**Computing for Analytics**

**Introduction**

Stock market data is fascinating to analyze and to predict/estimate the prices of stock based on past data. We found a five year long well-structured dataset of S&P 500 companies. The dataset was relatively clean and contained among other data, the opening and closing prices and volume traded on a daily basis from 2014 to 2018.

The problem statement we wanted to address was to estimate the stock prices for some of the famous companies as of present-day(October 9th) and then compare the estimated prices with the actual present-day prices.

We first imported the datasets from Kaggle for each company (500 number). After performing basic Exploratory Data Analysis, we realized that the data was relatively clean, and minimal data cleaning steps were required. However, the current prices were not included in the data. We, therefore, got a separate dataset for stock prices of current S&P 500 companies and merged these two datasets based on company ticker.

Since the last recorded stock price was till Feb 7, 2018, we performed 10000 Monte Carlo simulations for 609 days to get estimated stock prices as of October 9, 2019. Our main aim was to record the high computational steps in the algorithm and try to minimize the computational complexity by using different types of data structures.

One can consider things like how prices change over time and compare multiple stocks or create and graph new metrics from the data provided. From the data, detailed and explanatory stock statistics such as volatility and moving averages can be calculated.

**Computational Steps/ Setup**

**Outline:**

* Import the zip file, unzip the folder, and parse the folder which contains the CSV file. There were multiple steps of parsing within the folder which we completed using glob.
* Within the folder containing the zip files, there were 500 CSV from which we wanted to import the data. To do that we concatenated all the CSV files into one and read them into a data frame.
* To make the prediction, we added a derived column to the data frame using   
  all\_df['margin']= all\_df["close"] - all\_df["open"]
* The original dataset had 619k rows, to make the dataset smaller, we chose 11 recognizable companies from the list of 500 companies to get a smaller data frame of around 14k rows
* We then ran a loop for each company, created a sub-data frame of each company, calculated the probability of price going up, probability of price going down, probability of price remaining the same, mean price by which the stock went up in last 5 years, mean price by which the stock went down.  
  a) Probability of price going up= Number of days price went up in the last 5 years/ 1295  
  b) Probability of price going down= Number of days price went down in the last 5 years/ 1295  
  c) Probability of price going remaining the same= Number of days price remained the same in the last 5 years/ 1295  
  d) Mean Price by which the stock went up= Sum of margin on days when stock price went up/ number of days stock price went up  
  e) Mean Price by which the stock went down= Sum of margin on days when stock price went down/ number of days stock price went down
* We then ran a Monte Carlo simulation for each of the 11 companies with the probabilities and mean prices and got an estimated value of stock prices on Oct 9, 2019, stored them in a data frame and compared them with the actual values. (see Fig. 2)
* We timed the loop to calculate the time taken to store parameters using the data frame manipulation. We got a list of times in a list and plotted them. (see Fig. 1)
* We then converted the data to a dictionary to test the performance of the same computation using the dictionaries. We wanted to see how time for storing parameters for data manipulation varied with an increase in a number of companies while using a data frame and while using a dictionary.
* To get the data into the dictionary, we created a dictionary with keys as the company ticker symbol and each key has a list as its values. The value list consisted of 5 numbers for each company, i.e. [probability of price going up, probability of price going down, probability of price remaining the same, mean price by which the stock went up, mean price by which the stock went down]=  
  E.g. {“AMZN”:[0.51, 0.48, 0.01, 5.81, 4.33]}
* We then plotted the 2 lists which had times for data manipulation using the data frame and the one which had times for data manipulation using the dictionary.
* To see how the stock price of these 11 companies has progressed over the last 5 years (2013-2018), we plotted the prices of these companies using an interactive bubble scatter plot. (See Fig. 3)
* Using Bokeh, we plotted the stock prices of the 11 companies over the 5 years (2013-2018). The plot is an interactive plot that has the stock price for all the dates in those 5 years. (see Fig.4)

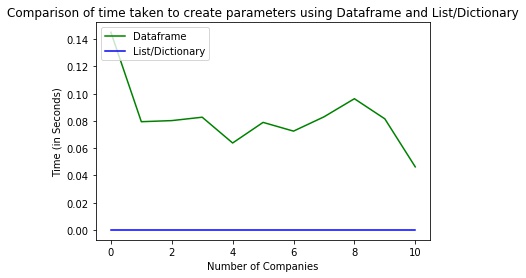
**Computational Challenges:**

* Creating parameters through the data frame was computationally challenging. It took a considerable amount of time to perform these actions, i.e. calculating probabilities and mean prices of going up and down.
* In order to tackle this, we converted the data from the data frame to a dictionary with keys as the company ticker symbol and each key has a list as its values. The value list consisted of 5 numbers for each company, i.e. [probability of price going up, probability of price going down, probability of price remaining the same, mean price by which the stock went up, mean price by which the stock went down]. The data manipulation is considerably faster in a dictionary than a data frame.

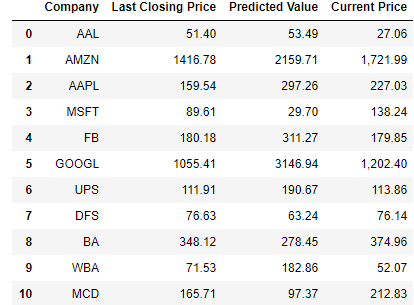
**Slowest Part of the Code:**

* The slowest part of the code is data manipulation using a data frame. We ran a for loop where we had to fetch values from a data frame, considering our data frame is smaller, the performance of the data frame is considerably slower than the dictionary.   
    
