Final Project

*An Interactive Stock Charting App Using D3.js and Angular*

Johns Hopkins University

605.462 Data Visualization

Zachary Rogers

05/14/2018

**Abstract**

Interactive web-based data visualizations are a popular means of communicating information to a user base. In particular, websites and mobile web applications have become more data-driven and interactive as consumers’ demand for more information and greater accessibility has increased over recent years. My personal passion for financial markets and quality market data analysis tools has inspired me to create an interactive stock charting tool using some of the visualization methods and languages we learned in this course. Furthermore, my application allows users to draw and sketch on the chart trend lines, technical patterns, or even write words; all in an effort to give the user more freedom to explore the data, analyze market trends, and eventually share their viewpoints with the world. My app is built using D3.js for the data visualization and interactive portion. I also leverage a font-end Javascript framework called Angular to build the website, UI, and handle the data loading. All code runs on the client-side in the web browser, which greatly reduces development and maintenance cost for servers and databases storage.

**Introduction**

For this final project, I will be creating an interactive chart for visualizing stock market price data. The visualization will be of daily prices with high, low, open, and close values for each data point. This is sometimes referred to as a “Candlestick” chart. The chart will also plot daily shares traded volume on a separate axis on the right hand side of the chart. I will use D3.js for this project, as I want to build a scalable solution that can run smoothly in a web browser without depending on a central server. I also prefer open source technologies, and it has been a fun challenge to build this. The stock chart will also be interactive. Users can mouse over the chart and get price and volume information for each candlestick. Furthermore, user will be able to draw on the chart with their mouse. This feature will allow users to draw trend-lines, call out certain price action, even write comments. Whatever the user of my app conceives of, they can sketch it on the chart. This can then be shared by simply taking a screen shot of the annotated chart.

The stock quote data is sourced from Yahoo! Finance and I am using the Angular JS web framework for the website and UI portion. Because the Yahoo! Finance data API was recently deprecated, I am unable load the data dynamically by querying the API. I was unable to find alternative, free data sources that is suitable for this project, so I resorted to loading a JSON formatted text file through the same asynchronous data pipeline in Angular.

**Background**

Throughout this course, we’ve explored various attributes of quality visualizations and several toolkits and languages common to building interactive data visualizations. I decided to use D3.js for this project, as I think it is the most flexible and powerful of all the approaches we covered. I have also had some minor prior experience learning about D3, so wanted to continue using the software and increase my expertise in the area. Since my project involved building a web based application, I needed additional HTML and JS code to create the user interface and pull in the stock-market data. I chose to use AngularJS as the front-end framework for this project. I have worked with Angular in the past and think it is a wonderfully powerful framework.

When I originally considered this project, I had planned to use the Yahoo! Finance historical stock market data API. Using Angular JS made sense for this data pull, as I could asynchronously query historical data from the API, and load that data into the D3 charting functions I wrote using Angular’s built in Factory directives. This process is very scalable and doesn’t disrupt the user interface or experience since the data is queried and stored asynchronously behind the scenes.

Unfortunately however, the Yahoo! Finance API has been deprecated and is no longer usable for this project. I investigated other data sources for stock quotes, but could not find a quality free service to use. As such, I resorted to loading my data from a JSON formatted text file. Currently the project is hard-coded to load Facebook (FB) historical stock quote data.

**Approach**

This project was focused around 3 main development components:

1. Getting data into scope such that D3 could build the visualization.

* + Originally I planned to use the Yahoo! Finance data API to query historical stock quotes by passing a ticker symbol through a text field in the UI to a RESTful asynchronous call to the API. However, the API has since been deprecated and can no longer be used. I investigated alternative data sources but was unable to find a suitable data feed for this project. Therefore, I downloaded historical data directly from Yahoo Finance in the format of a CSV file. I then used R to re-format the data as a JSON structured text file which is loaded asynchronously to construct the candlestick stock chart. For this project, I have 3 months of daily Facebook price and volume data.
    - This is an example of the quotes data structure:

[

{**"Date"**:**"2018-05-09"**,

**"Open"**:179.67,

**"High"**:183.01,

**"Low"**:178.7807,

**"Close"**:182.4,

**"Adj.Close"**:182.4,

**"Volume"**:18602915

},

{**"Date"**:**"2018-05-08"**,

**"Open"**:178.25,

**"High"**:179.04,

**"Low"**:177.11,

**"Close"**:178.92,

**"Adj.Close"**:178.92,

**"Volume"**:15272000

},

…

{ … }

]

2. Creating the chart using D3.js

* + I knew I wanted to use D3 for this project. It is a versatile and powerful language for building visualizations from data in the web browser.
  + My approach required modifying the chart axis for dates to have a continuous scale that skips weekends, since the stock market is closed on weekends. Without doing this, the chart would have gaps for weekends and holidays which I did not want to show. To remove these gaps, I had to use an Ordinal Scale axis rather than a time-series axis.
  + I also made extensive mouse interaction logic to create the chart crosshairs and also drawing on the chart. Every mouse move updates the location of the cross hairs by deleting them and re-drawing them with the cross hair’s center set at the the closing price of whichever candlestick is directly above or below the mouse location on the chart area.
  + Another feature I implemented is a information box I the top left corner of the chart which updates the High, Low, Open, and Close prices as well as the Volume traded for which ever candlestick the cross hair is set on. This updates real time as you move the mouse around the screen. I used D3 and some jQuery to make this work.
  + Coloring the chart candlesticks also required some custom logic. If the closing price is less than the opening price then the candlestick is red, otherwise it is green. Also if today’s closing price is less than yesterday’s, than today’s candle will be filled in with the red or green color. If it is greater than or equal to yesterday’s closing price then the candlestick is not filled. This style of candlestick coloring and logic follows what other professional stock charting tools use.
  + I also added a 2nd series to the chart which is the daily volume of shares traded. This series has a 2nd axis which I set to the right hand side of the chart. I also scaled it to be 1/3 the height of the chart area, so that the volume bars don’t overlap too much with the stock price candlesticks.

3. A UI and front-end code for the web app.

* + Since my visualization is embedded into a web app, I had to build the user interface that contains the web page layout and chart container as well as take input from the user for the ticker symbol query. The drawing feature is also triggered by a button click event in the UI. Furthermore, my data gathering is also handled asynchronously by the web app framework. I chose to use AngularJS to build these components and the overall website portion of the app.

**Results**

The result is an interactive stock chart that runs fully on the client-side browser. Using D3.js to build the chart and AngularJS Framework for the web app and data pipeline portion, I was able to build this cool and fun tool.

By downloading my source code and JSON data file. You should be able to launch my app locally on your computer and interact with the chart. As a reminder, Facebook stock (FB) is the only data currently supported in this project. Future iterations will include a stock quote data API connection.

Here’s a screenshot of the finished product!



**Conclusion**

In conclusion, I have showed that D3.js is a versatile and powerful language for working with data and scalable vector graphics directly in a web-browser’s DOM. As a javascript framework, D3 can work efficiently on the client-side in the web browser, thus freeing up costly server-side resources. While D3 is a great tool on its own, building a complete working application requires HTML, a User Interface, and some data pipeline to get the information I am plotting with D3. I chose AngularJS for this task as it is a great front end web framework that I have had some prior experience with.

Next Steps for this project include the continued development of this to hopefully launch a live version that anyone can use soon. Next steps include finding an API data source to load any stock or cryptocurrency price data. In the UI I would like to allow the ability to customize the time period as well as create charts of different granularity such as hourly or weekly candlesticks. I will also be working on making this more social so its easy to share your ideas with friends or on social media.

**Sources**

* @tomgertin. “D3 date-time series chart.” bl.ocks.org. http://bl.ocks.org/d3netxer/10a28b7aee406f4e7fce
* Scott Murray. (2013) Interactive Data Visualization for the Web. O’Reilly
* “An interactive Stock Comparison Chart with D3.” Scott Logic. <http://blog.scottlogic.com/2014/09/26/an-interactive-stock-comparison-chart-with-d3.htm>
* “D3.js Axes.” Dashing D3.js. https://www.dashingd3js.com/d3js-axes
* “Angular JS Factory Tutorial.” TutorialEdge.Net <https://tutorialedge.net/javascript/angularjs/angularjs-factory-tutorial/>
* “D3.JS Ordinal Scale.” bl.ocks.org. <https://bl.ocks.org/netzwerg/39cc14803c36404fdaaed7251fdae834>
* “d3.js line graph with dual Y axis.” bl.ocks.org. <http://bl.ocks.org/d3noob/e34791a32a54e015f57d>