DATA ANALYSIS OF COMMODITY PRICES

A thesis submitted in partial fulfillment
of the requirements for the degree of

MASTER OF TECHNOLOGY

in

Computer Science & Engineering

by

KAPIL THAKKAR

Entry No. 2014MCS2124

Under the guidance of

Dr. Aaditeshwar Seth

Department of Computer Science and Engineering,
Indian Institute of Technology Delhi.

Jan 2016.

Certificate

This is to certify that the thesis titled **DATA ANALYSIS OF COMMODITY PRICES** being submitted by **KAPIL THAKKAR** for the award of **Master of Technology** in **Computer Science & Engineering** is a record of bona fide work carried out by him under my guidance and supervision at the **Department of Computer Science & Engineering**. The work presented in this thesis has not been submitted elsewhere either in part or full, for the award of any other degree or diploma.

Dr. Aaditeshwar Seth
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Abstract

Supply demand imbalance, natural calamities etc. may not always be the reason behind the rise in the price of a commodity. It may be a result of artificial supply deficit planned intelligently by traders nexus to earn more profits through manipulation of supply of commodity and hence indirectly controlling their prices. Our attempt is to locate such hikes in prices which seem suspicious (we call them anomalies). We try to detect these anomalies by characterizing them and forming hypothesis based on their characteristics and then algorithmically try to find the days in time series during which it violates the stated hypothesis.

Acknowledgments

I would like to express my heartiest gratitude to our supervisors Dr. Aaditeshwar Seth for guiding this work with utmost interest, patience, care and scientific rigor. We thank him for setting high standards, giving us freedom to explore multiple facets of the problem and teaching us value of analytical thinking and hard work.

I would also like to thank Dipanjan Chakraborty who have helped a great deal by providing technical guidance and support whenever needed.

KAPIL THAKKAR

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Supply demand imbalance, natural calamities etc. may not always be the reason behind the rise in the price of a commodity. It may be a consequence of artificial supply deficit planned intelligently by traders nexus for profiteering through manipulation of supply of commodity and hence indirectly controlling their prices. Our attempt is to locate such hikes in prices which seem suspicious (we call them anomalies). To detect and analyse the characteristics of anomalies in the prices of commodities. Currently we have considered the case of onion and based on that we have developed one library which has multiple functions to detect anomalies in the time series.

1.2 Objective

Our objective is that we need to highlight anomalies which may be an indicator of illegal market manipulation act by traders nexus in the provided time series. For this purpose we have created library with set of functionalities to detect anomalies in the given input time series. Anomalies will be reported based on the hypothesis stated by the user. For different scenarios user can pass appropriate parameters to functions and functions will report anomalies to user based on that.

1.3 Relevance of Project

Anomaly detection techniques help to explore situations which might be different from the expected behaviour and could reveal interesting facts. It is used in many areas which are explained in detail in next chapter. The

project aims to raise potential red flags for days which are suspect of illicit market manipulation activities by set of traders. This type of monitoring system may help people monitor and hence control these illicit market manipulations better. For example, unnecessary hike in the prices may be due to wrong government policies, loopholes in the supply chain of commodity, intention of profiteering by traders etc. So our project will help journalist or end user interested in detecting such abnormal behaviour.

Chapter 2

Literature Survey

2.1 What is Anomaly Detection?

According to wikipedia, anomaly detection (or outliers detection) is the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset.

Here, our major focus is on detecting anomalies in the time series data. Time series usually considers data about price of some commodity, production, sell, etc. Usually, these time series follow some normal pattern. Some time series may be independent or behaviour of some may depend on other time series. It may also consist of seasonality and trend along with some noise. So, considering all these factors our aim will be to detect some points on time line during which these time series does not follow a normal pattern.

Reasons for presence of anomaly may be different depending upon the type of time series. We have considered time series of onion data as a use case. In case of onion presence of anomalies could be because of unseasonal rainfall, hoarding, price manipulation by traders' nexus, effect of import/export of onions, variation in production, etc. Being a seasonal crop, some part of onions are stored during harvest, so that demand can be met in lean season. But some traders hoard huge stocks for the purpose of profiteering. According to wikipedia, in economics, hoarding is the practice of obtaining and holding scarce resources, possibly so that they can be sold later to customers for more profit. So during this time also, we are able to see anomalies.

2.2 Onion Case

Onion is a staple ingredient for almost every Indian kitchen and hence its demand is almost constant throughout the year but not the supply. In order to supply onions throughout the year, they are stored during harvest and released into markets in lean seasons. Its importance can be well estimated by the fact that it is one among few essential commodities and often rise in its price has resulted into downfall of state and central government.

One major tragedy in onion market occurred in end of year 2010. The prices of the onion increased so much that it was out of reach from poor people. There was a study conducted by the CCI (Competitive Commission of India) for this case and they created report on that [18]. In this [18], they tried to find out the reasons behind this scenario. They came with the following things in their study:

- Large wholesalers/traders mainly operates in metropolitan city markets and large number of farmers dispose their bulk of produce in nearby markets because of absence of storage facility, immediate cash need for loans, family expenses, purchase of inputs of next season, etc.
- Concentration of large storage capacities with traders, Vertical Integration of various market functions by onion traders(one name, many roles), existence of established traders and barrier to new entry
- On December 23 of 2010, The Times of India published in an article that on Tuesday alone, wholesale traders in Delhi bought onion at about Rs.34 per kg while it was sold in retail at Rs. 80 per kg, the margin of Rs. 46 per kg or 135 %.
- In the weeks of November and December, wholesale price remains high, so retailers do not get much profit, but even after that when wholesale price go down, retailers particularly in metro cities, show strong rigidity in holding price and earn margin from 60 to 110 %. This clearly shows that along with traders, retailers also exploit the situation of crisis for their own benefits.

- If we take this forward, then government policies also had a great role in the December 2010 high price episode (export of 1.33 lakh tones onion in October 2010).

So if we consider its overall picture, then there was unseasonal rainfall in the month of September and October 2010, but after that also government policy regarding export of onion was unexplainable. The news article published in Times of India also questions why there is so much difference in the wholesale and retail price of onions. Study also suggests that all the traders operating in the market have experience of many years (20 years on an average) and this is sort of family business for them. Due to limited entries, there is also barrier for new traders in entering to the market. So no new person enters and because of existing traders monopoly, they operate in the market together by forming the nexus. So many times they can alter the prices in many regions so that farmer has to pay money they decide. Traders have monopoly in onion markets and due that prices do not follow normal behavior of demand-supply and goes out of the way.

2.3 Other Cases

2.3.1 Sugarcane Case

This [21] study on sugarcane shows the connections between Politicians and Sugar Mills in Maharashtra and explains how such connection may benefit to firms and politicians. Sugar mills are cooperatives and regions are formed according to mills present. Each sugarcane farmer has to sell his produce to the mill present in his region, he can not sell it to some other mill. Each mill pays its farmers a single price per metric tonne of cane every year, based on weight (not on the quality).

This study investigates how price of the sugarcane, paid to farmers, changes in the election year. Usually, chairman of the sugar mills are politicians who stands in election. They need funding for elections, so here author explains how sugarcane mills and election funding may be related. And if some chair

person wins the election then what is effect on prices offered to farmers. Some findings are:

- Prices are lowered by about Rs. 20 a ton in politically controlled mills during election years
- The results were robust to including rainfall and mill capacity as controls, as well as including mill-specific outcomes such as the recovery rate - sugar produced per unit cane, a measure of productivity - as well as various other mill level shocks such as mill breakdowns and cane shortages
- Price fall may be due to mill closure, i.e. mill is not operating for profitably, but politicians has kept mill open as a way to gain votes. But analysis shows that mill closure is not affected by political control
- Paying farmers Rs. 20 per ton less for their cane amounts to a total of Rs. 6 million
- Mills whose chairmen won national elections pay Rs. 80 per ton more in the year after elections
- Author finds that when the party affiliated with the mill chairmen is in power in Maharashtra, the mill pays Rs. 23 more in cane price and also Chairmen who win national elections seem to be able to keep their mills open far more successfully than chairmen who lose

So here author has strong belief that all facts indicates that funding for election campaign comes from these sugarcane mills if mill is politically controlled. Reason why farmers supports this may be that, with average probability 1/3rd of winning election, on an average farmer gets Rs. 27 on their principal of Rs. 20, so still in profit. The overall effect on farmer welfare is difficult to determine. On average, cane prices and recovery rates in politically connected mills are no different than those in non-politically connected mills, and the levels of public goods are no different either.

So this example explains how time series may go out of their normal behavior though there is no supply-demand crisis or any other case.

2.3.2 Builder-Politician Case

This [20] study shows in developing countries like India, where elections are costly and accountability mechanisms are poor, politicians often turn to private firms for illicit election finance. Land is one of the highly regularized sector in India which provides discretionary power in the hands of state (indirectly in the hands of politicians). It is easy for politicians to accumulate resources than to hide them. To hide these assets from scrutiny politicians often use real estates as medium because of its features which are absorptive capacity, liquid asset and Contract enforcement. As elections approach, however, builders are often compelled to provide politicians with money with which to contest elections; the mechanism can be a simple under-the-table transfer or an in-kind contribution. Although builders have to transfer funds back to politicians around elections, the transaction brings long-term benefits in terms of future goodwill. Based on this author have quoted following hypothesis:

- Cement consumption should exhibit a significant contraction during the month of the state election. Because builders are a leading source of election finance, one would expect activity in the sector to slow down during the month-long campaign period prior to Election Day.
- Contraction in cement consumption will be significant in national elections, though of a smaller magnitude than in state-level elections
- The magnitude of the contraction in cement consumption to be larger for dual elections than if only a state or national election is being held.
- cement consumption should exhibit a larger contraction in urban versus rural states.
- The contraction in cement consumption should be comparatively larger in more competitive elections

The paper was able to bring the quid pro quo relation between builders and politicians quantitatively.

Chapter 3

Study of Onion Data: Collection and Analysis

3.1 System

For now, we are only working over onion data. We have three actors in model: Farmers who are producers of onion, traders who are collectively responsible for supply of onions across country and consumers who purchase onions.

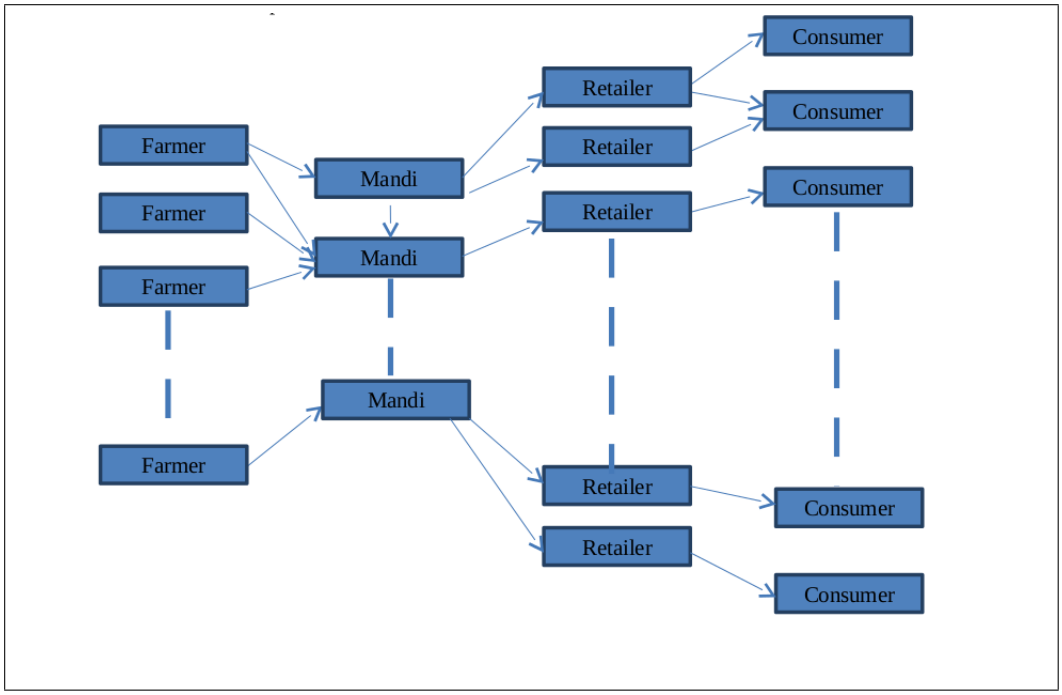

Figure 3.1: Normal Supply Chain

Farmers sell their produce to traders in nearest mandi offering better price. These traders sell these commodities to traders in other mandis or to retailers. Consumers purchase commodities for consumption from one of the retail stores. This way commodity reaches consumers from farmers following a huge chain of traders and retailers.

Under APMC act, mandis were established at different places across country so that farmers can sell their produce directly in mandi and get good returns (wholesale price). There are around 1500 mandis located in different places across country which log their daily arrival of onion, minimum, maximum, modal selling price per quintal of onion data to AGMARKNET. Retailers purchase from these mandis and sell to end customers at retail price. There are around 70+ centres across country which maintains retail price of onion on Ministry of consumer affairs website.

3.2 Data we have

We have following data:

1. Daily wholesale price of onion for 1514 mandis
2. Daily arrival of onion information for 1514 mandis
3. Daily retail price of onion for 76 centres
4. Dates and location for hoarding reports from news articles (Total 453 news articles)
5. Longitude and latitude of mandis and centres

Onion Data was collected from the the government websites,[1] for arrival and wholesale price data and [2] (Department of Consumer Affairs) for retail price data. Crawlers were written to collect the data from these datewise for the period of approximately 9.5 years, starting from 1st January 2006 to 6th July 2015. More data can be added simply by running crawlers again.

Note that news articles were collected two times. First time to study what is called anomaly in the case of onion and how news source says, on what basis, that there is anomaly. We collected some set of article by manual search. We studied this to understand the characteristics of anomalies, so that we can build our system on that. These articles are studied thoroughly

in this chapter in section 3.6. After that, when our system/library got ready, to match our results, we collected news articles rigorously. We have total 453 news articles regarding onion price from 1st December 2010 to 6th July 2015. First these articles were processed using Alchemy API **??** and Diffbot **??** to get the date, place and keywords in the article to analyse them. But we found that many of these collected articles were irrelevant, places fetched using Alchemy API was not up to the mark and date extracted from article was often wrong. Also, articles were related to onion prices. So, these APIs cannot tell us, whether article is abou price hike or price dropped. Plus it was difficult for us to fetch reason of price hike from these automatic analysis . Because of all these reasons, those articles were studied manually to find out reason stated by news articles for price hike, place mentioned in news article, number of days which are being compared to state that there is unusual price hike and any other comment if present in article. After studying each and every article manually, we found that only 267 articles are relevant to this project.

3.3 Normal market behavior

1. Wholesale price is inversely proportional to arrival of commodity. Higher production of crop will lead to more and more crop hitting market for sell. Hence more arrival which will result in surplus supply leading to drop in wholesale prices.

2. Retail price is directly proportional to wholesale price. Commodities reach customers through a long chain of traders and retailers, adding value at every stage of chain. So, retail price at which customers purchase commodities are more than wholesale.

Any divergence from these characteristics of normal market leads to suspicious price hike situations/anomalies.

3.4 What are the reasons for anomaly?

Primarily there are 3 main reasons of anomaly.

1. **Government Policies:** When the production is low in the country, still government allows the export of onion in large amount, or supports it by keeping low minimum price then the prices can rise up drastically.
2. **Unseasonal Rainfall:** Due to insufficient, heavy or unseasonal rainfall, onion crop may get affected and the produce is low and wholesale price may rise up. But, this reason still is validating that wholesale price is inversely proportional to arrival, it may be just prices will be little higher than what was supposed to be.
3. **Hoarding:** When traders/wholesalers store the onion and does not release the stock in the market in the expectation of the good prices in the future, it will create the artificial deficit in the market and will shoot up the onion prices in the retail market due to low arrival in the retail market. The reason people do this is to expect the higher prices in the time of low production or may be for security. For example, if it is expected that in some year the rainfall is not good, then people may predict production to be low in the future and so they will start storing onion so that they can gain more profit. It will also create deficit in the market and price will go up.

So our study will focus on detection of anomalies in data and if possible comment on the possible reason for the anomaly.

3.5 Mapping of wholesale price to retail price

Voronoi Diagram is used to map every mandi to nearest possible centre. The centres with retail data were considered fixed points and country was divided into 76 regions. All the mandis falling in that region are mapped to the respective centre.

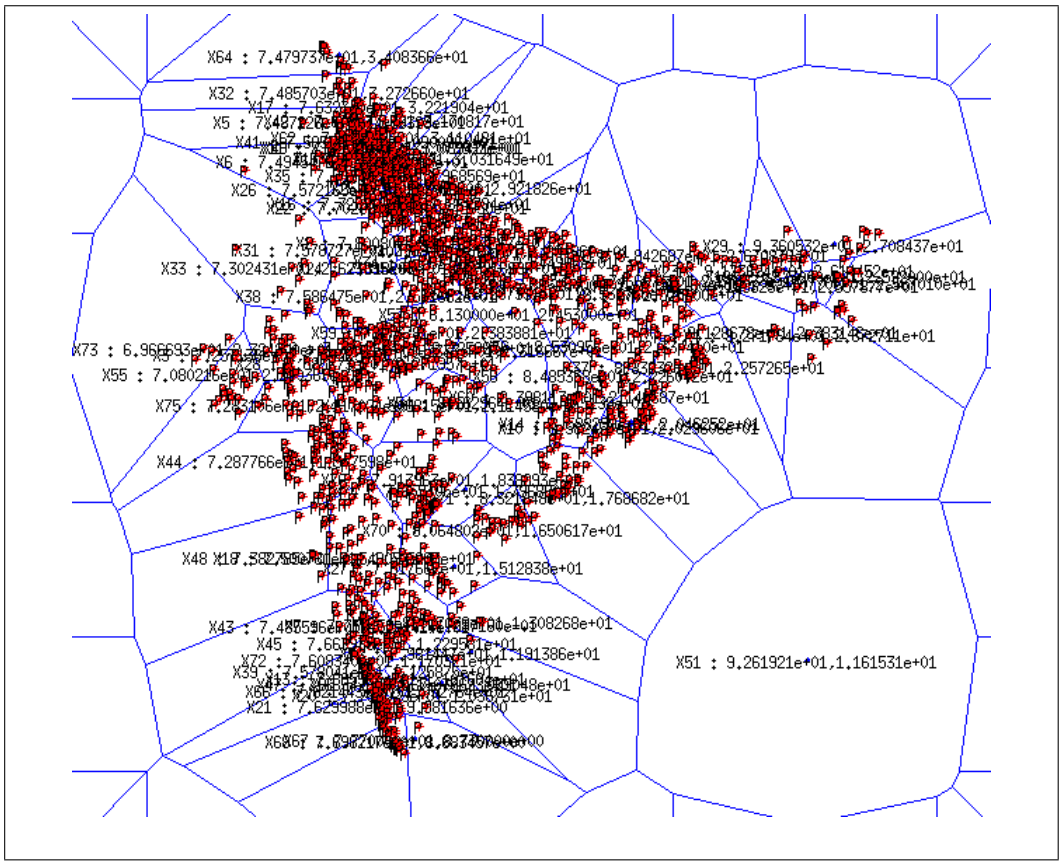

Figure 3.2: Voronoi Diagram

After all the mandis are mapped to their nearest centre, wholesale and arrival at every centre is computed. Wholesale price at centre is average of modal price of all mandis in its region and arrival was computed as the sum of the arrival at the mandis in its region. While calculating the wholesale price, the distance between centre and the mandi was not considered.

Corresponding to every date and location of hoarding news report, values like current year arrival, last year arrival, Percentage difference in wholesale-retail etc. were computed. Following table resulted from these computations.

3.6 How to Define Anomaly?

To answer this question, we went through the series of the news articles when the hoarding is in the news. Then looking over those articles, we try to see

that why they are reporting in news, what happened so that people are giving it name of hoarding and how reporters are making conclusion that it may be the class of the hoarding.

3.6.1 Summary Of News Atricles

First such incident was reported in the 1998. The article [14] dated on 21st September 1999 states as follows:

"Onions were retailed at Rs 6 a kilo two weeks back. Today, the price was almost 100 per cent up, hovering in the Rs 10-12 band in different parts of the country."

So this article is comparing the retail price of today with the price before 2 weeks. The rise upto 100% is what has come to notice. Also article says that,

"There is talk in the market that the government is likely to lift the ban on onion exports. Apparently, some traders are resorting to hoarding in anticipation of demand from markets abroad."

As stated previously also, government policy also plays a major role in this. After that Onion was in news in 2010. NDTV [15], TOI [5] and many more reported the incident. In 2010, unseasonal rainfall and the government policy on export price were also the reason for hike in the price. The report dated Dec 23, 2010, TOI states the follows (in Delhi):

"On Tuesday alone, wholesale traders in Delhi bought onions at about Rs 34 per kg while it was sold in retail at Rs 80 per kg. That's a margin of Rs 46 per kg or 135%!"

Here, they have compared the difference between the wholesale price and the retail price. The margin of 135% is reported. When we looked into data we have, we got the following results for Dec, 2010. (See figures 3.3 and 3.4)

So, as per our data, the maximum price difference observed was of $\tilde{1}00\%$. Note that the retail prices we have is the minimum price observed in the market.

NDTV on Dec 22, 2010 reported the following (NASIK):

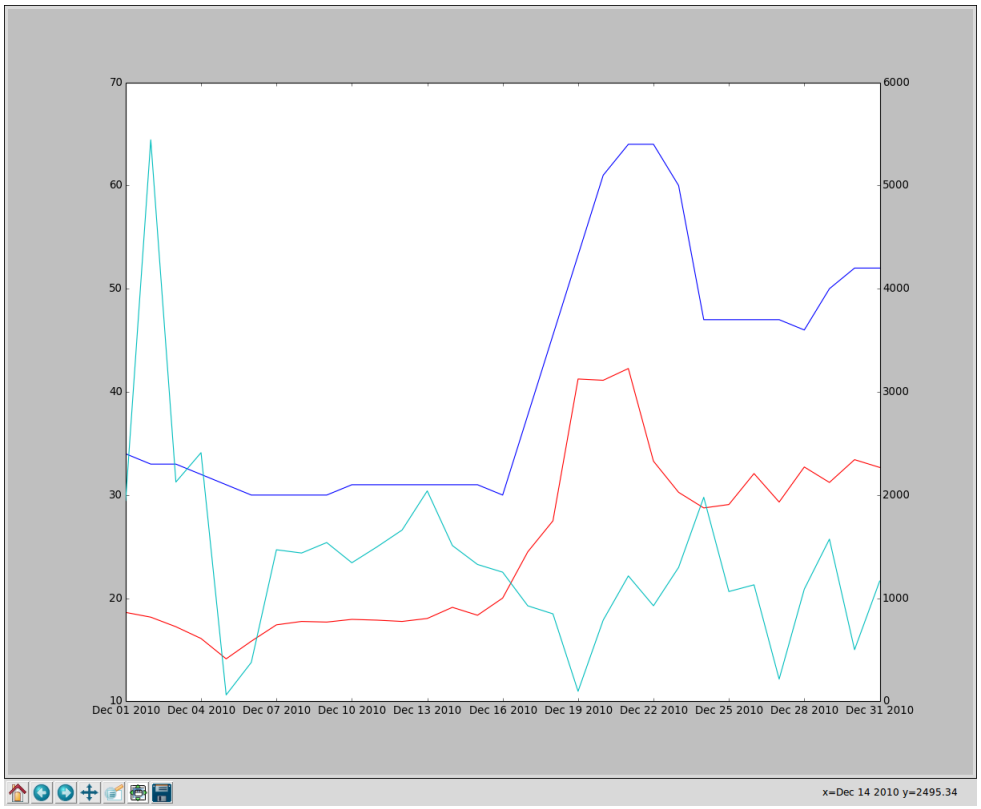

Figure 3.3: Delhi, Dec 2010. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival)

"The average purchase rate of a trader here is about Rs. 3,000, but they go to the cities and claim it's nearly Rs. 8,000 and that's how the rates go up. It's all the fault of the traders. They loot the people," said Sangdeorao Holkar, Director, National Agricultural Cooperative Marketing Federation of India.

Here also, as we see they have mentioned price difference between retail and wholesale in the market. It is approximately $\tilde{1}66\%$.

Report also adds the following:

The government, on a back foot, banned exports last evening. In less than 24 hours, prices in Lasalgaon crashed by 25 per cent.

