Candlestick Charting with Plot.ly

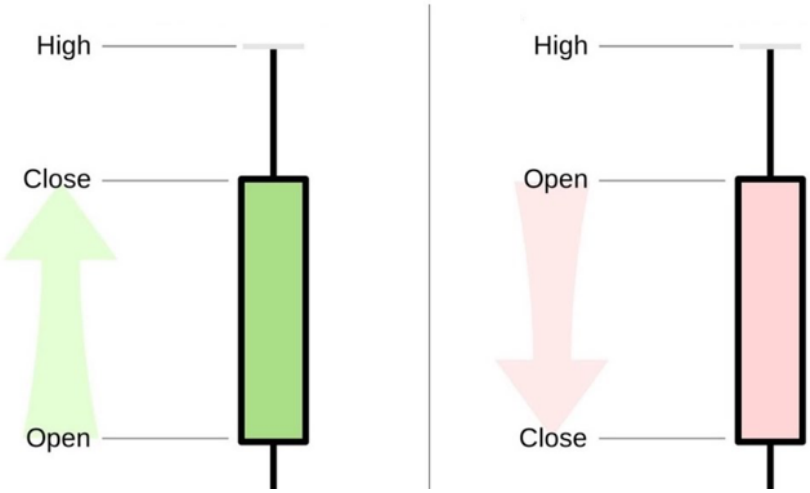
What are they?

Candlestick charts litter the computer screens of traders on trade floors. Invented in the 1700s in Japan for rice trading, are a common way for today's securities traders to show the movement of a stock, derivative, currency, or other security. Instead of a common line graph showing the progression of a price's movement up and down, a candlestick chart shows a collection of candlesticks where each stick shows four main pieces of information about price movement during a specified timeframe: open price, close price, high price, and low price. Not to be confused with box plots, which look similar, candlesticks often have a "body" with two whiskers.

Source: https://datavizcatalogue.com/methods/candlestick_chart.html (https://datavizcatalogue.com/methods/candlestick_chart.html)

How do you read them?

As you can see below, there are two basic types of candlesticks. Each candlestick represents a timeframe (e.g. one minute, twenty minutes, one day) as determined by the analyst. During that timeframe, the candlestick begins with the open price of that minute. If the price moves upwards, the closing price will be above the opening price and the box will be shaded the corresponding color (often green) to indicate upward motion in that timeframe. Conversely, if the price moves downwards, the closing price will be below the opening price and the box will be shaded the corresponding color (often red). The "whiskers" represent any price movements above/below the opening and closing prices within that range. For example, if a ticker starts the day of trading at $10, goes up to $12, then closes the day at $11, the space between $11 and $12 will be represented by what's called an "upper shadow" or "wick," while the space between $10 and $11 will be part of the box, or "body."



Source:
https://en.wikipedia.org/wiki/Candlestick_chart#/media/File:Candlestick_chart_scheme_03-en.svg (https://en.wikipedia.org/wiki/Candlestick_chart#/media/File:Candlestick_chart_scheme_03-en.svg)

Note: In addition to these two examples, one might see a straight line with wicks, indicating the price opened and closed at the same level, but moved around during the time period up and down. A straight "dash" with no wicks indicates the price stayed the same the whole time period, with open and close the same and no variance.

How and when are they used in practice?

Due to the dense amount of information provided by each candlestick, candlestick charts are often used by traders and technical analysts to track price patterns, momentum, and turnarounds. For example, if a trader sees consistent green candlesticks, followed by a red candlestick, that is called a "bull flag," and has corresponding trading strategies. An extremely long wick may indicate that the price moved up or down drastically during that timeframe, but was not able to hold. A candlestick with a short upper wick indicates that the open was near the high of the day. Technical analysts and traders are trained to know the multitude of trading indicators and patterns, and consequently know how to identify their next trade.

Though candlesticks are traditionally used in pricing, they are also sometimes seen to track volume of block trades. Since four points of data, all of the same type and tracking the same metric, are needed for each individual observation, this particular chart type's use is pretty limited. It would almost require some sort of time metric for the x axis, and four pricing points for each observation. In theory, this could potentially apply to something like, say, airline ticket prices over time (if prices change intraday/intraweek, etc.), but applications are fairly targeted towards the financial industry.

Plot.ly

Why Plot.ly?

Plotly's library is extremely robust. There are a number of fantastic features that make the outputs interactive and intuitive, with only minor tweaks needed in code to personalize the visualization. This library does a lot with only a little information. It is a free and open-source graphing library. Any sources, issue reporting, or contributions are through their GitHub (*https://github.com/plotly/plotly.py (https://github.com/plotly/plotly.py)*).

