Stock Price Data

In this project, we'll work with stock market data downloaded from Yahoo Finance using the yahoo_finance Python package. This data consists of the daily stock prices from 2007-1-1 to 2017-04-17 for several hundred stock symbols traded on the NASDAQ stock exchange, stored in the prices folder. We used the download_data.py script in the same folder as the Jupyter notebook to download all of the stock price data. Each file in the prices folder has a specific stock symbol for its name, and each contains the following information:

date — the date's date

close — the date's closing price

open — the date's opening price

high — the date's highest stock price during trading

low — the date's lowest stock price during trading

volume — the date's number of shares of traded

To read in and store all of the data, we will make use of a dictionary with the stock symbols (name of the file without the .csv extension) as the values and the value associated with each key as a DataFrame storing the data from the CSV file.

This pdf is classified information in which the functions provided can be used for stock analysis to work with Dr. Chris Cole. - Batman

We need to read all files stored in the prices folder. We use the os module.

In [ ]:
```
import pandas as pd
import numpy as np
import os
import datetime
import sqlite3
from collections import Counter
```

In [ ]:
```
stock_info = {}
[illegible]
```

In [ ]:
```
#Testing to ensure the data was read in correctly 'AAPL'
stock_info['aapl'].head(5)
```

	date	close	open	high	low	volume
0	2007-01-03	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
1	2007-01-04	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
2	2007-01-05	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3	2007-01-08	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
4	2007-01-09	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

In [ ]:
```
[illegible]
```

[illegible]

In [ ]:
```
[illegible]
```

[illegible]

Ratio Analysis: Determining Stock Stability Overtime

With the data read, we can now look into finding the following information:

The average closing price of each stock

The minimum average closing price over all stocks

The maximum average closing price over all stocks

[illegible]

Knowing the average opening and average closing of particular stocks can give us indications on how volatile a given stock might be with percentage points swinging in an upward or downward direction. [illegible]

For example:

[illegible]

From the above data we can conclude that from our 3 sample pulls [illegible]

[illegible]

Some questions are easier to answer by date instead of using the ticker. it will be easier to organize the trades by date. To do so, we'll calculate a dictionary where the keys are the dates and the values are a list of all trades from all stock symbols that occurred on that day.

In [19]:
```
#Will look for optimal ways to run this function. The time complexity is high
trades_by_day = {}

for ticker in stock_info: #O(N)  Time Complexity
    for index, row in stock_info[ticker].iterrows(): #O(N^2) Time complexity
        day = row["date"]
        volume = row["volume"]
        pair = (volume, ticker )
        if day not in trades_by_day:
            trades_by_day[day] = []
        trades_by_day[day].append(pair)
```

In [ ]:

Finding Profitable Stocks

Let's see which stocks would have been the most profitable to buy. We can do this by doing the following:

Subtracting the initial close price (first row) from the final close price (last row), then computing a percentage relative to the initial price. This tells us how much our initial investment would have grown or reduced.

In [209…
```
percentages = []

for ticker in stock_info:
    prices = stock_info[ticker]
    initial = prices.loc[0, "close"]
    final = prices.loc[prices.shape[0] - 1, "close"]
    percentage = 100 * (final - initial) / initial
    percentages.append((percentage, ticker))

percentages.sort()

percentages[-3:]
print ("most positive incline:", max(percentages))#This stock has the highest percentage
print("most negative decline:", min(percentages)) #Biggest decline in  stock from the lis
```

```
most positive incline: (7483.8389225948395, 'admp')
most negative decline: (-98.33424353725407, 'bont')
```

The stock 'admp' has the highest percentage in growth by percentage points the beginning close to the end close:

In [25]:
```
stock_info['admp']
```

Out[25]:

	date	close	open	high	low	volume
0	2007-01-03	0.059996	0.059996	0.059996	0.059996	1100
1	2007-01-04	0.059996	0.059996	0.059996	0.059996	0
2	2007-01-05	0.069996	0.059996	0.069996	0.059996	12800
3	2007-01-08	0.069996	0.069996	0.069996	0.069996	0
4	2007-01-09	0.059996	0.069996	0.069996	0.059996	600
...	...	...	...	...	...	...
2585	2017-04-10	4.600000	4.650000	4.700000	4.550000	189200
2586	2017-04-11	4.500000	4.600000	4.600000	4.500000	160800
2587	2017-04-12	4.500000	4.500000	4.550000	4.450000	227500
2588	2017-04-13	4.550000	4.500000	4.600000	4.400000	235800
2589	2017-04-17	4.550000	4.600000	4.640000	4.500000	196100

2590 rows × 6 columns

In [26]:
```
with open(os.path.join("prices", "admp.csv"), "r") as file:
    admp = pd.read_csv(file)
```