Navi Mumbai: Wholesale price
Tuesday: Rs. 60 per kg

Figure 3.4: Delhi, Dec 2010. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival,Black - Relative difference %)

Wednesday: Rs. 45 per kg

And this is finally what the consumer in Mumbai is paying:

Mumbai: Retail price
Tuesday: Rs. 60 to Rs. 70 per kg
Wednesday: Rs. 60 to Rs. 70 per kg

So as we can see, there is significant drop in the wholesale price, but no drop in the retail price, so that has been reported. Also difference in the hike in retail price, from 40 to 60 was reported in on week. What our data says:

As we can see from the Figures 3.5, 3.6, 3.7 and 3.8, the difference between retail and wholesale went as much high as 200% in both overall Maharashtra as well as in the Mumbai. Also, as per report [12], this trend was also

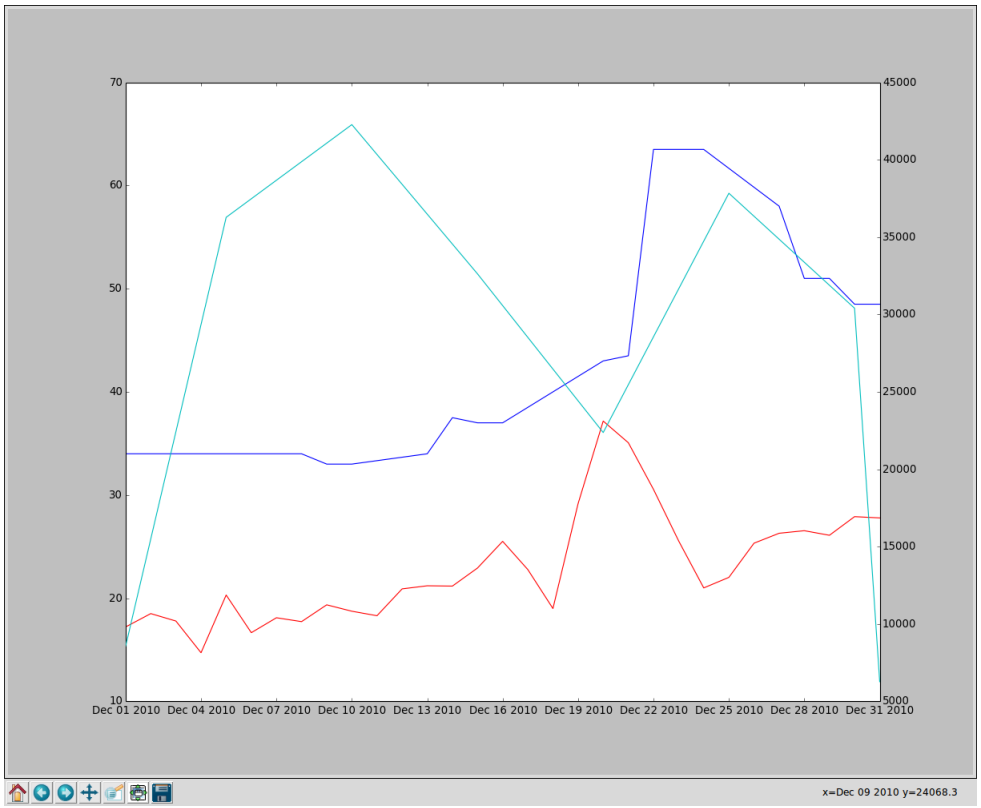

Figure 3.5: Maharashtra, Dec 2010. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival)

continued in the next month, i.e. January 2011, which can be seen in the following graph. (See figures 3.9 and 3.10)

Next hoarding news was reported in the last week July 2013. As per the Business Standard Report [3],

"Onion prices in Nashik, Pune and Ahmednagar have increased to Rs 2,400 a quintal as on July 21, compared to Rs 1,500-1,800 a quintal during the corresponding period of last year. Arrival of onions in Nashik, which contributes 35-40 per cent to the state production, has been 83,000 quintals compared with 82,000 quintals last year"

Also, In the month of August, September and October of 2013, the price

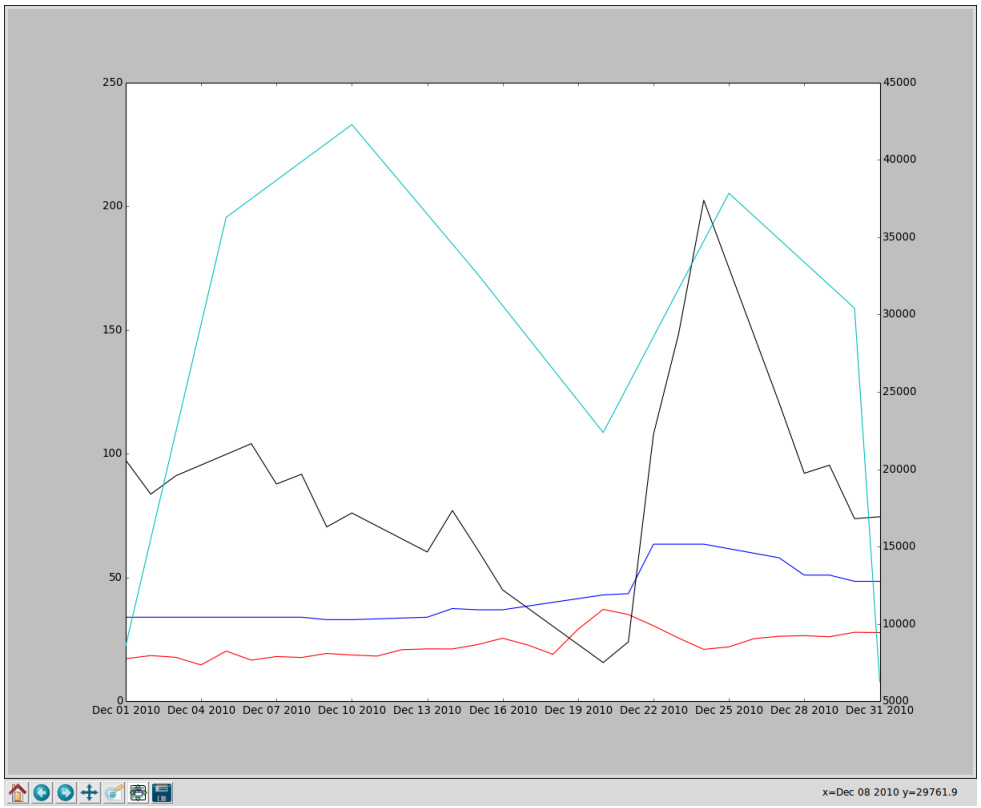

Figure 3.6: Maharashtra Dec 2010. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival, Black - Relative difference %)

rise of the Onion in various parts of countries like Banglore ([9]), New Delhi ([6],[10],[8]) and Gandhinagar ([[4]) was in the news. Reports of Banglore and New Delhi has just reported the hike in the retail price. DNA report on Gandhinagar says,

"Retail onion prices in the city have increased from around Rs. 40 per kg, a month ago, to Rs. 70/kg on Saturday. In the wholesale market, onion prices have increased around Rs. 35 per kg till last week to Rs. 45 per kg on Saturday. The wholesale price is Rs. 40 to 45 per kg. Ideally, in the retail market the price should not be more than Rs. 60 per kg"

Let's look at the data we have. As from the figures 3.11 and 3.12, the wholesale rates around the mandis present around the Mumbai in the month of July, 2012 was about Rs. 5/Kg, but during the same time period in the 2013,

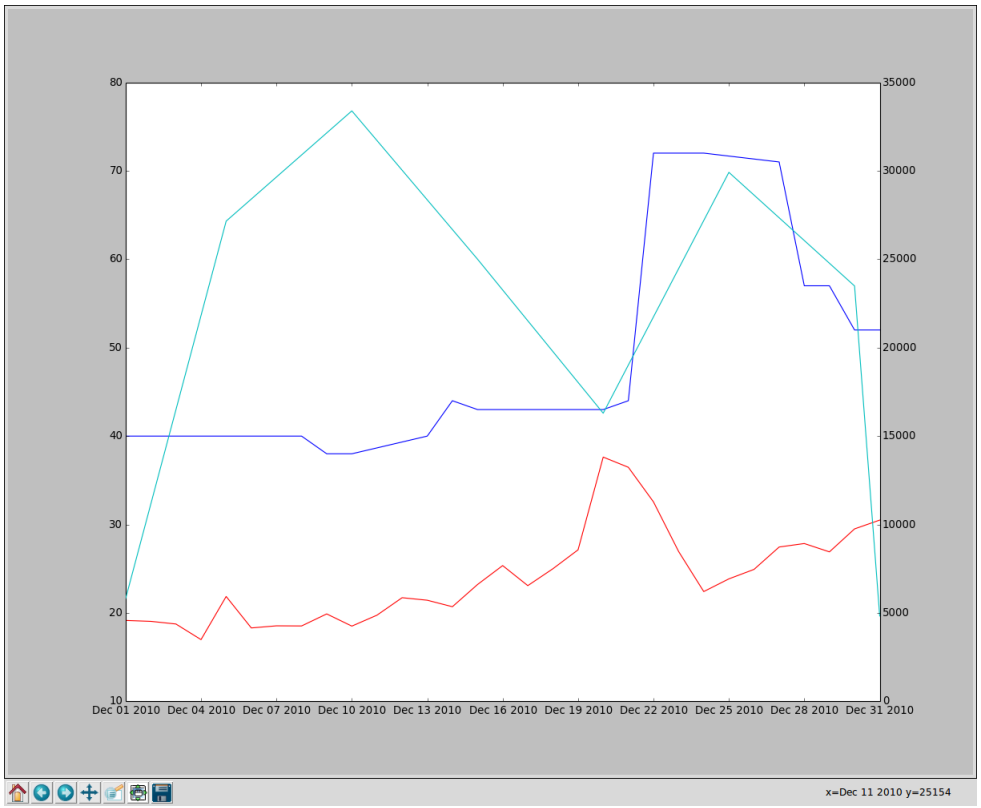

Figure 3.7: Mumbai , Dec 2010. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival)

the wholesale prices went from Rs. 15/Kg to Rs. 25/Kg. Although, there was decrement in the arrival by 7% as compared to July 2012, but the rise in the wholesale price is very much high.

Figures 3.13 and 3.14, states the scenario of Gujarat. There also the wholesale as well as the retail prices became suddenly almost double in the month of August 2013.

Also, in year 2014, price hike of Onions was in the news. There were reports from Mumbai [11], Kerala [7] and Hydrabad [13] which stated the hike in the prices of the onion. Let's look at the graph of the Mumbai for the year of 2014. As we can see from the graph (Figure 3.15) that for the period of July to October the wholesale price were decreasing, but still that was not

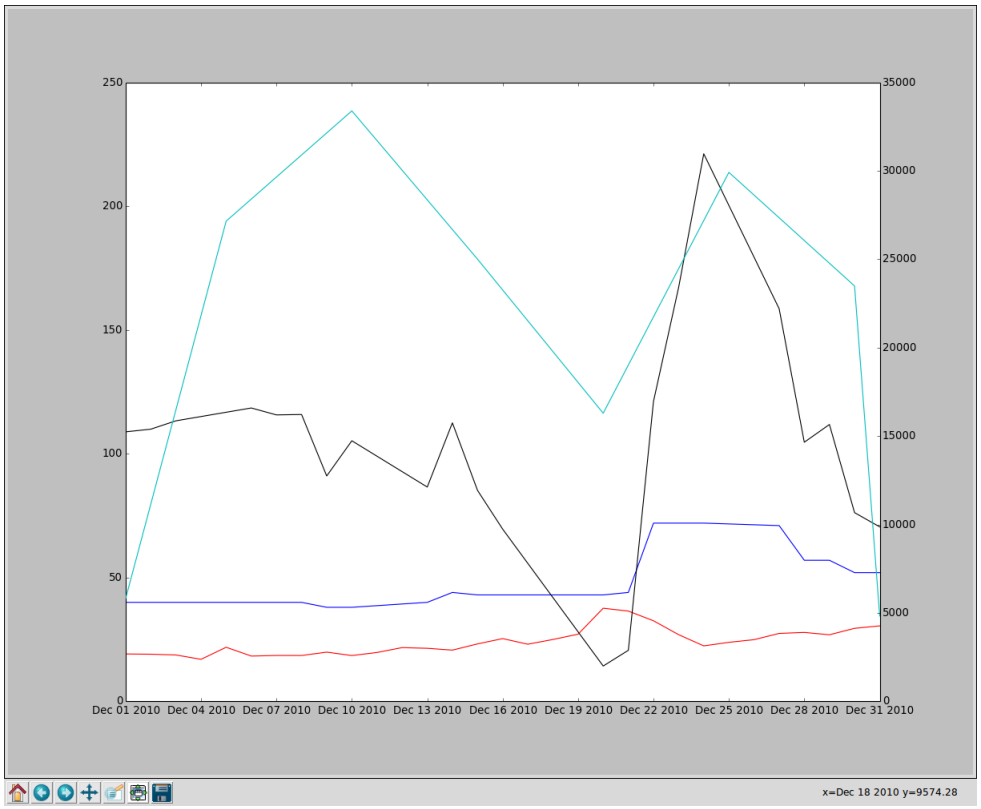

Figure 3.8: Mumbai, Dec 2010. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival Black - Relative difference %)

reflected in the retail price and instead of decreasing, the retail price kept on increasing.

3.7 Characteristics of anomaly

Hoarding of commodities in excess, results in anomaly. We segregated news articles on hoarding of onion and tried to spot some characteristics of data for anomalies.

So, major characteristics of hoarding spotted in newspapers are following:

1. Huge difference in wholesale and retail prices

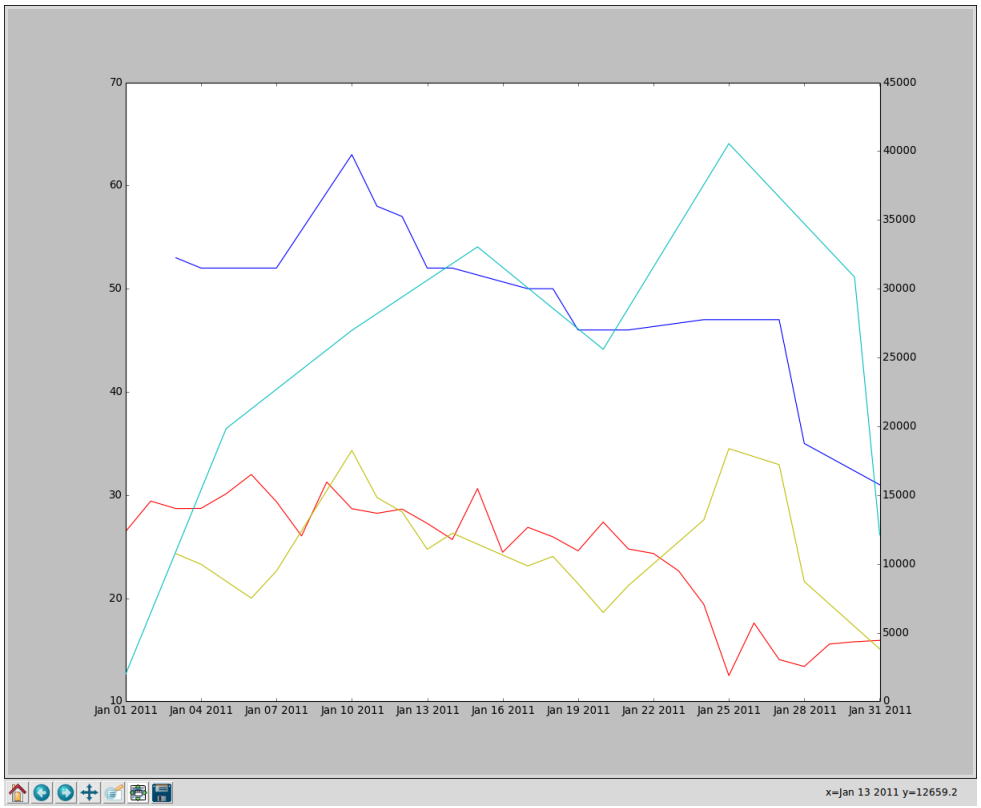

Figure 3.9: Mumbai , Jan 2011. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival)

2. Sudden rise in wholesale or retail price

3. Rise in wholesale prices when arrival is enough/high

We can also generalize anomaly cases as shown in table 3.1 and 3.2. The estimated arrival, wholesale and retail price data is labeled up, down and constant based on if it exceeds actual data value. Red flags are raised based on the following tables if the condition falls under the category marked with As.

3.8 Hypothesis

So from the above analysis of all news reports, we conclude four hypothesis.

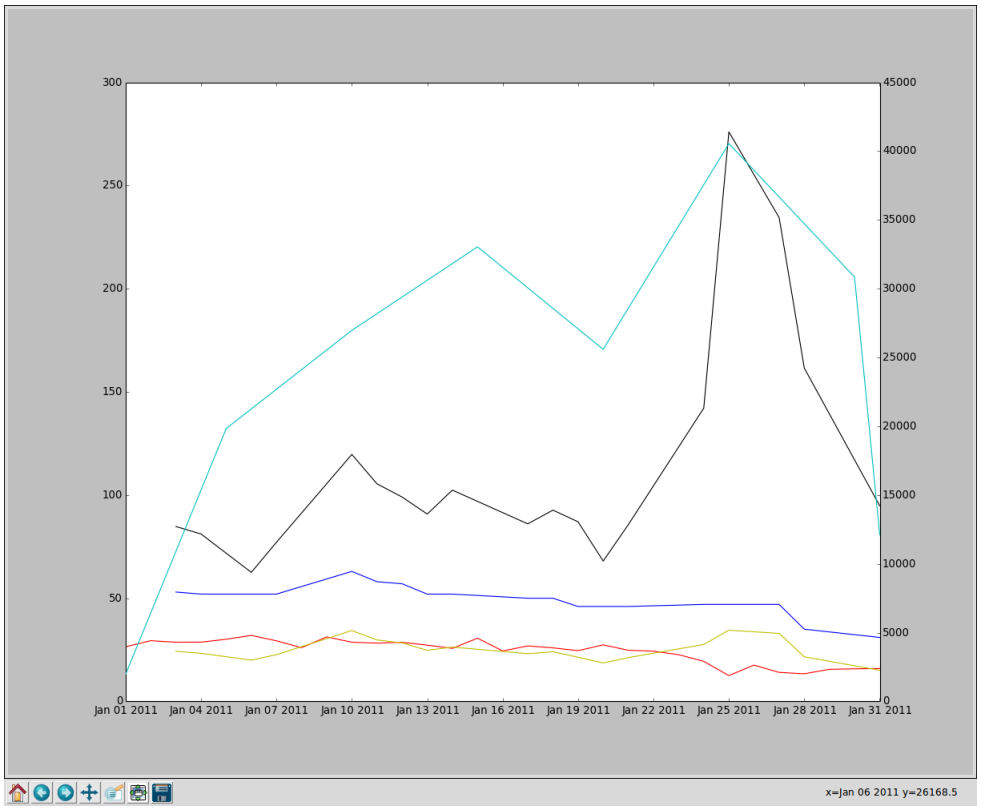

Figure 3.10: Mumbai, Jan 2011. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival Black - Relative difference %)

Wholesale price is inversely proportional to arrival in the market and we assume it is the only factor on which wholesale price is dependent. We also noticed in our literature review that most of the farmers don't hoard in India. So, from this relation, we have the following hypothesis:

H1. If there is increase in arrival pattern, there should be decrease in the wholesale price and if there is decrease in arrival pattern, there should be increase in the wholesale price consider-

W\A	↑	↔	↓
↑	-	A	A
↔	-	-	A
↓	-	-	-

Table 3.1: Anomaly Scenarios - 1

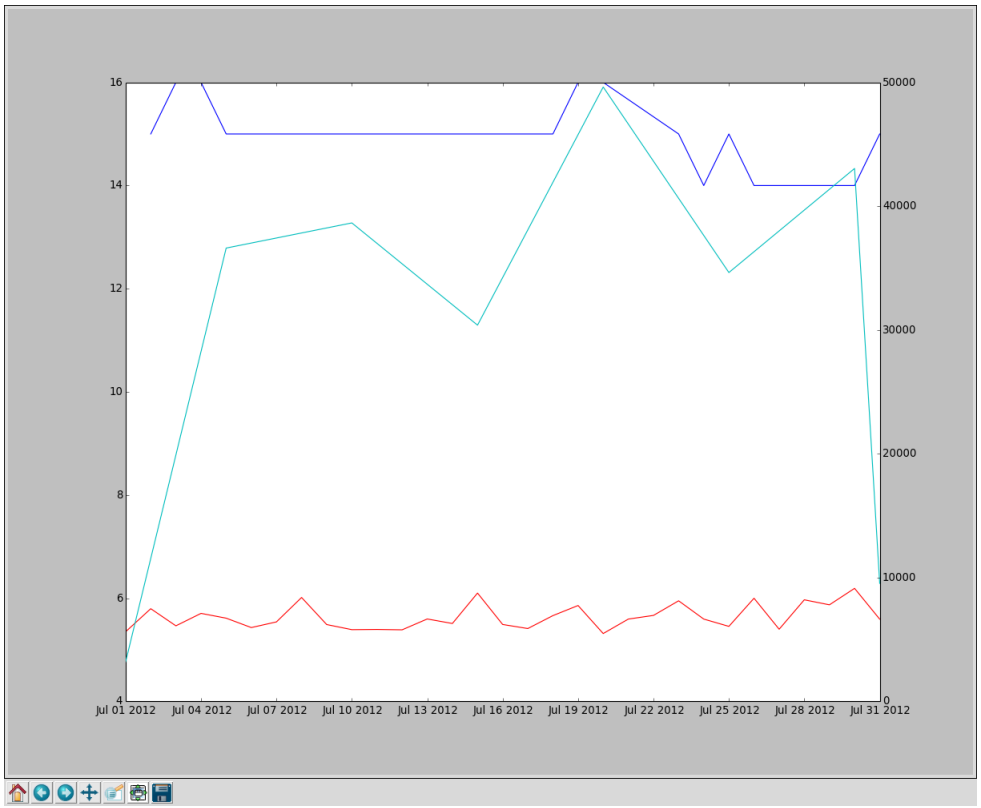

Figure 3.11: Mumbai , July 2012. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival)

ing a lag factor of 15 days.

Retailers purchase onions from the wholesale markets, mandis, etc. So, rate at retail level should be directly proportional to wholesale price in that region. We assume here that demand remains constant and there is no supply shock created because of excessive export of onion So from here we get the following hypothesis.

H2. If there is increase in the wholesale price of onion, then

W\A	↑	↔	↓
↑	A	A	-
↔	A	-	-
↓	-	-	-

Table 3.2: Anomaly Scenarios - 2

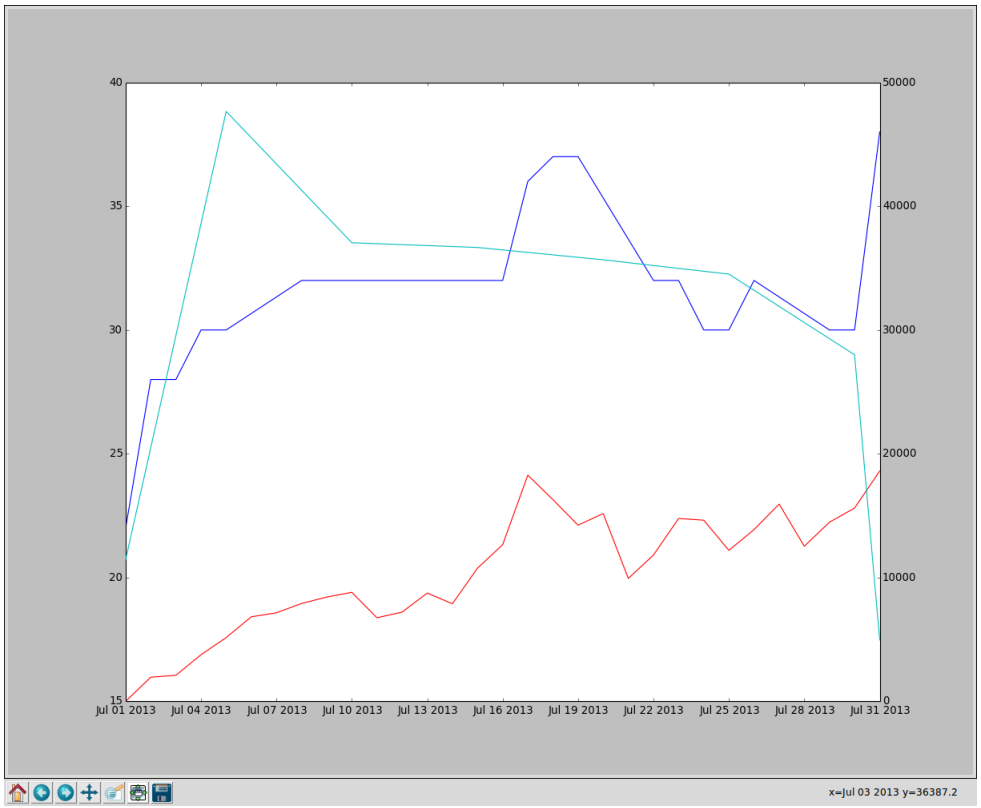

Figure 3.12: Mumbai , July 2013. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival)

there will be corresponding increase in the retail price and vice-versa assuming demand remains constant and there is no supply shock created because of excessive export of onion.

Similarly, we can state that if there is no change in the arrival for some period, then wholesale may remain same and if wholesale price remains same for some period than retail price may remain same. So from that we get following two hypothesis:

H3. The deviation between arrival - wholesale and wholesale-retail should not vary much compared to values for same time in past years.

Also, one should make a note of that, even in H1 and H2, when wholesale price or retail is increasing, then it should be in considerable amount. It should not be like, there is marginal increase in wholesale price and retail

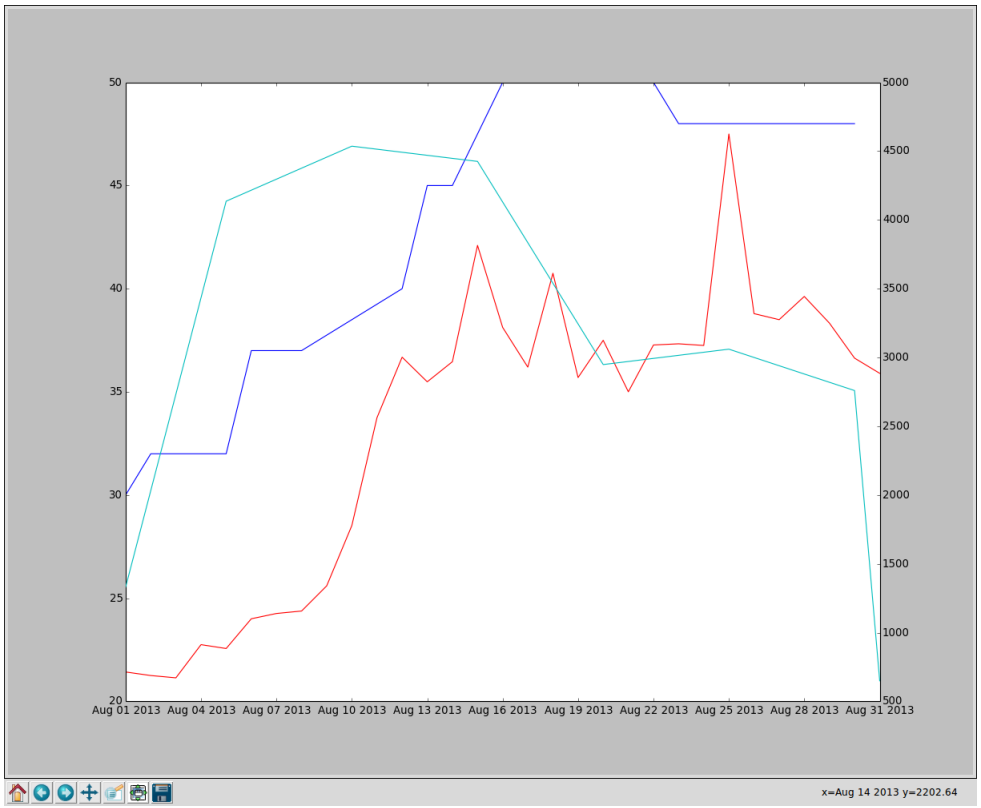

Figure 3.13: Ahmedabad , Aug 2013. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival)

price is boosting up or there is little decrement in arrival and wholesale price goes up by unacceptable level.

Also, we assume that mandis in the same region will behave in similar manner, because production and effect of other factors will be same in one region. So, based on that we have following hypothesis:

H4. Mandis in the same region should follow the same relationship between arrival and wholesale price, as that of, taken whole region combinely.

OR

Retail prices across various centres should follow same relationship among themselves taken effect of them combinely.

