CS261 Final Report

Team 29

Sylvester Cardorelle,**Tudor Cismarescu**, **Ollie Madelin**, **Josh Footman**, **Joseph Parkins**, **Anil Kiral**

March 2017

THE UNIVERSITY OF WARWICK

Contents

1 Installation Guide/ System Requirements

1.1 Setup

1.1.1 Node

1.1.2 SQL

1.1.3 Angular

1.1.4 Npm Installation

1.2 Running

2 User Guide

2.1 Analysis tab

2.2 Live Stream tab

2.3 Historical Data tab

3 FAQs

4 Research

After the analysis and design of the project, our software development team had developed a stronger idea of the technologies that would be used to design the system. This section further expands on the decisions and research that was made after the first deliverable that allowed us to finalise our choice of technologies.

4.1 Front-end

The **AngularJS** framework was used to build the frontend as planned. We considered other JavaScript frameworks, such as **React**, however we felt that Angular was the better choice being a full-fledged MVC (Model-View-Controller) framework, which has been in use for a much longer time than other JavaScript frameworks, thus boasting a detailed documentation and various fully-tested features. As our system is essentialy a single page application (SPA) which will update dynamically, AngularJS became a natural choice for our developers.
After this decision was made between the PM and Developers, our front-end developers took the initiative to intensively learn AngularJS using a multitude online resources. For example, Codecademy a programming teaching platform was used to learn Angular basics, whilst Angular Community pages were used to further expand their knowledge.

4.1.1 Graph Libraries

- Chart JS - responsive - sparse documentation - only supports 6 basic chart types - Plotly JS - used for financial graphs , well supported graph types e.g. candlestick graphs etc. - can export generated graph as png for further inspection - can zoom into generated graph and scrutinise specific areas - can save graph directly onto a cloud platform

4.2 Back-end

4.2.1 Server-Side Script

Django (Python) - no one in group was familiar , decided not to use python due to time restrictions Ruby on Rails(Ruby) - same as above Express (Node.JS) - minimalist , built on our exisiting skills of JavaScript Java was carried forward for the data analysis

4.2.2 Database

-decided between MySQL and SQLlite - however backend developers preferred to leave out database from the systems implementation this is because to avoid the bottleneck of querying data from the database rather than caching the data with Java abstract data structures and storing a data for a certain amount of time then storing historical data within the database that would not be useful in further algorithmic calculations

5 Design

5.1 User Interface

- color scheme - font size - abstraction - simplicity we wanted to develop something that did not require a manual essentialy, the apps fundamental operation can be grasped by the client with the assumption that they are computer literate.

5.2 Pseudocode

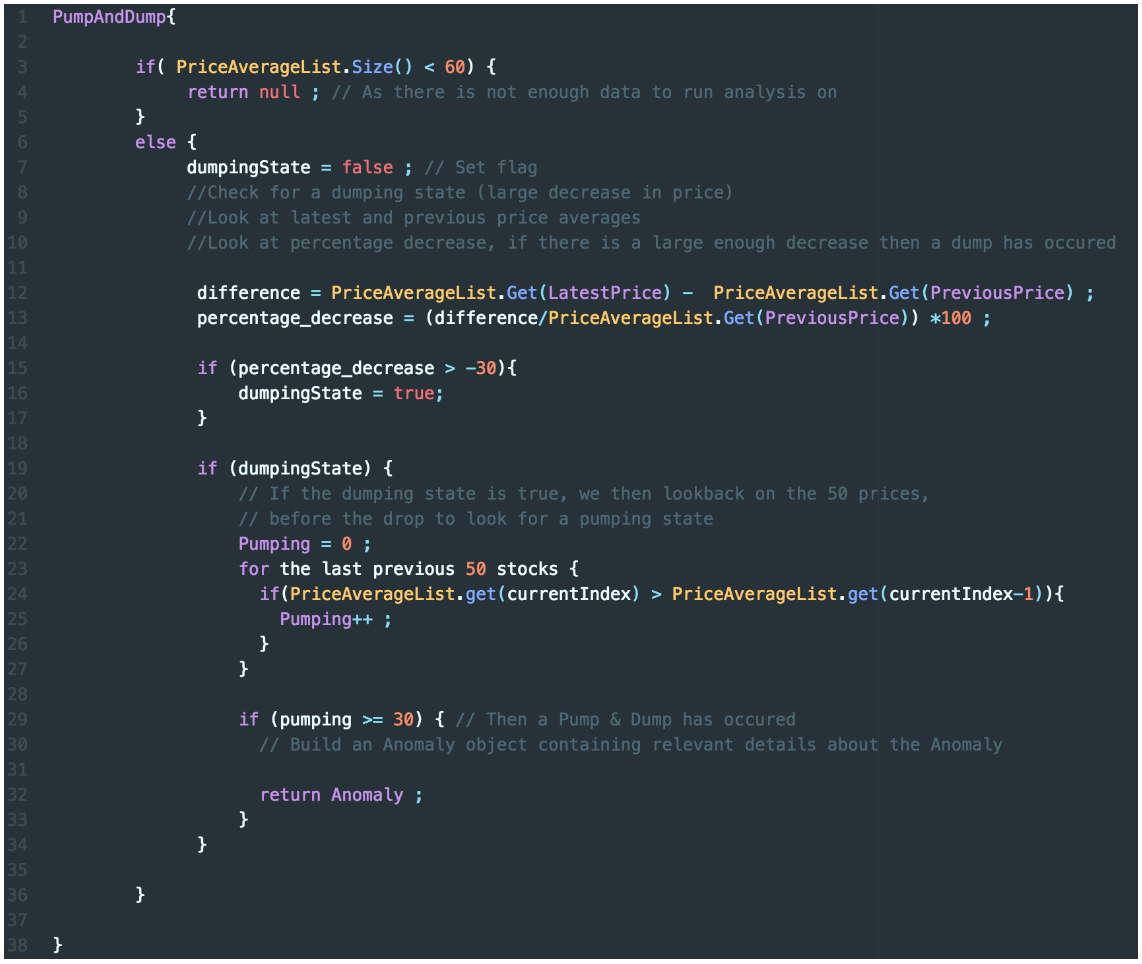
This section explores the type of algorithms that we used to analyse the input stream of data for anomalies.

5.2.1 Pump & Dump

Arguably, Pump & Dump was the most challenging type of price manipulation that we attempted to detect. After researching the characteristics of a Pump & Dump, our team concluded the following about Pump & Dump:

- A large batch of cancellation orders before the dumping of stocks by a person or group (**Non-Detectable with current data set**)
- A steady increase of stocks value from the average value, when the Pumping begins (**Detectable**)
- A sudden drop in price when the dumping of stocks occur (**Detectable**)

PICTURE OF PUMP DUMP EXAMPLE HERE By identifying the traits of a Pump & Dump, we were able to construct pseudocode that would later be transformed into Java code.



```
PumpAndDump{

        if( PriceAverageList.Size() < 60) {
             return null ; // As there is not enough data to run analysis on
        }
        else {
             dumpingState = false ; // Set flag
             //Check for a dumping state (large decrease in price)
             //Look at latest and previous price averages
             //Look at percentage decrease, if there is a large enough decrease then a dump has occured

              difference = PriceAverageList.Get(LatestPrice) -  PriceAverageList.Get(PreviousPrice) ;
              percentage_decrease = (difference/PriceAverageList.Get(PreviousPrice)) *100 ;

              if (percentage_decrease > -30){
                  dumpingState = true;
              }

              if (dumpingState) {
                  // If the dumping state is true, we then lookback on the 50 prices,
                  // before the drop to look for a pumping state
                  Pumping = 0 ;
                  for the last previous 50 stocks {
                    if(PriceAverageList.get(currentIndex) > PriceAverageList.get(currentIndex-1)){
                      Pumping++ ;
                    }
                  }

                  if (pumping >= 30) { // Then a Pump & Dump has occured
                    // Build an Anomaly object containing relevant details about the Anomaly

                    return Anomaly ;
                  }
              }

        }

}
```

Figure 1: Pump & Dump Pseudocode

5.2.2 Volume Spike

5.2.3 Fat Finger Errors

6 Implementation

6.1 Frontend

6.2 Backend

6.2.1 Parsing

7 Iterations

7.1 Iteration 1

7.2 Iteration 2

7.3 Iteration 3

7.4 Iteration 4

7.5 Iteration 5

8 Testing

8.1 Unit Testing

8.1.1 Angular Dependency Injection

8.2 Integration Testing

8.3 System Testing

8.4 User Acceptance Testing

8.4.1 Email Correspondence

9 Project Management

9.1 Overview

9.2 Roles

10 Evaluation