In [27]:
```
admp.columns
```

Out[27]:
```
Index(['date', 'close', 'open', 'high', 'low', 'volume'], dtype='object')
```

In [28]:
```
pip install pandasql
```

```
Requirement already satisfied: pandasql in /dataquest/system/env/python3/lib/python3.8/site-packages (0.7.3)
Requirement already satisfied: pandas in /dataquest/system/env/python3/lib/python3.8/site-packages (from pandasql) (1.0.5)
Requirement already satisfied: sqlalchemy in /dataquest/system/env/python3/lib/python3.8/site-packages (from pandasql) (1.3.20)
Requirement already satisfied: numpy in /dataquest/system/env/python3/lib/python3.8/site-packages (from pandasql) (1.18.5)
Requirement already satisfied: python-dateutil>=2.6.1 in /dataquest/system/env/python3/lib/python3.8/site-packages (from pandas->pandasql) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /dataquest/system/env/python3/lib/python3.8/site-packages (from pandas->pandasql) (2020.4)
Requirement already satisfied: six>=1.5 in /dataquest/system/env/python3/lib/python3.8/site-packages (from python-dateutil>=2.6.1->pandas->pandasql) (1.15.0)
WARNING: You are using pip version 20.2.4; however, version 22.3.1 is available.
You should consider upgrading via the '/dataquest/system/env/python3/bin/python3 -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [29]:
```
from pandasql import sqldf
max_close_date = sqldf("""
        SELECT date
        ,max(close)
        ,volume
        ,low
        FROM admp;
""")

max_close_date
```

Out[29]:

	date	max(close)	volume	low
0	2016-05-09	10.12	729500	8.91

We can see here that with the date of 05-09-2016 that "admp" had the highest close date in the 10 year range. What was so special about this stock on this date?

Lets look at the volume to associated with this date to see if the number of people trading played a factor in the max(close).

In [30]:
```
max_volume_date = sqldf("""
        SELECT date
        ,close
        ,max(volume)
        FROM admp;
""")

max_volume_date
```

Out[30]:

	date	close	max(volume)
0	2016-08-26	3.47	15210600

Here we can see that the max(volume) of traders does not coorelate to the number of the maximum close of $10.12 dated for 2016-05-09.

Lets look at the dates surrounding May.

In [138…
```
max_volume_date = sqldf("""
        SELECT date
        ,close
        ,volume
        FROM admp
        WHERE date > '2016-05-01'
        LIMIT 25;

""")

max_volume_date
```

Out[138…

	date	close	volume
0	2016-05-02	8.48	131000
1	2016-05-03	8.92	326400
2	2016-05-04	8.97	494200
3	2016-05-05	8.96	100900
4	2016-05-06	9.02	106200
5	2016-05-09	10.12	729500
6	2016-05-10	8.99	629300
7	2016-05-11	8.50	395500
8	2016-05-12	7.99	240100
9	2016-05-13	8.15	97300
10	2016-05-16	8.11	80300
11	2016-05-17	8.21	98700
12	2016-05-18	8.07	99100
13	2016-05-19	8.14	99000
14	2016-05-20	8.29	107700
15	2016-05-23	8.30	153600
16	2016-05-24	8.61	149200
17	2016-05-25	8.46	89500
18	2016-05-26	8.51	46600
19	2016-05-27	8.76	206600
20	2016-05-31	8.67	188000
21	2016-06-01	8.70	320300
22	2016-06-02	8.86	515800
23	2016-06-03	8.86	283700
24	2016-06-06	4.09	4966600

We can see that stock was at it's highest in the month of May. Those who invested prior reap the most benefits in this year at this given period.

Next let's look at this stock in depth to determine when to buy and when to sell.

Bollinger Bands Of 'ADMP'

Bollinger Bands are trend lines plotted above and below the SMA of the given stock at a specific standard deviation level.Bollinger Bands are great to observe the volatility of a given stock over a period of time. The volatility of a stock is observed to be lower when the space or distance between the upper and lower band is less. Similarly, when the space or distance between the upper and lower band is more, the stock has a higher level of volatility.

In [32]:
```
pip install termcolor
```

```
Requirement already satisfied: termcolor in /dataquest/system/env/python3/lib/python3.8/site-packages (2.1.0)
WARNING: You are using pip version 20.2.4; however, version 22.3.1 is available.
You should consider upgrading via the '/dataquest/system/env/python3/bin/python3 -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [33]:
```
import matplotlib.pyplot as plt
import requests
import math
import numpy as np
from termcolor import colored as cl
```

In his this portion of the case study, we will The first part is to calculate the SMA values of the "admp" stock and the second step will be to calculate the Bollinger Bands information.

