Retail Price Time Series Imputation

A Thesis

Submitted to the Faculty of Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of

Master of Science

in

Computer Science

University of Regina

By

Obaid Ullah Malik

Regina, Saskatchewan

June, 2014

UNIVERSITY OF REGINA

FACULTY OF GRADUATE STUDIES AND RESEARCH

SUPERVISORY AND EXAMINING COMMITTEE

Obaid Ullah Malik, candidate for the degree of Master of Science in Computer Science**,** has presented a thesis titled, ***Retail Price Time Series Imputation,*** in an oral examination held on May 29, 2014. The following committee members have found the thesis acceptable in form and content, and that the candidate demonstrated satisfactory knowledge of the subject material.

External Examiner:	Dr. Shaun Fallat, Department of Mathematics & Statistics
Supervisor:	Dr. Robert Hilderman, Department of Computer Science
Committee Member:	Dr. Howard Hamilton, Department of Computer Science
Committee Member:	Dr. Sandra Zilles, Department of Computer Science
Chair of Defense:	Dr. Tom Phenix, Department of Psychology

Abstract

A regular, discrete time series is an ordered sequence of coarse-grained observations taken at fixed time intervals. Here we consider regular, discrete, retail price time series datasets acquired through crowdsourcing. Crowdsourcing is a means of data collection whereby independent individuals push publicly-available information to an information consolidator who then distributes it back to the individuals for their collective mutual benefit. Crowdsourced datasets are typically incomplete due to missing observations and missing values for important attributes. In this thesis, we consider the problem of filling in missing values in crowdsourced, regular, discrete, retail price time series datasets using data imputation methods. We introduce a new method called Retail Price Time Series Imputation (RPTSI). The basic RPTSI method uses an ensemble of three constituent methods for imputing retail prices in a univariate time series dataset based upon retail prices that already occur in the time series: namely, Price Change Lookup, Central Moving Average, and Polynomial Interpolation. We also introduce four other methods that extend the basic RPSTI method by considering retail prices for similar products sold by the retailer and similar products sold by competitors: namely, RPTSI-C, RPTSI-CR, RPTSI-P, and RPTSI-PCR. The RPTSI-C method finds relationships between a retailer and competitor in terms of those days where they have similar price changes. The RPTSI-CR method finds the relationship between a retailer and a competitor in terms of those days where they have similar price change categories. The RPTSI-P method finds relationships among similar products sold by the retailer. The RPTSI-PCR method integrates the RPTSI-P and the RPTSI-CR methods to find

relationships in terms of similar product prices of the retailer while using prices obtained from competitors.

A crowdsourced dataset containing retail prices from retailers in four North American cities was used in a series of experiments to evaluate the five RPTSI methods. The results obtained were compared against those obtained using Multiple Imputation, Last Value Carried Forward, Mean Imputation, Moving Average, and Polynomial Interpolation. Mean Absolute Deviation (MAD) was used to measure the accuracy of the data filling methods. The RPTSI-CR and RPTSI-PCR methods outperformed Multiple Imputation and other aforementioned methods having the lowest MAD for univariate and multivariate time series, respectively.

Acknowledgment

I would like to express my gratitude to my supervisor, Dr. Robert Hilderman, for his insightful comments and suggestions. Your meticulous comments were of enormous assistance to me, and without your guidance and persistent help this thesis would not have been possible.

I would like to express my appreciation to, Dr. Howard Hamilton, who was an extremely active committee member. You have been a remarkable mentor for me. I owe you my deepest gratitude for the continuous morale and financial support for my graduate study and research. I would like to thank you for your patience, tolerance, knowledge, and constructive criticism during my thesis development. I will be indebted to you always.

I would like to thank Dr. Sandra Zillies, for her ideas while I was performing my research and her comments on my thesis. My sincere thanks also goes to my team mates for their continuous support.