So we have created a library which takes all these series as an input and try

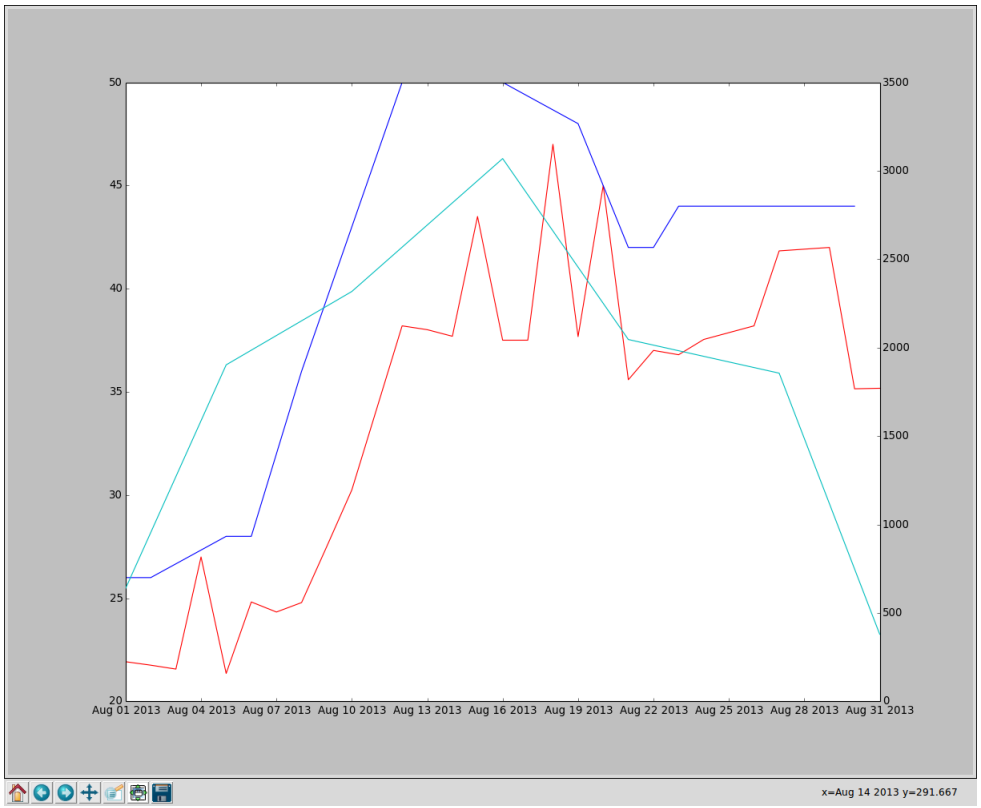

Figure 3.14: Rajkot , Aug 2013. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival)

to detect possibilities of above stated anomalies or highlight all the important unusual behavior of parameters seen in the input series.

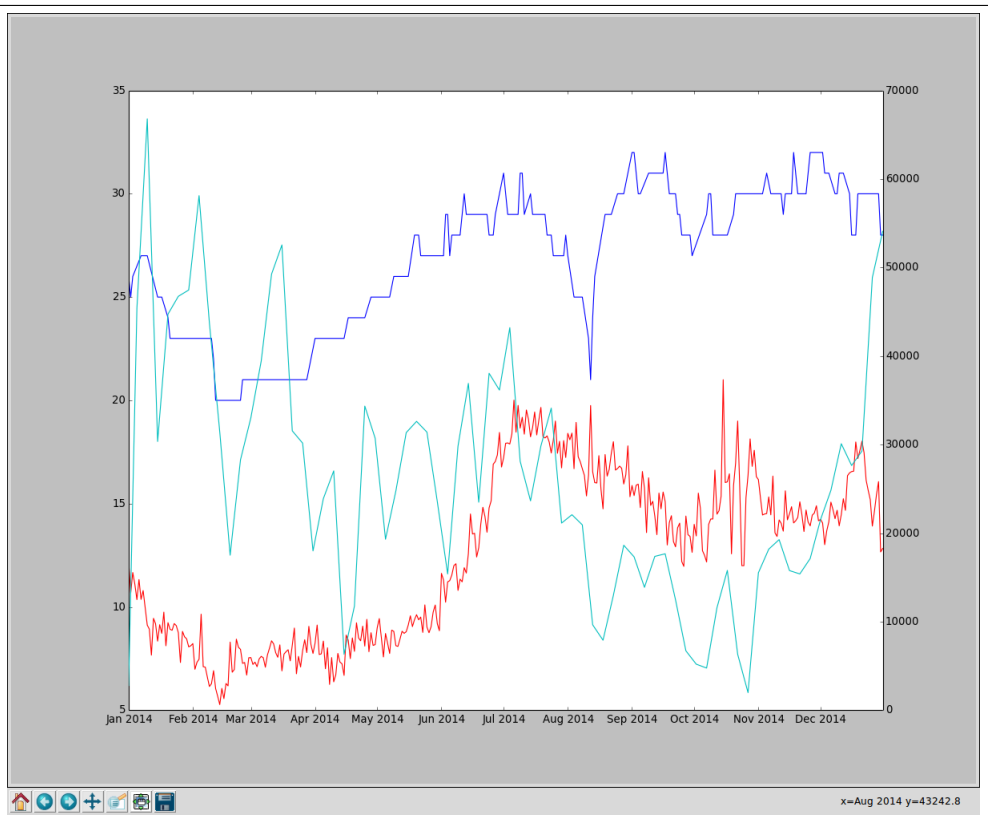

Figure 3.15: Mumbai , 2014. (Blue - Retail price, Red - Wholesale Price, Cyan - Arrival)

Chapter 4

Design and Framework

4.1 System Design

Figure 4.1 explains the overall design of the system and its modules.

We have multiple time-series as input. These time-series are to be read from file manually using python script and generate 2D list of the time-series. These 2D lists has first column as date and then we have one or more columns corresponding to one or more values for that date. In the next stage depending upon what time-series we need to compare, we select appropriate time-series and pass this as input to various functions. Note that it is required to have all time-series of equal length and time period for which time-series are considered should be same for all time-series.

We have implemented 5 functions till now as shown in figure 4.1 viz. Correlation, Slope Based, Linear Regression, Graph Based and Multivariate. Detail about each function is explained in the next section. These functions takes either 2 or multiple time-series as input and has various parameters. Depending upon how time-series should behave with respect to each other (directly propotional or inversaly propotional) parameteres are tuned. We take union of results of first three functions and rest of the two functions as shown in figure. After taking union, we take intersection of two results of union as shown in figure. This is final result produced by the system. This result states the anomalies in the time-series.

To verify result produced by system, we match it with the news articles present for the test case considered (here onion test case). We match system results with news articles present and check how well system is performing. Note that it is not necessary that for each and every anomaly case, we have

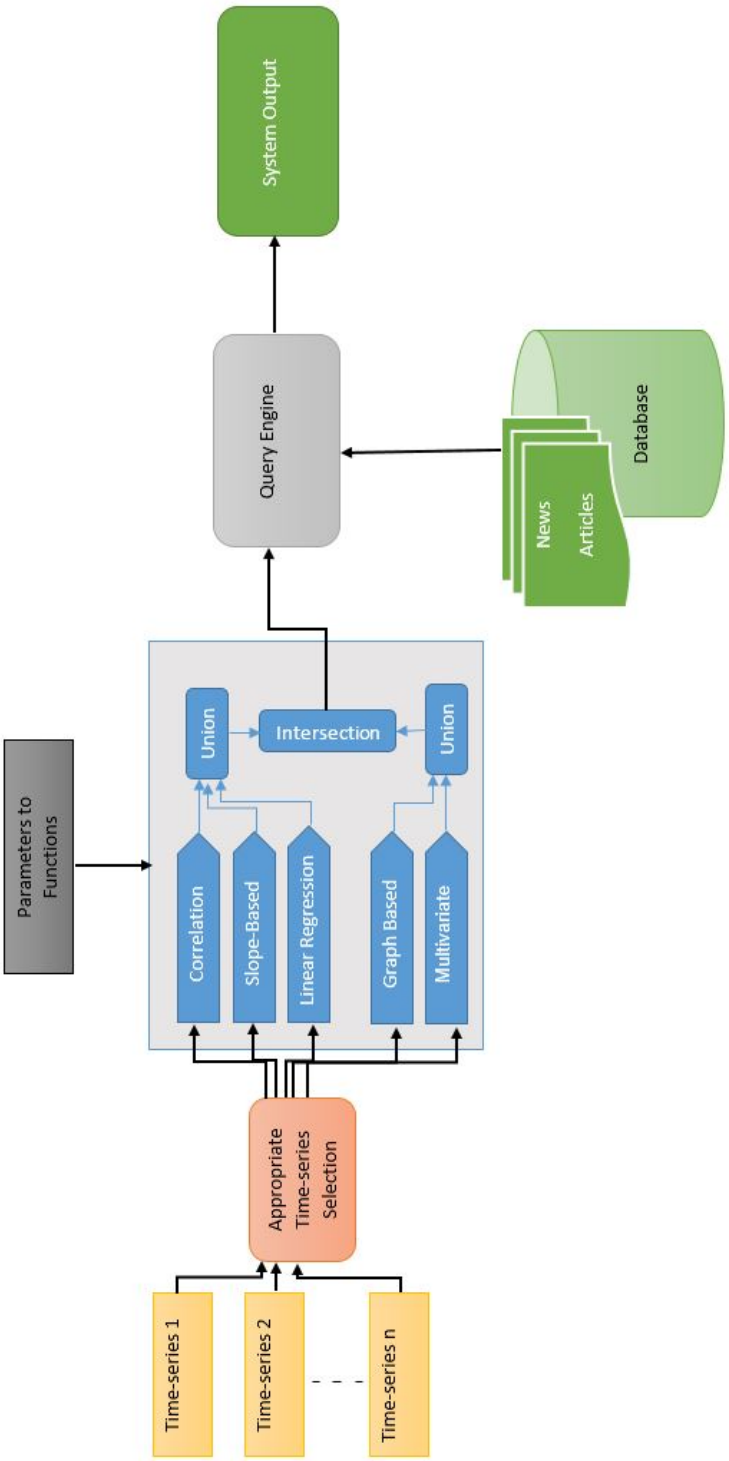

Figure 4.1: System Framework

news article present. If not, then we manually check behavior of time-series on that date. If some news articles do not match with system result then we also study, why they did not match and what are limitations of system.

4.2 Test Criterion for Hypothesis

In the last chapter, we stated some hypothesis. Now, here we state what tests can be performed to detect anomalies for each of the hypothesis.

4.2.1 Hypothesis 1

In this, we use three functions:

- Starting with granularity of year-wise, find out cross-correlation, considering various lag factor up to 15, between arrival and wholesale and check that overall it is positive or negative.
- Go on decrease the granularity. Next will be season-wise. For a particular season, how arrival and wholesale price are behaving. Point out where, this correlation becomes positive.
- Similar thing can be done for month-wise as well as fortnight data.
- Note that, correlation mentioned here is just one of the method. There can be various other methods to define the behavior of 2 time-series.
- **Slope Based Detection:** Calculate the slope of both time-series day-wise and compare. *Assumption:* Data is smoothed. Since day-wise is taken, if data is not smoothed then it may generate many spikes. So if data is not smoothed, then before applying this technique, either apply smoothing or take weekly average. If granularity of data is daywise, then smoothing can help. If data is reported once-twice weekly, then taking average weekly and then calculating slope day-wise may help.

- **Linear Regression Based Method:** We can train model using linear regression. Then find out the difference between actual and predicted value for each of the data point and plot a histogram. From these points we can try to find out points which are going very much out of the way using method like MAD test.

4.2.2 Hypothesis 2

- Starting with granularity of year-wise, find out cross-correlation, considering various lag factor up to 15, between arrival and wholesale and check that overall it is positive or negative.
- Go on decrease the granularity. Next will be season-wise. For a particular season, how arrival and wholesale price are behaving. Point out where, this correlation becomes positive.
- Similar thing can be done for month-wise as well as fortnight data.
- Note that, correlation mentioned here is just one of the method. There can be various other methods to define the behavior of 2 time-series.
- **Slope Based Detection:** Calculate the slope of both time-series day-wise and compare. *Assumption:* Data is smoothed. Since day-wise is taken, if data is not smoothed then it may generate many spikes. So if data is not smoothed, then before applying this technique, either apply smoothing or take weekly average. If granularity of data is daywise, then smoothing can help. If data is reported once-twice weekly, then taking average weekly and then calculating slope day-wise may help.
- **Linear Regression Based Method:** We can train model using linear regression. Then find out the difference between actual and predicted value for each of the data point and plot a histogram. From these points we can try to find out points which are going very much out of the way using method like MAD test.
- Spike Detection methods

4.2.3 Hypothesis 3

- One can use prediction based model like ARIMA [19] to test this hypothesis. If method mentioned in [19] is used then there is no need to align time-series into phase, as method mentioned in it take care of it. One can train the model considering some period for 8 years and can be tested on remaining 2 years. Difference in actual and predicted value is seen and one above some threshold value is reported. Also, while generating model, different window size can be considered. If some other technique is used, then one may need to align time series into common phase. That can be done using cross-correlation method, considering lag factor of around 15 days. Lag with the highest correlation value is considered.
- One can also apply graph based anomaly detection technique as mentioned in [17] to find out malicious behavior.

4.2.4 Hypothesis 4

Here, we need to combine data of multiple places into one, to find out the combined behavior and then compare it with each of the mandi. One of the technique, to combine multiple series is,

- For Arrivals: take sum from all Mandis
- For Wholesale-price: Take average
- Apart from center, if one wants to combine statewise, then for retail price also, one can take average for retail price

Other technique, to combine multiple time series is, subspace based transformation, as mentioned in [16]. Then, to compare behavior of each particular mandi time-series with the aggregated one, techniques mentioned to test hypothesis 1 or 2 can be used.

Note (Applicable to Onion Case): Results returned by H1 and H2, consists of all the time periods where arrival-wholesale price and wholesale

price-retail price pairs go out of line. There may be case where, for each time of that year it may be going out of line and might not be anomaly. So, to remove such false positives, we can take intersection of results of H1 and results of H3. Similar thing can be done with H2 also, by taking intersection of results of H2 and results of H3.

4.3 Anomaly Detection Library

Following methods are implemented in the library to detect anomalies in given timeseries:

4.3.1 Window Based Correlation

The method finds anomalies between two timeseries. If the two timeseries are supposed to be positively correlated, then it reports all the tenures where the timeseries deviates from showing positive correlation between them.
Similarly if the two timeseries are supposed to be negatively correlated, then it reports all the tenures where the timeseries deviates from showing negative correlation between them. The significant deviations are reported based on either the default significance value of 0.01 or it can be customized based on the need.

4.3.2 Slope Based Detection

The method calculates the rate of change of one timeseries with respect to the other. The tenures are reported where one timeseries varies to a large extent from the other.
The threshold is decided based on the MAD outlier detection test or it could be set manually.

4.3.3 Linear Regression

The method builds a linear model between two timeseries. Values are predicted based on this linear model and error is calulated. All the tenures with error values breaching the threshold are reported as anomaly.
The threshold can be decided by MAD outlier detection technique or it can be configured manually.

4.3.4 Graph Based Anomaly

Graph based anomaly detection technique considers each day as a node of a graph. Similar nodes are connected to each other by some weight. Similarity of nodes are calculated by making use of the values of that node i.e. value(s) of timeseries on that date. Based on this similarity, edge weights are also assigned. Then random walk algorithm is applied on this graph structure and connectivity value of each node is calculated. Graph nodes having the least connectivity values are reported as anomaly.
The method also takes into account the trend, seasonality etc of timeseries. Top few points (can be configured) are reported as anomaly by the system.

4.3.5 Multivariate- Vector Autoregressive

The method uses vector autoregressive framework for multivariate time-series analysis in order to forecast values. The framework treats all the variables as symmetrical and all the variables are modeled as if they influence each others equally.
Error percentage is calculated between actual and forecasted values. These error values are used to filter anomalous conditions from non-anomalous conditions by setting a threshold value which can be automatically computed by MAD outlier detection test or it can be manually configured to the required value.

Chapter 5

Results and Analysis

We executed our library functions on the onion data. This data consists of Wholesale Price, Retail Price and Arrival since 1st January 2006 to 6th July 2015. In this chapter, we will show results produced by our system and will analyse these results along with each method.

5.1 Results

We have performed 4 types of analysis and result for each of this method is as follows. Note that these are primary results. Data for 2 centres are considered - Mumbai and Delhi.

Here are some results related with Mumbai Center. Table 5.1 shows the result of anomalies reported by our system, with details about anomalies reported by each method. So here First 5 columns corresponds to each method. Column 6 is union of results of first 3 methods and column 7 is union of results of method 4 and 5, as described in table. Column 8 is intersection of results of column 6 and column 7, which is final result of our system.

Table 5.2 shows the result of number of articles matched with the dates reported by our system as anomaly for each method. So here First 5 columns corresponds to each method. Column 6 is union of results of first 3 methods and column 7 is union of result of method 4 and 5, as described in table. Column 8 is intersection of results of column 6 and column 7, which is final result of our system.

Note that total number of articles present for center Mumbai is **99** and all these articles are present after 2010. Apart from Graph Based Anomaly and Multivariate- vector autoregressive method, all methods are producing

results from 2006 onwards as input data is from that time. The following pie chart(see figure 5.1) shows the analysis of article showing what news articles states as the reason for the price hikes of onion.

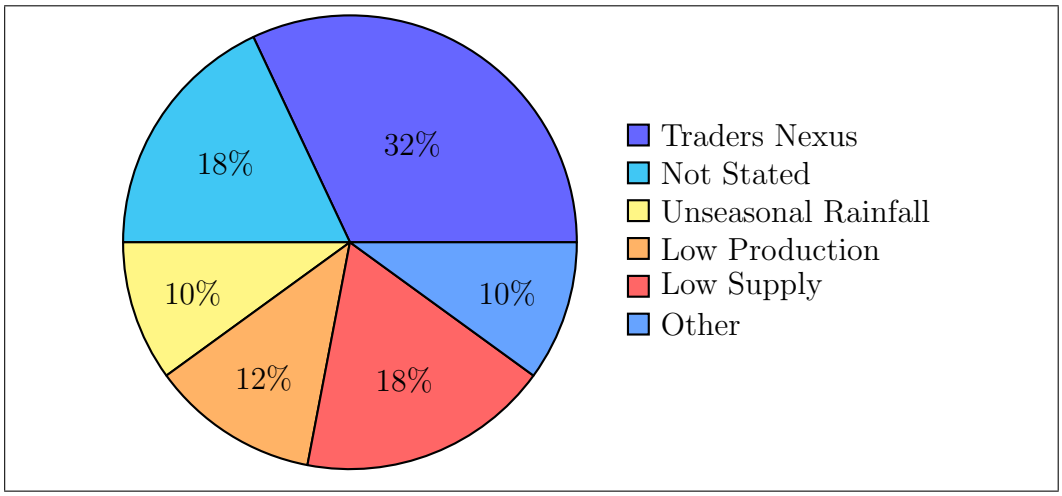

Figure 5.1: Reasons stated by news articles for onion price hike

While finding the news article match for every anomaly, we check for all the news articles 5 days before and 5 days after the reported date. In case we find any news article in this tenure, we consider it as a match for that date. For one date of anomaly, multiple articles may be present but for this analysis we have considered only the nearest article for that anomaly date. Following bar charts represents distribution of how far is the news article from the date of anomaly.

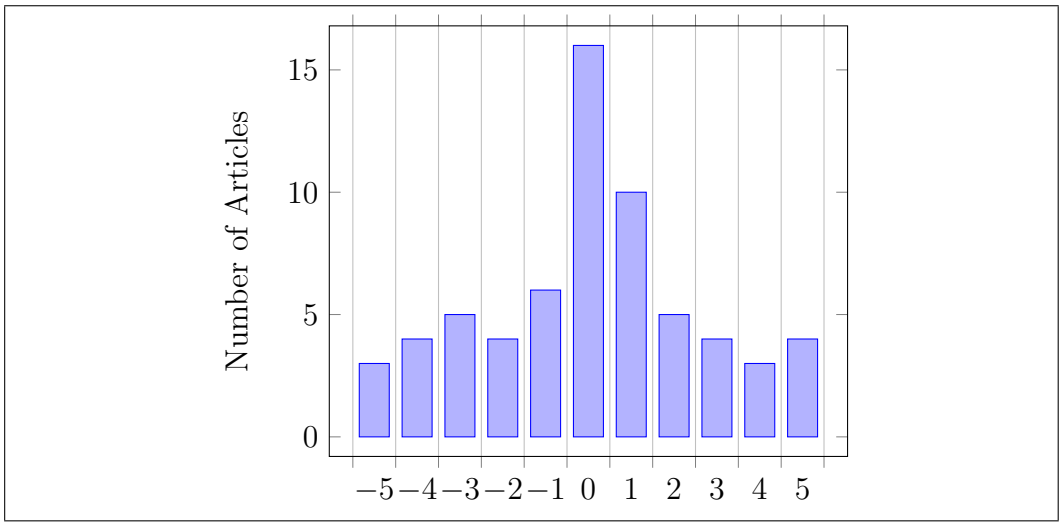

Figure 5.2: Article Distribution for Retail Price VS Average Retail Price

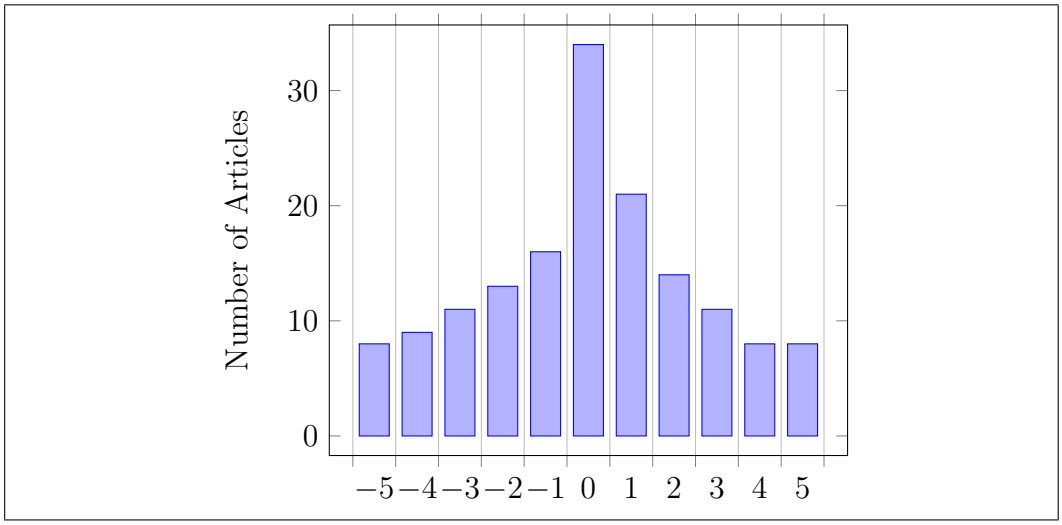

Figure 5.3: Article Distribution for Retail Price VS Arrival Of Onion

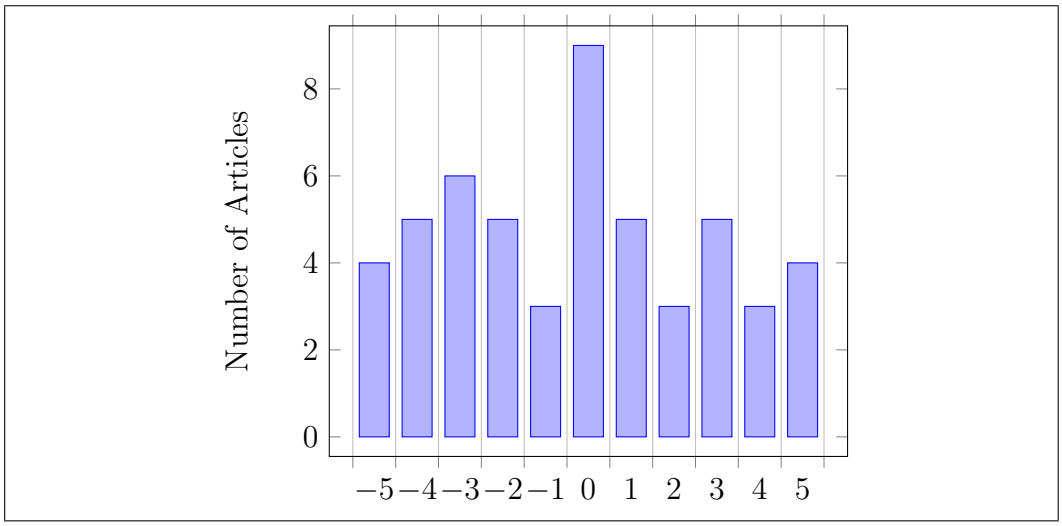

Figure 5.4: Article Distribution for Wholesale Price VS Retail Price

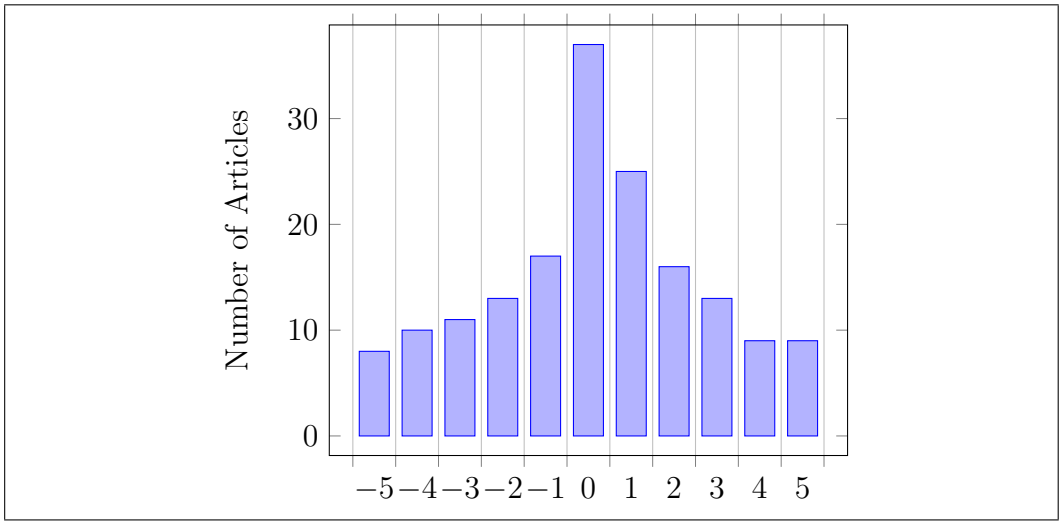

Figure 5.5: Article Distribution for Wholesale Price VS Arrival Of Onion

Now, we present detailed analysis for each of the different type of time-series. First type of such analysis is in table 5.3. This table shows distribution of news articles present (that matched with system results) year-wise for each method when retail price time series is compared with average retail price time series. Second type of such analysis is in table 5.4. This table shows distribution of news articles present year-wise for each method when retail price time series is compared with arrival data of onion time series. Third type of such analysis is in table 5.5. This table shows distribution of news

Methods	Slope Based (1)	Correlation (2)	Linear Regression (3)	Graph Based (4)	Multivariate (5)	1 U 2 U 3 (6)	4 U 5 (7)	6 ∩ 7
Retail Vs Average	742	255	353	300	177	1206	362	125
Retail Vs Arrival	420	795	353	500	167	1381	573	323
Retail Vs Wholesale	658	420	310	300	167	1243	367	160
Wholesale Vs Arrival	448	705	282	500	186	1315	586	332

Table 5.1: Anomalies Reported

Methods	Slope Based (1)	Correlation (2)	Linear Regression (3)	Graph Based (4)	Multivariate (5)	1 U 2 U 3 (6)	4 U 5 (7)	6 ∩ 7
Retail Vs Average	67	47	50	122	122	142	162	64
Retail Vs Arrival	42	74	167	121	119	220	159	153
Retail Vs Wholesale	30	55	40	117	119	107	150	52
Wholesale Vs Arrival	64	64	174	122	139	219	174	168

Table 5.2: Number of news articles matched with system

articles present year-wise for each method when retail price time series is compared with wholesale price time series. Fourth type of such analysis is in table 5.6. This table shows distribution of news articles present year-wise for each method when wholesale price time series is compared with arrival data of onion time series.

