John Hall and Brian Heath

Homework 6 - Project

December 18, 2017

CryptoGuru

We set out to build a website that would help individuals with technical trading experience in securities and bonds markets explore the world of cryptocurrencies. We recognized that there was an understandable apprehension for most traders when approaching cryptocurrencies, but we believed that with some practical tools, we could show that substantial parallels exist between trading securities and trading crypto.

The website primarily comprises a webapp implemented using Flask, the Python framework for web development. The functionality is divided between two pages: the landing page and Investment Analyzer page. The landing page contains the mission statement of the site, information on navigation, and live updates on tweets about Cryptocurrencies. The investment page provides tools for graphing and analyzing historical data on a selection of Cryptocurrencies.

The submission comprises the following file hierarchy:

* routes.py
* templates/
  + input\_test.php
  + investment.html
  + landing.html
  + layout.html
* static/
  + css/
    - bootstrap.min.css
    - bootstrap.min.css.map
    - carousel.css
    - custom.css
  + fonts/
    - Glyphicons-halflings-regular.eot
    - Glyphicons-halflings-regular.svg
    - Glyphicons-halflings-regular.ttf
    - Glyphicons-halflings-regular.woff
    - glyphicons-halflings-regular.woff2
  + Images/
  + js/
    - Bootstrap.js
    - Bootstrap.min.js
    - npm.js
* Code/
  + Analysis/
    - Analyzer.py
    - \_\_init\_\_.py
    - Input.py
  + Crawlers/
    - Coinmarketcap\_crawler.py
    - Log\_failures.csv
    - robots\_reader.py
  + Twitter/
    - Authorize.py
    - Credentials.txt
    - \_\_init\_\_.py
    - twitter\_searcher.py
* Data/

***routes.py***

The main launch point for the Flask webapp. This file helps us bridge the gap between Python and HTML by making Python functions and attributes directly accessible within our HTML code.

***templates/:***

* Directory containing all html templates rendered by *routes.py*.
* *layout.html* functions as the base point for the entire site, defining the navigation bar and footer, and importing relevant css stylesheets and js scripts.
* *landing.html* and *investment.html* extend *layout.html* and render the landing and investment page respectively.

***static/:***

* Directory containing static files in website style creation.
* *css/* contains the files which define the style of the site. *custom.css* is the file primarily used in describing unique aspects of the CryptoGuru site.
* *js/* *images/* and *fonts*/ describe helper javascript files, images, and fonts which are utilized by html and css stylesheets in normal functioning of the website.

***data/***

Directory containing csv files containing market data on select Cryptocurrencies. Data for this project is obtained by webcrawler (discussed below) and data is obtained for individual Cryptocurrencies on opening and closing prices, highs, lows, and market volumes. Data is stored in csv formatted files where each line contains the stored features for a given day of trading.

***Code/***

*Analysis:* The Crypto\_Currency\_Analyzer class constructs a dataframe of one or more currencies against a target feature (opening price, closing, etc…) indexed by trading date. The resulting collected data may be plotted using a standard line graph, as a rolling average with adjustable window average, normalized against a starting date, and analyzed over any arbitrary time period for which data is available. The dataframe is constructed using the Python Pandas library and plotting conducted with the Python Matplotlib library. The analysis code further includes Input.py which includes helper functions to ensure the correct formatting of inputs to the Analyzer code.

*Crawlers*: Our crawlers implement spiders that pull historical trading data from the coinmarketcap.com website. They are optimized to only pull historical data from the coinmarketcap.com site if there was an update that has not yet been included in our data files. We use this data to update our Tableau charts and our investment analysis tool. In the future, we will automate these updates. Successfully crawled currencies are stored in the Data directory while a log\_file is kept of currencies which could not be successfully crawled.

*Twitter***:** Our Twitter API implements the Python Twitter dictionary to collect the most recent tweet regarding a topic we choose. The twitter\_search method returns the handle and tweet ID of the latest tweet to routes.py. From here we use the return values to construct a visually appealing Twitter box.

**Test Strategy**

*Python Code:* The Analysis classes used in this project make use of the Python library Doctest where appropriate. For example, the helper functions of Input.py use the Doctest library to show clear examples of desired output as well as test for proper formatting of the results of the functions. Because a large portion of this project involved the production of visual results (i.e., graphs of large data sets) straightforward unit tests could not be conducted for all elements of the project. Instead, much of the implementation of the web crawlers, analyzer, and twitter search code required try/except blocks which ensured usable inputs, desirable interactions with relevant APIs, and raised meaningful error messages when functionality was not as expected.

*HTML/CSS/Flask*: We were unable to find appropriate software to run traditional tests for HTML and CSS. Most software that exists simply ensures that there is no extraneous code related to style, and we used that throughout to ensure that our CSS was correct.

We approached testing as a way to iteratively check that our code would be bug free in both an isolated environment (the divisions in which it existed), and in the greater Flask construct. For HTML and CSS, we developed blocks of the project in isolation, and when functional independently we tested the code against the existing site. Furthermore, we iteratively checked that the code we were writing on our local host would function appropriately on the Heroku server, which is what we used to host our final Flask construct.

*Twitter*: The Twitter API offered a unique opportunity to test because there were restrictions that were unknown to us at the time of coding. Specifically, Twitter restricts the number and frequency of API calls, and upon identifying this restriction we were able to test multiple workarounds to ensure that content always appeared on our page. Once a system was put in place, we also ran this segment through our standard test suite (both the local host and the Heroku host environments) to ensure proper functionality.

**Work Distribution**

*John*

* Build the Flask Blueprint
* Build the Analyzer and Twitter Search scripts
* Integrate the Analyzer

*Brian*

* Design the web page and curate media
* Integrate third party components into the page
* Integrate the Twitter Search API

*Shared*

* Git management
* Testing
* Hosting

**Github Link:** https://github.com/halljw/QuantitativeCryptocurrency