Last but not least, special thanks to my family: my parents Naimat Ullah and Humaira Naz, for all the sacrifices they have made on my behalf. Your prayers were what sustained me thus far. At the end, I would also like to express appreciation to my sisters for sharing my responsibilities and filling in space for me at times when I was not available.

Table of Contents

List of Tables

List of Figures

1. Introduction

In this thesis, we consider the problem of filling in missing data in regular time series datasets acquired through crowdsourcing. Before describing this problem in more detail, we pause to define the terms "time series dataset" and "crowdsourcing".

A *time series dataset* consists of an ordered sequence of observations captured over time. A time series can be regular or irregular, univariate or multivariate, and continuous or discrete. A *regular* time series refers to a collection of observations gathered at fixed time intervals. Yearly sales data and monthly product consumption data are two examples of a regular time series. An *irregular* time series, on the other hand, refers to a collection of observations gathered at irregular time intervals. For example, earthquake occurrence data may be represented as an irregular time series, as earthquakes do not occur at fixed time intervals. A *univariate* time series refers to single-valued (scalar) observations recorded over time. For example, a univariate time series dataset describing the apparent annual consumption of lead in the United States from 1900 to 2011 is shown in Figure 1.1 [1].

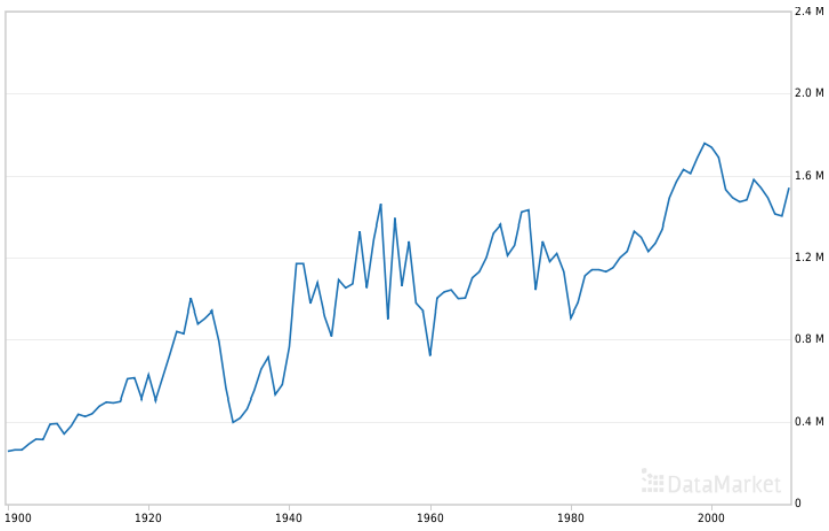


Figure 1.1: A univariate time series showing lead consumption [1]

A *multivariate* time series differs from a univariate time series in that each observation of a time series is a vector-valued observation of time series variables. A multivariate (bivariate) time series dataset is shown in Figure 1.2, where two variables, apparent consumption and imports of lead in the United States, are observed yearly from 1900 to 2011 [1]. A *continuous* time series is a sequence of fine-grained observations that approximate a continuous process. A *discrete* time series is a sequence of more coarse-grained observations, typically gathered at some standard temporal interval, such as once a day or once a year. Thus, a continuous time series is merely a special case of a discrete time series where the loss of information is minimized by more frequent observations. In this thesis, we discuss regular, discrete time series with univariate and multivariate analyses.

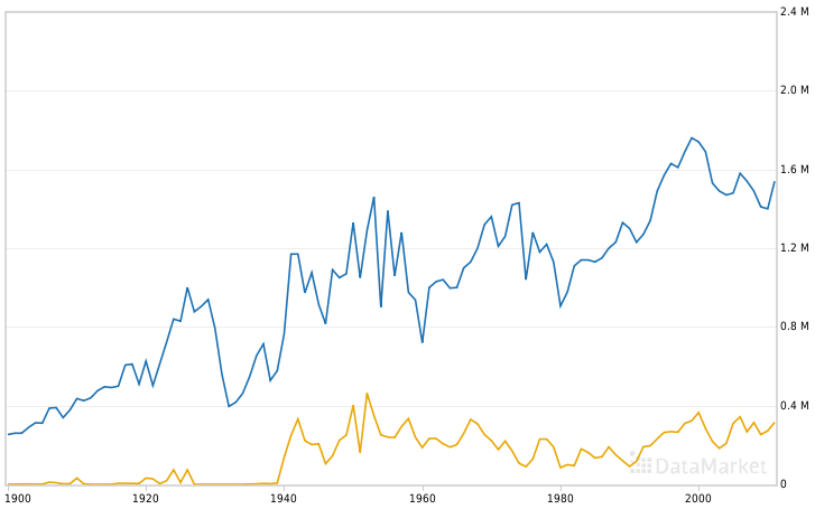


Figure 1.2: A simple multivariate (bivariate) time series [1]

Analysis of a time series often involves detecting its principal components, which include trend, seasonal, cyclical, and irregular effects. The *trend* in a time series is the general direction or movement, which typically refers to a tendency to increase, decrease, or stay the same. A *seasonal* effect refers to regular and fixed period fluctuations, either

rises or falls, in data values that are related to the calendar within a short period of one year (e.g. daily rainfall, monthly fuel consumption, or higher frequency of road accidents in the winter each year). Seasonality can also refer to systematic effects that do not occur exactly at fixed intervals but are related to the calendar, (e.g. Easter or the Muslim Eid festival). If a time series has periodic fluctuations, with rises and falls over varying periods unrelated to the calendar or with periods longer than one year, then a *cyclic* effect may be detected. For example, cyclic behaviour may occur every five years. A time series may also have an *irregular* effect, which affects the data values but is not systematic.

Figure 1.3 shows a graph of the number of airline passengers from 1949 to 1961 [2]. The graph has an overall increasing trend, because of the gradual increase in the number of passengers. High spikes every summer demonstrate seasonal factors. As well, there are random fluctuations that can be categorized as irregular components of the time series.

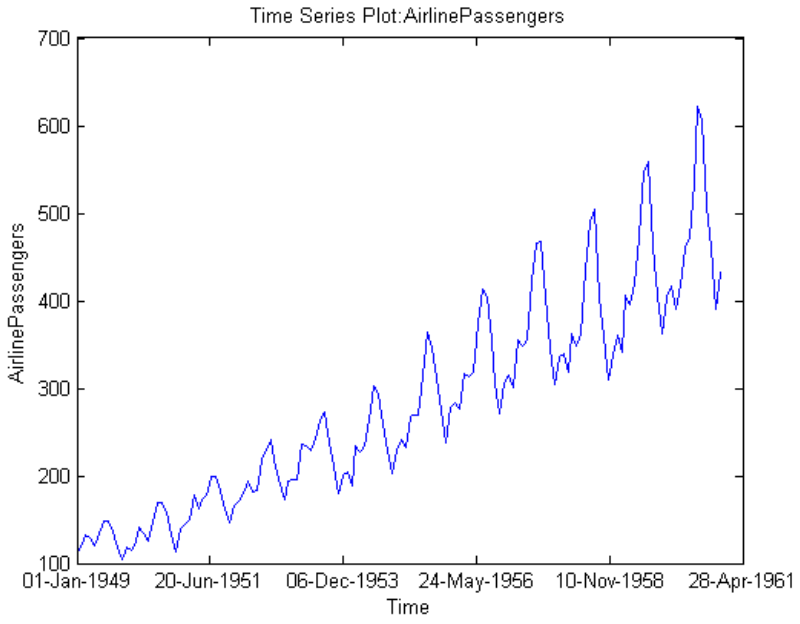


Figure 1.3: Airline passenger dataset [2]

Crowdsourcing was first introduced in the late 1990s by Lévy [3]. It has evolved into a new means of data collection that is being utilized by corporations in an effort to generate higher profits. Some companies, such as Threadless.com and iStockphoto.com, base their business models on crowdsourcing [4]. Estellés-Arolas and González-Ladrón-de-Guevara give a detailed definition of crowdsourcing, which can be summarized as "a type of participative online activity in which an individual, institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, … the voluntary undertaking of a task [which] entails mutual benefit" [5]. To emphasize the fact that crowdsourcing is available to the general public, we define *crowdsourcing* to be a collective group effort where independent individuals share information with an information distributor to make it available for public use for everyone's mutual benefit.

In this thesis, we study time series datasets collected through crowdsourcing. These datasets are typically incomplete and inconsistent. Generally, data preprocessing is needed to help to deal with these problems [6]. Inconsistency is typically addressed by data cleaning, which is beyond the scope of this thesis. Incompleteness in a dataset may be due to either missing observations or observations with missing values for one or more of the input variables. We focus on the problem of incomplete data, and in particular on missing data for univariate or multivariate time series datasets generated by crowdsourcing. Broadly speaking, missing observations can be handled in two ways: (1) by ignoring the missing information and carrying on without it or (2) by filling in the missing values. Missing values for variables in observations can be handled in either of these two ways, or by removing any such observations.

Here, we consider algorithms that fill in missing values, which are called *data filling* (or *imputation*) *algorithms*. These algorithms fill in missing data using a variety of techniques that essentially guess what value should have been present and add it to the dataset. We introduce and evaluate a new data filling algorithm, called RPTSI, which has two primary features. First, it uses a combination of traditional methods in a novel way. These traditional methods are used with slight variations and restrictions. Secondly, when specialized for a class of price-imputation applications, it allows information regarding local competitors and the prices of related products to be incorporated to obtain the most suitable values for missing observations. The focus of this research is to fill in the missing values in a dataset of retail prices with high accuracy.

The RPTSI method can work with any dataset with high variation, frequently reported prices but in this thesis it is evaluated on a dataset containing daily retail prices for petroleum products from multiple stores in multiple cities. An overarching goal of the owner of the dataset is to predict prices based on existing data. Using a dataset with missing values generates predictions of unacceptably low accuracy, because an error in the predicted price of even a few cents is considered to be significant. As well, many well-known time series algorithms were inapplicable because of the problem of missing values. Thus, data filling was used to create a complete dataset that would allow for better price prediction but that is beyond the scope of this thesis and we focused on filling missing values as close as possible to actual values.

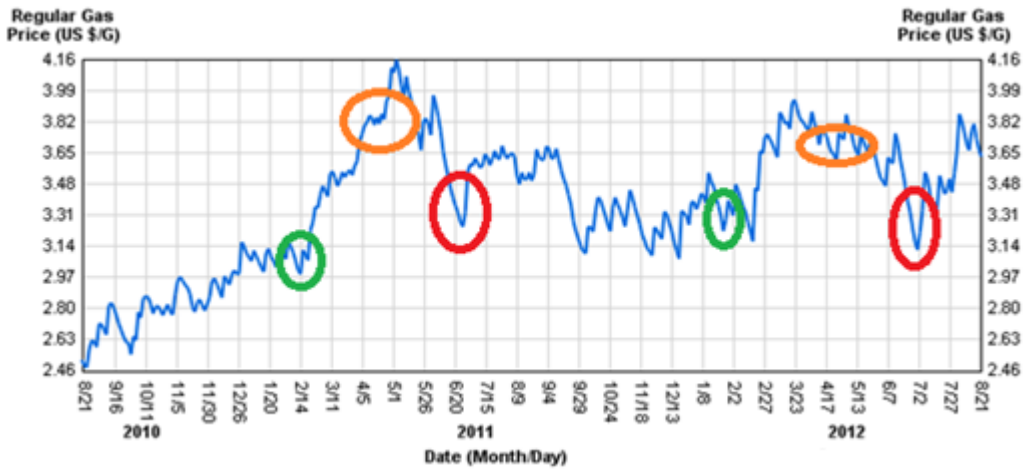


Figure 1.4: Observed trends in City1 gas prices (original in color)

The price of a petroleum product at a store depends on numerous factors, which must be integrated before the price can be predicted. For example, some factors potentially relevant to such prices are the brand, location, competitors, and available facilities (car wash, convenience store, tire shop, etc.). Figure 1.4 shows a graph of the average price of ProductA in a US city, referred to as City1, for a duration of two years, August 2010 – August 2012 [7]. The continual variation in prices between consecutive days illustrates the instability of the price, as do the big jumps. For example, the price rose rapidly from $3.14 to $4.16 in a period of two months starting in February 2011. The trend of the data is increasing. A repetitive pattern of increases or decreases typically indicates seasonality. However, not all significant changes are repeated, suggesting that there are significant irregular effects. Preliminary analysis of the data has revealed a few similarities between the two years that are depicted in the graph. The pair of green circles indicate similar behaviours during the same months in the two years. The pair of red circles indicate two other matching behaviours. However, the orange circles indicate months where the gas prices did not follow a pattern. This unpredictability may have

been caused by an unobserved factor, which adds to the complexity of making predictions.

The dataset used in this thesis was collected using crowdsourcing. Individual users report petroleum product prices, which are combined to form a pool of information for each store. To support the crowdsourcing process, two questions are repeatedly posed concerning the dataset.

1. Is the price reported by the user correct?
2. For the current day, if the price has not yet been reported, what price should be provided to a user who requests it?

Either of these questions can be answered if an accurate prediction can be made for the price of every product at every store. For each price reported by a user, the validity of the price can be assessed by comparing it to the predicted price for the day. For any inquiry about the price at a store, if no value has yet been reported for the day, then the price can be predicted.

The nonlinearity of price changes that makes price prediction difficult, also makes price imputation difficult. The characteristics of our dataset motivated the use of methods different from the ones used for datasets where change occurs more consistently and gradually.

A preliminary analysis of the dataset showed that the assumption of complete data did not hold for a major portion of the data. Although many stores had prices reported for them every day, these reports pertained to four distinct petroleum products (ProductA, ProductB, ProductC, and ProductD), and the price of each product was not reported every

day. For brevity, we refer to the portion of the dataset relevant to City1 as the City1 dataset, the portion relevant to City2 as the City2 dataset, and so forth.

Table 1.1 shows the number of missing values for ProductA in the dataset for the two cities. For example, only 9 out of 267 stores in City1 and only 5 out of 781 in City2 had complete data. Of 661 days of data, only 57.37% of stores in City1 and only 26.89% of stores in City2 had 500 or more days of data.

Table 1.1: Missing data analysis for City1 and City2

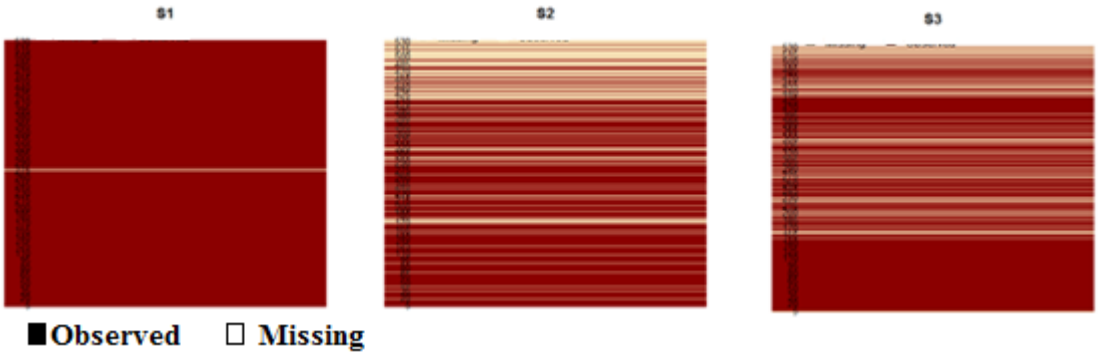
	(City1)		**(City2)**	
Condition	**#Stores**	**%**	**#Stores**	**%**
Total Stores	267	100%	781	100%
Stores with complete data (661 days)	9	3.37%	5	0.64%
Stores with at least 650 days (missing at most 11 days)	26	9.74%	12	1.53%
Stores with at least 600 days (missing at most 2 months)	73	27.34%	69	8.83%
Stores with at least 500 days (missing at most 5 months)	153	57.37%	210	26.89%

Table 1.2: Distribution of missing data for City1 and City2

Missing Data	**5%**	**10%**	**15%**	**20%**	**25%**
City1 Stores	18.28%	25.37%	35.71%	46.64%	52.99%
City2 Stores	4.35%	7.55%	13.09%	19.84%	26.50%

In Table 1.2, the percentages of stores that had various percentages of missing data are shown for City1 and City2. In City1, 18.28% of the stores were missing at most 5%

of the data, and 52.99% of the stores were missing at most 25%. In City2, these numbers were lower; only 4.35% of the stores had at most 5% missing data and only 26.50% had at most 25%. The remaining stores had higher percentages of missing values. Other cities showed similar statistics, with some stores missing up to 85% of the data.



(a) Store 1 **(b) Store 2** **(c) Store 3**

Figure 1.5: Plots of missing price values for three stores in City1

The plots of Figure 1.5 show the missing values for three stores in City1 over a period of 530 days. The stores have different numbers of missing values and show different patterns among the missing observations. Each plot shows the distribution of missing values and observed values. The *y*-axis represents the day, starting with 1 as the first day (shown at the bottom) and 530 as the last day (shown at the top). The first store had only one missing value. The second one had many missing values, with frequent gaps near the end of the dataset. The third store did not have any missing values in the first quarter of the dataset, but was later missing many values.

Overall, the numbers and patterns of missing values suggested that if a time series forecasting method that requires complete data is used for prediction, it would not be applicable to large portions of the datasets. Therefore, a method was needed to fill in

missing values in order to increase the portion of the datasets that could be used for predictions.Retail selling of petroleum products is a competitive business. Information from competing stores can potentially be added to the data of an individual store to improve the imputation. One piece of the information that was not recorded directly, but which could be mined from the data, was the relationships between stores. Previous work by Ahmad on the same dataset show that prices at a given store are affected by those of its competitors [8]. Ahmad identified a set of competitors for each store based on information implicit in the dataset. Spatial competitors were found by multiple methods, including straight line distance within a maximum radius, driving distance, and same route competitors. The use of competitors allowed more accurate predictions to be made, which led us to consider information about competitors while imputing missing values.

Products in the dataset other than ProductA were not included while working with univariate time series. The prices of ProductB and ProductC can be added to the dataset, converting the time series dataset from univariate to multivariate. From here, relationships between different variables of the multivariate time series can be mined and used to get accurate imputations. This research focuses mainly on the univariate dataset with ProductA values because of the dataset owner's interest in imputing ProductA values. Also, the other two products, namely ProductB and ProductC, have higher percentages of missing data, which makes imputation of their values difficult. Nonetheless, the prices for ProductB and ProductC can be used when imputing values for ProductA. Therefore, we use two datasets, a univarite time series dataset for most of the research, and multivariate time series dataset in two of the algorithms.

The remainder of the thesis is organized as follows. In Chapter 2, we discuss background concepts and related work. In Chapter 3, we discuss the RPTSI algorithm for univariate time series and present a step by step walkthrough of it. We also discuss a few variations of the algorithm for both univariate and multivariate time series that are applicable when certain conditions are met by the dataset. In Chapter 4, we present an experimental evaluation of the proposed method and its variants. In each of the experiments, we used the retail price dataset mentioned in this chapter. We also compare the performance of several existing methods with the proposed RPTSI algorithm and its variants. Chapter 5 concludes our work and suggests further research relevant to the problem of filling in missing data.