Distribution of anomalies present year-wise, for each method is also shown in table. Result for various analysis is described in tables 5.7, 5.8, 5.9 and 5.10.

Such results for different cities can also be calculated.

5.2 Analysis of Each Method

In this section, we try to analyse strengths, weaknesses and limitations of every method. So, we will describe each method one by one. Note that we have articles from 2010 onwards, so we will be focussing on anomalies reported after 2010 and compare them with the news articles we have.

Note that all the graphs and results described in following sections

Distribution of All Articles	Articles Present	Slope Based (1)	Correlation (2)	Linear Regression (3)	Graph Based (4)	Multivariate (5)	1 U 2 U 3 (6)	4 U 5 (7)	6 ∩ 7
2010	6	0	0	8	10	0	8	10	3
2011	3	0	0	0	12	0	0	12	0
2012	2	1	0	0	0	0	1	0	0
2013	54	42	14	18	86	119	61	125	61
2014	23	24	0	24	14	3	39	15	0
2015	11	0	33	0	0	0	33	0	0

Table 5.3: Retail Price VS Average Retail Price

Distribution of All Articles	Articles Present	Slope Based (1)	Correlation (2)	Linear Regression (3)	Graph Based (4)	Multivariate (5)	1 U 2 U 3 (6)	4 U 5 (7)	6 ∩ 7
2010	6	17	0	17	17	0	17	17	17
2011	3	1	0	12	12	0	12	12	12
2012	2	0	0	0	0	0	0	0	0
2013	54	10	39	126	92	119	137	130	124
2014	23	14	35	12	0	0	54	0	0
2015	11	0	0	0	0	0	0	0	0

Table 5.4: Retail Price VS Arrival data of onion

Distribution of All Articles	Articles Present	Slope Based (1)	Correlation (2)	Linear Regression (3)	Graph Based (4)	Multivariate (5)	1 U 2 U 3 (6)	4 U 5 (7)	6 ∩ 7
2010	6	6	0	2	11	0	6	11	6
2011	3	1	0	0	4	0	1	4	1
2012	2	2	7	0	0	0	7	0	0
2013	54	7	21	18	91	119	45	124	44
2014	23	14	27	20	11	0	48	11	1
2015	11	0	0	0	0	0	0	0	0

Table 5.5: Retail Price VS Wholesale Price

Distribution of All Articles	Articles Present	Slope Based (1)	Correlation (2)	Linear Regression (3)	Graph Based (4)	Multivariate (5)	1 U 2 U 3 (6)	4 U 5 (7)	6 ∩ 7
2010	6	17	0	17	17	0	17	17	17
2011	3	1	0	12	12	0	12	12	12
2012	2	0	0	0	0	0	0	0	0
2013	54	25	15	125	93	123	129	129	123
2014	23	21	49	20	0	16	61	16	16
2015	11	0	0	0	0	0	0	0	0

Table 5.6: Wholesale Price VS Arrival data of onion

Distribution of All Articles	Slope Based (1)	Correlation (2)	Linear Regression (3)	Graph Based (4)	Multivariate (5)	1 U 2 U 3 (6)	4 U 5 (7)	6 ∩ 7
2006	133	30	65	0	0	204	0	0
2007	14	15	30	0	0	53	0	0
2008	42	15	0	0	0	47	0	0
2009	82	15	0	0	0	96	0	0
2010	72	45	28	37	0	142	37	13
2011	56	60	115	57	0	182	57	9
2012	84	0	33	13	0	100	13	5
2013	77	15	56	134	161	125	189	90
2014	140	0	26	40	16	155	47	8
2015	42	60	0	19	0	102	19	0

Table 5.7: Distribution of Anomalies reported by system for Retail Price VS Average Retail Price

Distribution of All Articles	Slope Based (1)	Correlation (2)	Linear Regression (3)	Graph Based (4)	Multivariate (5)	1 U 2 U 3 (6)	4 U 5 (7)	6 ∩ 7
2006	35	185	0	0	0	199	0	0
2007	63	85	0	0	0	148	0	0
2008	28	105	0	0	0	126	0	0
2009	28	45	0	0	0	73	0	0
2010	48	60	46	143	0	107	143	56
2011	36	60	40	165	0	128	165	65
2012	77	45	0	81	0	118	81	25
2013	59	90	168	111	161	263	178	172
2014	46	105	99	0	6	204	6	5
2015	0	15	0	0	0	15	0	0

Table 5.8: Distribution of Anomalies reported by system for Retail Price VS Arrival data of onion

Distribution of All Articles	Slope Based (1)	Correlation (2)	Linear Regression (3)	Graph Based (4)	Multivariate (5)	1 U 2 U 3 (6)	4 U 5 (7)	6 ∩ 7
2006	126	0	0	0	0	126	0	0
2007	77	30	0	0	0	107	0	0
2008	70	15	0	0	0	85	0	0
2009	63	75	0	0	0	126	0	0
2010	62	30	2	53	0	92	53	8
2011	50	45	67	75	0	155	75	34
2012	72	82	64	7	0	160	7	0
2013	50	38	44	105	161	124	166	81
2014	60	75	99	60	6	182	66	37
2015	28	30	34	0	0	86	0	0

Table 5.9: Distribution of Anomalies reported by system for Retail Price VS Wholesale Price

Distribution of All Articles	Slope Based (1)	Correlation (2)	Linear Regression (3)	Graph Based (4)	Multivariate (5)	1 U 2 U 3 (6)	4 U 5 (7)	6 ∩ 7
2006	0	65	0	0	0	65	0	0
2007	42	100	0	0	0	142	0	0
2008	21	120	0	0	0	141	0	0
2009	56	60	0	0	0	109	0	0
2010	83	75	51	142	0	162	142	62
2011	64	60	34	165	0	157	165	67
2012	42	45	0	81	0	87	81	12
2013	80	45	157	112	152	246	164	157
2014	46	105	40	0	34	162	34	34
2015	14	30	0	0	0	44	0	0

Table 5.10: Distribution of Anomalies reported by system for Wholesale Price VS Arrival data of onion

are for centre Mumbai. Mumbai is in Maharashtra state, which is the largest producer of onion in India. Also in graphs, Yellow highlighted region corresponds to anomalies reported by system, red region corresponds to dates for which our system reported anomaly and news article was present for that and blue region corresponds to date for which news article was present but that date was not reported as anomaly by our system.

5.2.1 Slope Based Anomaly Detection

The main functionality of this method is to find change in one variable with respect to other. Given two time-series, here we try to find, between two points in time series, how much dependent variable changed corresponding to independent variable. If this change is huge, then it is reported as anomaly.

We have four types of analysis which are as following:

1. **Retail Price vs Average of Retail Price**: Here, we first take average of retail price at all centres and than compare change in retail price with respect to change in average of retail price for different time window.

2. **Retail Price vs Arrival of Onion**: Here, we try to find change in retail price with respect to change in arrival of onion for different windows.

3. **Retail Price vs Wholesale Price**: Here, retail price is dependent on wholesale price and we try to find change in retail price with respect

to change wholesale price for different windows.

4. **Wholesale Price vs Arrival of Onion**: Here, we try to find change in wholesale price with respect to change in arrival of onion for different window size.

So, in each of the case, we try to find change with respect to another, and if this change is huge, crossing threshold than it is reported as anomaly. Note that in analysis 1 and 3 stated above, both the time series are directly proportional to each other and in the analysis 2 and 4 both the time series are inversely proportional to each other. So, limitations faced by this method for analysis 1 and 3 will be similar and for analysis 2 and 4 will be similar. While describing this method, each analysis will be referenced by its corresponding number.

First we will start with analysis 1 and 3. Following are the few observations:

- In case of analysis 1, dates are reported as anomalies if retail price at centre is increasing more as compared to average retail price and in case of analysis 3, it is reported if retail price at centre is increasing more as compared to wholesale price.
  Following tenures are some cases reported by this method:
  - *Analysis 1*: June 2010, August 2010, May 2011, June 2011, May Jun Nov 2012, Apr May 2013 (Prices went too high as compared to average) (See Figure 5.6)
  - *Analysis 3*: Apr July Oct Dec 2010, Jan 2011, May June 2014 (See Figure 5.7)

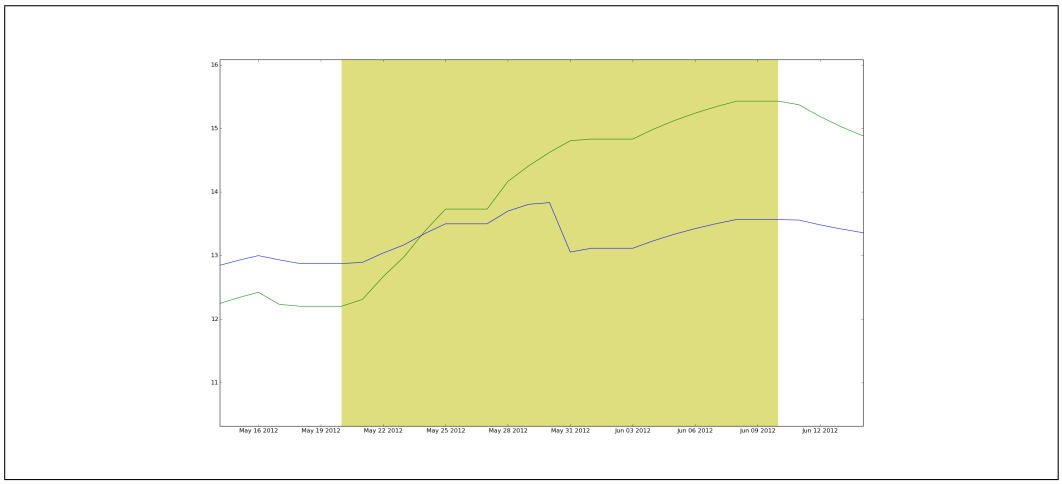

Figure 5.6: Slope Based Anomaly Detection (Green line - Centre Retail Price, Blue Line - Average Retail Price)

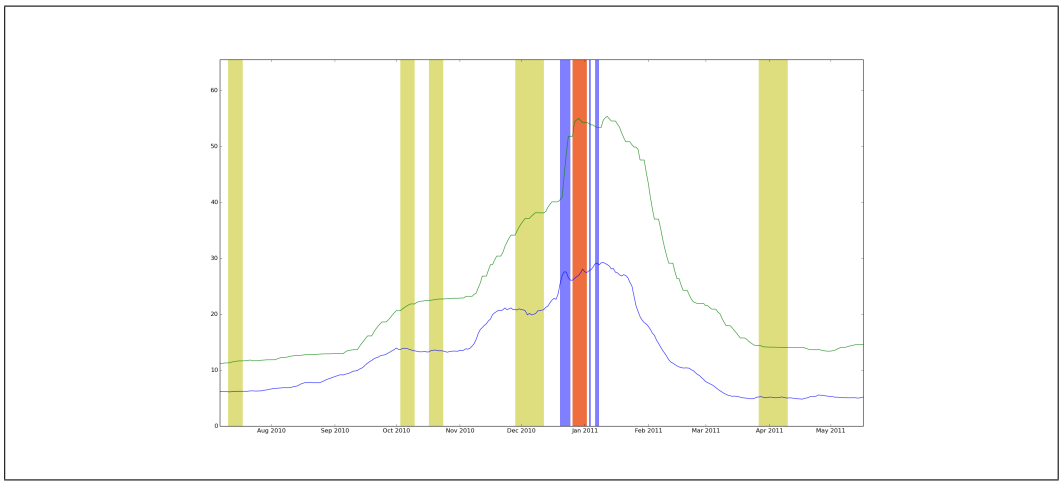

Figure 5.7: Slope Based Anomaly Detection (Green line - Retail Price, Blue Line - Wholesale Price)

So, even though we do not have articles for these anomalies, but method is behaving as it should be.

- One of the limitation for this method is that there are some cases where drop in retail price for one center is quite huge as compared to drop in average retail price (in case of *analysis 1*) or wholesale price (in case of *analysis 3*). This is good thing for customers at that centre and should not be treated as anomaly. But in this case, slope value goes high and

that's why our method reports that tenure as anomaly.
Following tenures are some cases reported by this method:

- *Analysis 1*: January 2010, June Aug 2012, Jan 2013, Feb Oct 2014, Feb 2015 (See Figure 5.8)
- *Analysis 3*: Feb May 2010 (See Figure 5.9)

Figure 5.8: Slope Based Anomaly Detection (Green line - Centre Retail Price, Blue Line - Average Retail Price)

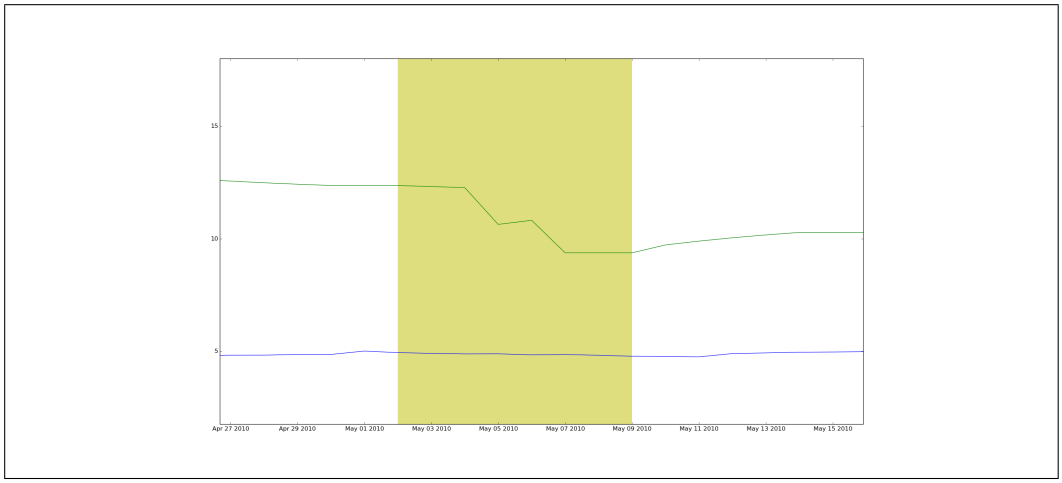

Figure 5.9: Slope Based Anomaly Detection (Green line - Retail Price, Blue Line - Wholesale Price)

- Other observation is related to why few anomalies were reported in news but not by our system. Reason for that is we are comparing

relative change in two time series. Now for some dates, where news article is present but our system did not report, value of both time series increased together. Although, prices went too high, but still relative change, i.e. slope value remained relatively low as compared to others. So that was not reported by our system.

Following tenures are some cases reported by this method:

- *Analysis 1*: Dec 2010, Jan Feb 2012, June 2013, May June 2015 (See Figure 5.10)
- *Analysis 3*: Feb 2013, July 2014 (See Figure 5.11)

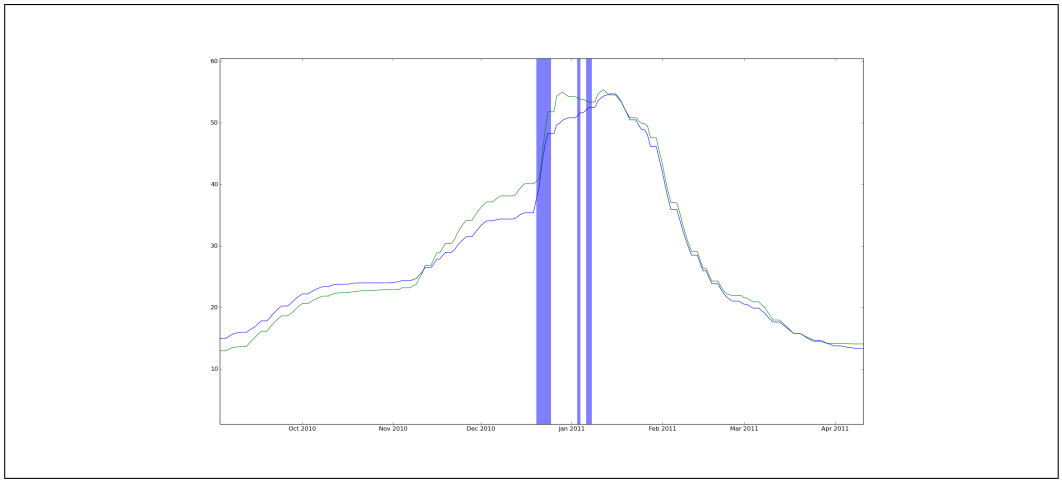

Figure 5.10: Slope Based Anomaly Detection (Green line - Centre Retail Price, Blue Line - Average Retail Price)

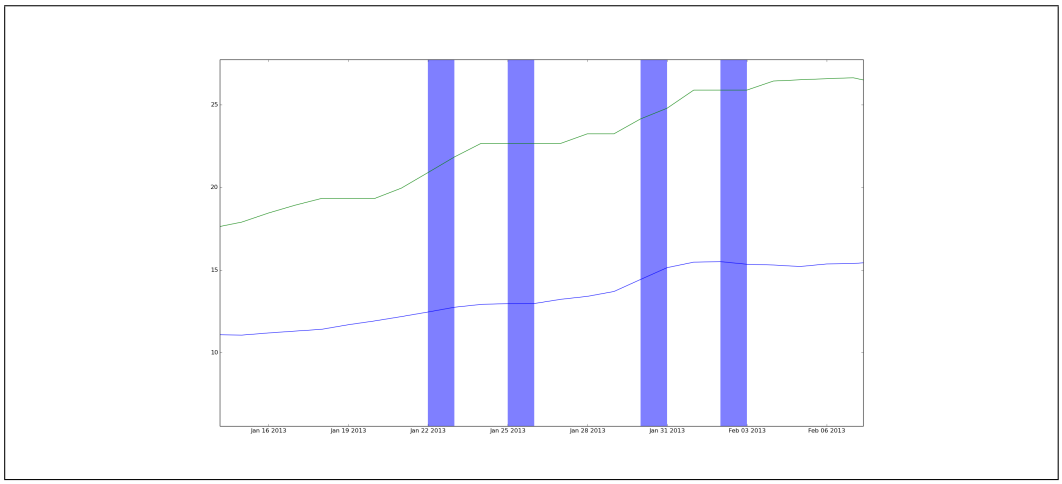

Figure 5.11: Slope Based Anomaly Detection (Green line - Retail Price, Blue Line - Wholesale Price)

- In some cases, original retail price was running less than average retail price for some time and then suddenly prices in the centre increased drastically. So such cases were reported as anomaly in this method, which is quite normal. Such cases were found in *Analysis 1* for Nov 2011, Feb Mar 2012 and Dec 2014. (See Figure 5.12)

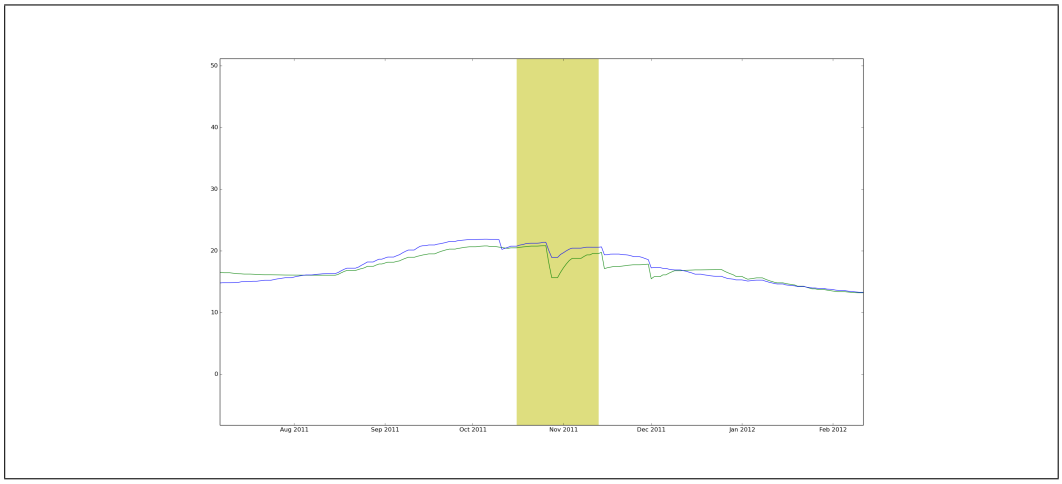

Figure 5.12: Slope Based Anomaly Detection (Green line - Centre Retail Price, Blue Line - Average Retail Price)

- In some cases, tenure reported as anomaly is quite large, because situations were abnormal for long time. But is not necessary that news articles should be present for such a large tenure and anomaly reported

was justifiable. For *Analysis 1*, such tenure was reported for March end to May start 2014. (See Figure 5.13)

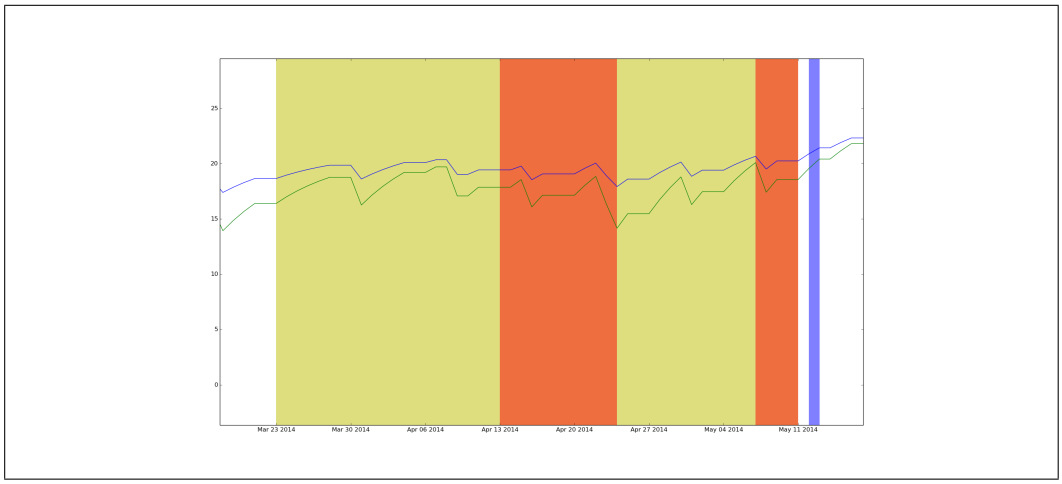

Figure 5.13: Slope Based Anomaly Detection (Green line - Centre Retail Price, Blue Line - Average Retail Price)

- In *Analysis 1* for June 2014, method has reported tenure upto mid June when prices started increasing but it remained high and due to that some news articles are present around June 21. We have missed because at that time relative slope value became normal. But since prices were high, it was covered by news articles. (See Figure 5.14)

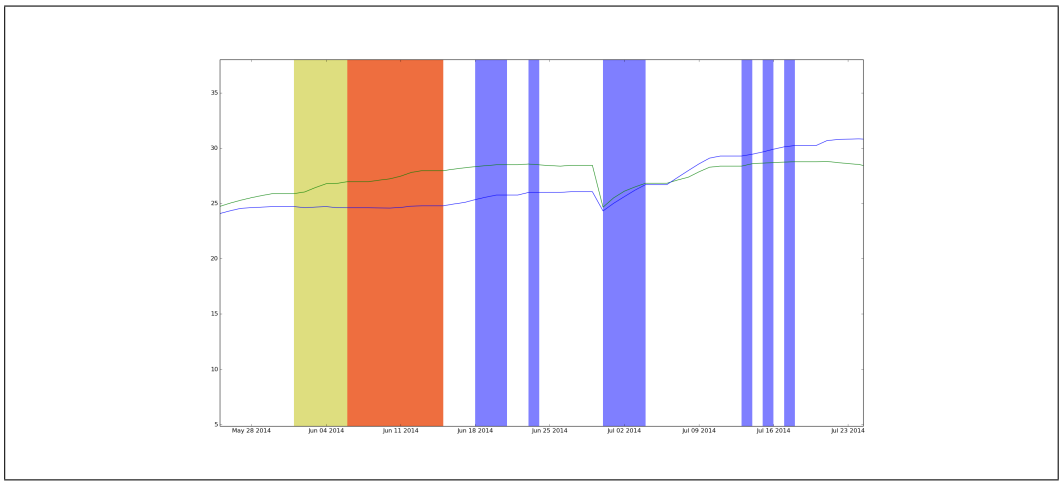

Figure 5.14: Slope Based Anomaly Detection (Green line - Centre Retail Price, Blue Line - Average Retail Price)

- There exist some cases in *Analysis 3* where retail price were decreasing but wholesale price kept on increasing, this created negative slope value, whereas in this scenario, we were looking for only positive slopes and that's why this method missed it. Such periods were in July Aug Sept 2013, Nov Dec 2013. (See Figure 5.15)

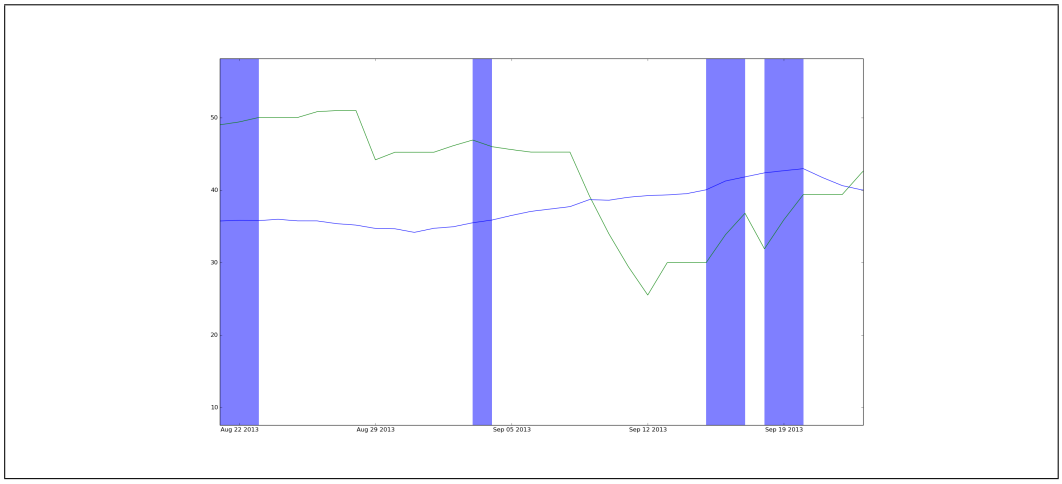

Figure 5.15: Slope Based Anomaly Detection (Green line - Retail Price, Blue Line - Wholesale Price)

Now we present few observation for Analysis 2 and 4.

