Project Report  
Correlation between Public Sentiment and Stock Prices.

short line

Team 5:  
Arjun Rao, Jinali Gandhi, Kanchan Markandeya

# **Introduction**

The aim of this project is to investigate the relationship between public sentiment and performance of financial markets using Big Data Analytics tools. The development of this project was pragmatically divided into four parts:

**1. Aggregation of Public Sentiment (Opinion) on topics of interest -**

Social media is gradually reflecting and influencing behaviour of other complex systems. Every single day large amounts of text is transmitted online through a diverse variety of social media channels. These channels serve as a huge corpus of valuable information about virtually every topic thats exists.

Through twitter alone over 400 million tweets are published each day. Though each tweet may not provide significant information, but it is argued that the aggregation of these tweets by applying the filter of words of our choice, an overall public opinion, mood or public sentiment can be obtained for the topics of choice.

**2. Aggregation of Financial performance data of companies of interest -**

The information of a company’s financial performance is readily available on the internet. After going through multiple options we discovered Yahoo API as one of the most famous API’s used for analysis of financial data. Hence, we chose Yahoo API to get the historic and current market performance data of companies of choice.

**3. Methods to find the correlation between the two -**

We used ‘Statistical Correlation algorithms’ called Pearson Correlation and Spearman Correlation to find out the association between the public sentiments and market performance as well find the degree (weight) of relationship between the two.

**4. Development of a front end web-application and visualization of results -**

We developed an interactive web - application using Django framework and connecting it to Amazon RDS for MySQL. Our applications gives the detailed information about the company performance and the public sentiment about that company as well as interactive visualizations developed using javascript and Tableau to help understand the correlation better.

The detailed explanation of the processes and analysis of each step we undertook while developing the project is provided in the subsequent sections of this report.

# **Collection of Data**

## Twitter Data

Twitter has made available 2 public API’s to interact with its data i.e. tweets and several attributes about tweets. These are-

* Search API
* Streaming API

Any server side scripting language such as Python or PHP or Ruby can be used to make requests to Twitter API’s and the results are retrieved in JSON format, which can be parsed to get the required labels of your choice. Both the API’s support functions to filter using specific tags.

Initially, we started with the implementation of Search API but soon we were acquainted with its limitations which include-

* Search API looks back into the past for data. It searches against sampling of recent tweets published back in time within 7 days
* The API can retrieve only 450 tweet counts per run
* The API runs only for a 15 minute time interval and disconnects from twitter platform automatically

Since our goal was to get a continuous stream of tweets for the companies of our choice, Search API was not helping our cause. So we looked into Streaming API as our option to get tweets.

Streaming API provides real time flow of tweets. It streams tweets as they occur. So you can access tweets that are being tweeted real time in the world.

We implemented Streaming API in Python using Tweepy - open source library to connect to Twitter platform. Data retrieved from the API is in JSON format and has several attributes of tweets including ID, location, user ID, name, profile title etc. We used Python packages - Json and TextBlob to parse this data to get the labels of our choice - timestamp and tweet text.

We ran this python script 24/7 for 5 companies of our choice - IBM, Tesla, United Airlines, American Airlines and Snap Inc. All the data that was collected was pushed to HDFS.

## Yahoo Data

Yahoo finance is the most popular API for retrieving financial data. Through yahoo finance one has access to multitude of information based on financial performance of almost all publicly listed companies such as all Fortune 500 companies and so on. There are many advantages while using Yahoo API. It allows us to get the current data as well historical data dump of the companies of interest.

We used ‘get\_historical()’ command to get the data of the 5 companies of interest, filtering by the date range we chose. We collected twitter data for 2 months, hence similarly we collected Yahoo data for the same dates across the same 2 months.

We retrieved the entire stock quotes of the 5 companies using a Python script. The retrieved data was in JSON format. We then stored these JSON objects into a variable and parsed it to get the relevant variables such as ‘Open Price’, ‘Close Price’, ‘Day High’ and ‘Day Low’ each filtered by dates.

# **Sentiment-Analysis** In general terms, sentiment refers to feelings. This includes the attitudes, emotions and the opinion being expressed in either speech or a given piece of text. As humans, we have a very strong intuition to decipher the sentiment being expressed by listening to someone speak or reading a piece of text. However, in the face of extremely large volume of data being processed it is not feasible to go through the entire text manually. Hence, we use the help of software to automate the process of defining the sentiment. This automation is known as sentiment analysis.

Sentiment analysis (sometimes known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine. Generally speaking, sentiment analysis aims to determine the attitude of a speaker, writer, or other subject with respect to some topic or the overall contextual polarity or emotional reaction to a document, interaction, or event. The attitude may be a judgment or evaluation (see appraisal theory), affective state (that is to say, the emotional state of the author or speaker), or the intended emotional communication (that is to say, the emotional effect intended by the author or interlocutor). A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature/aspect level—whether the expressed opinion in a document, a sentence or an entity feature/aspect is positive, negative, or neutral. Advanced, "beyond polarity" sentiment classification looks, for instance, at emotional states such as "angry", "sad", and "happy".

## Sentiment analysis algorithm

At the basic level, sentiment analysis works in the following way:

1. Obtain text to perform sentiment analysis upon.
2. Prepare a list of all possible words which indicate positive sentiment (positive word list/lexicon).
3. Prepare a list of all possible words which indicate negative sentiment (negative word list/lexicon).
4. Compare original text with both lexicons and find count of positive matches and negative matches.
5. Classify sentiment as positive if positive words are more prevalent.
6. Classify sentiment as negative if negative words are more prevalent.

However, The inherent nature of social media content poses serious challenges to practical applications of sentiment analysis. Some of these challenges stem from the sheer rate and volume of user generated social content, combined with the contextual sparseness resulting from shortness of the text and a tendency to use abbreviated language conventions to express sentiments. A comprehensive, high quality lexicon is often essential for fast, accurate sentiment analysis on such large scales. An example of such a lexicon that has been widely used in the social media domain is the Linguistic Inquiry and Word Count (LIWC, pronounced “Luke”) (Pennebaker, Francis, & Booth, 2001; Pennebaker, Chung, Ireland, Gonzales, & Booth, 2007).

The tool we used for our project is called VADER (Valence Aware Dictionary and sEntiment Reasoner). VADER is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media. It is fully open-sourced under the [MIT License].

VADER has several important features which distinguishes it from other sentiment analysis tools like NLTK, Social Sentiment Analysis, Stanford CoreNLP toolkit etc. Some of the most important features are as follows:

1. Combination of qualitative and quantitative methods to produce, and then empirically validate, a gold-standard sentiment lexicon that is especially attuned to microblog-like contexts.
2. Combination of lexical features with consideration for five generalizable rules that embody grammatical and syntactical conventions that humans use when expressing or emphasizing sentiment *intensity.*
3. Supplemented with additional lexical features commonly used to express sentiment on social media text like emoticons, acronyms and slang.
4. Has been validated by humans.

In our project, after getting the tweet data in Python using Twitter Streaming API and Tweepy, we passed each and every tweet through the VADER sentiment analysis tool in order to get the sentiment score. We get a positive score, negative score and a compound score for every tweet passing through the algorithm.

1. Positive Score: Proportion of text that falls in the positive category.
2. Negative Score: Proportions of text that fall in the negative category.
3. Compound Score: Computed by summing the valence scores of each word in the lexicon, adjusted according to the rules, and then normalized to be between -1 (most extreme negative) and +1 (most extreme positive). It is a 'normalized, weighted composite score', and is the score that we passed on for Sentiment Aggregation that follows next.

# **Sentiment Aggregation**

The entire concept of sentiment analysis is to learn the majority public opinion about a topic. To understand how any idea or product is perceived by the audience in general or to understand the average emotions of people for that particular topic it is really important you receive and gauge the performance of the product based on an unbiased aggregated sentiment score.

Similarly, for this project our aim was to learn the average sentiment of people about a company on a day-to-day basis. We did not want a biased opinion of a single individual, but we required the average opinion, hence to achieve the average sentiment for a day we aggregated the composite sentiment score based on each day.

Until that point, we had the fetched\_tweets file for each of the company for different days in the span on two months. Every such file consisted of date of that particular day and computed composite score of each tweet. Our aim now was to computed a aggregated average of the composite scores of each date. Using a pyspark script we computed the aggregation of the composite score based on each day. This aggregated composite score now gives us the overall sentiment of people for a particular company and on a particular day over a span of two months.

The pyspark script, parsed the fetched\_tweets files from the HDFS where we stored every tweets file for every company collected each day. After accessing the .txt files from HDFS the pyspark script split the records into <key,value> pairs. The key is the date and value is the list of composite scores for that date. We then parsed it to compute the average composite score based on each unique key (date).

Finally, we print the output on terminal while saving it again on HDFS as well locally. This computed aggregate is also easier to then find the correlation values.

# **Correlation**

To understand the correlation and association between the sentiment of people and market performance or stock prices of a company on any particular day, we implemented two famous ‘Statistical Correlation Algorithms’ namely Pearson Correlation and Spearman Correlation. For our system, the aim was to find the correlation between ‘*aggregate sentiment*’ and ‘*open price of stock*’ for the same date.

Pearson correlation basically is a measure of linear correlation between two variables. The value of Pearson correlation lies between -1 and 1. A value of 1 implies that a linear equation describes the relationship between two variablesperfectly, with all data points lying on a line for which second variable increases as first increases. A value of −1 implies that all data points lie on a line for which second variabledecreases as first variable increases. A value of 0 implies that there is no linear correlation between the variables.

Similarly, Spearman correlation is a non-parametric version of Pearson’s correlation. Spearman correlation not only shows us the association of the variables but also ranks or weights them. Spearman correlation helps one understand the ‘strength’ of the relationship. It accesses the monotonic whether the variables are linear or not.

We wrote a pyspark script to perform correlation and used libraries such as ‘Statistics’ and ‘MLUtils’. The two parameters i.e aggregate sentiment and open price which were previously saved as text files were passed as arrays and the correlation functions were called. The output of both the values were printed on the terminal and later saved in our database. This step was performed for all 5 companies.

Once, we were equipped with both the correlations values for all companies, we then used tableau to develop visualizations showing the values for all 5 companies. These visualizations were then appended to our front-end web application.

From the visualizations, we could observe that IBM had the highest correlation values followed by Tesla. We could also observe that American Airlines had negative correlation values even when it was a trending public topic.

# **Development of Final Application**

For our final deliverable application, we have used a combination of three of the most widely used tools for web development (Amazon RDS for MySQL + Django + HTML). For any basic web application, the two most important parts are the back-end and front-end.

## Back-end

Any website or dynamic web application is a sum of layers—structure, design and content, and functionality. The technology and programming that “power” a site—what the end user doesn’t see but what makes the site run—is called the back end. Consisting of the **server**, the **database**, and the **server-side application**s, it’s the behind-the-scenes functionality—the brain of a site.

### Server and database:

For the database, the most common and easy-to-understand technology is MySQL. Hence, we decided to use MySQL as our database and decided to fetch the data from the tables and display them on our web pages in the Front-End. In order to do so, we used Amazon Relational Database Service (RDS). Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks. Amazon RDS provides six familiar database engines to choose from, including Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, and Microsoft SQL Server. We chose MySQL due to our familiarity with it and its ease of use.

### Server-side application:

Django is an extremely popular and fully featured server-side web framework, written in Python. It has been rapidly gaining popularity for its ease of use and pragmatic design. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Following are some of the key features which made us decide to use Django as our server-side application language -

1. Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Eg - user authentication. HTML templates, forms, admin interface etc.
2. Security : Django helps developers avoid many common security mistakes by providing a framework that has been engineered to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, avoiding common mistakes. It also enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-site request forgery and clickjacking.
3. Django uses a component-based architecture wherein each part of the architecture is independent of the others, and can hence be replaced or changed if needed.
4. Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the Model View Controller (MVC) pattern).
5. Django is written in Python, which runs on many platforms. Thus, it can run applications on many flavours of Linux, Windows, and Mac OS X. Furthermore, Django is well-supported by many web hosting providers, who often provide specific infrastructure and documentation for hosting Django sites.

Our Django application (called twitter\_and\_stocks) has three principle components -

1. **Models**: A model is the single, definitive source of information about the data. It contains the essential fields and behaviors of the data that is stored. Generally, each model maps to a single database table.
   1. Each model is a Python class that subclasses django.db.models.Model.
   2. Each attribute of the model represents a database field.
   3. With all of this, Django gives you an automatically-generated database-access API
2. **Views**: A view is a callable which takes a request and returns a response. This can be more than just a function, and Django provides an example of some classes which can be used as views (ListView, TemplateView, DetailView, FormView etc). These allowed us to structure the views and reuse code by harnessing inheritance. In the view, we request information from the model created before and pass it to a template.
3. **Templates**: Being a web framework, Django needs a convenient way to generate HTML dynamically. The most common approach relies on templates. A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted. We used **Jinja2,** one of the most used template engines for Python. It is inspired by Django's templating system but extends it with an expressive language that gives template authors a more powerful set of tools.

## Front-end

This is the part of the web that you can see and interact with. The frontend usually consists of two parts: the web design and front end web development. Front-end web development is the practice of producing HTML, CSS and JavaScript for a website or Web Application so that a user can see and interact with them directly. In our project, Django allowed us to integrate Jinja2 HTML templates directly with the models and views which enabled us to rapidly produce our web pages and produce the desired behaviour using CSS and JavaScript.

### HTML

HTML (HyperText Markup Language) is the most basic building block of the Web. It describes and defines the content of a webpage. In our project, we used Django supported Jinja2 HTML templates. A template is a file that we can re-use to present different information in a consistent format. The “information” being referred to is just the data from the Models. In the View, we take that information and pass it to the Template.

In Jinja2 Templates, we can also perform several additional operations like conditional rendering, looping through lists/matrices, HTML escaping system for XSS prevention etc. This enables us to display content dynamically without having to write complicated code in the models or views.

### CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language.It is used to set the visual style of web pages and user interfaces written in HTML. We wrote custom CSS to style the pages according to our own needs and also used **Bootstrap**, a free and open-source front-end web framework. Bootsrap contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Using it enabled us to develop good-looking layouts and easily add basic functionality like page headers, navigation bars using its built-in components.

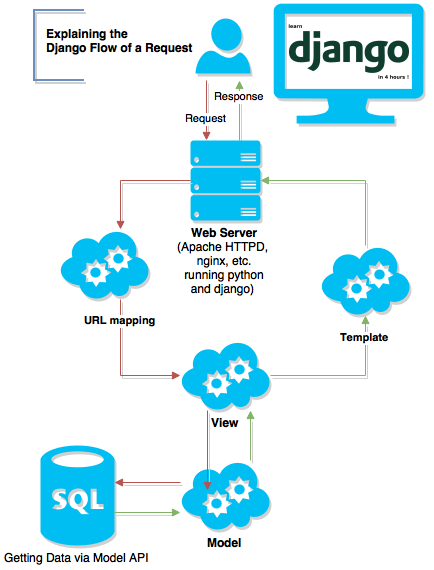
### JavaScript

JavaScript (JS) is a lightweight interpreted or JIT-compiled programming language with first-class functions. It is most well-known as the scripting language for Web pages. JavaScript is a prototype-based, multi-paradigm, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles. However, in our project, we have used **jQuery**, a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery's syntax makes it easier to navigate a document, select DOM elements, create animations, handle events etc.

jQuery is also a requirement for **Chart.js**, a powerful JavaScript library which builds graphs via the canvas element using the data supplied to it. In our project, all the plots depicting the daily stock price and Twitter sentiment have been displayed using Chart.js.

## End Product

Finally, using all the above mentioned tools and technologies, we get a fully functioning, interactive web application that can be run on local server or remote server (which requires additional configuration). Below is a high level view of how everything ties up together



**Findings and Takeaways**

## Time Delay

In our findings, we saw that the calculated Pearson and Spearman coefficients of correlation are quite low (and even negative) for the companies. This basically informs us that there is not a very strong relationship between the public sentiment about a company and its stock price. However, if we check the plots for daily stock price and the daily aggregate sentiment for a company, we notice something.

Sometimes, a drop in the stock price of a company is preceded by a drop in public sentiment by a couple of days. Similarly, a rise in the stock price is preceded by a rise in the public sentiment. In a few cases, The change (positive and negative) in stock price for a company causes the public sentiment to change accordingly over the next few days.

This shows that there is a cause and effect type of behavior wherein, sometimes, a change in public opinion causes a rise or dip of the stock price and sometimes, the change in sentiment is a *result* of a positive or negative change of a company’s stock price. This is a key observation which could not have been made without running the appropriate visualizations and could have been missed if we had stuck only to finding a numerical value for correlation.

Real Time Analysis

As of now, we are not aware of any public API that provides real time stock market price change data. If there were any and if we could access this data, then the current scenario of aggregating the sentiment for a particular day would not have to be done. We could then correlate minute-by-minute tweet and price. This allows for a finer analysis. Also, time series regression techniques could have been used for fitting models on the minute details of data and predict correlation or future stock price change.

## Scale up

Finally, the system we developed is a completely scalable system. For most computations, we used pyspark, for storage we used HDFS, for our database we used Amazon RDS for MySQL and for the front end we deployed django along with tableau visualizations. All these technologies provide a robust architecture to upscale the entire web application. Due to limitation of time and processing power we could collect the data for 5 companies for over 2 months, but what we deployed for 5 companies can be easily extended to cover multiple companies for example, Fortune 500 companies and so on.

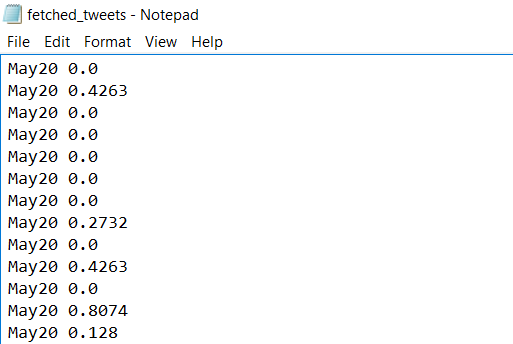
**Appendix**

**Exhibits:**

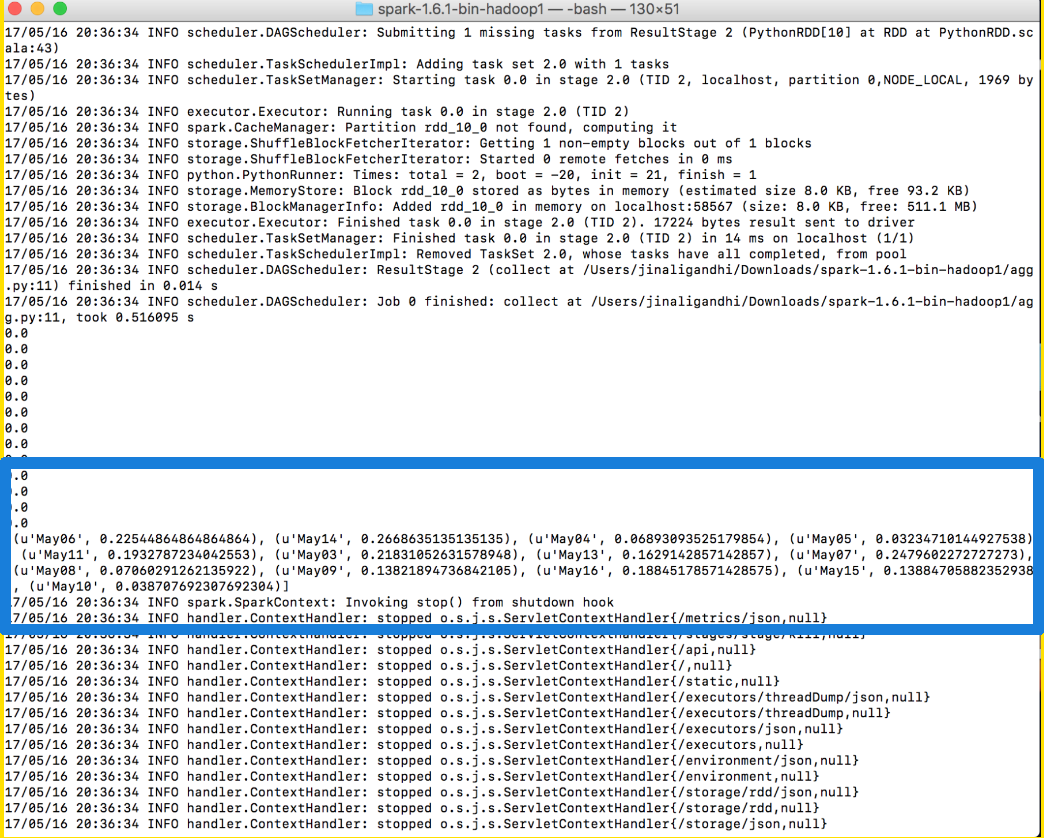
**Exhibit 1: Collection of tweets using streaming API**



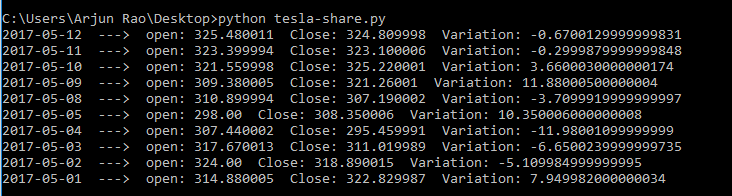
**Exhibit 2: fetched\_tweets text file**



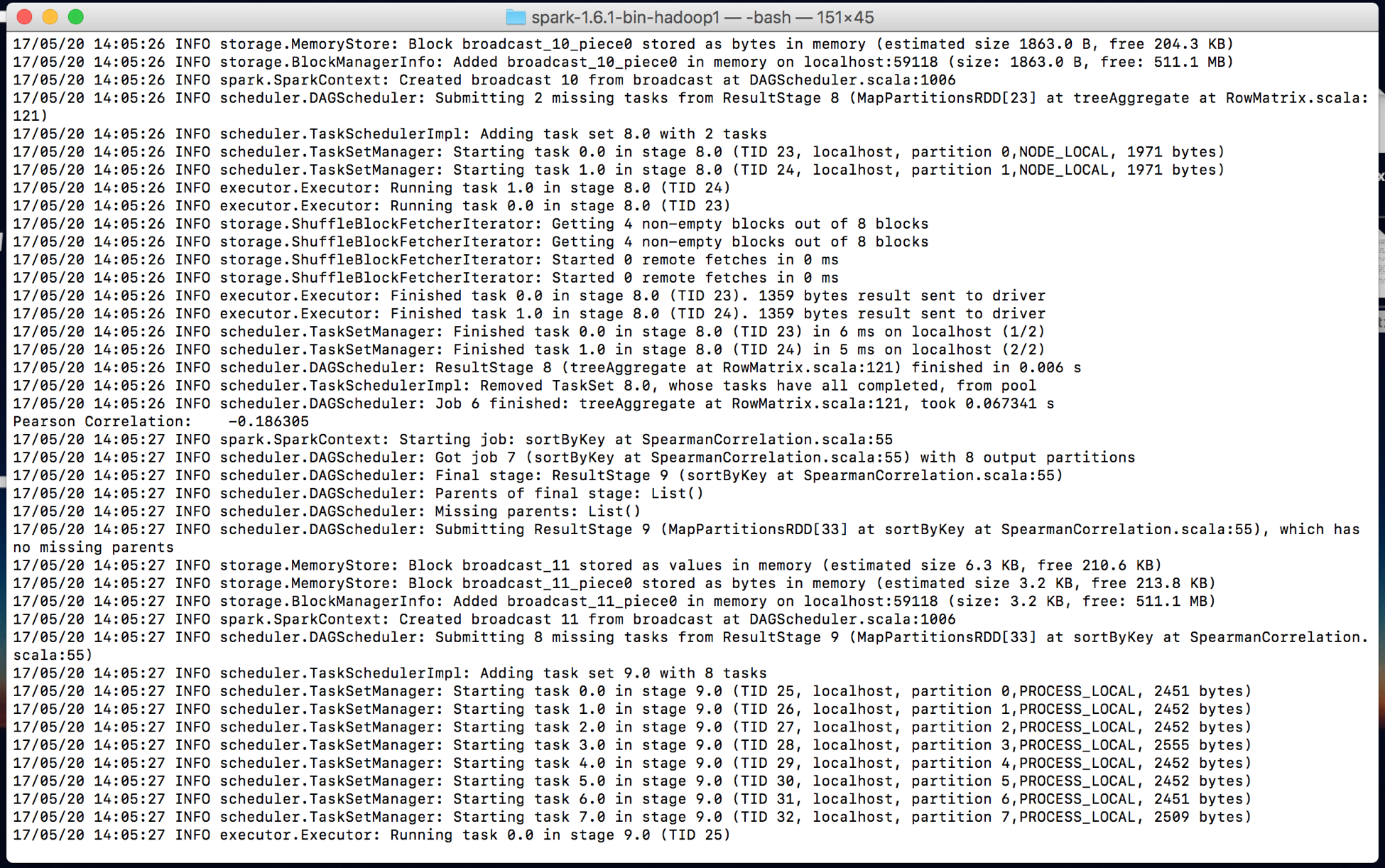
**Exhibit 3: Output of pyspark script ‘agg.py’**

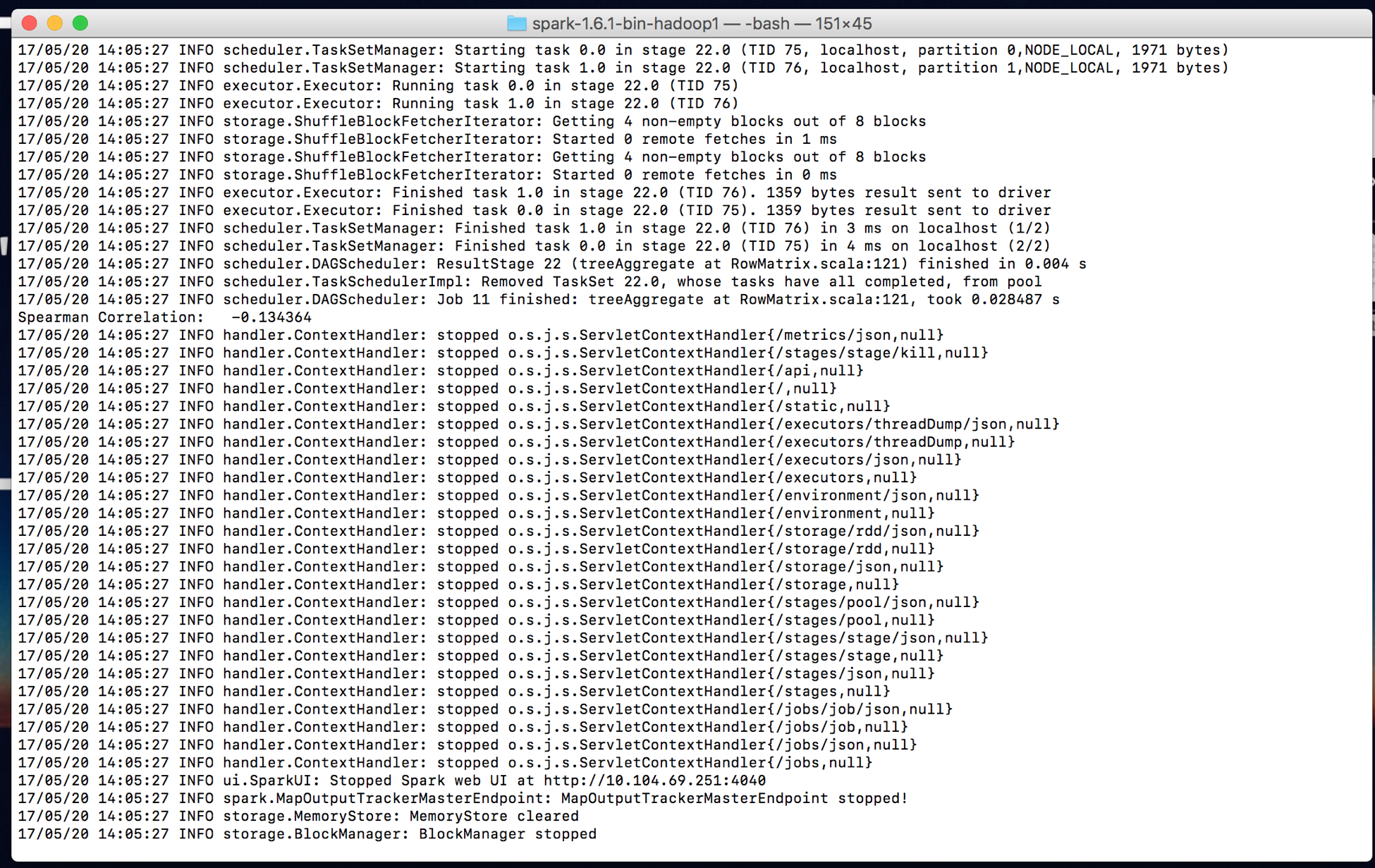


**Exhibit 4: Get financial data from Yahoo**



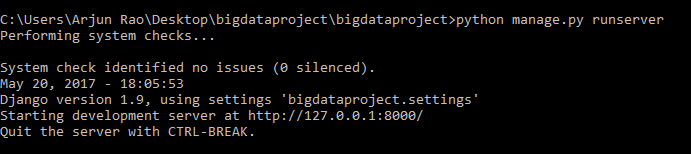
**Exhibit 5: Output of the correlations.py script**



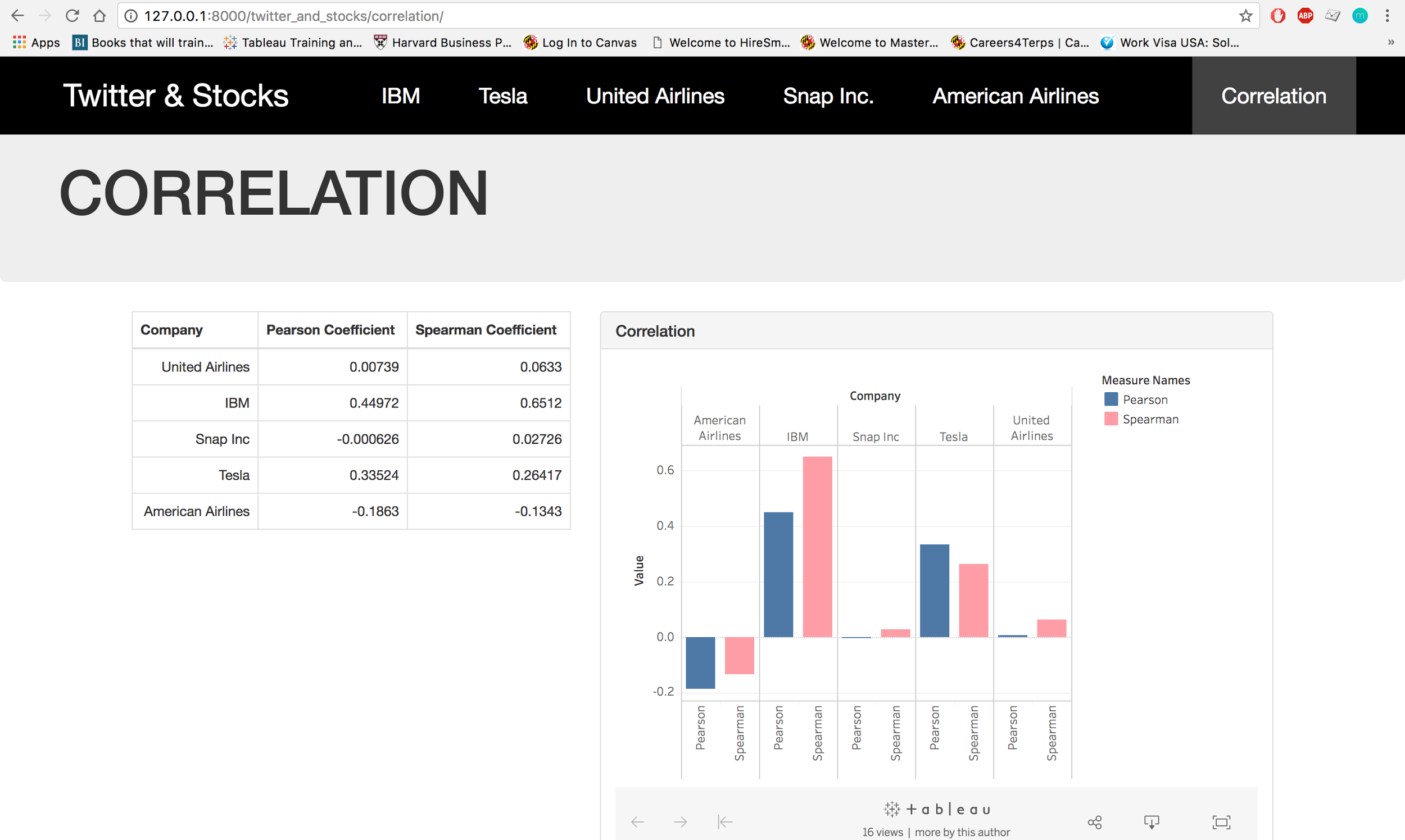


**Exhibit 6: Run the Django application**

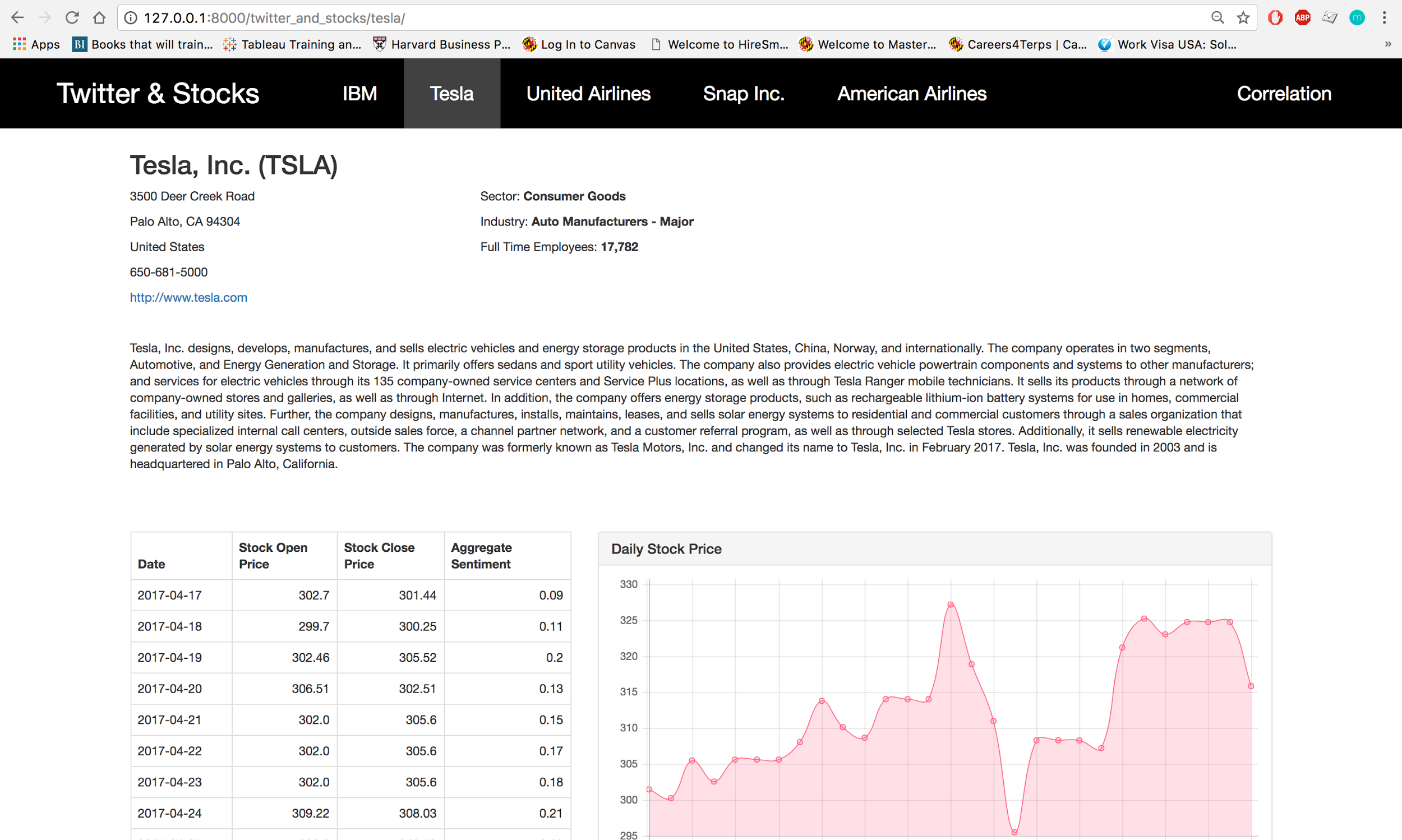
navigate terminal.PNG

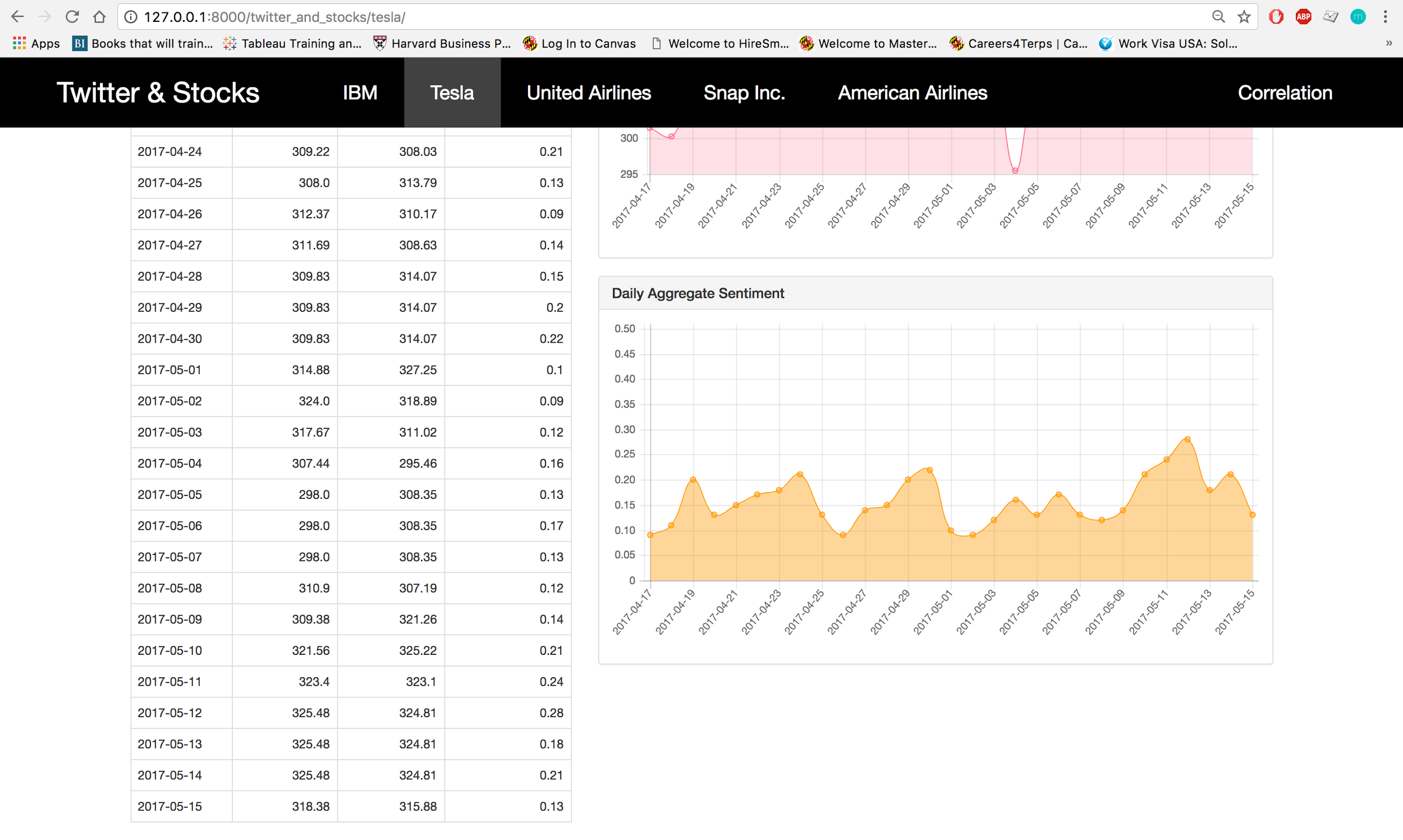


**Exhibit 7: Correlation Visualization on the Web-application**



**Exhibit 8: Front-end pages for every company with graphs showing sentiment values and stock prices**





**References**

Twitter API.  
<https://dev.twitter.com/docs>

Twitter Search API: Limitations.  
<https://dev.twitter.com/rest/public/search>

Twitter Streaming API.  
<http://adilmoujahid.com/posts/2014/07/twitter-analytics/>

yahoo-finance 1.4.0  
<https://pypi.python.org/pypi/yahoo-finance>

VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text  
Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.

Amazon Relational Database Service (RDS).   
<https://aws.amazon.com/rds/mysql/>

Django Tutorial Part 3: Using models. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Models>

Django writing Views   
<https://docs.djangoproject.com/en/1.11/topics/http/views/>

Learn Django – Chapter 3: Views, Templates and URLs  
<http://slash4.net/blog/django-tutorial/learn-django-views-templates-and-urls.html>

Django Templates  
<https://docs.djangoproject.com/en/1.11/ref/templates/>

JINJA  
<http://jinja.pocoo.org/>

Bootstrap (front-end framework)  
<https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)>

Bootstrap  
<http://getbootstrap.com/>

**Acknowledgement**

We would like to express our gratitude to Prof. Kunpeng Zhang for his guidance, support and encouragement which helped us in completion of this project.