The library integrates well with Jupyter, and even has special installation instructions to ensure that Jupyter renders all of its charts inline and with full functionality.

It was founded by Alex Johnson, Jack Parmer, Chris Parmer, and Matthew Sundquist, whose backgrounds are in science, energy, and data analysis and visualization. It has been featured as an innovative company at multiple conferences.

How do you install it?

We will be using Plot.ly for this demonstration. If you do not have this on your computer already, you will need to install it.

Plotly.py can be installed using pip by typing the following into your terminal:

for pip:

```
pip install plotly==4.3.0
```

for conda:

```
conda install -c plotly plotly=4.3.0
```

To ensure compatibility with Jupyter notebooks, you should additionally install the following:

for pip:

```
pip install "notebook>=5.3" "ipywidgets>=7.2"
```

for conda:

```
conda install "notebook>=5.3" "ipywidgets>=7.2"
```

This will ensure that all charts render properly inline.

If you have any difficulties, the following link directs to Plotly's installation page:
https://plot.ly/python/getting-started/ (https://plot.ly/python/getting-started/)

Now let's get started.

In [1]:
```
# Let's import the main libraries we will need.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas_datareader import data
```

Dataset

We will use AAPL (Apple, Inc.) pricing data over a 5 year span. Yahoo! Finance provides Open, High, Low, Close data for all csv's, so it is easy to set up and explain these charts in relation to the data with minimal edits. It is also free, and API access is robust and simple.

Typically, traders would use intraday data for trading decisions, but those can require expensive feeds for tick-by-tick data. It is not unheard of to see daily data as well, for example on resources like https://finviz.com (https://finviz.com), a popular TA site. Conceptually, a trading day as a single candlestick is easy to understand, so we will emulate that sort of idea to start.

In [2]:
```
# And let's grab some data to work with.
# We will pull the daily price data for AAPL for the last 5 years from

aapl = data.DataReader('AAPL', 'yahoo', start='2014-11-01', end='2019-

# If you have trouble, a .csv file has been provided with the same ini
#aapl = pd.read_csv('assets/AAPL.csv')

aapl.head()
```

Out[2]:

Date	High	Low	Open	Close	Volume	Adj Close
2014-11-03	110.300003	108.010002	108.220001	109.400002	52282600.0	100.084625
2014-11-04	109.489998	107.720001	109.360001	108.599998	41574400.0	99.352760
2014-11-05	109.300003	108.129997	109.099998	108.860001	37435900.0	99.590630
2014-11-06	108.790001	107.800003	108.599998	108.699997	34968500.0	99.875427
2014-11-07	109.320000	108.550003	108.750000	109.010002	33691500.0	100.160294

Now, a note for anyone not familiar with finance: We see a "Close" and an "Adj Close" column. As prices are closer to current, the deviation is often minimal, but over extended periods of time we can start to see drift between the two. So which one do we use?

Prices in the "Adjusted Close" column are adjusted for dividends and stock splits, while the "Close" column merely represents the posted end of day price. Adjusted prices more accurately represent the return during that time period, as it includes the amount also earned via dividends, etc.

So, when tracking price movement in general, especially short term, "Close" is likely the better dataset to use. When tracking overall returns of a symbol "Adj Close" is likely the better dataset to use. However, these should always be determined on a case-by-case basis.

For this dataset, since we are merely tracking price movements and learning to visualize, we will stick to "Close," but I often find it a good practice to incorporate a boolean arg to toggle

will stick to "Close," but I often find it a good practice to incorporate a boolean arg to toggle between the two in real-life practice, as it makes any changes faster and easier to implement.

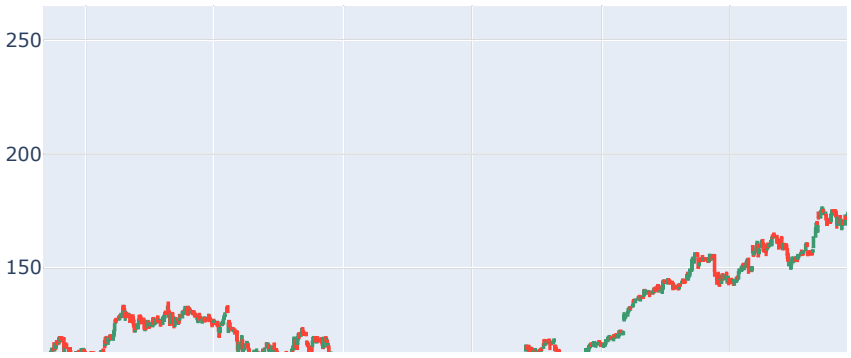
In [3]:
```
# Let's see how this pricing data would look as a standard line graph:

def plot_pricing(df):
    data = df['Close']
    data.plot()
    plt.xlabel("Date")
    plt.ylabel("Price")
    plt.title("Closing Stock Price")

plot_pricing(aapl)
```

Okay, so we see the general past movement of AAPL's closing stock price. As a trader, I could potentially pull some information from this. But before I place a million dollars of client funds into Apple, Inc., I want to know a little bit more about some of this movement. This is just the closing price. What if AAPL is extremely volatile mid-day and a client needs to get out of a position while it is down extremely far? Let's see if I can get more information using candlesticks.