- If change in retail price or change in wholesale price is more as compared to change in arrival (prices went too high, even for small drop in arrival), then it is reported as anomaly by this method.

  Following tenures are some cases reported by this method:

  - *Analysis 2*: Oct Dec 2010, May Jun 2013 (See Figure 5.16)
  - *Analysis 4*: Sept Oct Nov 2010, Jun 2013, Jun 2015 (See Figure 5.17)

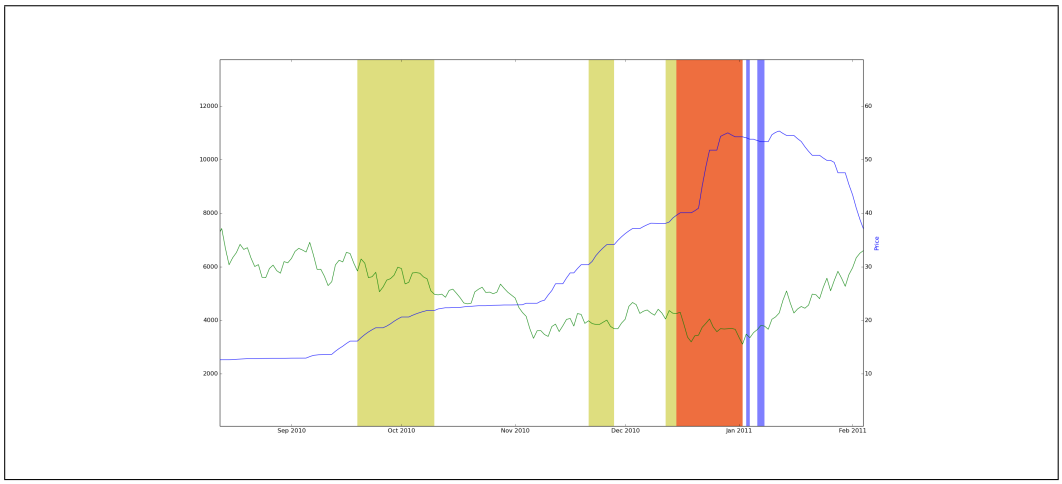

Figure 5.16: Slope Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Retail Price)

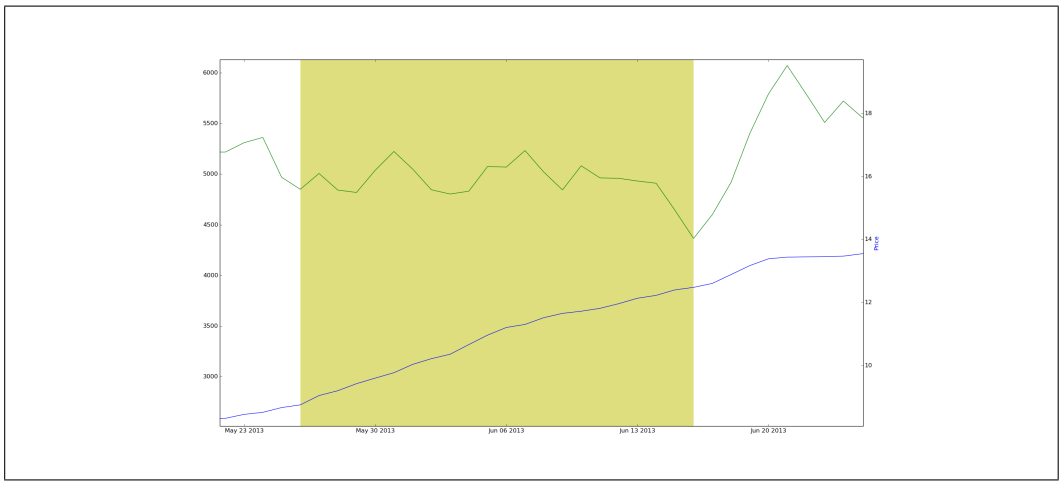

Figure 5.17: Slope Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Wholesale Price)

- But in above described scenario, when change in price is high, but prices are decreasing and when arrival is increasing, and if drop in price is too high, then also it will be reported as anomaly. So this is limitation of this method and reports false positives in this case.
  Following tenures are some cases reported by this method:
  - *Analysis 2*: Feb Mar 2011, Jan Feb 2014 (See Figure 5.18)
  - *Analysis 4*: Oct Dec 2011, Jan 2014, Mar 2011 (See Figure 5.19)

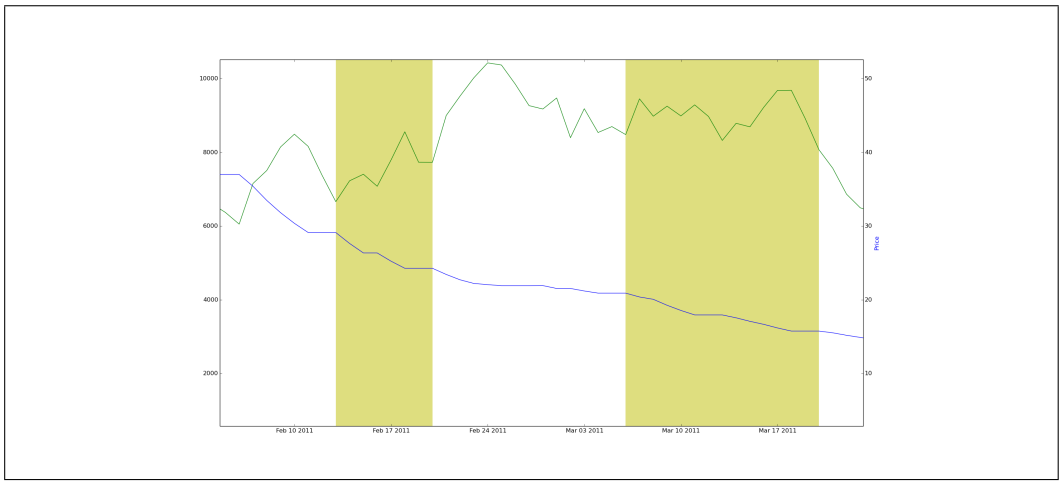

Figure 5.18: Slope Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Retail Price)

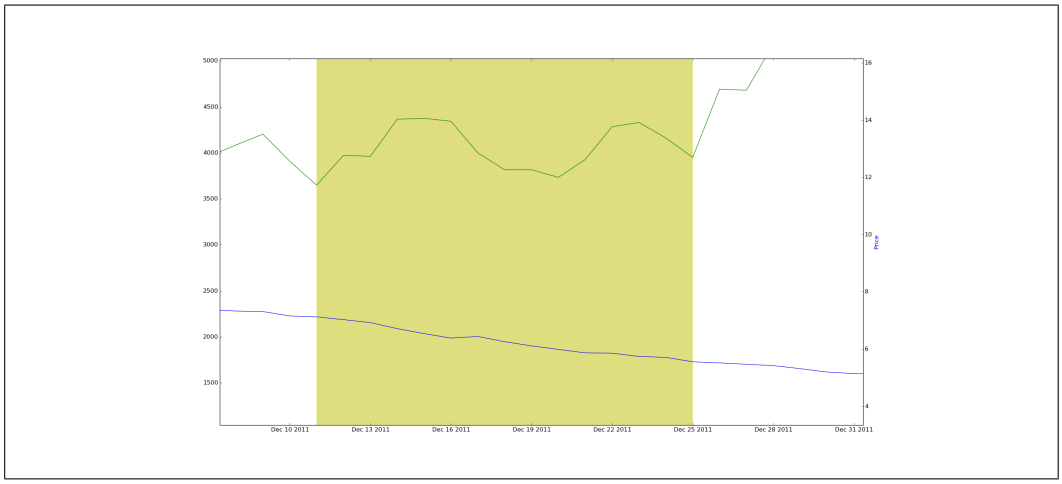

Figure 5.19: Slope Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Wholesale Price)

- One more limitation of this method is when arrival is increasing but along with that retail or wholesale price is also increasing. This will result into a positive slope and in this scenario we are only looking for negative slope and that's why, this will not be reported as anomaly and news articles corresponding to this tenure will not be matched by results of this method.

  Following tenures are some cases reported by this method:

– *Analysis 2*: Jan Feb July Nov 2013, June July 2014 (See Figure 5.20)

– *Analysis 4*: Jan Feb 2013, July 2014, June 2015 (See Figure 5.21)

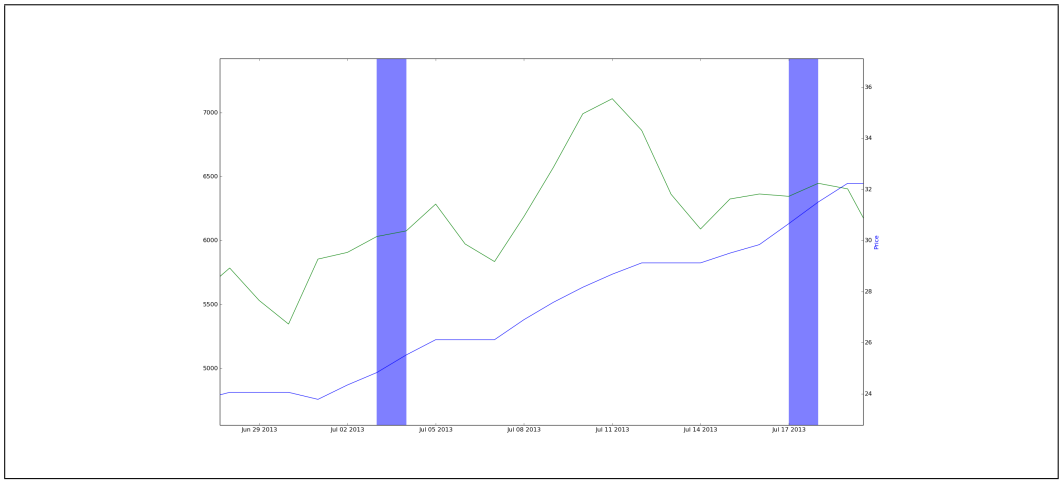

Figure 5.20: Slope Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Retail Price)

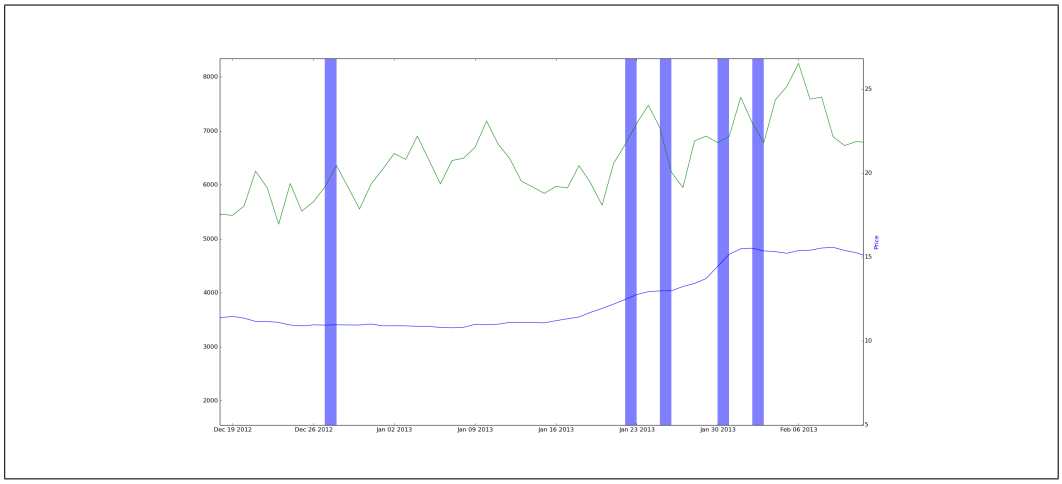

Figure 5.21: Slope Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Wholesale Price)

- There exist some cases where due to low arrival, prices went high and when slowly arrival started entering into market, prices were going down slowly. This period of slow decrement of prices is not reported by this method but since prices were still high, this system could not

report dates for news articles corresponding to this tenure. Such cases occurred in *Analysis 2* for Dec 2010 and Jan 2011 (See Figure 5.22)and in *Analysis 4* for Nov 2013 (See Figure 5.23). Also, note that these articles were mainly on Pakistan banned exports and article on inflation stating that inflation rate went high and onion prices are playing an important role in this.

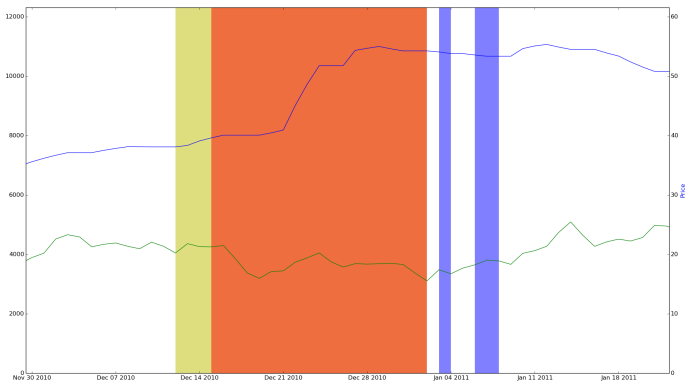

Figure 5.22: Slope Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Retail Price)

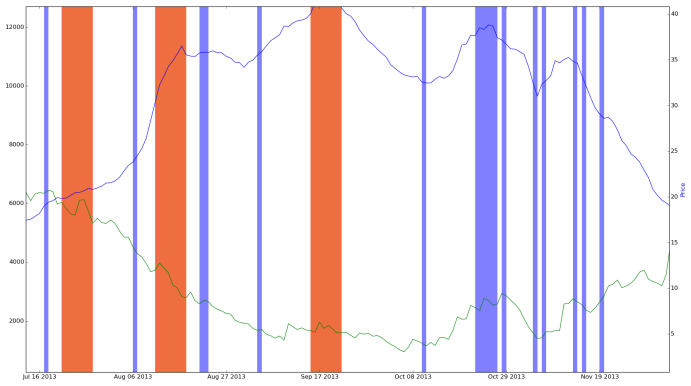

Figure 5.23: Slope Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Wholesale Price)

- In some of the cases where arrival fell drastically and due to that retail price went high drastically. Since retail prices went high too much, it

got reported in news articles but this was expected as arrival was less. But here both the changes were high, so ultimately slope value was not so high and was not reported by system. Such cases in *Analysis 2* exist for Aug Sept 2013 (See Figure 5.24).

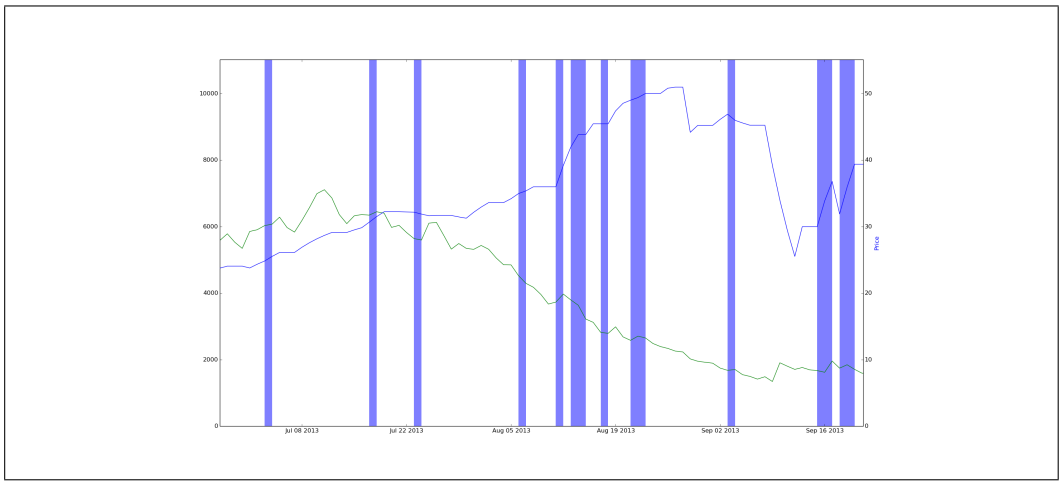

Figure 5.24: Slope Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Retail Price)

- Another limitation of this method is when retail price remained constant and there was change in arrival. As retail price was constant, slope value became zero and method did not report them and due to that few news articles could not be matched by dates reported by this method for example in *Analysis 2*, this thing occurred for June July 2015 (See Figure 5.25).

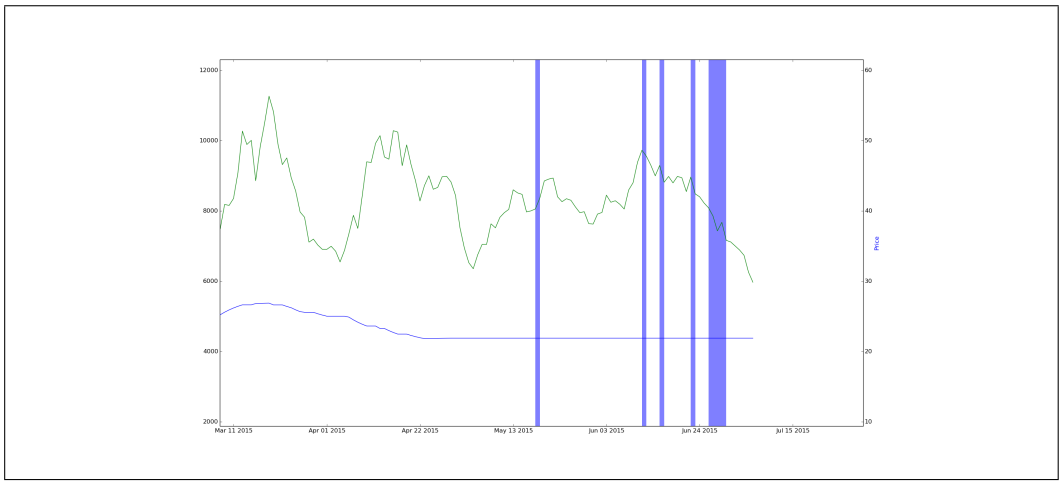

Figure 5.25: Slope Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Retail Price)

Also, note one thing that this method reports anomaly as whole window of few days (here 7 days). So because of that too, method tends to report more anomaly dates.

5.2.2 Linear Regression

The main functionality of this method is to find what should be ideal value of the dependent variable given value of independent variable. This method first finds out linear relationship between 2 variables, whose time series is given as input and one of them is dependent on another. After finding out this equation, we see for a given value of independent variable what should be ideal value of dependent variable and note down the relative difference. If this difference is large, then it is reported as anomaly.

We have four types of analysis which are as follows:

1. **Retail Price vs Average of Retail Price**: Here, we first take average of retail price at all centres as independent variable and retail price as dependent variable.

2. **Retail Price vs Arrival of Onion**: Here, we take retail price as dependent variable and arrival of onion as independent variable.

3. **Retail Price vs Wholesale Price**: Here, we take retail price as dependent variable and Wholesale Price as independent variable.

4. **Wholesale Price vs Arrival of Onion**: Here, we take Wholesale price as dependent variable and arrival of onion as independent variable.

So, in each of the case, we try to find relative difference between ideal value and its real value, and if it is huge (crosses threshold) then it is reported as anomaly. Now, note that in analysis 1 and 3 stated above, both the time series are directly proportional to each other and in the analysis 2 and 4 both the time series are inversely proportional to each other. So, limitations faced by this method for analysis 1 and 3 will be similar and for analysis 2 and 4 will be similar. While describing this method, each analysis will be referenced by its corresponding number.

First we will start with analysis 1 and 3. Following are the few observations we made:

- This method will report any tenure as anomaly when there is large gap i.e. more than expected between retail price of a center and average retail price (for *Analysis 1*) or wholesale price (for *Analysis 3*).

  Following tenures are some cases reported by this method:

  - *Analysis 1*: Dec 2010, Near to Jan 2011, May June July 2011, Jan May June 2012, June 2013 (See Figure 5.26)
  - *Analysis 3*: Feb Mar 2011, Jan 2012, June 2012, Jan Feb 2014, Apr 2015 (See Figure 5.27)

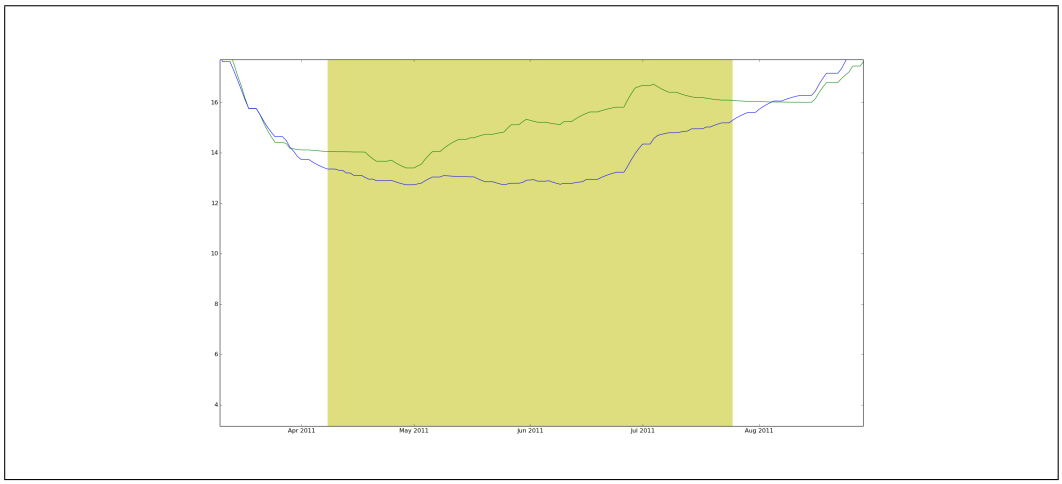

Figure 5.26: Linear Regression (Green line - Centre Retail Price, Blue Line - Average Retail Price)

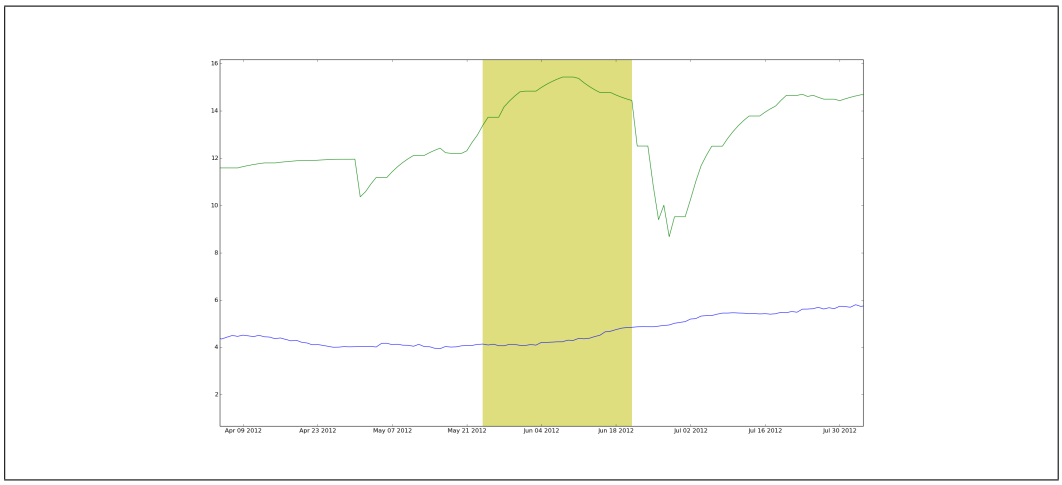

Figure 5.27: Linear Regression (Green line - Retail Price, Blue Line - Wholesale Price)

- One limitation of this method is that, if both the series have high values for some time period and difference between them is not so huge then that will not be reported as anomaly.

  Following tenures are some cases reported by this method:

  – *Analysis 1*: Jan 2011, Jan Feb Aug Sept Oct Nov 2013, July 2014, June July 2015 (See Figure 5.28)

– *Analysis 3*: Feb 2013, Aug Sept Oct Nov 2013, June July 2015 (See Figure 5.29)

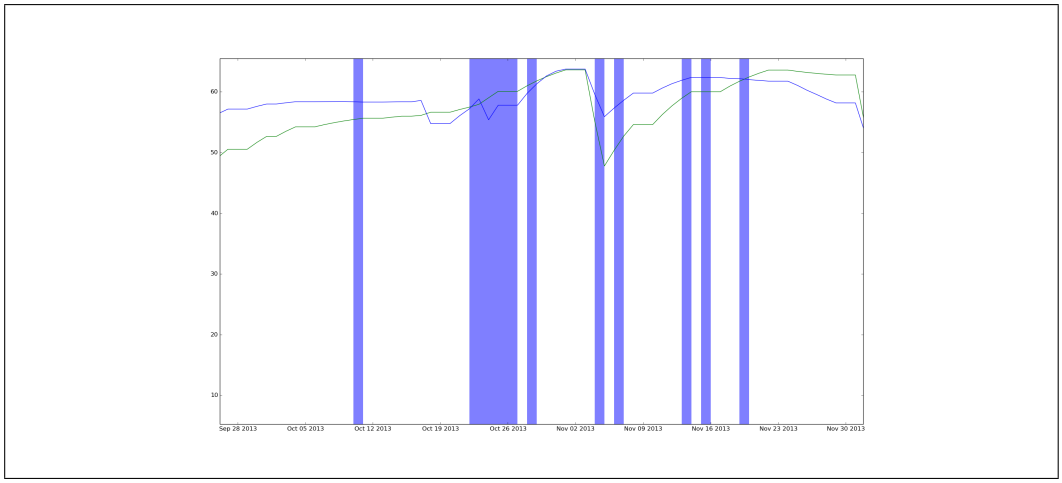

Figure 5.28: Linear Regression (Green line - Centre Retail Price, Blue Line - Average Retail Price)

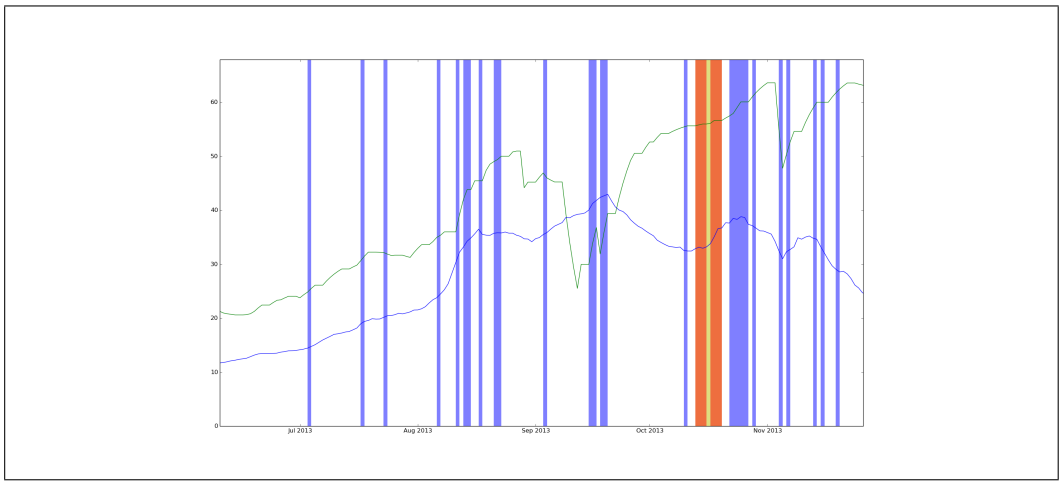

Figure 5.29: Linear Regression (Green line - Retail Price, Blue Line - Wholesale Price)

- Note that in the tenure of Oct Nov 2013 (for *Analysis 1*) prices are usually high and as the prices are high, tolerance level also increases little bit. So, even if for some difference it is reported as anomaly at lower price, it is not necessary that for the same difference, it will be reported as anomaly at higher prices. (See Figure 5.28)

Now we present few observation for Analysis 2 and 4.

- Here, in this method, it tries to predict what should be retail price or wholesale price based on the arrival of the product. So if the price is too high for the given arrival then it will be reported as anomaly. Following tenures are some cases reported by this method:
  - *Analysis 2*: Dec 2010, Jan Feb 2011, Aug Sept Oct Nov 2013, Oct Dec 2014 (See Figure 5.30)
  - *Analysis 4*: Dec 2010, July Aug Sept Oct Nov Dec 2013, July 2014 (See Figure 5.31)

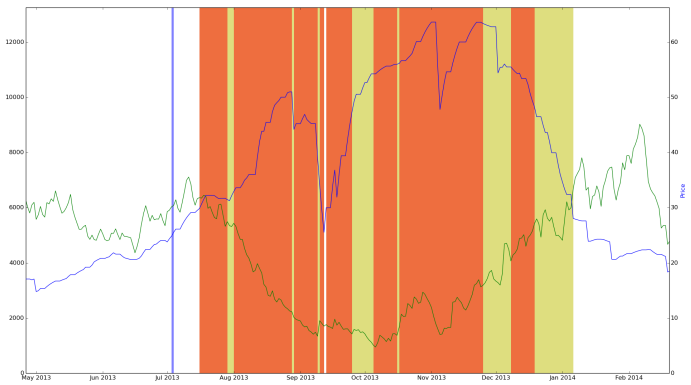

Figure 5.30: Linear Regression (Green line - Arrival Data of Onion, Blue Line - Retail Price)

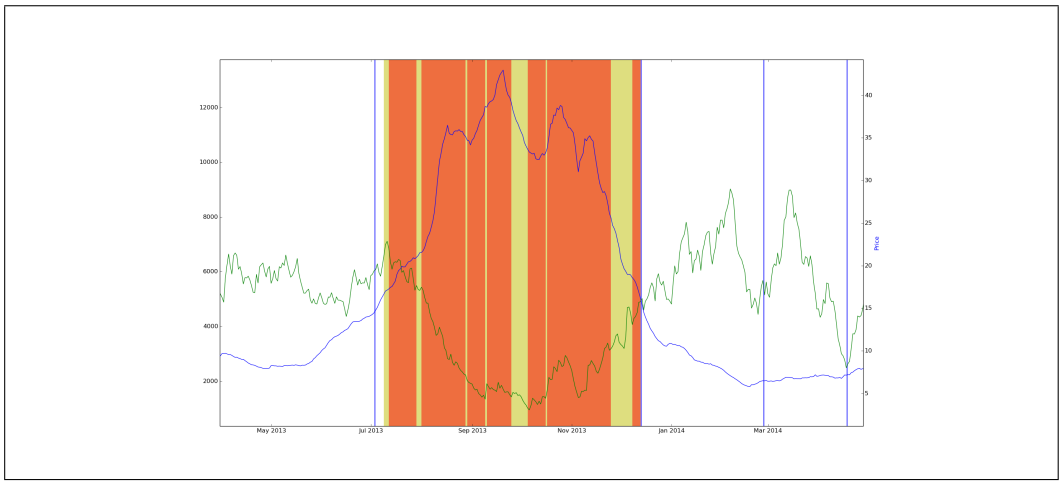

Figure 5.31: Linear Regression (Green line - Arrival Data of Onion, Blue Line - Wholesale Price)

- Now, this method has also missed few of the articles for this analysis as well. Now, looking at the graphs we could not interpret what may be exact reason why they were missed. But method may have found prices to be moderate and that's why they might have been missed.

  Following tenures are some cases reported by this method:

  - *Analysis 2*: Jan Feb 2013, July 2014, June July 2015 (See Figure 5.32)
  - *Analysis 4*: Jan Feb 2013, June July 2013, June 2015 (See Figure 5.33)

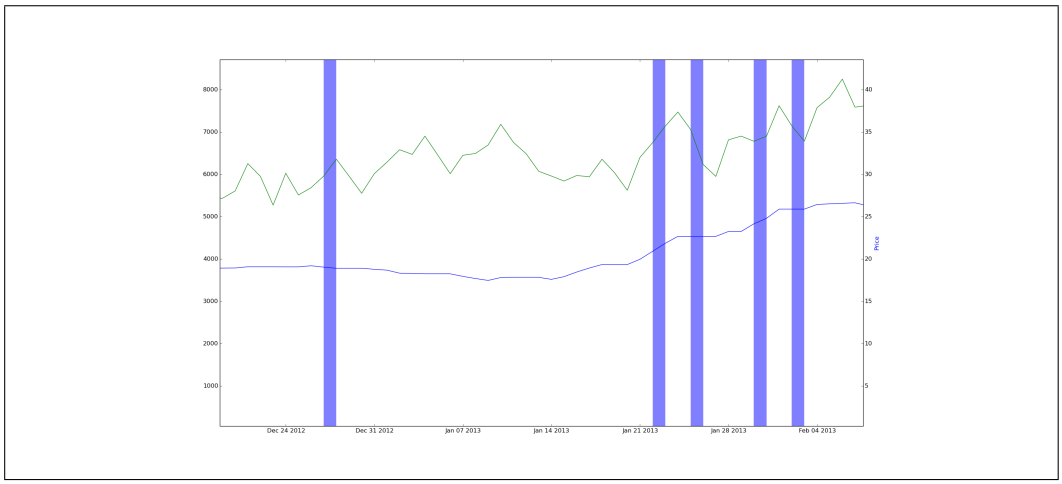

Figure 5.32: Linear Regression (Green line - Arrival Data of Onion, Blue Line - Retail Price)

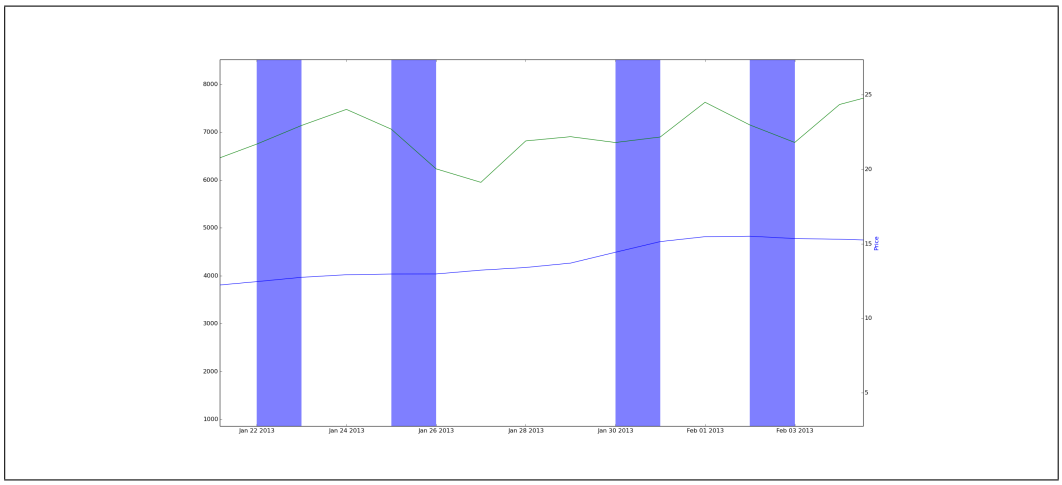

Figure 5.33: Linear Regression (Green line - Arrival Data of Onion, Blue Line - Wholesale Price)

5.2.3 Window Based Correlation

This method checks if the provided two timeseries moves in tandem or not. If not, then what are the time periods when they are not following the desired behavior. In order to find such time periods in the timeseries, the whole timeseries is divided into smaller windows and correlation value computed for that window is used to determine if that period is anomalous or not. The P-value of 0.01 is used in order to check the significance of correlation value.

The function is tested with different set of timeseries. Following are the four tests which were performed:

1. **Retail Price vs Average of Retail Price**: This test tries to find if any centre deviates from other centres abruptly. Average of timeseries of all the centres is taken as a representative timeseries for the behavior of all the centres which is compared with every centre in order to find time periods where these two did not move in tandem
2. **Retail Price vs Arrival of Onion**: Retail timeseries is expected not to move in tandem with Arrival timeseries. So, the time periods with positive correlation values are spotted in this.
3. **Retail Price vs Wholesale Price**: Retail timeseries is expected to move in tandem with wholesale timeseries. So, the time periods with negative correlation values are spotted in this.
4. **Wholesale Price vs Arrival of Onion**: Wholesale timeseries is expected to not move in tandem with Arrival timeseries. So, the time periods with positive correlation values are spotted in this.

The two timeseries are first aligned with each other at the maximum lag value. Then the entire timeseries is divided into small time periods of 15 days in order to locate anomalous situations through their respective correlation values.

Observations when retail price timeseries is compared with average of retail price timeseries:

- The timeseries showed the initial shift/lag of zero days which means both the timeseries are best aligned without any lag.

  Some of the tenures for which the method reported anomalies are:

  - 2008-03-06 to 2008-03-20 (See Figure 5.34)
  - 2009-09-12 to 2009-09-26 (See Figure 5.35)

– 2011-07-04 to 2011-08-02 (See Figure 5.36)

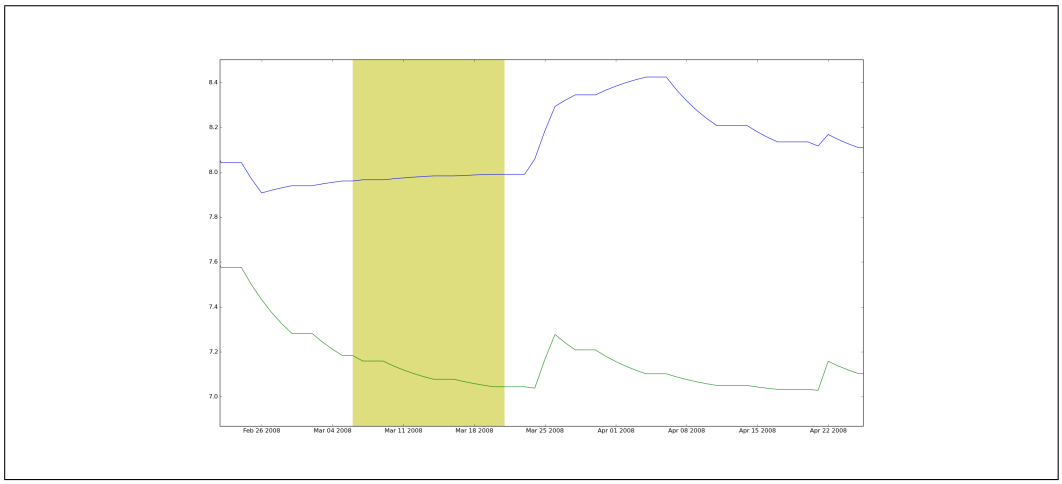

Figure 5.34: Window based correlation (Green line - Centre Retail Price, Blue Line - Average Retail Price)

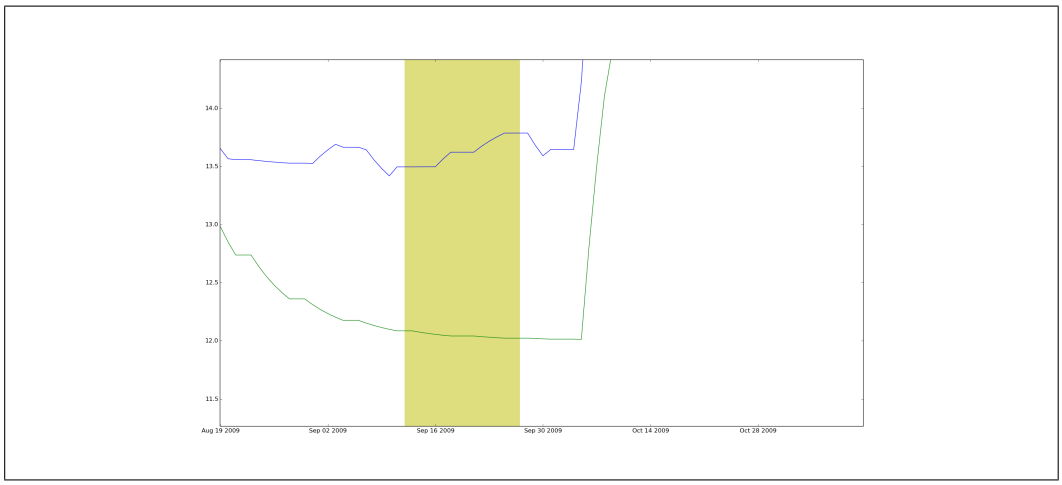

Figure 5.35: Window based correlation (Green line - Centre Retail Price, Blue Line - Average Retail Price)

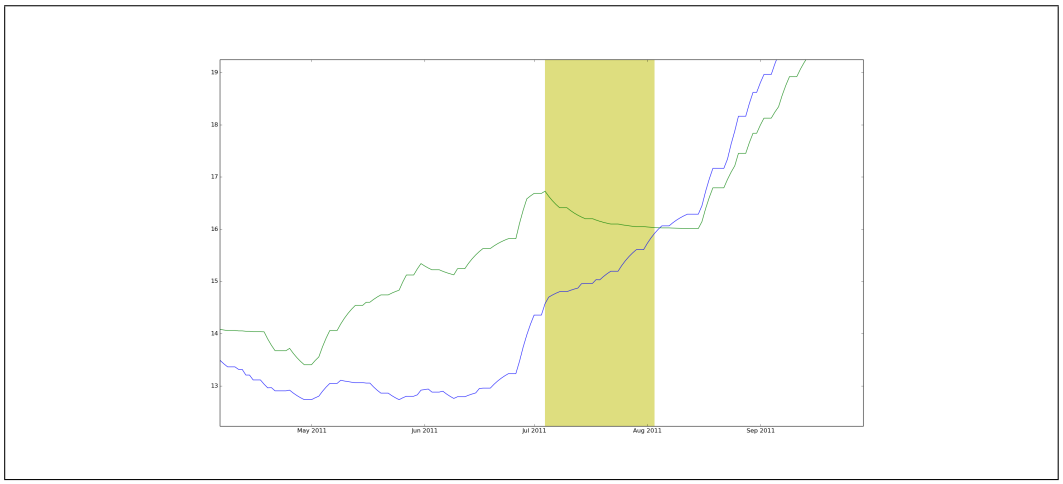

Figure 5.36: Window based correlation (Green line - Centre Retail Price, Blue Line - Average Retail Price)

But looking closely in the data, it is found that these tenures are reported because the two series were not in tandem which was mostly because prices of delhi faced some fluctuations whereas Mumbai prices did not show evident fluctuations. One of the reason for fluctuation in prices of Delhi over Mumbai could be because Delhi is dependent on other states for Onion whereas Mumbai does not have such issue.

Some of the tenures which went unnoticed by method:

- 2010-12-20 to 2010-12-25 (See Figure 5.37)
- 2011-01-06 to 2011-01-08 (See Figure 5.38)
- 2012-12-27 to 2013-01-05 (See Figure 5.39)

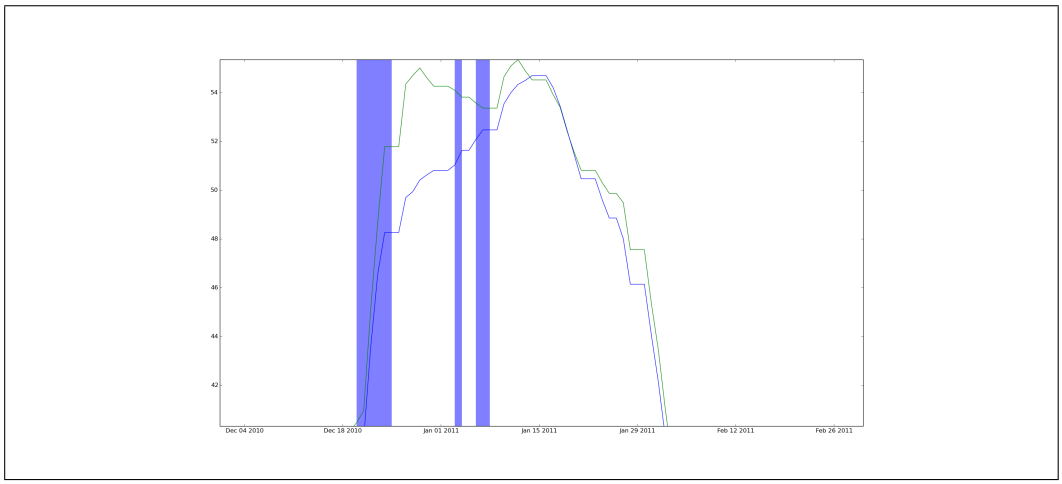

Figure 5.37: Window based correlation (Green line - Centre Retail Price, Blue Line - Average Retail Price)

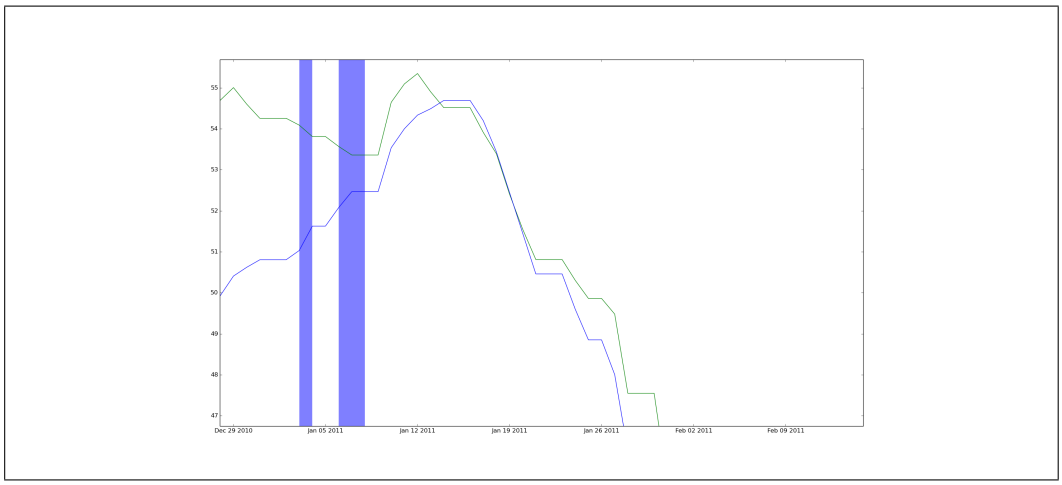

Figure 5.38: Window based correlation (Green line - Centre Retail Price, Blue Line - Average Retail Price)

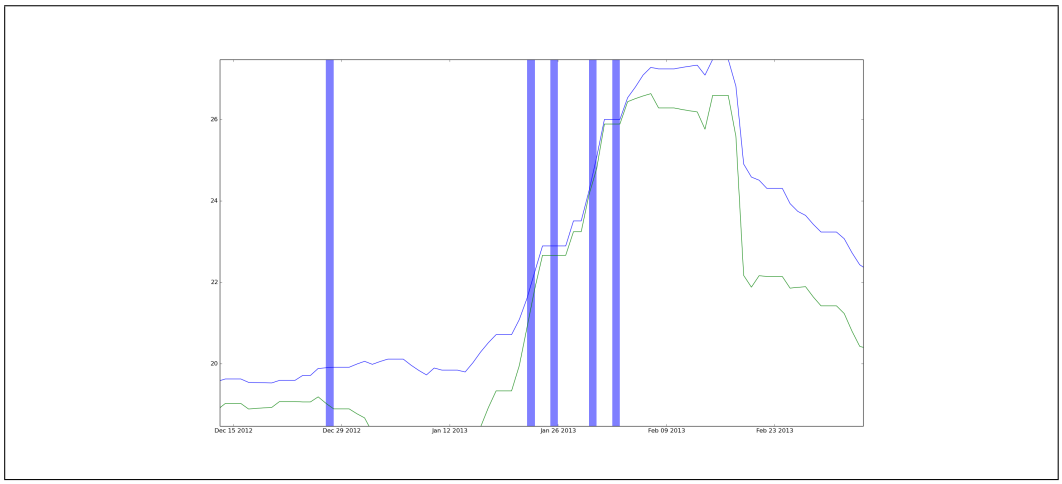

Figure 5.39: Window based correlation (Green line - Centre Retail Price, Blue Line - Average Retail Price)

Some of the tenures which were reported by news reports as well as method:

- 2013-10-06 to 2013-10-15 (See Figure 5.40)
- 2013-10-17 to 2013-10-21 (See Figure 5.40)

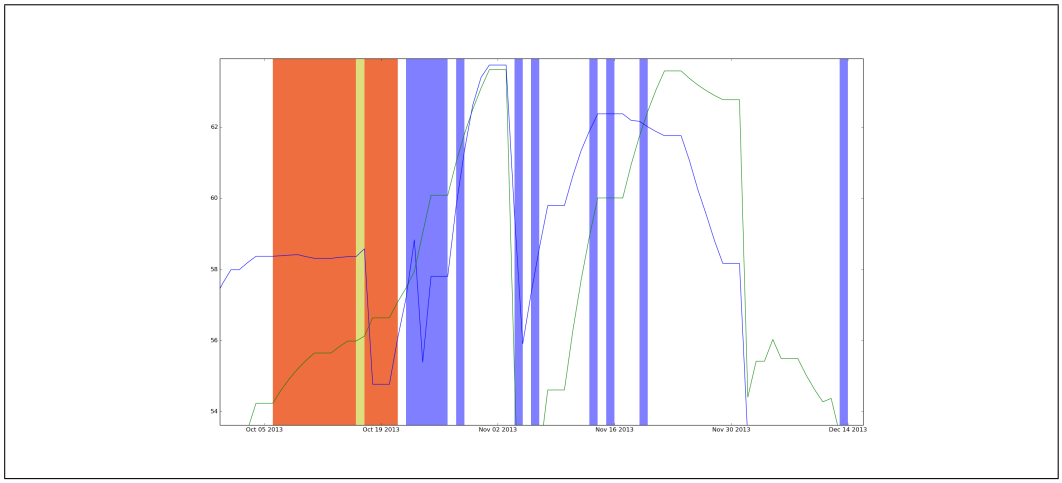

Figure 5.40: Window based correlation (Green line - Centre Retail Price, Blue Line - Average Retail Price)

Other interesting observations that were found while conducting above mentioned tests are as following:

- The lag chosen while comparing retail and arrival for Mumbai came as 15 days which means it takes 15 days of time for arrival to impact retail.
- The lag chosen by method while comparing retail and arrival for Delhi came out to be -9 days which gives hints that retail prices are impacting arrival of Onion which is not ideal in real time scenario.
- The lag chosen while comparing retail and wholesale for Mumbai comes out as -15 which means retail prices are impacted in approx. 15 days after change in the wholesale prices.
- The lag chosen while comparing retail and wholesale for Delhi comes out as -4 which is clearly very small than Mumbai. It might be because of smaller supply chain in Delhi compared to Mumbai.
- The lag chosen while comparing wholesale and arrival for Mumbai came as 11 days which means it takes 11 days of time for arrival to impact wholesale.
- The lag chosen by method while comparing wholesale and arrival for Delhi came out to be -14 days which gives hints that wholesale prices are impacting arrival of Onion which is not ideal in real time scenario.

Limitations of method:

- The method reports all the tenures where series are not in sync or are in sync(as needed). Like in case of 2008-03-06 to 2008-03-20 for retail vs average, though the correlation went really high because of opposite directions but the fluctuation in prices were not very prominent.
- If all the timeseries follows same anomalous behavior then it is not possible to find anomaly in any.
- If the anomaly is for very small tenure compared to the selected window size then it is likely to be missed or go unnoticed. In case of 2011-01-06 - 2011-01-08

5.2.4 Multivariate Time Series- Vector Autoregressive

The method uses vector autoregressive framework for multivaraiate time-series analysis in order to forecast values by using all the related variables/timeseries that can impact the timeseries. The error in the predicted value with the original value helps in determining anomalous points from the timeseries. MAD test is applied in order to fix the threshold value for the error. 60% (This can be configured as per need) of the data is used for calibration and rest of the data is checked for anomalous points. So, the start date for the anomalous points begins after 2011-09-16
The function is tested with different set of timeseries. Following are the four tests which were performed:

1. **Retail Price vs Average of Retail Price**: All the centres should ideally move in tandem. So test is performed over every centre with other centres helping in predicting values.

2. **Retail Price vs Arrival of Onion**: Arrival is one of the major deciding factor for retail. So the predictions are made for retail based on arrival.

3. **Retail Price vs Wholesale Price**: Retail prices are predicted based on the wholesale prices.

4. **Wholesale Price vs Arrival of Onion**: Wholesale timeseries is predicted on the basis of arrival timeseries.

Observations when retail price timeseries is compared with average of retail price timeseries:

- The threshold selected by this method for error values are -19.0358099572 and 107.697818954 which means all the data points with error values less than -19.0358099572 and greater than 107.697818954 are reported by the method.