  However, if we had a large enough data set, the performance of the dictionary and data frame will produce a similar performance. If we keep on increasing the size of data to include more than 1000 companies, the performance would be similar and at a certain point, the performance of a data frame will be faster than the dictionary. Found a quote from a Stack Overflow3 user “*A dictionary is to a DataFrame as a bicycle is to a car. You can pedal 10 feet on a bicycle faster than you can start a car, get it in gear, etc, etc. But if you need to go a mile, the car wins.*”

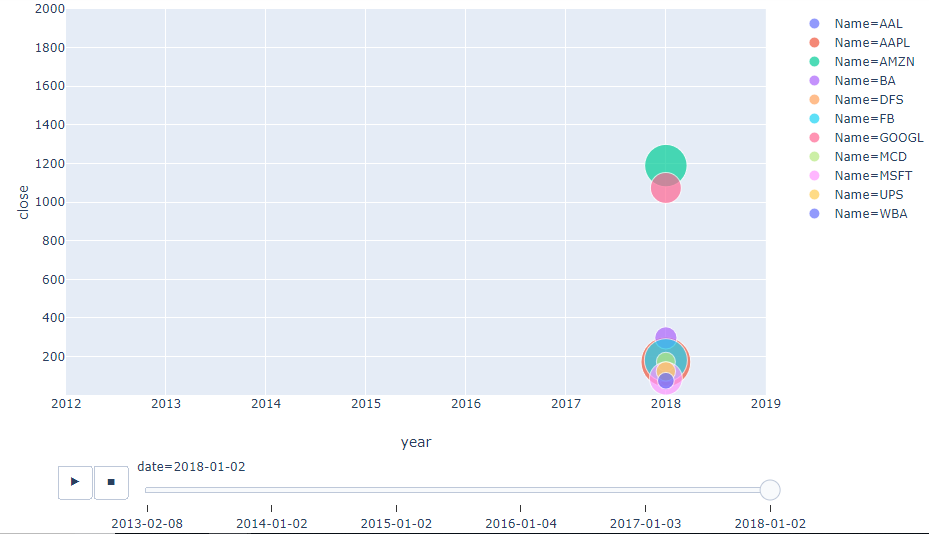
**Results**

****

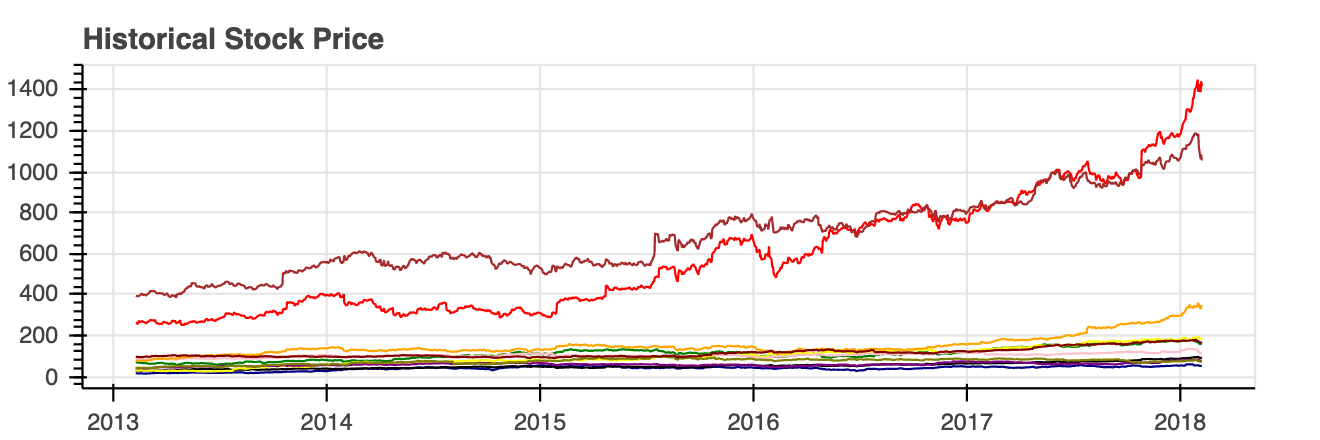
***Fig.1 Plot of comparison of time taken to create the parameters using Dataframe and List/Dictionary***

****

***Fig.2 Comparison of the stock price predicted using Monte-Carlo simulation and the stock price as on Oct-09-2019***

****

***Fig.3 Snapshot of an animated interactive bubble scatter plot using plotly - The animation will show how the stock price of the 11 companies varied through the years. The size of the bubble is the value of the shares traded (Closing price \* Number of shares traded)***

****

***Fig.4 Interactive Bokeh plot of Historical Stock Price - By moving the cursor across the lines, the plot will show the close price for all 11 companies on a specific date.***

**Conclusion**

From the analysis of the time-complexity using the time package, we found that the most time-consuming part of the code was data manipulation using data frames. For a small data frame, the performance of a dictionary is much better than that of a data frame. We therefore concluded that the better option would be to convert the data from the data frame to a dictionary with keys as the company ticker symbol and each key with a list as its values.

However, if we used a large enough dataset to include 1000 companies, the performance of both data frame and dictionary would be similar, and after a certain point (incrementing the dataset to include more data), data frame will perform faster than a dictionary.

For the Monte Carlo simulation, we found that regardless of using dictionary or data frame, the same performance will be the same, since the same values (probabilities of stock = going up, going down, no change) are fed in the for loop and are then appended in a list.

**Reference:**

1.The stock price for S&P 500 companies on Oct 9, 2019: <https://www.suredividend.com/sp-500-stocks/>

2.Dataset and info. On the dataset and some summary on possible problem statements: <https://www.kaggle.com/camnugent/sandp500>

3. Stack Overflow quote

<https://stackoverflow.com/questions/22084338/pandas-dataframe-performance/32658847>