Plotting Basic Candlesticks

To plot candlesticks, we will use Plotly's graphing library. We will refer to "open," "high," "low," and "close," as a collective, as OHLC from this point forward.

In [4]:
```
# Import the plotly library:
import plotly.graph_objects as go

def basic_candlestick(df):
    fig = go.Figure(data=[go.Candlestick(x=df.index,        # Set inde
                                         open=df['Open'],   # Set oper
                                         high=df['High'],   # Set high
                                         low=df['Low'],     # Set low
                                         close=df['Close']) # Set the
                          ])

    # Show the figure designed above:
    fig.show()

basic_candlestick(aapl)
```



This is the bare minimum output without any particular edits or annotations, which is pretty amazing! All you had to input was an x-axis, and which columns should be read for the OHLC. This is the main reason I chose this library for this demonstration. As noted above, it

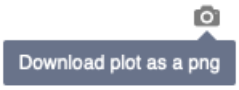
can do quite a lot with very little.

You'll notice in the top right of the plot you have a menu:

Menu Buttons

These buttons provide great functionality within the Jupyter notebook environment.

Download PNG file directly from the menu:



Selection and Panning Tools

Zoom Buttons

Autoscale

Reset Axes

Toggle Spike Lines

These help you triangulate the datapoint in relation to the x and y axes by extrapolating lines out to the axes ticks.

Hover Options

When hovering over a datapoint, you can select to either show the information of the closest datapoint, or two compare multiple datapoints.

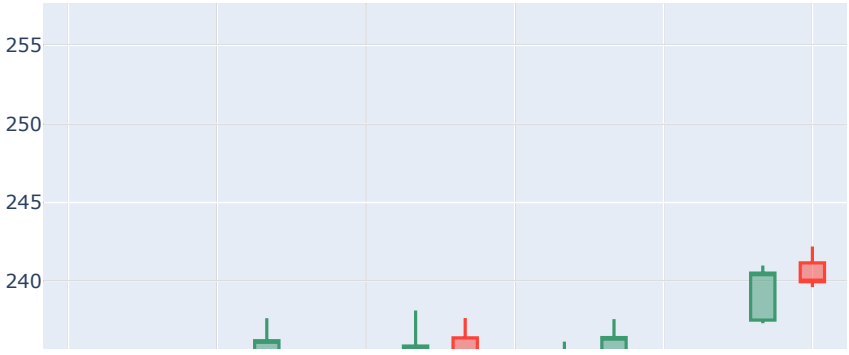
Rangeslider

One really great functionality of the plotly candlestick library is the ability to use the range slider at the bottom by dragging the small rectangles in. This is so you can zoom in on a certain time period. This isn't particularly helpful on this chart, though, because the range of prices spans about $150 within the 5 years, but the y-axis does not adjust, so we have trouble seeing some of the wicks, and instead just end up with a fancier Christmas-colored line plot. Since candlestick charts are generally used for shorter-term trading and entry positions, and don't typically need 5 years of data, let's condense this a bit into something we can make interpretations from.

Simplifying and Making it More Readable

Let's drop the slider since we won't need it on the smaller dataset, and only include the most recent 20 days of trading, to see if we can make out anything more meaningful.

In [5]:
```
# Reduce the dataset to include only n
def reduce_eod_prices(df,n=20):
    df.sort_index(ascending=True, inplace=True)
    df = df.tail(n)
    return df

def basic_candlestick(df, slider=False):
    fig = go.Figure(data=[go.Candlestick(x=df.index,        # Set x ax
                                         open=df['Open'],   # Set oper
                                         high=df['High'],   # Set high
                                         low=df['Low'],     # Set low
                                         close=df['Close']) # Set the
                          ])

    # Add a logic to include the slider
    if slider == False:
        # This line of logic allows us to update the x-axis range slid
        # by turning it off (to "False")
        fig.update_layout(xaxis_rangeslider_visible=False)

    # Show the figure designed above:
    fig.show()


aapl20 = reduce_eod_prices(aapl)
basic_candlestick(aapl20, slider=False) # We will turn off the slider
```



Cool! So we have narrowed down our dataset to only include the last n=20 trading days. Now we can better see the wicks and day-to-day patterns. For example, October 31 saw a $10 intraday drop from open, but ended up bouncing back by close. With further research, we actually find that Apple reported earnings on October 30th after-hours, explaining the strong movement in the next trading day, and the days of trading before and after likely

represents sentiment going into earnings, and the reaction after. Putting the data into a visible context is important with candlestick charting, because it helps identify trends.

Back to the coding aspect.... The interesting thing about that particular edit was the "update_layout" module. This can be further expanded to include a bunch of edits to the chart using Plotly's documentation. *(Found here: https://plot.ly/python/reference/#candlestick (https://plot.ly/python/reference/#candlestick) .)* But I will review some of the more commonly used features that a standard technical analyst may find useful.

Customizing the Candlestick Chart

First, let's add some titles for clarification, and add some shapes and annotations into the "update_layout" module.

In [6]:
```
def adv_candlestick(df, ticker='Company'):
    fig = go.Figure(data=[go.Candlestick(x=df.index,        # Set x ax
                                         open=df['Open'],   # Set oper
                                         high=df['High'],   # Set high
                                         low=df['Low'],     # Set low
                                         close=df['Close']) # Set the
                          ])

    fig.update_layout(
        xaxis_rangeslider_visible=False,
        # Set a chart title
        title='{} 20-Day Price Chart'.format(ticker),

        # Set a y-axis title
        yaxis_title='Price',
        # Set an x-axis title
        xaxis_title='Date',

        # Let's set a line marker to denote the start of the month.
        shapes = [dict(
            x0='2019-11-01', # Select x-tick to start on (in this case
            x1='2019-11-01', # Select x-tick to end on (in this case,

            y0=0, # Select y value to start (bottom of chart)
            y1=1, # Select y value to end (top of chart)

            xref='x', # From documentation:
            # "Sets the shape's x coordinate axis. If set to an x axis
            # id (e.g. "x" or "x2"), the `x` position refers to an x
            # coordinate. If set to "paper", the `x` position refers
            # to the distance from the left side of the plotting area
            # in normalized coordinates where 0 (1) corresponds to
            # the left (right) side."

            yref='paper', # From documentation:
            # "Sets the annotation's y coordinate axis. If set to an y
            # axis id (e.g. "y" or "y2"), the `y` position refers to
            # an y coordinate If set to "paper", the `y` position
            # refers to the distance from the bottom of the plotting
            # area in normalized coordinates where 0 (1) corresponds
            # to the bottom (top)."

            layer='below', # Push the line behind the candlestick to t
            line_width=1,
        )],

        # Let's annotate that marker to denote what it means
        annotations=[dict(
            x='2019-11-01', # Mark where the annotation should be
            y=0.02, # Mark how high up the annotation should be placed
            xref='x', # Sets shape of x coordinate axis
            yref='paper', # Sets shape of y coordinate axis
            showarrow=False, # This shows an arrow to the point at the
```

```
            xanchor='right', # Which side of the annotation is "anchor
            # for spacing, we "anchor" it on the right.
            text='November ' # Denoting which month is being marked as
        )]
        )

    # Show the figure designed above:
    fig.show()

adv_candlestick(aapl20, ticker='AAPL')
```

Annotating breaks in time periods is useful in trading, especially when big options-expiration dates are coming up, or you want to quickly investigate how prices move around "op-ex" and earnings reports.

Shading Areas Within the Chart

One other thing for this particular dataset that may be useful is to denote weekends. Often on trading platforms, one might see greyed-out portions to note after-hours trading or weekends where trading is not occurring. Since Plotly uses a one-tick-per-date method here, there is not a great way of eliminating weekends, as they naturally fall on the x-axis. We can instead grey them out to note that it is a weekend with no trading, and clarify that there is no missing data.

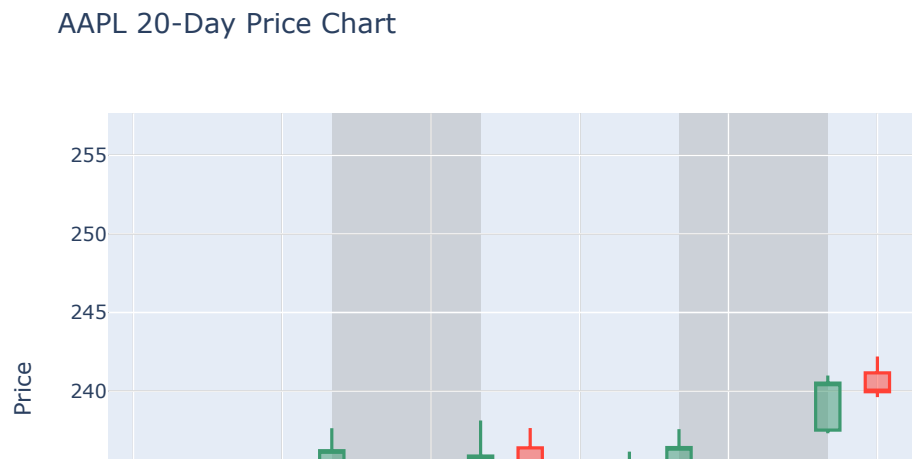
We will modify the above code under the "shapes" arg to create grey boxes where weekends would be

would be:

In [7]:

```
def adv_candlestick(df, ticker='Company'):
    fig = go.Figure(data=[go.Candlestick(x=df.index,        # Set x ax
                                         open=df['Open'],   # Set oper
                                         high=df['High'],   # Set high
                                         low=df['Low'],     # Set low
                                         close=df['Close']) # Set the
                          ])

    fig.update_layout(
        xaxis_rangeslider_visible=False,
        # Set a chart title
        title='{} 20-Day Price Chart'.format(ticker),

        # Set a y-axis title
        yaxis_title='Price',
        # Set an x-axis title
        xaxis_title='Date',

        # Let's set a line marker to denote the start of the month.
        shapes = [dict(
            x0='2019-11-01',
            x1='2019-11-01',
            y0=0,
            y1=1,
            xref='x',
            yref='paper',
            layer='below',
            line_width=1,
        ),
            # Let's create a dictionary for each grey-out period
                  dict(
            x0='2019-10-11', # Expand the box along the x-axis from
            x1='2019-10-14', # the end of Friday to the beginning of M
            y0=0, # Fill the entire height of the chart
            y1=1, # Fill the entire height of the chart
            xref='x',
            yref='paper',
            layer='below',
            line_width=0, # Don't want to have an outline on the box,
            fillcolor='grey', # Grey fill of the box
            opacity=0.25 # Increase the transparency of the box so it'
        ),
            # Repeat for following weekends:
                  dict(
            x0='2019-10-18',
            x1='2019-10-21',
            y0=0,
            y1=1,
            xref='x',
            yref='paper',
            layer='below',
            line_width=0,
            fillcolor='grey',
            opacity=0.25
        ),
                  dict(
            x0='2019-10-25',
            x1='2019-10-28',
            y0=0,
            y1=1,
            xref='x',
            yref='paper',
            layer='below',
            line_width=0,
            fillcolor='grey',
            opacity=0.25
        )
                  ],

        # Let's annotate that marker to denote what it means
```

```
        annotations=[dict(
            x='2019-11-01',
            y=0.02,
            xref='x',
            yref='paper',
            showarrow=False,
            xanchor='right',
            text='November '
        )]
        )

    # Show the figure designed above:
    fig.show()

adv_candlestick(aapl20, ticker='AAPL')
```



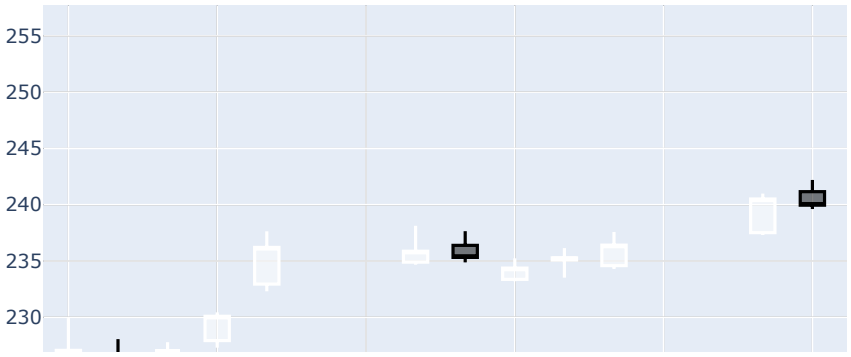
Now, more functionally in a larger dataset, you could create a function that scans the dataset and returns a list of dictionaries that use datetime to pass through the same settings for each weekend module.

So, thus far, Plotly has been fairly comprehensive with the ability to show basic data. What else can we do with it?

Changing Candlestick Colors

Well, another basic edit is to change the color of the indicators. Candlestick charts are sometimes shown with black and white, but this is often difficult as wicks can get lost to the backgrounds. If needed, this can be accomplished by adding the following code to the original figure setup:

In [8]:
```
df = aapl20 = reduce_eod_prices(aapl)

fig = go.Figure(data=[go.Candlestick(x=df.index,
                                     open=df['Open'],
                                     high=df['High'],
                                     low=df['Low'],
                                     close=df['Close'],
                                     increasing_line_color='white', #
                                     decreasing_line_color='black') #
                      ])
fig.show()
```



Since this is rare to see in practice and is difficult to see some of the wicks, let's go back to the original and make some additions.

Adding Subplots and Overlays

Volume Bar Charts

Commonly, these candlestick charts are overlayed with volume bar charts to show if a movement had heavy trading associated with it or not. Let's create a list of colors corresponding to whether the price moved up or down that we will use for these bars.

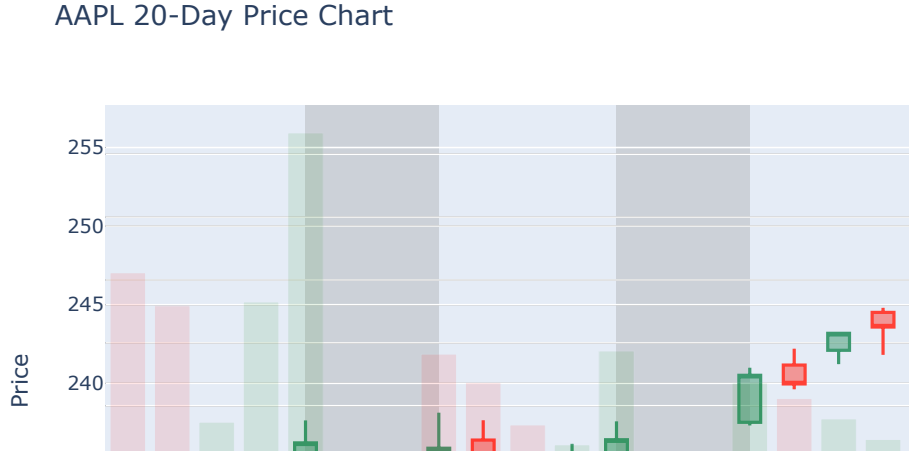
In [9]:
```
colors = []

# Run through the dataset and create a list of colors corresponding to
```

```
# on row n is greater than the close on the previous row.
for i in range(len(aapl20.Close)):
    if i != 0:
        if aapl20.Close[i] > aapl20.Close[i-1]:
            colors.append('green')
        else:
            colors.append('red')
    else:
        colors.append('red')
```

Now let's use those colors to create a volume bar chart. We will do so by editing the first part of the code where we put in the data inputs.

In [10]:
```
# We will need to be able to create subplots to separate the axes. Let
from plotly.subplots import make_subplots

def adv_candlestick(df, ticker='Company'):
    # First, create a figure with subplots and a secondary y-axis
    # With help from https://plot.ly/python/multiple-axes/
    fig = make_subplots(specs=[[{"secondary_y": True}]])

    # Add two traces: one for the candlestick chart...
    fig.add_trace(
        go.Candlestick(x=df.index,        # Set x axis (typically time
                       open=df['Open'],   # Set open as the "Open" col
                       high=df['High'],   # Set high as the "High" col
                       low=df['Low'],     # Set low as the "Low" colum
                       close=df['Close'], # Set the close as the "Clos
                       name='AAPL'),      # Give it a name to differen
                       secondary_y=False  # Set it to the first y-axis
    )

    # ...and a second trace for the volume bar chart.
    fig.add_trace(
        go.Bar(x=df.index,                # Use the same x-axis (dates
               y=df['Volume'],            # Use Volume as the y-coordi
               marker=dict(color=colors), # Pull in the colors list fr
                                          # at each point on the x-axi
               name='Volume',             # Name the chart data
               opacity=0.1),              # The bars are overbearing i
        secondary_y=True                  # Set to secondary y-axis
    )

    fig.update_layout(
        xaxis_rangeslider_visible=False,
        # Set a chart title
        title='{} 20-Day Price Chart'.format(ticker),

        # Set a y-axis title
        yaxis_title='Price',
        # Set an x-axis title
        xaxis_title='Date',

        # Let's set a line marker to denote the start of the month.
        shapes = [dict(
            x0='2019-11-01',
            x1='2019-11-01',
            y0=0,
            y1=1,
            xref='x',
            yref='paper',
            layer='below',
            line_width=1,
        ),
            # Let's create a dictionary for each grey-out period
                  dict(
            x0='2019-10-11',
            x1='2019-10-14',
            y0=0,
            y1=1,
```

```
            xref='x',
            yref='paper',
            layer='below',
            line_width=0,
            fillcolor='grey',
            opacity=0.25 #
        ),
            # Repeat for following weekends:
                  dict(
            x0='2019-10-18',
            x1='2019-10-21',
            y0=0,
            y1=1,
            xref='x',
            yref='paper',
            layer='below',
            line_width=0,
            fillcolor='grey',
            opacity=0.25
        ),
                 dict(
            x0='2019-10-25',
            x1='2019-10-28',
            y0=0,
            y1=1,
            xref='x',
            yref='paper',
            layer='below',
            line_width=0,
            fillcolor='grey',
            opacity=0.25
        )
                 ],

        # Let's annotate that marker to denote what it means
        annotations=[dict(
            x='2019-11-01',
            y=0.02,
            xref='x',
            yref='paper',
            showarrow=False,
            xanchor='right',
            text='November '
        )]
        )

    # Show the figure designed above:
    fig.show()

adv_candlestick(aapl20, ticker='AAPL')
```



Awesome! Now we can see which movements upwards and downards had the most momentum behind them. It looks like October 11 had a large gap up on the open, substantiated by high volume. There are a few anomolies here. For example, on October 7th, 15th and 24th we see a mismatch of colors between the candlesticks and volume bars. This is because, for example, on October 24th the price closed above the previous day's price (as we coded it to identify), but below it's own opening price. So, it gapped up after hours (green volume bar), then fell throughout the day (red candlestick). This is fine, as long as the trader understands what it means and why it occurred. Alternatively, they could change the code to say the open > or < close for the same day and get the same colors as the candlesticks.

Overlaying Analyst Tools into the Chart

Analysts and traders use a ton of tools and patterns to determine if a price is reasonable or not, if a timing is right or not, etc. Let's figure out how to incorporate a simple moving average into our chart.

We will start by defining a simple moving average, which standard conventions have at a window of 20 time periods, though this can be determined by an analyst on a case-by-case basis.

We will then apply this moving average to the *original* dataset (before we reduced it to 20 trading days) so we can see the rolling average reflected in our first few datapoints in early October as well.

Create a function for the metric

In [11]:
```
def simple_moving_average(df, window=20):
    df['SMA'] = df['Close'].rolling(window=window).mean()
    return df

aapl_sma = simple_moving_average(aapl)
print(aapl_sma[['Close', 'SMA']][14:24])
```

```
                 Close         SMA
Date
2014-11-21  116.470001         NaN
2014-11-24  118.629997         NaN
2014-11-25  117.599998         NaN
2014-11-26  119.000000         NaN
2014-11-28  118.930000         NaN
2014-12-01  115.070000  113.374500
2014-12-02  114.629997  113.635999
2014-12-03  115.930000  114.002499
```

```
2014-12-04  115.489998  114.333999
2014-12-05  115.000000  114.648999
```

The above shows where the window of 20 finally has enough datapoints to calculate an SMA. It will be the equally-weighted average of the closing price over the course of the previous 20 datapoints, as a rolling window. In trading, the current price crossing this line is significant because it indicates a trend up or down in relation to what it had been previously doing.

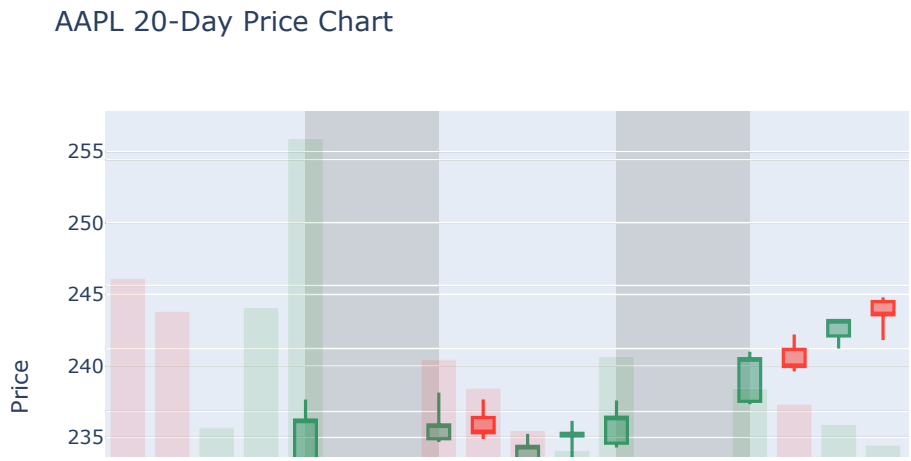
Overlay the metric onto the chart

Let's incorporate this as another trace overlay after our volume and candlestick data entries.

