3/31/24, 4:24 AM Hands-on Activity 9.4 Introduction to Seaborn.ipynb - Colaboratory

## 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"

mag magType place tsunami parsed\_place 🚃 **2018-09-28 10:02:43.480** 7.5 mww 78km N of Palu, Indonesia

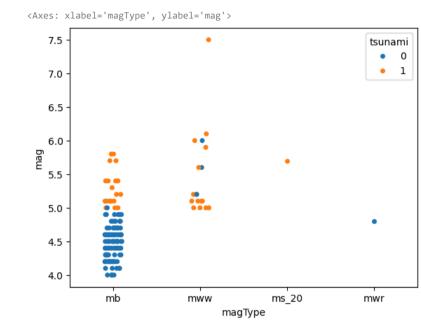
#### stripplot()

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( x='magType', y='mag',

data=quakes.query('parsed\_place == "Indonesia"')



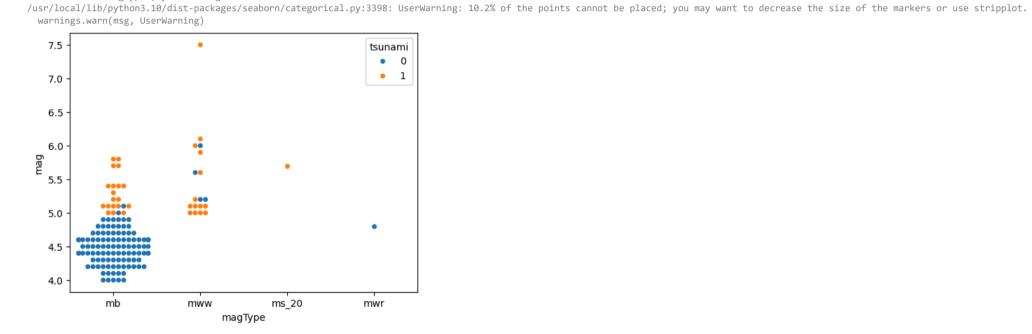
## swarmplot()

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( x='magType',

y='mag', hue='tsunami', data=quakes.query('parsed\_place == "Indonesia"')

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

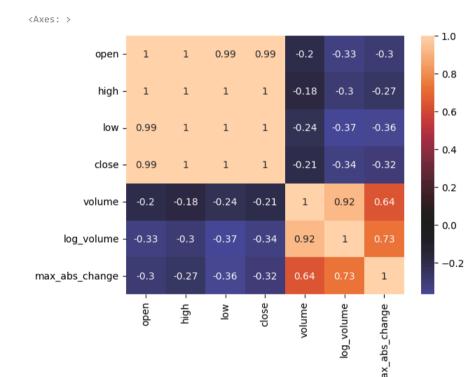


## Correlations and Heatmaps

#### heatmap()

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

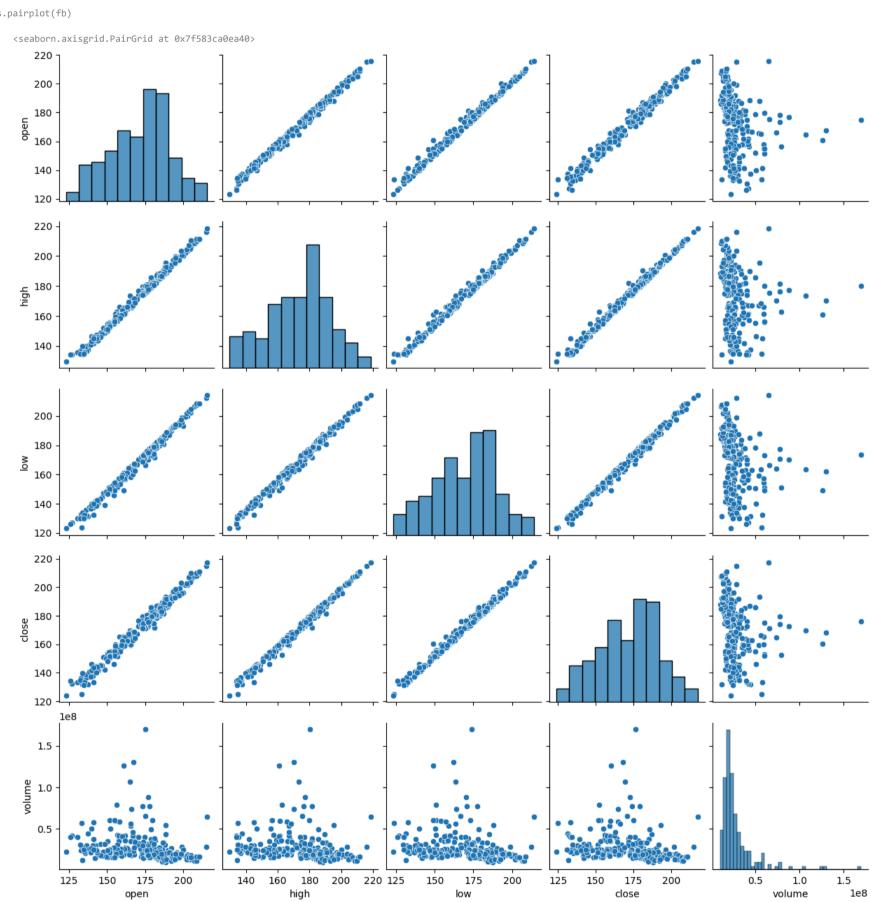
sns.heatmap( fb.sort\_index().assign(
log\_volume=np.log(fb.volume), max\_abs\_change=fb.high - fb.low ).corr(), annot=True, center=0



## v pairplot()

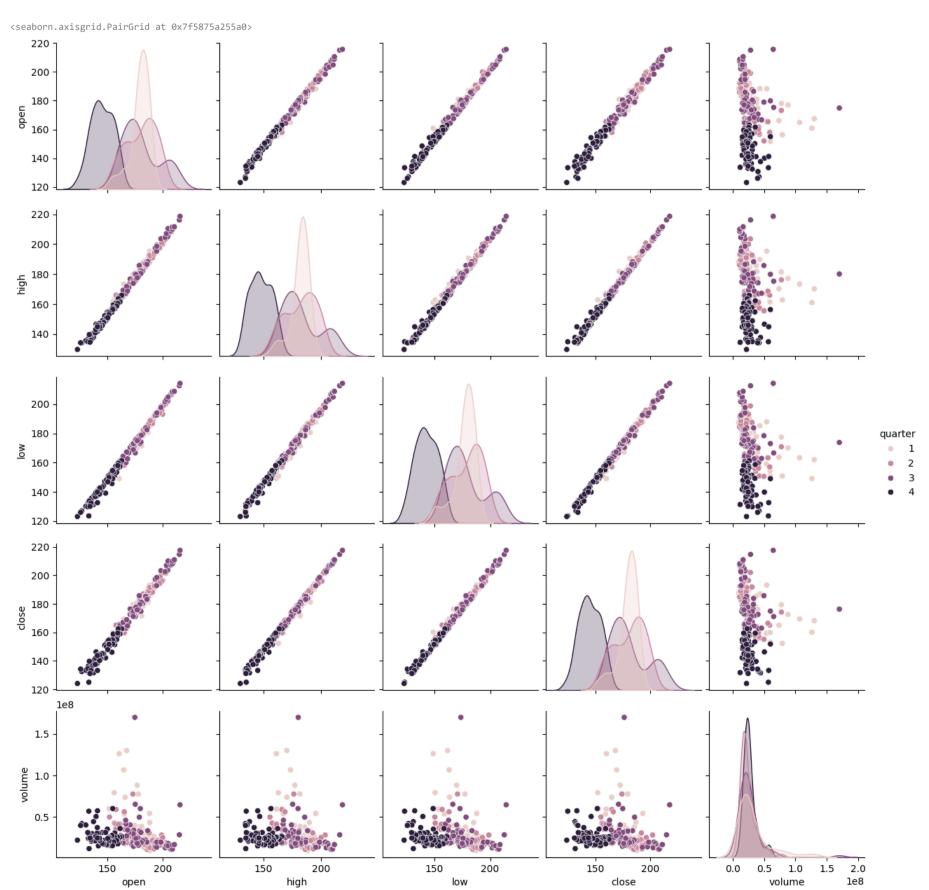
The pair plot is seaborn's answer to the scatter matrix we saw in the pandas subplotting notebook

## sns.pairplot(fb)



## 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'



## 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

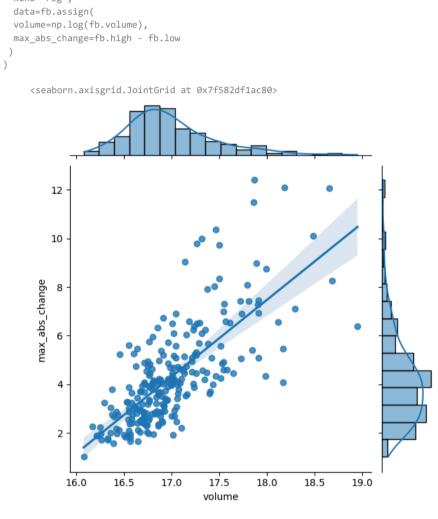
on the sides: https://colab.research.google.com/drive/1xu8YK1vIaSJzesFrikGB2ORnh6PzFero#scrollTo=Wnw72U8T\_xfG&printMode=true

```
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>
          16.0 16.5 17.0 17.5 18.0 18.5 19.0
                                 volume
```

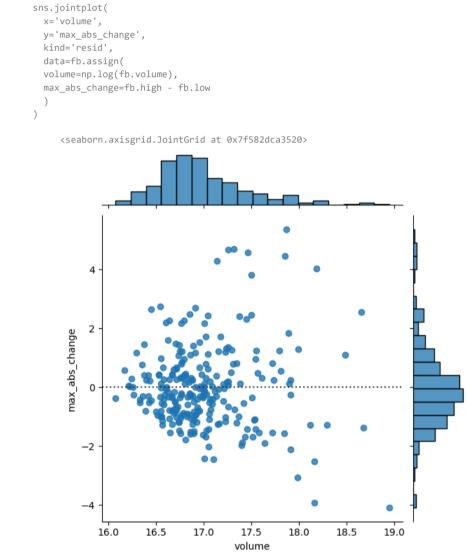
#### By changing the kind argument, we can change how the center of the plot is displayed. For example, we can pass kind='hex' for hexbins:

```
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>
           16.0 16.5 17.0 17.5 18.0 18.5 19.0
If we specify kind='reg' instead, we get a regression line in the center and KDEs on the sides:
```

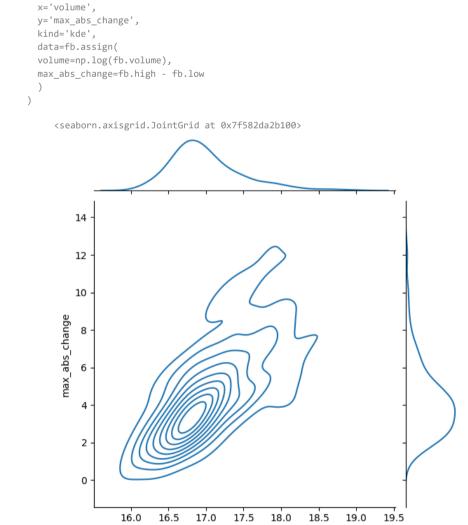
sns.jointplot( x='volume', y='max\_abs\_change', kind='reg',



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



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



volume

## Regression plots

sns.jointplot(

#### 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:')

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

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

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) 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.

We can use Implot() 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'

OPEN EXAMPLES

2/3

3/31/24, 4:24 AM Hands-on Activity 9.4 Introduction to Seaborn.ipynb - Colaboratory

quarter = 3

quarter = 4

# <seaborn.axisgrid.FacetGrid at 0x7f582da2ba30> quarter = 1

quarter = 2

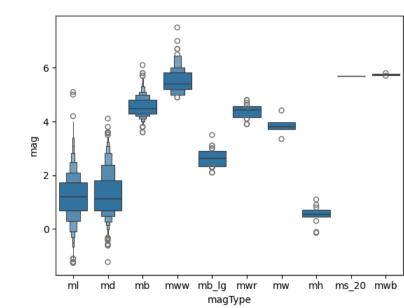
#### **Distributions**

### v boxenplot()

box plot that shows additional quantiles

sns.boxenplot( x='magType', y='mag', data=quakes[['magType', 'mag']]

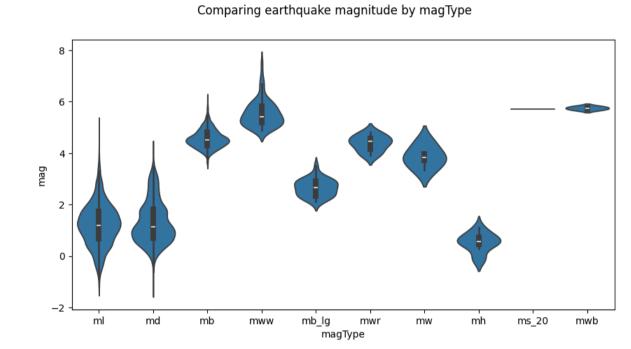
plt.suptitle('Comparing earthquake magnitude by magType') Text(0.5, 0.98, 'Comparing earthquake magnitude by magType') Comparing earthquake magnitude by magType



#### 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:

