Plotting with Pandas

The plot() method is available on Series and DataFrame objects. Many of the parameters get passed down to matplotlib. The kind argument let's us vary the plot type

Setup

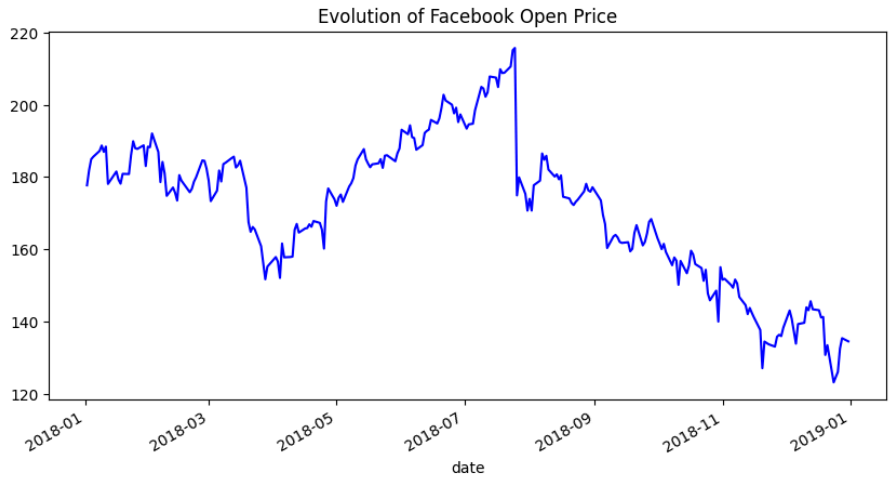
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
fb = pd.read_csv(
    '/content/fb_stock_prices_2018.csv', index_col='date', parse_dates=True
    )
quakes = pd.read_csv('/content/earthquakes.csv')
```

Evolution over time

Line plots help us see how a variable changes over time. They are the default for the kind argument, but we can pass kind='line' to be explicit in our intent:

```
fb.plot(
    kind='line',  # Type
    y='open',  # Column for y-axis data
    figsize=(10, 5),  # Size of the plot
    style='b-',  # Line style and color
    legend=False,  # Disabling legend
    title='Evolution of Facebook Open Price'  # Title
)
```

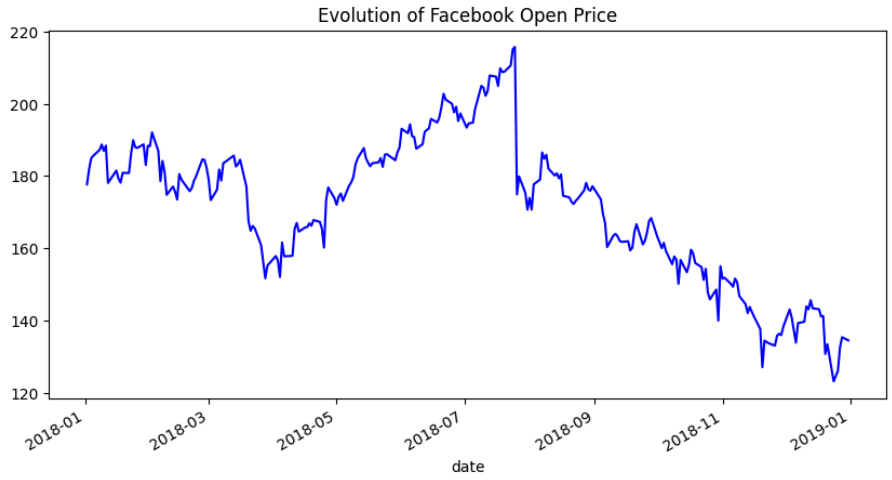
```
<Axes: title={'center': 'Evolution of Facebook Open Price'}, xlabel='date'>
```



We provided the style argument in the previous example; however, we can use the color and linestyle arguments to get the same result:

```
fb.plot(
    kind='line',  # Type
    y='open',  # Column for y-axis data
    figsize=(10, 5),  # Size of the plot
    color='blue',  # Line color
    linestyle='solid',  # Line style
    legend=False,  # Disabling legend
    title='Evolution of Facebook Open Price'  # Title
)
```

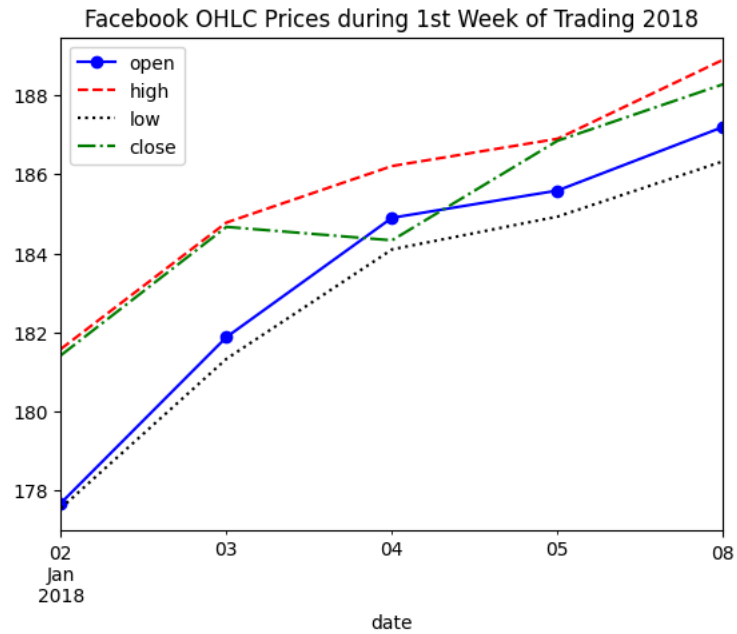
```
<Axes: title={'center': 'Evolution of Facebook Open Price'}, xlabel='date'>
```



We can also plot many lines at once by simply passing a list of the columns to plot:

```
fb.iloc[:5,].plot(
    y=['open', 'high', 'low', 'close'], # Columns for y-axis data
    style=['b-o', 'r--', 'k:', 'g-.'], # Line styles for each column
    title='Facebook OHLC Prices during 1st Week of Trading 2018' # Title
    )
```

```
<Axes: title={'center': 'Facebook OHLC Prices during 1st Week of Trading 2018'},
xlabel='date'>
```



Creating subplots

When plotting with pandas, creating subplots is simply a matter of passing (rows, columns):

```
fb.plot(
    kind='line',  # Type
    subplots=True,  # Creating subplots
    layout=(3, 2),  # Layout of subplots (3 rows, 2 columns)
    figsize=(15, 10),  # Size of the overall plot
    title='Facebook Stock 2018'  # Title
)
```

```
array([[<Axes: xlabel='date'>, <Axes: xlabel='date'>],
       [<Axes: xlabel='date'>, <Axes: xlabel='date'>],
       [<Axes: xlabel='date'>, <Axes: xlabel='date'>]], dtype=object)
```

Note that we didn't provide a specific column to plot and pandas plotted all of them for us

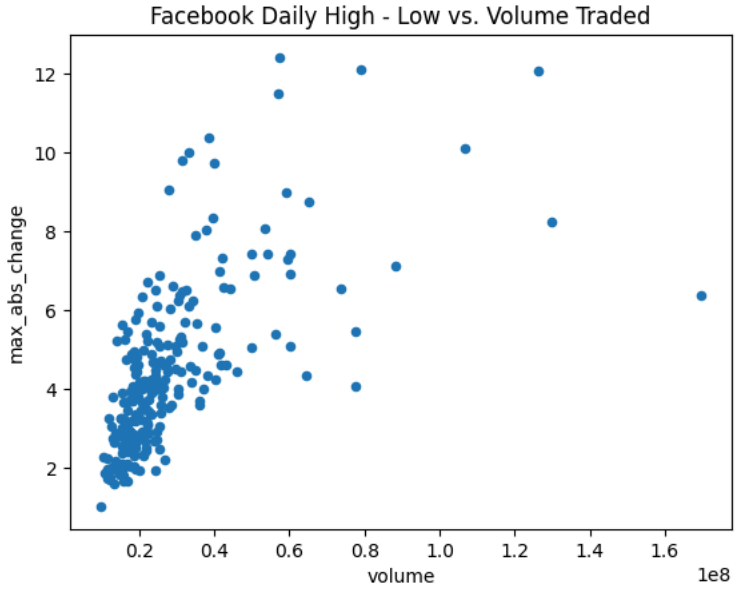
Visualizing relationships between variables

Scatter plots

We make scatter plots to help visualize the relationship between two variables. Creating scatter plots requires we pass in axis and a column for the y-axis:

```
fb.assign( # Calculating maximum absolute change and plotting it against volume traded
    max_abs_change=fb.high - fb.low
    ).plot(
        kind='scatter', x='volume', y='max_abs_change', # Data for x-axis, y-axis
        title='Facebook Daily High - Low vs. Volume Traded' # Title
        )
```

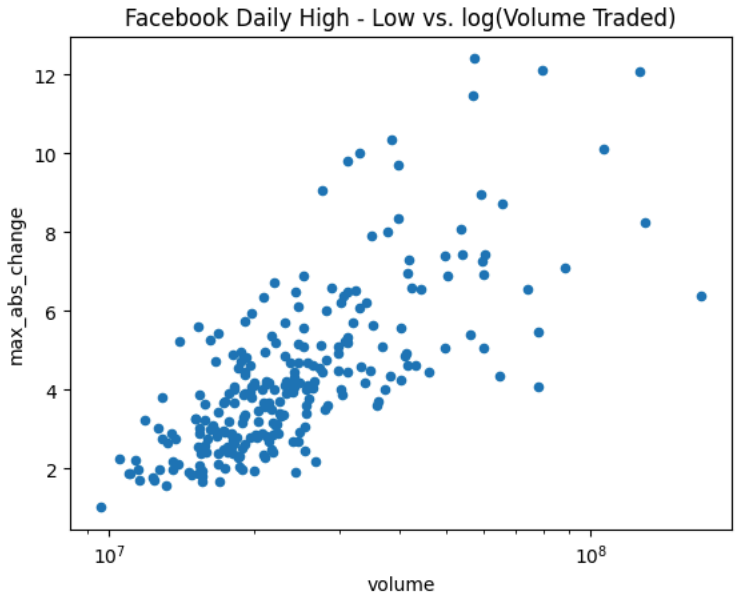
```
<Axes: title={'center': 'Facebook Daily High - Low vs. Volume Traded'},
xlabel='volume', ylabel='max_abs_change'>
```



The relationship doesn't seem to be linear, but we can try a log transform on the x-axis since the scales of the axes are very different. With pandas, we simply pass in logx=True:

```
fb.assign(
    max_abs_change=fb.high - fb.low
    ).plot(
        kind='scatter', x='volume', y='max_abs_change',
        title='Facebook Daily High - Low vs. log(Volume Traded)',
        logx=True # Using logarithmic scale for x-axis
        )
```

```
<Axes: title={'center': 'Facebook Daily High - Low vs. log(Volume Traded)'},
xlabel='volume', ylabel='max_abs_change'>
```



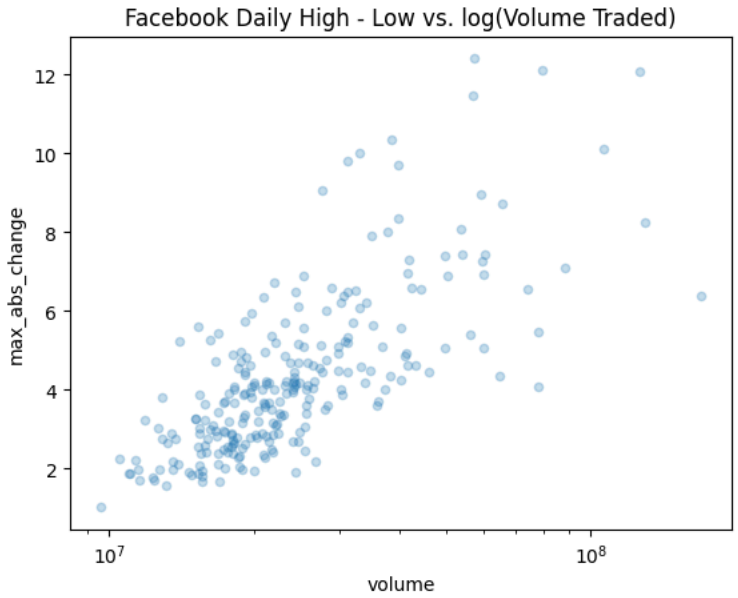
With matplotlib, we could use plt.xscale('log') to do the same thing

Adding Transparency to Plots with alpha

Sometimes our plots have many overlapping values, but this can be impossible to see. This can be addressed by increasing the transparency of what we are plotting using the alpha parameter. It is a float on [0, 1] where 0 is completely transparent and 1 is completely opaque. By default this is 1, so let's put in a lower value and re-plot the scatter plot:

```
fb.assign(
    max_abs_change=fb.high - fb.low
    ).plot(
        kind='scatter', x='volume', y='max_abs_change',
        title='Facebook Daily High - Low vs. log(Volume Traded)',
        logx=True, alpha=0.25 # Adjusting transparency of data points
 )
```

```
<Axes: title={'center': 'Facebook Daily High - Low vs. log(Volume Traded)'},
xlabel='volume', ylabel='max_abs_change'>
```

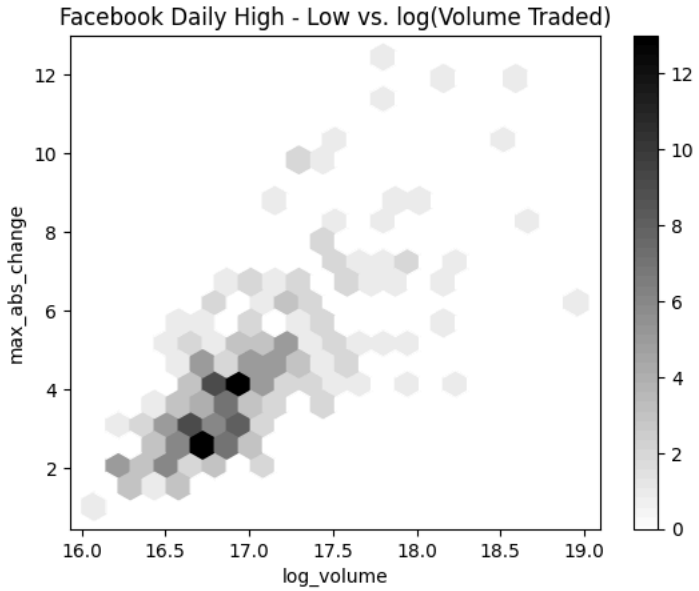


Hexbins

In the previous example, we can start to see the overlaps, but it is still difficult. Hexbins are another plot type that divide up the plot into hexagons, which are shaded according to the density of points there. With pandas, this is the hexbin value for the kind argument. It can also be important to tweak the gridsize , which determines the number of hexagons along the y-axis:

```
fb.assign(
    log_volume=np.log(fb.volume),
    max_abs_change=fb.high - fb.low
    ).plot(
        kind='hexbin',
        x='log_volume',
        y='max_abs_change',
        title='Facebook Daily High - Low vs. log(Volume Traded)',
        colormap='gray_r',
        gridsize=20,
        sharex=False # we have to pass this to see the x-axis due to a bug in this version of pandas
        )
```

```
<Axes: title={'center': 'Facebook Daily High - Low vs. log(Volume Traded)'},
xlabel='log_volume', ylabel='max_abs_change'>
```

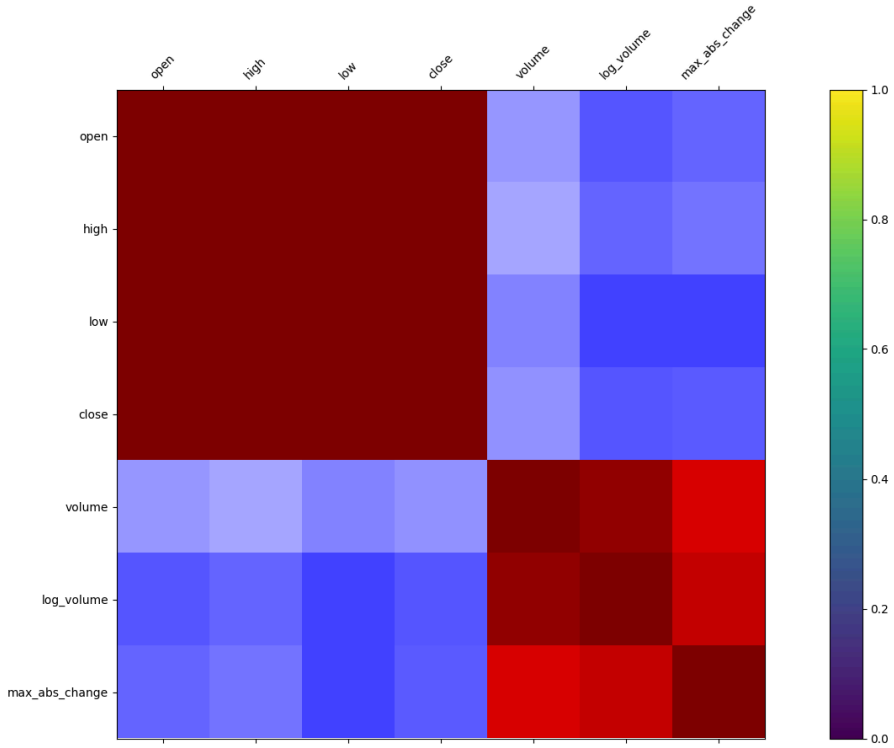


Visualizing Correlations with Heatmaps

Pandas doesn't offer heatmaps; however, if we are able to get our data into a matrix, we can use from matplotlib:

```
fig, ax = plt.subplots(figsize=(20, 10))
fb_corr = fb.assign(
    log_volume=np.log(fb.volume),
    max_abs_change=fb.high - fb.low
).corr()
im = ax.matshow(fb_corr, cmap='seismic')
fig.colorbar(im.set_clim(-1, 1))
labels = [col.lower() for col in fb_corr.columns]
ax.set_xticklabels([''] + labels, rotation=45)
ax.set_yticklabels([''] + labels)
```

```
<ipython-input-10-06bb5e4493b6>:7: MatplotlibDeprecationWarning: Unable to determine Axe
  fig.colorbar(im.set_clim(-1, 1))
<ipython-input-10-06bb5e4493b6>:9: UserWarning: FixedFormatter should only be used toget
  ax.set_xticklabels([''] + labels, rotation=45)
<ipython-input-10-06bb5e4493b6>:10: UserWarning: FixedFormatter should only be used toge
  ax.set_yticklabels([''] + labels)
[Text(0, -1.0, ''),
 Text(0, 0.0, 'open'),
 Text(0, 1.0, 'high'),
 Text(0, 2.0, 'low'),
 Text(0, 3.0, 'close'),
 Text(0, 4.0, 'volume'),
 Text(0, 5.0, 'log_volume'),
 Text(0, 6.0, 'max_abs_change'),
 Text(0, 7.0, '')]
```




```
fb_corr.loc['max_abs_change', ['volume', 'log_volume']]
```

```
volume        0.642027
log_volume    0.731542
Name: max_abs_change, dtype: float64
```

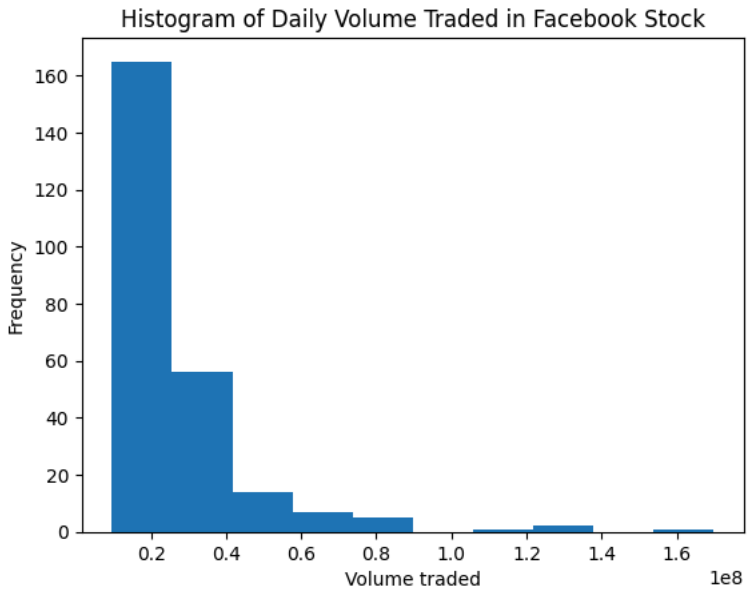
Visualizing distributions

Histograms

With the pandas plot() method, making histograms is as easy as passing in kind='hist':

```
fb.volume.plot(
    kind='hist',
    title='Histogram of Daily Volume Traded in Facebook Stock'
    )
plt.xlabel('Volume traded') # label the x-axis (discussed in chapter 6)
```

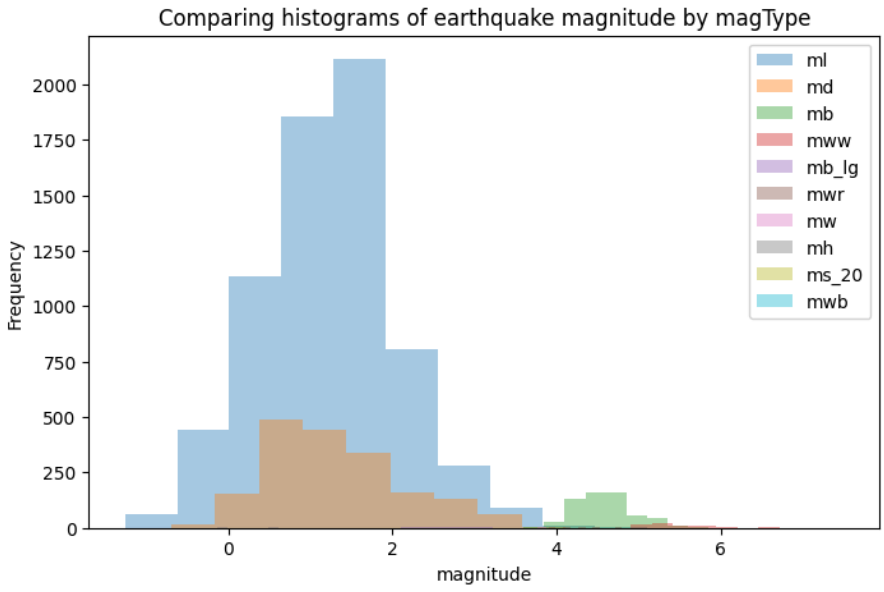
```
Text(0.5, 0, 'Volume traded')
```



We can overlap histograms to compare distributions provided we use the alpha parameter. For example, let's compare the usage and magnitude of the various magTypes in the data:

```
fig, axes = plt.subplots(figsize=(8, 5))
for magtype in quakes.magType.unique():
  data = quakes.query(f'magType == "{magtype}"').mag
  if not data.empty:
    data.plot(
        kind='hist', ax=axes, alpha=0.4,
        label=magtype, legend=True,
        title='Comparing histograms of earthquake magnitude by magType'
        )
plt.xlabel('magnitude') # label the x-axis (discussed in chapter 6)
```

```
Text(0.5, 0, 'magnitude')
```

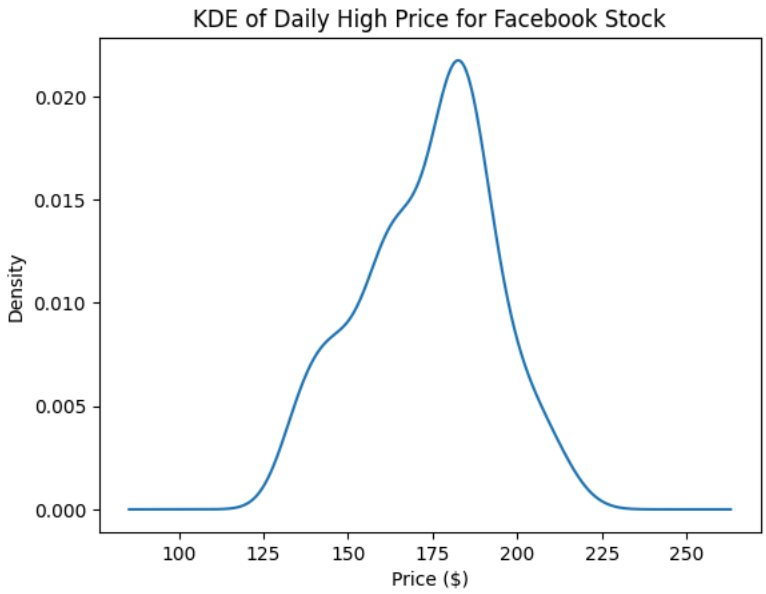


Kernel Density Estimation (KDE)

We can pass kind='kde' for a probability density function (PDF), which tells us the probability of getting a particular value:

```
fb.high.plot(
    kind='kde',
    title='KDE of Daily High Price for Facebook Stock'
    )
plt.xlabel('Price ($)') # label the x-axis (discussed in chapter 6)
```

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

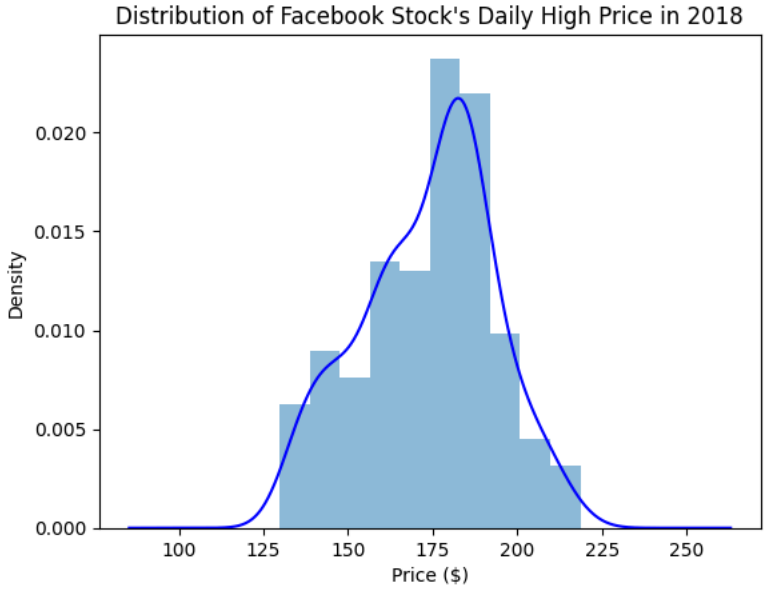


Adding to the result of plot()

The plot() method returns a matplotlib Axes object. We can store this for additional customization of the plot, or we can pass this into another call to ax argument to add to the original plot. It can often be helpful to view the KDE superimposed on top of the histogram, which can be achieved with this strategy:

```
ax = fb.high.plot(kind='hist', density=True, alpha=0.5)
fb.high.plot(
    ax=ax, kind='kde', color='blue',
    title='Distribution of Facebook Stock\'s Daily High Price in 2018'
)
plt.xlabel('Price ($)') # label the x-axis (discussed in chapter 6)
```

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

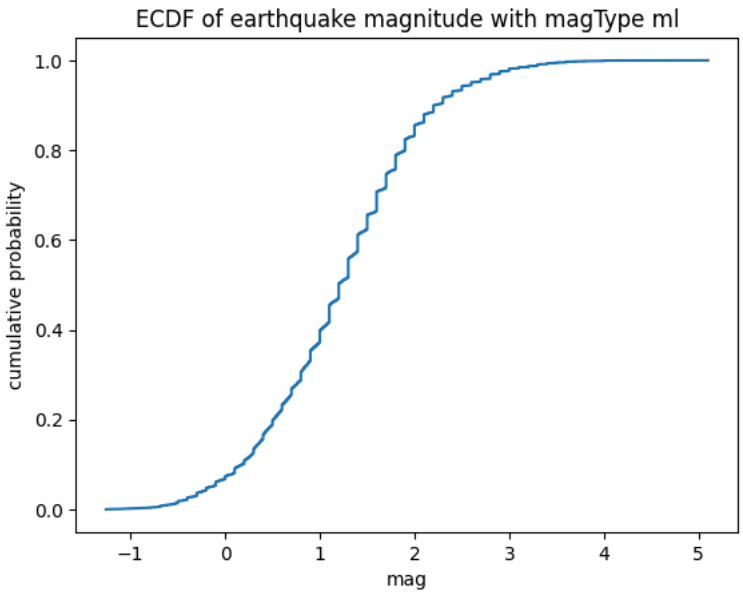


Plotting the ECDF

In some cases, we are more interested in the probability of getting less than or equal to that value (or greater than or equal), which we can see with the cumulative disribution function (CDF). Using the statsmodels package, we can estimate the CDF giving us the empirical cumulative distribution function (ECDF):

```
from statsmodels.distributions.empirical_distribution import ECDF
ecdf = ECDF(quakes.query('magType == "ml"').mag)
plt.plot(ecdf.x, ecdf.y)
# axis labels (we will cover this in chapter 6)
plt.xlabel('mag') # add x-axis label
plt.ylabel('cumulative probability') # add y-axis label
# add title (we will cover this in chapter 6)
plt.title('ECDF of earthquake magnitude with magType ml')
```

```
Text(0.5, 1.0, 'ECDF of earthquake magnitude with magType ml')
```

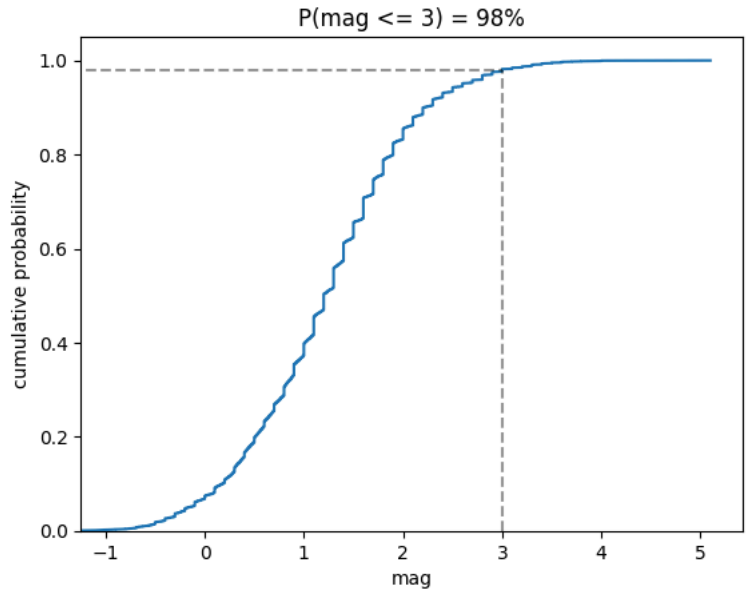


This ECDF tells us the probability of getting an earthquake with magnitude of 3 or less using the ml scale is 98%:

```
from statsmodels.distributions.empirical_distribution import ECDF
ecdf = ECDF(quakes.query('magType == "ml"').mag)
plt.plot(ecdf.x, ecdf.y)
# formatting below will all be covered in chapter 6
# axis labels
plt.xlabel('mag') # add x-axis label
plt.ylabel('cumulative probability') # add y-axis label
# add reference lines for interpreting the ECDF for mag <= 3
plt.plot(
    [3, 3], [0, .98], 'k--',
    [-1.5, 3], [0.98, 0.98], 'k--', alpha=0.4
)
# set axis ranges
plt.ylim(0, None)
plt.xlim(-1.25, None)
# add a title

plt.title('P(mag <= 3) = 98%')
```

```
Text(0.5, 1.0, 'P(mag <= 3) = 98%')
```

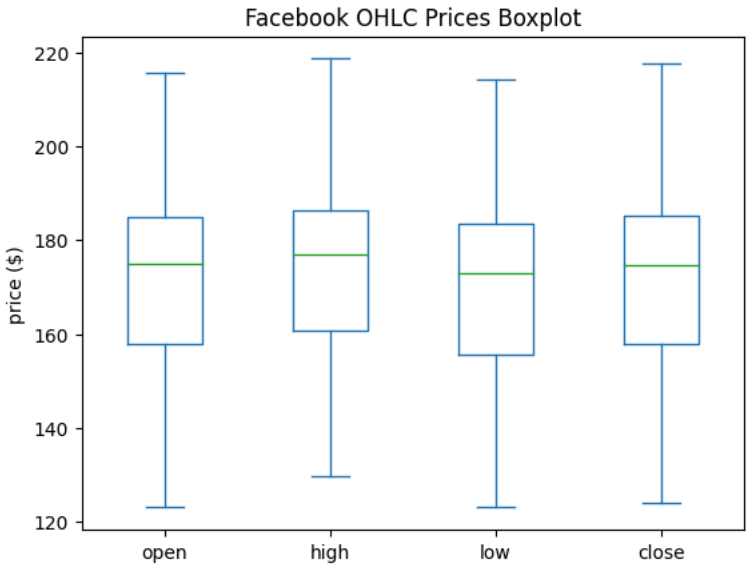


Box plots

To make box plots with pandas, we pass kind='box' to the plot() method:

```
fb.iloc[:,:4].plot(kind='box', title='Facebook OHLC Prices Boxplot')
plt.ylabel('price ($)') # label the x-axis (discussed in chapter 6)
```

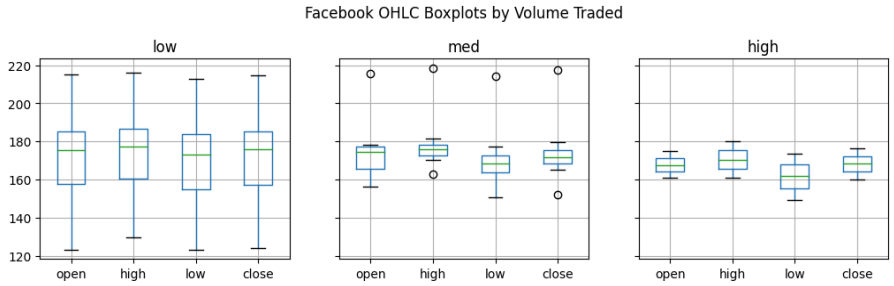
```
Text(0, 0.5, 'price ($)')
```



This can also be combined with a groupby():

```
fb.assign(
    volume_bin=pd.cut(fb.volume, 3, labels=['low', 'med', 'high'])
    ).groupby('volume_bin').boxplot(
        column=['open', 'high', 'low', 'close'], # Columns for boxplot
        layout=(1, 3), figsize=(12, 3)# Layout of subplots and size of the overall plot
        )
plt.suptitle('Facebook OHLC Boxplots by Volume Traded', y=1.1) # Adding title
```

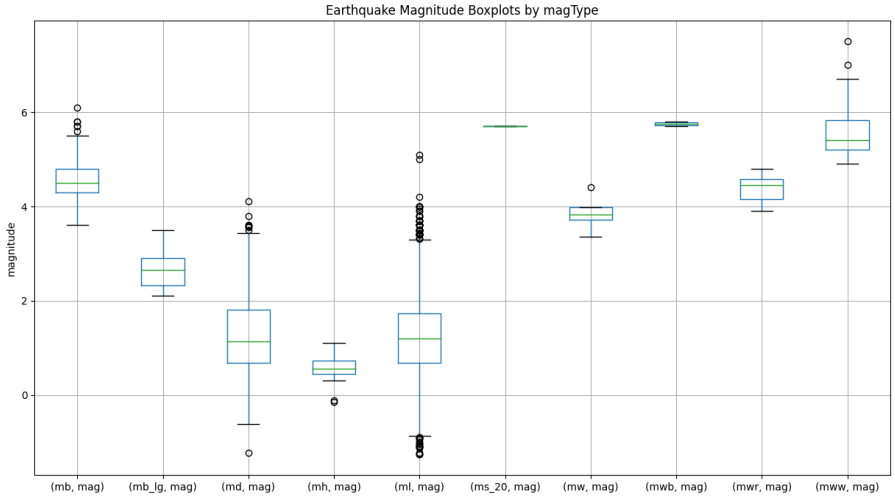
```
Text(0.5, 1.1, 'Facebook OHLC Boxplots by Volume Traded')
```



We can use this to see the distribution of magnitudes across the different measurement methods for earthquakes:

```
quakes[['mag', 'magType']].groupby('magType').boxplot(
    figsize=(15, 8), subplots=False
    )
plt.title('Earthquake Magnitude Boxplots by magType')
plt.ylabel('magnitude') # label the y-axis (discussed in chapter 6)
```

```
Text(0, 0.5, 'magnitude')
```



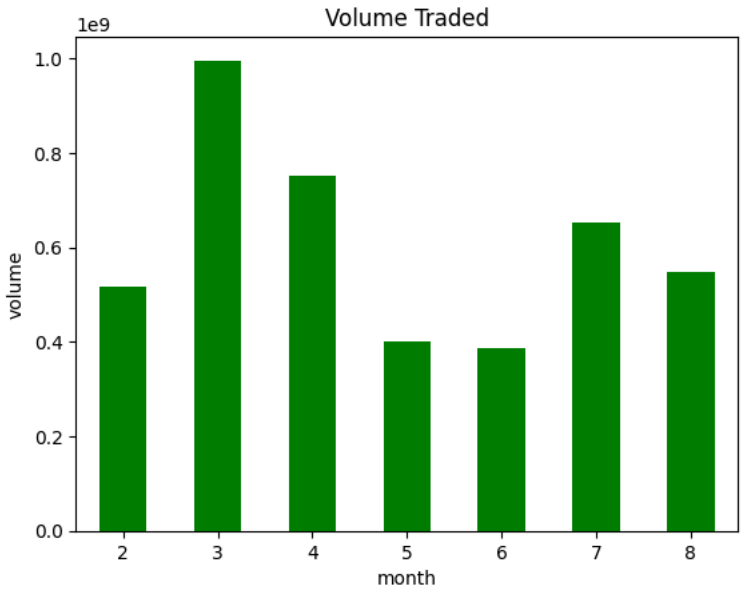
Counts and frequencies

Bar charts

With pandas, we have the option of using the Facebook stock over time:

```
fb['2018-02':'2018-08'].assign(
    month=lambda x: x.index.month
    ).groupby('month').sum().volume.plot.bar(
        color='green', rot=0, title='Volume Traded'
        )
plt.ylabel('volume') # label the y-axis (discussed in chapter 6)
```

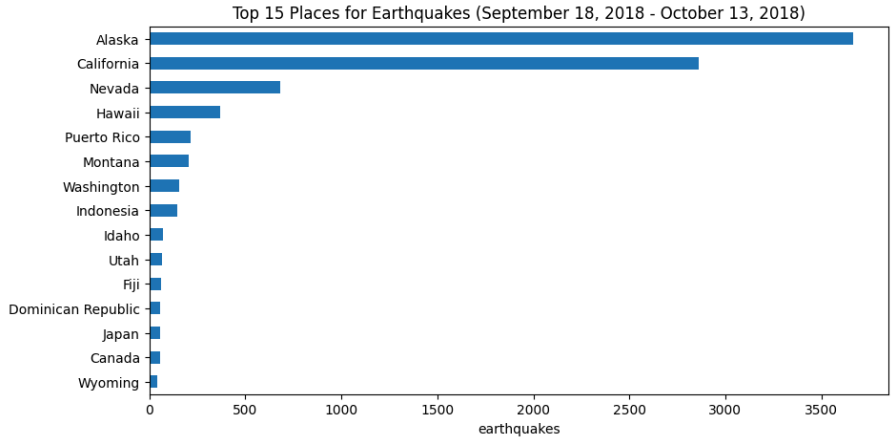
```
Text(0, 0.5, 'volume')
```



We can also change the orientation of the bars. Passing earthquakes in our data:

```
quakes.parsed_place.value_counts().iloc[14::-1,].plot(
    kind='barh', figsize=(10, 5),
    title='Top 15 Places for Earthquakes '\
    '(September 18, 2018 - October 13, 2018)'
    )
plt.xlabel('earthquakes') # label the x-axis (discussed in chapter 6)
```

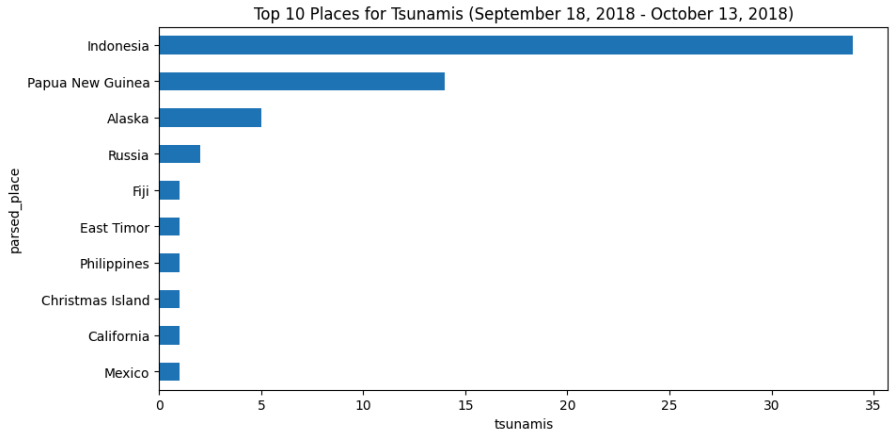
```
Text(0.5, 0, 'earthquakes')
```



We also have data on whether earthquakes were accompanied by tsunamis. Let's see what the top places for tsunamis are:

```
quakes.groupby('parsed_place').tsunami.sum().sort_values().iloc[-10::,].plot(
    kind='barh', figsize=(10, 5),
    title='Top 10 Places for Tsunamis '\
    '(September 18, 2018 - October 13, 2018)'
)
plt.xlabel('tsunamis') # label the x-axis (discussed in chapter 6)
```

```
Text(0.5, 0, 'tsunamis')
```



Seeing that Indonesia is the top place for tsunamis during the time period we are looking at, we may want to look how many earthquakes and tsunamis Indonesia gets on a daily basis. We could show this as a line plot or with bars; since this section is about bars, we will use bars here:

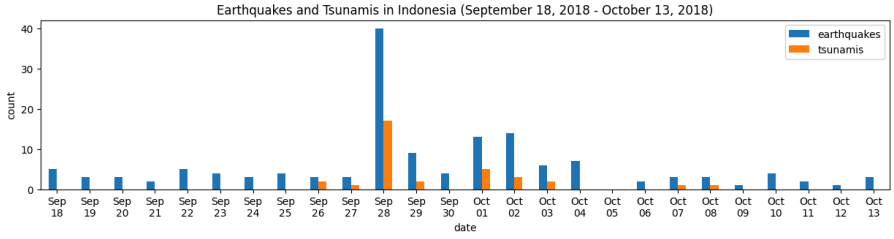
```
# Filtering earthquakes in Indonesia, aggregating by day, and plotting
indonesia_quakes = quakes.query('parsed_place == "Indonesia"').assign(
    time=lambda x: pd.to_datetime(x.time, unit='ms'),  # Converting time to datetime
    earthquake=1  # Adding column to count earthquakes
).set_index('time').resample('1D').sum()  # Resampling data to daily frequency and summing up earthquakes

# Formatting index for better visualization
indonesia_quakes.index = indonesia_quakes.index.strftime('%b\n%d')

# Plotting earthquakes and tsunamis in Indonesia
indonesia_quakes.plot(
    y=['earthquake', 'tsunami'],  # Columns for y-axis data
    kind='bar',  # Type
    figsize=(15, 3),  # Size
    rot=0,  # Rotation of x-axis labels
    label=['earthquakes', 'tsunamis'],  # Legend labels
    title='Earthquakes and Tsunamis in Indonesia (September 18, 2018 - October 13, 2018)'  # Title
)

# Labeling the axes
plt.xlabel('date')
plt.ylabel('count')
```

```
<ipython-input-24-2f70e6217184>:5: FutureWarning: The default value of numeric_only in D
  ).set_index('time').resample('1D').sum()  # Resampling data to daily frequency and sum
Text(0, 0.5, 'count')
```



Using the kind arugment for vertical bars when the labels for each bar are shorter:

```
quakes.magType.value_counts().plot(
    kind='bar', title='Earthquakes Recorded per magType', rot=0
    )
# label the axes (discussed in chapter 6)
plt.xlabel('magType')
plt.ylabel('earthquakes')
```

```
Text(0, 0.5, 'earthquakes')
```