In [12]:
```
def adv_candlestick(df, ticker='Company'):
    fig = make_subplots(specs=[[{"secondary_y": True}]])

    # Add Candlestick chart
    fig.add_trace(
        go.Candlestick(x=df.index,
                       open=df['Open'],
                       high=df['High'],
                       low=df['Low'],
                       close=df['Close'],
                       name='AAPL'),
                       secondary_y=False
    )

    # Add Volume Bar chart
    fig.add_trace(
        go.Bar(x=df.index,
               y=df['Volume'],
               marker=dict(color=colors),
               name='Volume',
               opacity=0.1),
        secondary_y=True
    )

    # Add SMA line
    fig.add_trace(
        go.Scatter(x=df.index,                  # Set index as x-axis (
                   y=df['SMA'],                 # Set SMA to y-axis
                   mode='lines',                # Create a line graph
                   line=dict(width=1),          # Set the width of the
                   marker=dict(color='blue'),   # Set the color of the
                   name='SMA'),                 # Name the line for the
        secondary_y=False                       # Set to first y-axis,
    )

    fig.update_layout(
        xaxis_rangeslider_visible=False,
        # Set a chart title
        title='{} 20-Day Price Chart'.format(ticker),

        # Set a y-axis title
        yaxis_title='Price',
        # Set an x-axis title
        xaxis_title='Date',

        # Let's set a line marker to denote the start of the month.
        shapes = [dict(
            x0='2019-11-01',
            x1='2019-11-01',
            y0=0,
            y1=1,
            xref='x',
            yref='paper',
            layer='below',
            line_width=1,
        ),
        # Let's create a dictionary for each grey out period
```

```
            # Let's create a dictionary for each grey-out period
                  dict(
                x0='2019-10-11',
                x1='2019-10-14',
                y0=0,
                y1=1,
                xref='x',
                yref='paper',
                layer='below',
                line_width=0,
                fillcolor='grey',
                opacity=0.25
            ),
                # Repeat for following weekends:
                      dict(
                x0='2019-10-18',
                x1='2019-10-21',
                y0=0,
                y1=1,
                xref='x',
                yref='paper',
                layer='below',
                line_width=0,
                fillcolor='grey',
                opacity=0.25
            ),
                      dict(
                x0='2019-10-25',
                x1='2019-10-28',
                y0=0,
                y1=1,
                xref='x',
                yref='paper',
                layer='below',
                line_width=0,
                fillcolor='grey',
                opacity=0.25
            )
                      ],

            # Let's annotate that marker to denote what it means
            annotations=[dict(
                x='2019-11-01',
                y=0.02,
                xref='x',
                yref='paper',
                showarrow=False,
                xanchor='right',
                text='November '
            )]
            )

    # Show the figure designed above:
    fig.show()
```

In [13]:
```
# Let's first cut down the AAPL w/ SMA dataset to 20.
aapl20sma = reduce_eod_prices(aapl_sma)

# And then chart the reduced + SMA dataset.
adv_candlestick(aapl20sma, ticker='AAPL')
```



Great! Now we have a chart showing a 20-day simple moving average in relation to the price movements! What's even more interesting is that on October 31st, the price actually seemed to bounce off of the SMA, showing it did not break downwards. Had it, you may have seen more of a trend downward, showing it was crossing down. This is what is great about candlestick charts!

10 Simple Rules Analysis

Rule 1: I believe that this demonstration accurately tells a story. I tried to narrate the development process as I went through, describing the topic as a whole, the uses/relevance of each of the additions, and is readable by someone in the general public. Code is commented to describe what the additions are/steps being taken at each point, and clarifies any confusing parts.

Rule 2: I did my best to document the process, not just the final result. As I went through the steps, I tried to note any changes to the code, as well as any rationale as to why. Most of the coding process was just thinking through how one would try to pass a certain concept into the procedural code, but I found myself much more fluid this time around in execution. Plotly has a fairly comprehensive documentation site, so most errors were fixed with a simple formatting change. The only things involving serious trial-and-error were

sizing/coloring/transparency issues.

Rule 3: I did my best to keep one concept per cell, although I'd like to draw attention to Rule 4 below in the "Room for Improvement" section as to how I think this could have been better done. I feel my demonstration adequately broke up main points as they went along and showed the major changes, how they affect the code, what their purposes is, and how to execute them with adequate annotation/comments.

Rule 8: I believe that I created a reasonable expectation of being able to explore the data I used here. The .csv file is included in the zip file, and an alternative to access it was provided with an API line of code. All assets (photos, data, etc.) should be included in the zip file. It is not a particularly large or cumbersome dataset, so this is easy to include.

Room for Improvement

Rule 4: I think that modularlizing my code actually worked against me here. I previously never used functions for every step, but have so far been taught to in the MADS program so they could pass through the autograders, so I did so here as well. In hindsight, this may have been better suited for a step-by-step not modularized code, as in review the changes seemed to get lost in the growing length of the function. Instead, I should have started off with basics, i.e. "Here is how to set up the basic plot. |Break| Now, let's change one color. |Break| Now, let's add this line/bar/etc. |Break| etc." Instead, the modularization of the code ends up creating more code than necessary, and makes it harder to focus on the actual change being made for that section.