  Some of the tenures for which the method reported anomalies are:

– 2013-07-19 to 2013-09-11 (See Figure 5.41)

– 2013-09-17 to 2014-01-06 (See Figure 5.42)

– 2014-12-03 to 2014-12-15 (See Figure 5.43)

Figure 5.41: Vector Autoregressive (Green line - Centre Retail Price, Blue Line - Average Retail Price)

Figure 5.42: Vector Autoregressive (Green line - Centre Retail Price, Blue Line - Average Retail Price)

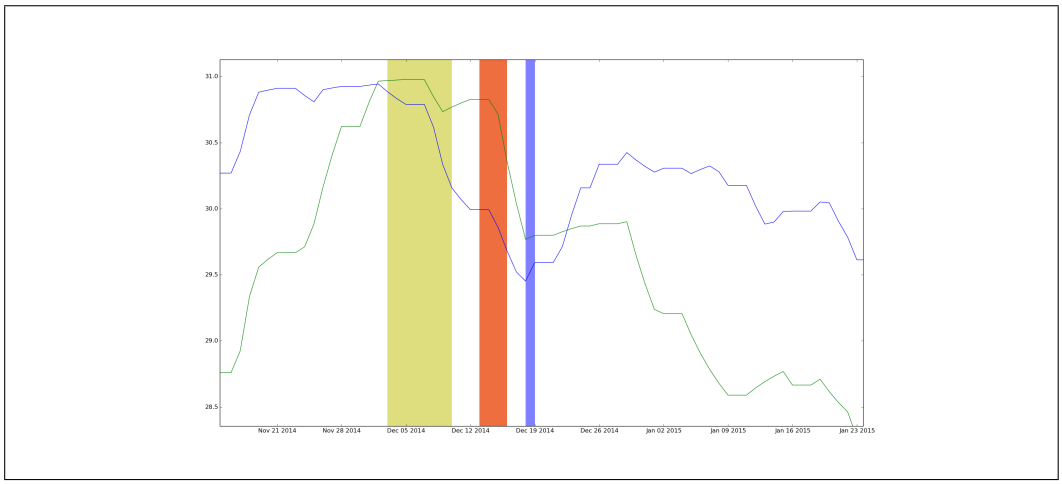

Figure 5.43: Vector Autoregressive (Green line - Centre Retail Price, Blue Line - Average Retail Price)

No data points were reported against news articles present in May, June, July 2014 despite of large error value. The threshold selected by the MAD test was higher than the error values. This could be corrected by manually setting threshold values which can be account for these data points.

Other interesting observations that were found while conducting above mentioned tests are as following:

- Some of the data points like July 2014 data were not captured despite of error value close to MAD threshold. These can be captured by lowering the threshold.
- For news articles in Dec 2012,Jan 2013, Feb 2013 the values does not show high error values despite of news articles reporting the crisis. The possible reason could be because similar trend have been seen for this tenure in the data.

Limitations of method:

- The method depends on the MAD Test in order to set threshold. So, even though the error value is close to threshold but less than it won't be reported. Other methods could be also used in order to decide threshold.

5.2.5 Graph Based Anomaly Detection

This method, treats each day as a node of a graph, and connects with other nodes if nodes are similar. This connecting edge is given similarity value and random walk is performed to get connectivity of each node. Node with the least connectivity values are reported as anomaly. Note that for the previous methods, we had threshold values either defined by user or calculated by using MAD test. But here we do not have that and we just ask method to report "n" number of nodes with least connectivity values.

The working of this method is quite complex and can not be generalised. For detailed information go through the paper **??**. So we will just represent, how method has performed on the different analysis.

For **Retail Price vs Average of Retail Price** (See Figure 5.44) and **Retail Price vs Wholesale Price** (See Figure 5.45), this method has performed well. For **Retail Price vs Average of Retail Price**, every tenure of anomaly has been matched with some news articles.
The anomalies which were not matched with news articles were part of large tenure which had some matching with news articles and usually, this tenure is large and for every date news articles are not present.
Few articles are missed, that might be due to limited number of points chosen. If number of points are increased, than it might be covered as well.
For **Retail Price vs Wholesale Price**, apart from Jan 2013, July 2014, June 2015, all anomalies are matching with some news articles.

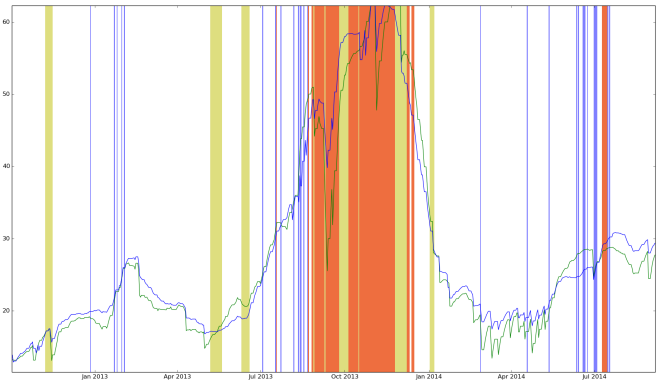

Figure 5.44: Graph Based Anomaly Detection (Green line - Centre Retail Price, Blue Line - Average Retail Price)

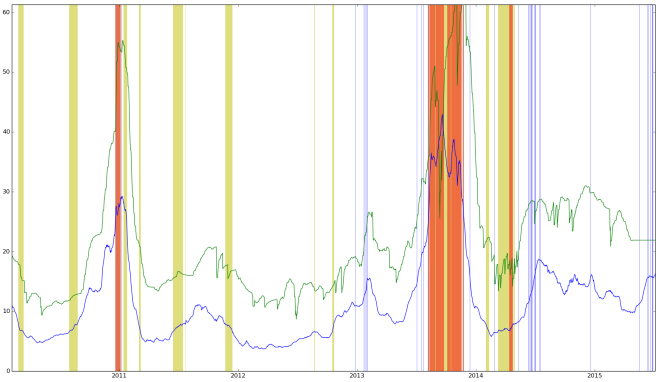

Figure 5.45: Graph Based Anomaly Detection (Green line - Retail Price, Blue Line - Wholesale Price)

For **Retail Price vs Arrival of Onion** (See Figure 5.46) and **Wholesale Price vs Arrival of Onion** (See Figure 5.47), this method is not producing good results. Many points are reported as anomaly which are close to each other. And due to limited number of points, number of anomalies matching with news articles are quite less. The reason might be because of fluctuations seen in the arrival timeseries. Figures 5.48 and 5.49 describe results of these both analysis for Delhi centre.

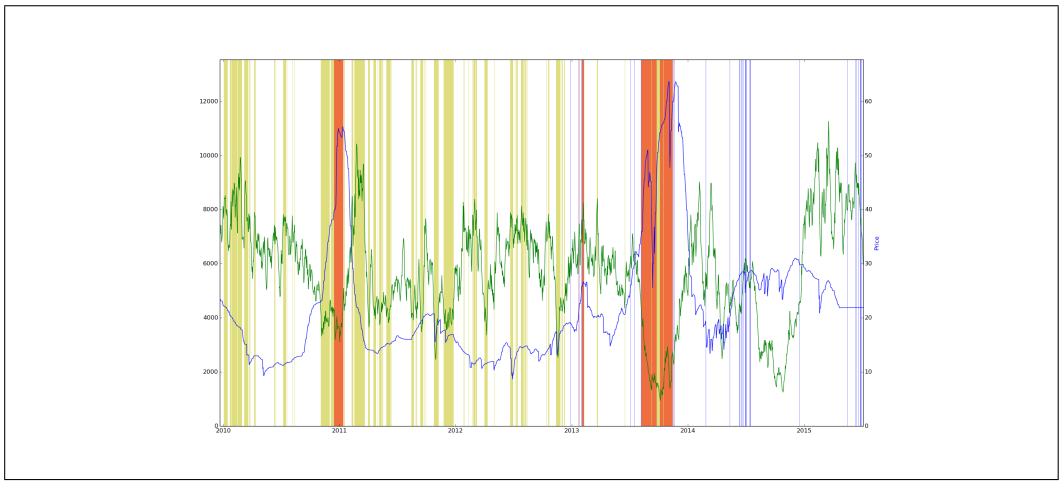

Figure 5.46: Graph Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Retail Price)

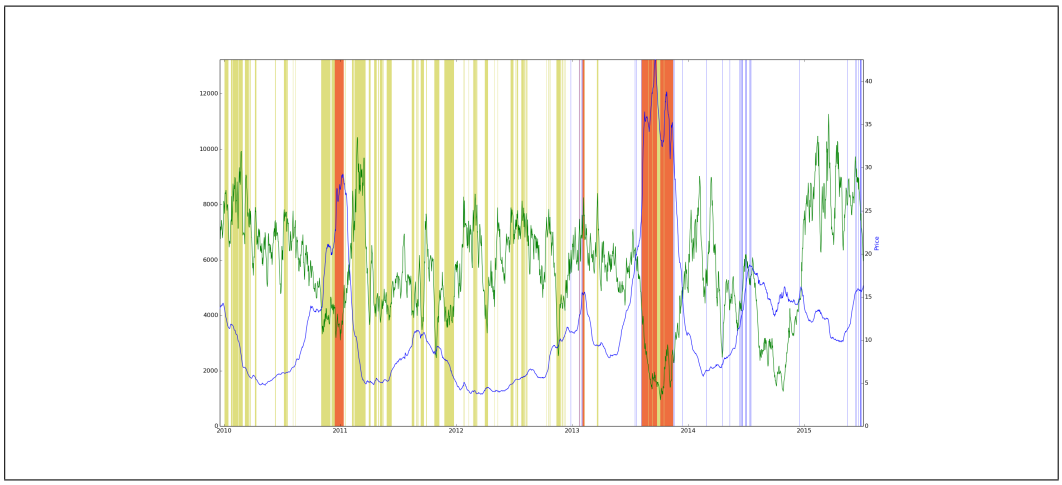

Figure 5.47: Graph Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Wholesale Price)

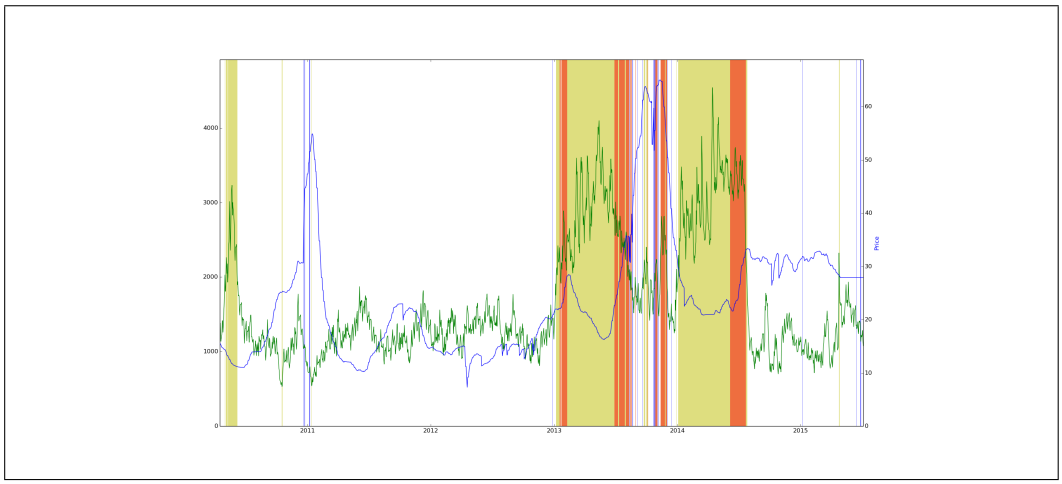

Figure 5.48: Graph Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Retail Price)

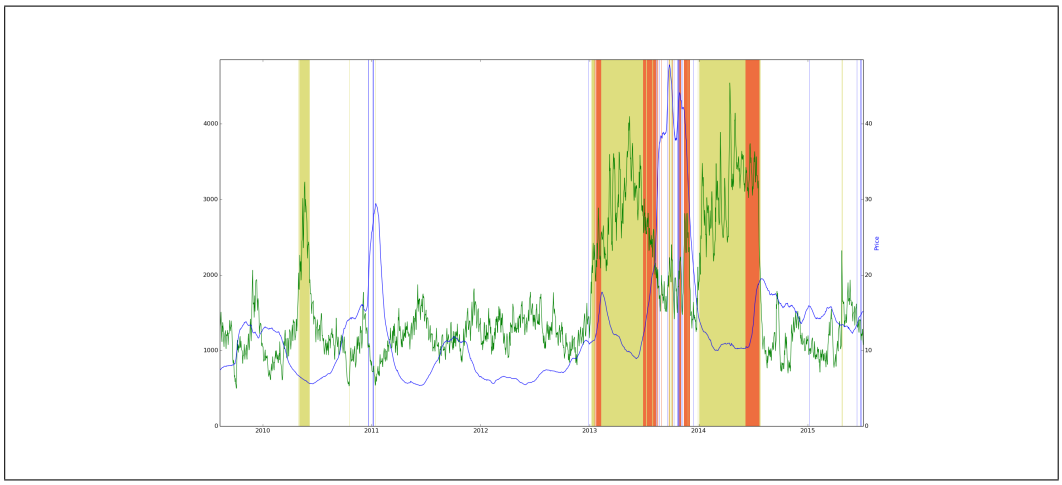

Figure 5.49: Graph Based Anomaly Detection (Green line - Arrival Data of Onion, Blue Line - Wholesale Price)

Chapter 6

Conclusion

Often rise in the price of commodity are considered as a result of unseasonal rainfall, increased demand, unrealistic government policies, supply deficit etc. But supply chain deficits and natural calamities can not only be held responsible for every hike in price of commodity. Sometimes the hike in prices are man-made with the intension of earning more profits with illicit means like excess hoarding, manual creation of supply crunch etc. As we have observed in the previous chapter as well that for the Mumbai centre, 32% of the news articles state treders' nexus as reason for price hike of onions.

These manual interventions to the supply chain of commodity for sake of earning more profit can be found by analyzing time series data of dependent factors since these have distinguishing characteristics. These characteristics can be found by various statistical techniques.

So, for given time series first one needs to understand normal behavior before going for detection of anomalies. Time series may have periodicity, seasonality, trends or may have complete randomness. So method to detect anomaly should be designed in a such a way that functionalities covered by it does not miss any type of time series. So, here we have tried to build up a system keeping multiple functionalities for one hypothesis, such that it can cover multiple aspects of any time series. Results produced by this are quite justibiable and acceptable.

This project can further be extended by adding new methods for various hypothesis. One such method is Spike Detection, which can be used for Hypothesis 2, to enhance the results. First we can generate series of relative difference between retail price and wholesale price. Then we apply spike detection method over it. If this difference becomes very large in the short duration of time, then it can be reported as anomaly. Reason to report this

as anomaly is that there exists few news articles which reports such type of behaviour as anomaly.

One can also consider value chain of any product, like let's say car. Then price of various componenets in this chain starting from raw material, raw parts and final product price, etc can be collected and one can find if there exists any anomaly at any point of time if price of final product goes up.

Bibliography

[1] Agriculture marketing. http://agmarknet.dac.gov.in/. (Visited on 01/04/2016).

[2] fcainfoweb.nic.in/pmsver2/reports/report_menu_web.aspx. http://fcainfoweb.nic.in/PMSver2/Reports/Report_Menu_web.aspx. (Visited on 01/04/2016).

[3] Flying squads to tackle onion hoarding in maharashtra — business standard news. http://www.business-standard.com/article/markets/flying-squads-to-tackle-onion-hoarding-in-maharashtra-113072200152_1.html. (Visited on 01/05/2016).

[4] Govt wakes up from slumber to prevent onion hoarding — latest news & updates at daily news & analysis. http://www.dnaindia.com/ahmedabad/report-govt-wakes-up-from-slumber-to-prevent-onion-hoarding-1876027. (Visited on 01/05/2016).

[5] The great onion robbery: 135% mark-up from mandi to retail - times of india. http://timesofindia.indiatimes.com/india/The-great-onion-robbery-135-mark-up-from-mandi-to-retail/articleshow/7147837.cms. (Visited on 01/05/2016).

[6] Hoarding pushing onion prices up, govt finds - times of india. http://timesofindia.indiatimes.com/india/Hoarding-pushing-onion-prices-up-govt-finds/articleshow/21872501.cms. (Visited on 01/05/2016).

[7] Kerala yet to initiate action against onion hoarders - the hindu. http://www.thehindu.com/news/national/kerala/kerala-yet-to-initiate-action-against-onion-hoarders/article6189123.ece. (Visited on 01/05/2016).

[8] No onion hoarding in delhi, don't politicise price rise: Sheila:ibnlive videos. http://www.ibnlive.com/videos/politics/sheila-alerts-646704.html. (Visited on 01/05/2016).

[9] Onion crisis worsened by hoarding? http://www.ndtv.com/india-news/onion-crisis-worsened-by-hoarding-531668. (Visited on 01/05/2016).

[10] Onion price rise may be due to hoarding: Centre - times of india. http://timesofindia.indiatimes.com/india/Onion-price-rise-may-be-due-to-hoarding-Centre/articleshow/22672225.cms. (Visited on 01/05/2016).

[11] Onion price rise: Nashik farmers put blame on hoarders - timesofindia-economictimes. http://articles.economictimes.indiatimes.com/2014-06-20/news/50739208_1_lasalgaon-apmc-onion-prices-farmers. (Visited on 01/05/2016).

[12] Onion prices rise again at vashi agricultural produce marketing committee — latest news & updates at daily news & analysis. http://www.dnaindia.com/mumbai/report-onion-prices-rise-again-at-vashi-agricultural-produce-marketing- (Visited on 01/05/2016).

[13] Rains, hoarding make vegetables costlier. http://www.deccanchronicle.com/140720/nation-current-affairs/article/rains-hoarding-make-vegetables-costlier. (Visited on 01/05/2016).

[14] Rediff on the net business news: Onion crisis looms as trader-exporter nexus waits for the kill. http://www.rediff.com/business/1999/sep/21onion.htm. (Visited on 01/05/2016).

[15] Why onion prices are so high. http://www.ndtv.com/india-news/why-onion-prices-are-so-high-442629. (Visited on 01/05/2016).

[16] Deepthi Cheboli. *Anomaly detection of time series*. PhD thesis, University of Minnesota, 2010.

[17] Haibin Cheng, Pang-Ning Tan, Christopher Potter, and Steven A Klooster. Detection and characterization of anomalies in multivariate time series. In *SDM*, pages 413–424. SIAM, 2009.

[18] PG Chengappa, AV Manjunatha, Vikas Dimble, and Khalil Shah. Competitive assessment of onion markets in india. *Institute for Social and Economic Change. Competition commission of India*, 1:86, 2012.

[19] SC Hillmer, DF Larcker, and DA Schroeder. Forecasting accounting data: A multiple time series analysis. *Journal of Forecasting*, 2(4):389–404, 1983.

[20] Devesh Kapur and Milan Vaishnav. Quid pro quo: Builders, politicians, and election finance in india. *Center for Global Development Working Paper*, (276), 2011.

[21] Sandip Sukhtankar. Sweetening the deal? political connections and sugar mills in india. Technical report, Mimeo, 2008.

Appendix A

Window Based Correlation

A.1 Introduction

This technique is basically applied on two time-series. Let's say we have two time series as series1 and series2. So, in this method, we first find correlation at various lags between these two time series. User can specify minimum and maximum lag to consider. So, for each value between minimum and maximum lag, we find correlation values.

After finding correlation values at all lags, we consider that lag at which correlation value is higher, among all previously calulated correlation values, at all lags. Let's say that lag be "x". So, depending upon that "x", we shift series1 or series2. If "x" is positive, we move series2 by "x" units and if it is negative than we shift series1 by $|x|$ units.

Now, we are ready to apply window correlation. Take window size,"w" as input. First window will be from 1st element to w'th element of both the time series after aligning by lag "x". Find correlation for this window between two time-series and save it in an array. Now, slide window by "w" elements and calculate correlation value again and so on. Now, we have correlation values at multiple windows.

Now, let's say both the series should have been positively correlated. So, what we do is, we choose threshold by MAD test if not provided to us, and find all correlation values which are below that threshold and report all those windows as anomaly. Similarly, for negative correlation values above the threshold value is reported as anomaly.

A.2 Related Functions

A.2.1 correlation(arr1, arr2, maxlag, pos, neg)

This function calculates correlation between arr1 and arr2 at all possible lags between -maxlag to +maxlag, as specified by pos and neg parameters.

- Input Parameters
  1. arr1 *(list)* : Input series 1 as a list of float values
  2. arr2 *(list)* : Input series 2 as a list of float values
  3. maxlag *(int)* : maximum (maxlag) and minimum (-maxlag) lag to consider while calculating correlation betweem arr1 and arr2
  4. pos *(int, 1 or 0)* : To consider positive lag or not, i.e. 1 to maxlag, if value is 1, than positive lag will be considered, else not.
  5. neg *(int, 1 or 0)* : To consider negative lag or not, i.e. -maxlag to -1, if value is 1, than negative lag will be considered, else not.
- Output *(list)* :
  Returns list of tuples of the form

  (lag, correlation value at this lag)

A.2.2 getMaxCorr(arar1,positive_correlation)

If both the series are positively correlated than we will be interested in maximum positive correlation or if both series are negatively correlated than we will be interested in minimum negative correlation, which is specified by positive_correlation parameter.
This function takes list of tuples of the form (lag, correlation value at this lag) as input. Returns lag value at which correlation value is maximum, if positive_correlation is True, and returns lag at which correlation value is minimum if positive_correlation is False.

- Input Parameters

  1. arr1 *(list)* : list of tuples of the form
     (lag, correlation value at this lag)
     i.e. correlation values at various lags
  2. positive_correlation *(boolean, "True" or "False")* :
     - True: If value of this parameter is True than it will return lag at which correlation value if maximum (positive)
     - False: If value of this parameter is False than it will return lag at which correlation value if minimum (negative)

- Output *(Tuple)* :
  returns single tuple of the form (lag,correlation value at this lag), i.e. lag at which optimum correlation value is found along with correlation value.

A.2.3 correlationAtLag(series1, series2, lag, window_size)

This function first aligns two series by given lag. If lag is positive than it shifts start of series2 else start of series1. After aligning both the series according to lag, this function calculates correlation between both series at all windows.

window_size states size of the window. So, we will start with first window taking first "window_size" elements from each series and will calculate correlation. We will save this correlation value in list and will slide to next window. Next window will start after "window_size" elements. In such a way, we calculate, correlation at all windows and return the list of correlation values.

- Input Parameters

  1. series1 *(list)* : Input series 1 as a list of float values
  2. series2 *(list)* : Input series 2 as a list of float values

3. lag *(int)* : lag at which series needs to be adjusted as explained above
4. window_size *(int)* : window size to be considered

- Output *(list)* :
  Returns list of correlation values (of float type) for all windows calculated at given lag

A.2.4 WindowCorrelationWithConstantLag(arr1, arr2, window_size,maxlag, positive_correlation, pos, neg)

This is sort of driver function, which will call above 3 functions. This function will first get lag at which series needs to be adjusted. Than using this lag, it will calculate correlation values at all windows and will return it.

- Input Parameters

1. arr1 *(list)* : Input series 1 as a list of float values
2. arr2 *(list)* : Input series 2 as a list of float values
3. window_size *(int)* : window size to be considered while calculating window correlation
4. maxlag *(int)* : maximum (maxlag) and minimum (-maxlag) lag to consider while calculating correlation betweem arr1 and arr2, to align both the series
5. positive_correlation *(boolean, "True" or "False")* :
   - True: This suggest that both the series are positively correlated
   - False: This suggest that both the series are negatively correlated
6. pos *(int, 1 or 0)* : If value of this parameter is 1 than we will consider positive values for lag, i.e. 1 to +maxlag to align both the series initially

7. neg *(int, 1 or 0)* : If value of this parameter is 1 than we will consider negative values for lag, i.e. -maxlag to -1 to align both the series initially

- Output *(list)* :
  Returns tuple of the form (lag,array) Where lag is lag value for which whole series is shifted and then at that lag, we have calculated correlation for all window. Correlation value for all windows is stored in array.

A.2.5 anomaliesFromWindowCorrelationWithConstantlag(arr1, arr2, window_size=15,maxlag=15, positive_correlation=Tr pos=1, neg=1, default_threshold = True, threshold = 0):

This is main function of this method. This is driver of whole method. Using previously stated methods, it will first gather correlation values at different windows. Than depending upon which type of threshold is to be used, it will filter out anomalies. If default threshold is to be used, than it will be calculated using MAD test on the correlation values at each window, else threshold provided by user will be used.

Correlation values not satisfying threshold will be reported along with the date range of that window.

- Input Parameters

1. arr1 *(list)* : Input series 1 as a list of tuples of the form (date,value)
2. arr2 *(list)* : Input series 2 as a list of tuples of the form (date,value)
3. window_size *(int)* : window size to be considered while calculating window correlation
4. maxlag *(int)* : maximum (maxlag) and minimum (-maxlag) lag to consider while calculating correlation betweem arr1 and arr2, to align both the series

5. positive_correlation *(boolean, "True" or "False")* :
   - True: This suggest that both the series are positively correlated
   - False: This suggest that both the series are negatively correlated
6. pos *(int, 1 or 0)* : If value of this parameter is 1 than we will consider positive values for lag, i.e. 1 to +maxlag to align both the series initially
7. neg *(int, 1 or 0)* : If value of this parameter is 1 than we will consider negative values for lag, i.e. -maxlag to -1 to align both the series initially
8. default_threshold *(boolean, "True" or "False")* : whether to use default threshold or not. If True, default threshold will be used using MAD test on calculated correlation values for all windows.
9. threshold *(float)* : if default_threshold is False, than this user provided threshold will be used.

- Output *(list)* :
  This function filter out anomalies and returns them. This function returns List of tuples of the form
  (start_date,end_date,correlation_value),
  where (start_date, end_date) specifies range of the window and correlation_value is value of correlation of that window

A.3 Description

Putting all together, here is the summary:

Function "WindowCorrelationWithConstantLag", first makes use of "correlation" function, to calculate correlation values at all lags to find out at which lag it needs to be align. Result of "correlation" function is passed to "getMaxCorr" function. Which will return lag at which optimum value

of correlation is present. This output will be used by "correlationAtLag" function, to caluclate correlation at all windows after aligning both series by input lag. So, in this way "WindowCorrelationWithConstantLag" combines these three functions and returns correlation value at each window.

Function "anomaliesFromWindowCorrelationWithConstantlag" is the main driver. This function calls "WindowCorrelationWithConstantLag" and gets the correlation values at all windows and filters out anomalies (either using threshold calculated by MAD test or by user provided threshold) and returns them in the format of (start_date,end_date,correlation_value), where (start_date, end_date) specifies range of the window and correlation_value is value of correlation of that window.

Appendix B

Slope Based Detection

B.1 Introduction

The method works on two time-series. It finds the ratio of steepness at two different points in the time-series. Let's say we have two time series as series1 and series2. So in this method, we first find the rate of change in the time-series values for both time-series followed by taking ratio of these rate of change. i.e suppose we have two points on time-series 1 as y_{11} and y_{12} and on time-series 2 as y_{21} and y_{22}. Rate of change between these points is calculated as following

$$S_1 = \frac{y_{12} - y_{11}}{y_{11}}$$

,

$$S_2 = \frac{y_{22} - y_{21}}{y_{21}}$$

Ratio of steepness (**rs**) is calculated as

$$rs = \frac{S_1}{S_2}$$

The **rs** is calulated between first and last point of every window of size "w" provided as input.

Now, we have rate of change in the steepness(**rs**) for every window. The outliers are detected by the threshold value provided by user or if not, then threshold is computed using MAD test on the all **rs** calculated above.

If the two time-series are expected to move in tandem, then all the points with **rs** greater than threshold are reported whereas if the two time series should not move in tandem then all the points with **rs** less than threshold are reported.

B.2 Related Functions

B.2.1 slopeBasedDetection(series1,smoothed1,series2,smoothed2 next_val_to_consider, default_threshold, threshold, what_to_consider)

This function smoothes the provided time series data using exponential moving average(if needed) and calculates **rs** for first and last point of every window of size next_val_to_consider.

- Input Parameters

  1. series1 *(list)* : Input series 1 as a list of float values
  2. smoothed1 *(boolean)* : Whether series1 is smoothed or not? If not (value of this parameter is *False*) smoothing will be done
  3. series2 *(list)* : Input series 2 as a list of float values
  4. smoothed2 *(boolean)* : Whether series2 is smoothed or not? If not (value of this parameter is *False*) smoothing will be done
  5. next_val_to_consider *(int)* : indicates the size of window or next point in time-series to calculate slope of steepness. Default is 7 days.
  6. default_threshold *(int)* : Whether to consider default threshold or not. If *True*, the threshold is calculated using MAD Test.
  7. threshold *(int)* : Threshold value to consider if default_threshold is set to *False*
  8. what_to_consider *(int)* : Can be either 1,0 or -1. If the series are supposed to move in tandem, 1 is set otherwise -1 is set. In case we don't know the correlation between two, 0 is set.

- Output *(list)* :
  Returns list of tuples of the form

(first,second,slope_value)

where first and second are the array index of passed series for which the **rs** is calculated.

B.2.2 anomalyDatesSlopeBaseddetetion(slopeBasedResult,any_s

This function basically takes result of slopeBasedDetection as input along with any series which is list of tuples of the form (date, value), and gives date to each anomaly.

The result returned by slopeBasedDetection function just provides index of data point, which is reported as anomaly. But we have time series, so we need to provide date, instead of index of data point. So, this function basically, attaches each anomaly with its date and returns it.

- Input Parameters
  1. slopeBasedResult *(list)* : This is list of anomalies reported by slopeBasedDetection function.
  2. any_series *(list)* : Any list/series of tuples in the format (Date,Value), date will be used from this series to find date against each anomaly.
- Output *(list)* :
  Returns list of tuples of the following form:
  (start_date,end_date,slope_value)

B.2.3 slopeBased(series1,smoothed1,series2,smoothed2,next_val_ default_threshold, threshold, what_to_consider)

This is main function of this anomaly detection technique. This function first calls "slopeBasedDetection" function, gets list of anomalies. After that, it calls "anomalyDatesSlopeBaseddetetion" function to attach date with each anomaly and than returns result.

- Input Parameters

  1. series1 *(list)* : Input series 1 as a list of tuples of the forms (date, value)
  2. smoothed1 *(boolean)* : Whether series1 is smoothed or not? If not (value of this parameter is *False*) smoothing will be done
  3. series2 *(list)* : Input series 2 as a list of tuples of the forms (date, value)
  4. smoothed2 *(boolean)* : Whether series2 is smoothed or not? If not (value of this parameter is *False*) smoothing will be done
  5. next_val_to_consider *(int)* : indicates the size of window or next point in time-series to calculate slope of steepness. Default is 7 days.
  6. default_threshold *(int)* : Whether to consider default threshold or not. If *True*, the threshold is calculated using MAD Test.
  7. threshold *(int)* : Threshold value to consider if default_threshold is set to *False*
  8. what_to_consider *(int)* : Can be either 1,0 or -1. If the series are supposed to move in tandem, 1 is set otherwise -1 is set. In case we don't know the correlation between two, 0 is set.

- Output *(list)* :
  Returns list of tuples of the following form:
  (start_date,end_date,slope_value)
  where, start_date and end_date are points on which the steep was computed along with the slope_value which was spotted as outlier.

B.3 Description

slopeBased function is called to find anomalies based on the rate of change in value. It calls slopeBasedDetection method to compute the slopes between points in windows. Anomaly points are reported based on the parameters

provided to the function.
In order to map dates against every anomaly points instead of array index, anomalyDatesSlopeBaseddetetion is called which provides (start_date,end_date,slope_value) as final output.

Appendix C

Linear Regression

C.1 Introduction

This technique is applied on two time series where one is independent variable and other is dependent variable. Let's say independent variable is "x" represented by series1 and "y" is dependent variable which is represented by series2, where y=f(x).

So, in this method, given values of both variables at different points, i.e. given many pairs of (x,y), which are represented here by series1 and series2, this technique tries to find relation between x and y, i.e. it tries to find best suitable function y=f(x), which can best fit given data. Note that this function can only find linear relation between two variables, i.e. it can find relation such as y = mx + c, where "m" and "c" are some variables, which are found by this method, which can best represent these two series.

After finding that function, for a given value of "x" one can predict, what should be ideal value of "y". So, this technique basically works on this principle. After finding that function, we again apply same function of the given series of "x" and try to predict corresponding series of "y" and see the relative difference between actual "y" series and predicted "y" series. If this relative difference is too high or too low or both (depending upon what user needs), we return those values as anomalies. To decide, whether value is too high or too low, we set up threshold. This threshold can be given by user or can be set automatically by using MAD test on the series generated by taking relative difference. Values beyond this threshold are reported as anomalies.

C.2 Related Functions

C.2.1 linear_regression(x_series, y_series, param = 0, default_threshold = True, threshold = 0)

This function takes two time series, x_series and y_series as input, where x_series is series corresponding to "x" variable (independent variable) and y_series is series corresponding to "y" variable (dependent variable, dependent on "x"). Given these two series, it first finds out best linear relationship between these two variables and as described in the above section, it finds relative difference between predicted and actual "y" series and the ones which are beyond threshold value are reported as anomaly.

As described above, threshold value may be calculated by MAD test on relative difference values by keeping "default_threshold" as "True", and if it is false, user will provide threshold value, by setting up "threshold" parameter above.

Note that i'th value in y_series should be corresponding to i'th value in the x_series.

- Input Parameters

    1. x_series *(list)* : List of float values representing "x" variable (independent variable)
    2. y_series *(list)* : List of float values representing "y" variable (dependent variable)
    3. param *(int, 1 or 0 or -1)* :
       Defines what to be treated as anomaly depending on its value as follows:
       0: Values going out of range, both with positive and negative error
       1: Values with positive errors
       -1: Values with negative errors
       (Here error is relative difference crossing threshold value, positive

error is relative difference which is positive and crossing positive threshold value and vice-versa).

4. default_threshold *(boolean, True or False)* : If this is set as "True", than threshold will be calculated using MAD test, if False, than user given threshold value will be used.
5. threshold *(float)* : Here, user can provide threshold value if, default_threshold is False.

- Output *(Tuple)* :
  returns Following tuple: (result,regression_object)

  Where, "result" is list of tuples which are anomaly according to linear regression test of following format:

  (Index_of_Data_Point,x_value,y_value,predicted_y_value,
  difference_between_predicted_and_actual_y_value)

  "regression_object" is an object of linear regression test, which represents y=f(x) = mx + c, which can be used to regenerate predicted values for plotting graphs afterwards or for some other task.

  Format of using: regression_object.predict(x_value), where x_value is just one value, for which we need corresponding ideal "y" value.

C.2.2 anomalies_from_linear_regression(result_of_lr, any_series)

This function basically takes result of "linear_regression" as input along with any series which is list of tuples of the form (date, value), and gives date to each anomaly.

The result returned by "linear_regression" function just provides index of data point, which is reported as anomaly. But we have time series, so we need

to provide date, instead of index of data point. So, this function basically, attaches each anomaly with its date and returns it.

- Input Parameters

  1. result_of_lr *(list)* : This is list of anomalies reported by "linear_regression" function. Note that here we are just passing list of anomalies only and not the regression object, i.e. we are passing just first element of tuple returned by "linear_regression" function.
  2. any_series *(list)* : Any list/series (x_series or y_series) of tuples in the format (Date,Value), date will be used from this series to attach each anomaly with its corresponding date.

- Output *(list)* :
  Returns list of tuples of the following form:

  (date,x_value,y_value,predicted_y_value, difference_between_predicted_and_actual_y_va

C.2.3 linear_regressionMain(x_series, y_series, param = 0, default_threshold = True, threshold = 0)

This is main function of this anomaly detection technique. This function first calls "linear_regression" function, gets list of anomalies. After that, it calls "anomalies_from_linear_regression" function to attach date with each anomaly and than returns result.

- Input Parameters

  1. x_series *(list)* : List of tuples of the format (date,value) representing "x" variable (independent variable)
  2. y_series *(list)* : List of tuples of the format (date,value) representing "y" variable (dependent variable)

3. param *(int, 1 or 0 or -1)* :
   Defines what to be treated as anomaly depending on its value as follows:
   0: Values going out of range, both with positive and negative error
   1: Values with positive errors
   -1: Values with negative errors
   (Here error is relative difference crossing threshold value, positive error is relative difference which is positive and crossing positive threshold value and vice-versa).
4. default_threshold *(boolean, True or False)* : If this is set as "True", than threshold will be calculated using MAD test, if False, than user given threshold value will be used.
5. threshold *(float)* : Here, user can provide threshold value if, default_threshold is False.

- Output *(list)* :
  Returns list of tuples of the form

  (start_date,end_date,difference_between_predicted_and_actual_y_value)

  Note that here, start_date is equal to end_date, as we are working day-wise in this technique, instead of any window.

C.3 Description

Putting all together, here is the summary:

"linear_regressionMain" is the main function of this technique, which calls 2 other functions and returns result. First it calls, "linear_regression" function, gets list of anomalies. After that, it calls "anomalies_from_linear_regression" function to attach date with each anomaly and than returns result.

Appendix D

Graph Based Anomaly Detection Technique

D.1 Introduction

This technique was introduced by [17]. We have used R implementation given by authors of this book [**?**]. So, here by using python script, we will be just calling R script with appropriate arguments and will be using result provided by that script.

Graph based anomaly detection technique considers each day as a node of a graph. Similar nodes are connected to each other by some weight. Similarity of nodes are calculated by making use of the values of that node i.e. value(s) of timeseries on that date. Based on this similarity, edge weights are also assigned. Then random walk algorithm is applied on this graph structure and connectivity value of each node is calculated. Graph nodes having the least connectivity values are reported as anomaly.

Note that previous techniques, like Window Correlation, Slope Based and Linear Regression techniques, can take only 2 time series as input. They also don't consider historical values, trend or seasonality. It just makes prediction on the given present data. Whereas, this Graph based anomaly detection technique, can take multiple time series as input and also considers trends, seasonality as well, as explained in research paper [17].

So, here, we take multiple time series as input. Out of them, one will be dependent on rest of the others. We will call R script, it will print result in one csv file. We read that CSV file and return result. Note that here we do not have threshold value. We just give number of points with the least connectivity value and function returns them. If in future, one wants to add threshold value on connectivity than function can be modified according to

that as well.

D.2 Related Functions

D.2.1 graphBasedAnomalyCall(dependentVar, numberOfVals, timeSeriesFileNames)

This function calls the R Script "graphBasedAnomaly.R". This function takes multiple time series as input, which are stored in files, whose names are stored in "timeSeriesFileNames" list. This time-series files are generated by us only. Out of these time series, one will be for dependent variable and others will be corresponding to independent variable. So variable, "dependentVar" represents which time series/variable is dependent.

This function executes R script and writes output to the file named "GraphBasedAnomalyOp.csv".

- Input Parameters
  1. dependentVar *(int)* : Index of the dependent variable, where dependentVar = function of independantVars
  2. numberOfVals *(int)* : Each CSV contains how many values? That is each time series has how many values?
  3. timeSeriesFileNames *(list)* : Names of the files in which series is stored. File should contain only series values.
- Output: This function does not generate any output. R Script will write output to CSV file as stated before.

D.2.2 generateCSVsForGraphBasedAnomaly(lists, dateIndex, seriesIndex)

In python code, we have time-series as a list. This list is list of tuples, in which first value of tuple is date and than we have more than one values in

the same tuple, representing different time-series. For example, if we have test-case as onion, than for one city we have 3 time series along with date, which is represented as list of tuples of the the form (date, arrival, wholesale price, retail price). But, for R script, we just need time series values. So this function will take series of time series in variable "lists", where lists[i] will represent one timeseries or multiple time series for one object (like explained previously we can have multiple time series for one city).

dateIndex will say which tuple number for the list lists[i] represents date and seriesIndex represents, if lists[i] represents multiple series than which one to take out of them. This can be explained by example as follows:

Let's say, we have lists as folows:

```
[
[(1-1-2010, x1, y1, z1), (2-1-2010, x2, y2, z2), ... ],
[(1-1-2010, x1, y1, z1), (2-1-2010, x2, y2, z2), ... ],
[...], ...
];
```

So, here we have time-series corresponding to two entities, which can be accessed via lists[0] and lists[1]. Now, lists[0] gives us 3 time-series for one entity. But let's say, here we need only one corresponding to "y" time series. So, give dateIndex as 0 here and seriesIndex as 2. So, this function will create 2 CSVs, one for each entity. Each CSV will have values [y1, y2, y3, ...]. One line will contain one value in file.

Note that it is not necessary to have multiple time-series for on entity. We can have just simple structure as follows:

```
[
[(1-1-2010, x1), (2-1-2010, x2), ... ],
[(1-1-2010, y1), (2-1-2010, y2), ... ],
[...], ...
];
```

So here, we have two time-series as x and y, and we can than give dateIndex as 0 here and seriesIndex as 1. This will create two CSVs, one for "x" and other for "y".

After creating these CSVs, this function returns names of the file created.

- Input Parameters

  1. lists *(list)* : List of time-series, where lists[i] = list of tuple of the form (date, val1 [, val2, val3, ...])
     where date is in form of string and values in square brackets are optional.
  2. dateIndex *(int)* : column number of date in list of tuple (starting with 0)
  3. seriesIndex *(int)* : column number of series in list of tuple (starting with 0)

- Output *(Tuple)*:
  returns tuple of the form, (dates,fileNames),

  Where,
  fileNames: Generated multiple CSVs, corresponding to each series for the input of R script. Returns name of these files.
  dates: Separated date from the series, so that later we can combine result of the R script (anomalies) with dates.

D.2.3 getAnomalies(dates,resultFile, numOfPtsReqd)

This function does the work of combining result of R script with the date. Result generated by R will be in some file, which is passed here as resultFile parameter. This will have indices for each day. So using this we append dates to it. So now, we have connectivity value for each date. This function sorts them according to connectivity value and returns the number of points required stated by parameter numOfPtsReqd, which has low connectivity value.

- Input Parameters

  1. dates *(list)* : List of dates, returned by "getAnomalies" function.
  2. resultFile *(string)* : Path of file to which output of R script is written
  3. numOfPtsReqd *(int)* : Number of anomalous points required

- Output *(list)*:
  reurns list of tuples of the form:
  (start_date, end_date, connectivity_value)
  Note that, here start_date will be same as end_date, as this function returns results day-wise.

D.2.4 graphBasedAnomalyMain(lists, dependentVar, numOfPtsReqd, dateIndex=0, seriesIndex=1)

This is the main function of this method. This function makes call to other functions, uses the output of one, as a input to other, combines all functions and returns generated output.

- Input Parameters

  1. lists *(list)* : List of time-series, where lists[i] = list of tuple of the form (date, val1 [, val2, val3, ...])
     where date is in form of string and values in square brackets are optional.
  2. dependentVar *(int)* : Index of the dependent variable, where dependentVar = function of independantVars
  3. numOfPtsReqd *(int)* : Number of anomalous points required
  4. dateIndex *(int)* : column number of date in list of tuple (starting with 0)
  5. seriesIndex *(int)* : column number of series in list of tuple (starting with 0)

D.3 Description

Putting all together, here is the summary:

"graphBasedAnomalyMain" is the main function. First, it calls "generateCSVsForGraphBasedAnomaly", which will generate files for each series, which will be used as input for R script. It also generates list of dates. Now, this list of file is passed to function, "graphBasedAnomalyCall", which will execute R script and will generate output in predefined file. This file name, along with dates and number of anomaly points required is passed to function "getAnomalies", which will return output in required format.

Appendix E

Multivariate Time Series Anomaly Detection Technique

E.1 Introduction

The method uses vector autoregressive framework for multivaraiate time-series analysis in order to forecast values. The framework treats all the variables as symmetrical and all the variables are modelled as if they influence each others equally.

VAR generates forecast values for all the variables in recursive manner. Since VAR works on only stationary series, lag needs to be found so that the series could be differenced in order to make them stationary.

The code is implemented in R and python is used to call the R script with Appropriate arguments and process the intermeddiate results generated from the script. Forecast and vars library are used in R to implement VAR model for multiple time-series.

All the interrelated time-series are passed to R script using a csv file. 60% (This can be configured as per user need) of the passed data for every time-series is consumed for modelling the time-series. Rest of the time-series data is used to find the anomalies in the system by finding predicted values and range of higher and lower predicted values.

Multiple csv files (one for each varaiable) are generated as an output to the R script call which has actual values of variables along with the predicted ,lower and higher values of prediction. All the points which does not fall in the forecasted range and the percentage differnce between the actual and forecasted value breached threshold are reported as anomalies.

Note that the threshold is computed with MAD test on the percentage of differnce between actual and forecasted value. If in future, one wants to add

threshold value than function can be modified according to that as well.

Refer [**?**] for more detailed information.

E.2 Related Functions

E.2.1 MultivariateAnomaly(fileName,hd, paramCount,fileStart)

This function is inside R script which takes a fileName containing all the related time-series and generate output files (one file for every component) containing predicted, actual, lower and higher forecasted values.

The name of output files starts with fileStart appended with the sequence number of variable. Like if "RetailWsArrival" is passed as fileStart and there are two variables or series in input file. The names of generated file will be : RetailWsArrival1.csv and RetailWsArrival2.csv

- Input Parameters
    1. fileName *(string)* : Name of the file which contains all the inter-related time series for model
    2. hd *(boolean)* : Whether the CSV file contains header for columns or not
    3. paramCount *(int)* : Number of variables in file
    4. fileStart *(string)* : Prefix for the name of output files to be generated
- Output: This function will write output to CSV file as stated before.

E.2.2 multivaraiateAnalysis(args)

This function calls the R script through python passing args as the input to the R script.

- Input Parameters

1. args *(list)* : list of strings which serve as input to R script

- Output: No output.

E.2.3 csvTransform(filePath,startDate)

This function segregate the anomalies from all the other points for which the forcast was generated by R script function named "MultivariateAnomaly".

- Input Parameters

  1. filePath *(string)* : Path of output file generated by R script "MultivariateAnomaly" which contains actual, forecasted, lower and higher values
  2. startDate *(string)* : The date from which the forecast was generated by R script

- Output *(list)*:
  returns list of tuples of the form:
  (start_date, end_date, percDiff)
  Where percDiff is the percentage differnce between the actual and forecasted value. Note that, here start_date will be same as end_date, as this function returns results day-wise.

E.3 Description

Putting all together, here is the summary:

"MultivariateAnomaly" is the R Script function which is called by python function "multivaraiateAnalysis". It generates forecasted values based on the model generated by 60% of the input data. Lastly, csvTransform processes the generated file in order to report anomalies which are falling outside the range of lower and higher forecast value and breach the threshold calculated using MAD test on percentage of differnce between actual amd forecasted value.

Appendix F

Utility

F.1 Introduction

Here, we explain some related funtions which are used by stated anomaly detection techniques. Note that some of these functions can be used to process the results of the anomaly techniques.

F.1.1 Functions used by Anomaly detection techniques

- convertListToFloat(li)
  Given list of elements, this function type casts all the elements to float type.
  - Input Parameters
    1. li *(int)* : list of elements
  - Output *(list)*:
    Returns list of elements after converting each element into float

- getColumnFromListOfTuples(lstTuples,i)

  This function returns i'th element of all tuples as a list.
  - Input Parameters
    1. lstTuples *(int)* : list of tuples. Tuples has no fixed format
    2. i *(int)* : index of which tuple element to return, index starting from zero
  - Output *(list)*:
    Returns list of elements after fetching i'th element from each tuple.

- findAverageTimeSeries(timeSeriesCollection)

  It takes 2D list of element, where each element of timeSeriesCollection is one time series. It returns average of all time series. (first element of resltant time series will be average of first element of all time series)

  For example let,
  timeSeriesCollection: [
  [1,2,3], # Timeseries 1
  [4,5,6], # Timeseries 2
  [7,8,9] # Timeseries 3
  ]

  This function will return,
  [4,5,6]

  - Input Parameters
    1. timeSeriesCollection *(list)* : 2D array of float elements.
  - Output *(list)*:
    Returns list after taking average of all time series.

- writeToCSV(lstData,fileName)

  This writes the list of tuples into the file provided as input.

  - Input Parameters
    1. lstData *(list)* : list of tuples that needs to be written in csv file.
    2. fileName *(string)* : Name of file in which the list of tuples needs to be written.
  - Output *(file)*:
    Generates a csv file with data written in that.

- concateLists(lstData)

  This function converts the list of lists into a single list of tuples.

  For example let,
  timeSeriesCollection: [
  [1,2,3], # Timeseries 1
  [4,5,6], # Timeseries 2
  [7,8,9] # Timeseries 3
  ]

  This function will return,
  [
  (1,4,7), # Timeseries 1
  (2,5,8), # Timeseries 2
  (3,6,9) # Timeseries 3
  ]

  - Input Parameters
    1. lstData *(list)* : list of different lists
  - Output *(file)*:
    Return a single list of tuples.

- cleanArray(array)

  This function removes "nan" (Not a number values) from the list.

  - Input Parameters
    1. array *(list)* : list of float type elements elements.
  - Output *(list)*:
    Returns list of float elements after removing "nan" elements.

- MADThreshold(array)

This function is used to calculate threshold value using Median Absolute Deviation outlier detection method.

- Input Parameters
  1. array *(list)* : Array of integers, real numbers, etc
- Output *(float)*:
  threshold value computed using MAD Test.

- smoothArray(array, alpha = 2.0/15.0)
  This function smooths input array by exponential moving average technique.
  - Input Parameters
    1. array *(list)* : Array of integers, real numbers, etc
    2. alpha *(float)* : smoothening factor of exponential average smoothing
  - Output *(list)*:
    Array which is exponentially smoothed

- csv2array(filePath)

  This function reads csv file into list
  - Input Parameters
    1. filePath *(string)* : Path of file to be read
  - Output *(list)*:
    list of rows in the csv file

- getColumn(array, column_number)

  This function fetches a particular column from a list of tuples.
  - Input Parameters
    1. array *(list)* : List of tuples

2. column_number *(int)* : The index of the column number to be fetched from list of tuples

– Output *(list)*:
list of elements corresponding to the column

- formatCSV2Array(z)

This function returns the same list with changed datatypes of elements within it.

– Input Parameters

1. z *(list)* : List of tuples

– Output *(list)*:
return same array changing data type of second column to float

F.1.2 Functions used to process results

- intersection(numOfResults, list1, resultOf1, list2, resultOf2, list3 = [], resultOf3="linear_regression", list4=[], resultOf4="graph_based", list5=[], resultOf5="spike_detection" , list6=[], resultOf6="multiple_arima")
This function is used to take intersection of results of multiple methods (2 or more), which are passed as list here. This function requires minimum of 5 arguments.

– Input Parameters

1. numOfResults *(int)* : number of lists you are passing, minimum 2
2. $list_i$ *(list)* : list represnting result of the i'th algorithm. Note that this is list of tuples of the format,
(startDate, endDate, value)
3. $resultOf_i$ *(string)*: this variable states, $list_i$ is of which algorithm, it can be from following:
(slope_based, linear_regression, graph_based, spike_detection, multiple_arima)

  - Output *(list)*:
    This function returns intersection of all lists. Returned value is list of tuples of the form:
    (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)

- intersectionOfFinalResults(list1, list2)

  This function takes intersection of 2 lists (where each list is list of tuple) and returns result. Note that these input lists are generated as a output of "intersection" method.

  - Input Parameters
    1. list1 *(list)* : List of tuples of the format:
       (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)
    2. list2 *(list)* : List of tuples of the format:
       (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)
  - Output *(list)*:
    This function returns intersection of list1 and list2. Returned value is list of tuples of the form:
    (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)

- unionOfFinalResults(list1, list2)
  This function takes union of 2 lists (where each list is list of tuple) and returns result. Note that these input lists are generated as a output of "intersection" method.

  - Input Parameters
    1. list1 *(list)* : List of tuples of the format:
       (date, correlation value, slope_based value, linear_regression

value, graph_based value, spike_detection value, multiple_arima value)

2. list2 *(list)* : List of tuples of the format:
   (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)

– Output *(list)*:
  This function returns union of list1 and list2. Returned value is list of tuples of the form:
  (date, correlation value, slope_based value, linear_regression value, graph_based value, spike_detection value, multiple_arima value)

- mergeDates(li)

  This function merges overlapping time period. The list, which it takes as input, "li", is list of tuples, of the format,
  (startDate, endDate, Value).

  So if, we have overlapping period or two time periods are adjacent, for example if one tuple is (1-1-2015, 15-1-2015, 5) and other tuple is (15-1-2015, 30-1-2015, 6), than this function will produce output as, (1-1-2015, 30-1-2015, 5).

  – Input Parameters

    1. li *(list)* : List of tuples of the format:
       (startDate, endDate, Value)
       Note that here startDate and endDate, are of type *datetime* and Value is of type *float*.

  – Output *(list)*:
    This function returns list of tuples of the form:
    (startDate, endDate, Value)

- resultOfOneMethod(array)

This function just converts format of the result list. Usually, anomaly detection methods returns list of tuples of the format,
(startDate, endDate, Value)
this function will convert it to list of tuples of the format,
(date, value)

Basically, all the dates between startDate and endDate will be added to the result list.

- Input Parameters
  1. array *(list)* : List of tuples of the format:
     (startDate, endDate, Value)
     Note that here startDate and endDate, are of type *datetime* and Value is of type *float*.
- Output *(list)*:
  This function returns list of tuples of the form:
  (date, Value) Note that here date is of type *datetime* and Value is of type *float*.