In [154…
```
#Calculating SMA values: 'admp' values over 100 periods

def sma(data, window):
    sma = data.rolling(window = window).mean()
    return sma
admp['sma_1000'] = sma(admp['close'], 1000)
admp.tail()
```

Out[154…

	date	close	open	high	low	volume	sma_100	upper_bb	lower_bb	sma_500	sma_5000	sma_2(
2585	2017-04-10	4.60	4.65	4.70	4.55	189200	3.4397	4.640937	2.238463	3.17440	4.24128	3.43
2586	2017-04-11	4.50	4.60	4.60	4.50	160800	3.4567	4.669421	2.243979	3.18325	4.24052	3.45
2587	2017-04-12	4.50	4.50	4.55	4.45	227500	3.4742	4.696188	2.252212	3.19265	4.23954	3.47
2588	2017-04-13	4.55	4.50	4.60	4.40	235800	3.4922	4.724075	2.260325	3.20195	4.23930	3.49
2589	2017-04-17	4.55	4.60	4.64	4.50	196100	3.5089	4.752466	2.265334	3.21115	4.23908	3.50

Standard deviation tells you how spread out the data is. It is a measure of how far each observed value is from the mean. In any distribution, about 95% of values will be within 2 standard deviations of the mean.

In [162…
```
#Calculating Bollinger Bands: Bollinger Bands values of "Admd"
#bb = Bollinger Bands
def bb(data, sma, window):
    std = data.rolling(window=window).std() #returns the standard deviation
    upper_bb = sma + std * 2
    lower_bb = sma - std * 2
    return upper_bb, lower_bb



admp['upper_bb'], admp['lower_bb'] = bb(admp['close'], admp['sma_100'], 100)
admp.tail()
```

Out[162…

	date	close	open	high	low	volume	sma_100	upper_bb	lower_bb	sma_500	sma_5000	sma_2(
2585	2017-04-10	4.60	4.65	4.70	4.55	189200	3.4397	4.640937	2.238463	3.17440	4.24128	3.43
2586	2017-04-11	4.50	4.60	4.60	4.50	160800	3.4567	4.669421	2.243979	3.18325	4.24052	3.45
2587	2017-04-12	4.50	4.50	4.55	4.45	227500	3.4742	4.696188	2.252212	3.19265	4.23954	3.47
2588	2017-04-13	4.55	4.50	4.60	4.40	235800	3.4922	4.724075	2.260325	3.20195	4.23930	3.49
2589	2017-04-17	4.55	4.60	4.64	4.50	196100	3.5089	4.752466	2.265334	3.21115	4.23908	3.50

Inside the function, we are using the 'rolling' and the 'std' function to calculate the standard deviation of the given stock data and stored the calculated standard deviation values into the 'std' variable. Next, we are calculating Bollinger Bands values using their respective formulas, and finally, we are returning the calculated values. We are storing the Bollinger Bands values into our 'admp' dataframe using the created 'bb' function as seen above.

Plotting Bollinger Bands values

In [163…
```
%matplotlib inline
```

In [164…
```
plt.rcParams["figure.figsize"] = (10,5)
```

In [165…
```
admp['close'].plot(label = 'CLOSE PRICES', color = 'skyblue')
admp['upper_bb'].plot(label = 'UPPER BB 20', linestyle = '--', linewidth = 1, color = 'bl
admp['sma_20'].plot(label = 'MIDDLE BB 20', linestyle = '--', linewidth = 1.2, color = 'g
admp['lower_bb'].plot(label = 'LOWER BB 20', linestyle = '--', linewidth = 1, color = 'bl
plt.legend(loc = 'upper left')
plt.title('ADMP BOLLINGER BANDS')
plt.show()
```

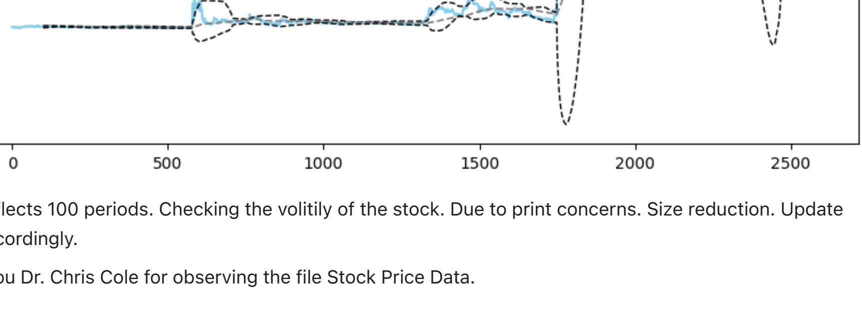

Table reflects 100 periods. Checking the volitily of the stock. Due to print concerns. Size reduction. Update table accordingly.

Thank you Dr. Chris Cole for observing the file Stock Price Data.

RQQ

In [ ]:

In [ ]: