

```
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')
```

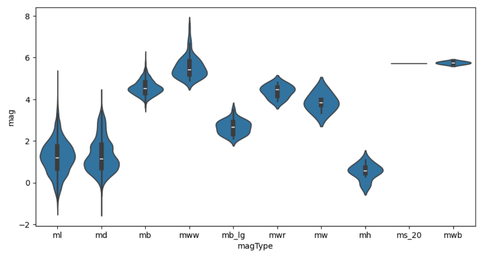


violinplot()

Box plots lose some information about the distribution, so we can use violin plots which combine box plots and KDEs:

```
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-30-796155bd611f>: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')
```



Faceting

We can create subplots across subsets of our data by faceting. First, we create a rows and which one along the columns). Then, we call the FacetGrid specifying how to layout the plots (which categorical column goes along the map() method of the FacetGrid and pass in the plotting function we want to use (along with any additional arguments).

histograms showing the distribution of earthquake magnitude in California, Alaska, and Hawaii faceted by magType and parse_placed:

```
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')
```