2. Data Filling Methods for Regular Time Series

Researchers continue to face the problem of missing data. In the past few years, significant efforts have been made to develop methods to deal with this issue. Interest has increased in the area for the simple fact that analysis conducted on a dataset where missing values have been ignored will lead to biased results [9]. If the variables involved in the analysis of the sample dataset have missing values, then the results are prone to false inferences, which clearly suggests that missing values need to be handled before performing any further analysis. Depending on the nature of a time series, various methods have been discussed in the literature to fill in missing data effectively.

Before we look at any methods for data-filling, we should first understand the data and be aware of its characteristics and the mechanism that it follows. Characteristics here refer to the completeness of data.

2.1. Mechanisms for Missing Data

The mechanism causing the data to have missing values needs to be identified before filling in any missing values. Missing-data mechanisms are used to clarify and give some sense of how the data is missing in the dataset. Rubin presents three mechanisms to distinguish and categorize missing data: (1) Missing Completely At Random (MCAR), (2) Missing At Random (MAR), and (3) Missing Not At Random (MNAR) [10].

The mechanisms and dataset distributions specified by Rubin are discussed in this section. Since 1976, when these mechanisms were first identified, they have frequently been used in research to identify the *missingness* and to specify the assumptions made for any proposed data filling method. As a means of illustrating the mechanisms used for

missing data, the sample dataset of Figure 2.1 is referenced throughout this section. It has five variables named *Day*, *Outlook*, *Temperature*, *Windy*, and *Humidity*.

Day	Outlook	Temperature	Windy	Humidity
1	rainy	68	false	65
2	sunny	80	true	93
3	overcast	83	false	80
4	overcast	70	false	65
5	rainy	68	false	80
6	rainy	65	true	70
7	overcast	60	false	65
8	sunny	72	false	95

Figure 2.1: Sample dataset for missing data mechanisms

2.1.1. Missing Completely At Random

Data is *Missing Completely At Random* (MCAR) if the probability that the value for a variable *x* is missing does not depend on any observed variables. For example, consider a wearable device connected to a wireless network that records and uploads the vital signs of a patient every thirty minutes, including Blood Pressure and Heart Rate. If the device skips a few recording points at random, then the patient's record will have missing data. This data would be considered as MCAR because no explanation can be found for the values being missing. Figure 2.2 gives an example of MCAR data from the sample dataset. Suppose that the Humidity variable is missing a value in each row with a probability of 0.5 that is unrelated to all other variables. The probability that the value is missing for the Humidity variable is related neither to other values of the Humidity variable nor to other variables of the dataset.

Day	Outlook	Temperature	Windy	Humidity
1	rainy	68	false	65
2	sunny	80	true	?
3	overcast	83	false	?
4	overcast	70	false	65
5	rainy	68	false	?
6	rainy	65	true	70
7	overcast	60	false	65
8	sunny	72	false	?

The values of Humidity are missing with a probability of 0.5, unrelated to all other variables.

Figure 2.2: Missing Completely At Random (MCAR) example

2.1.2. Missing At Random

Data is *Missing At Random* (MAR) if the probability that a value for variable *x* is missing depends on the other observed variables (e.g. *y* or *z*), but does not depend on the values of *x* itself. Continuing with the previous example, if the wearable device does not record Blood Pressure when the Heart Rate is 190 or above, then the patient's record will have missing values for the Blood Pressure variable. Figure 2.3 illustrates MAR data for the sample dataset where the value of Humidity is generally missing when the value of Outlook is overcast and the Temperature variable has a value of less than 80.

Day	Outlook	Temperature	Windy	Humidity
1	rainy	68	false	65
2	sunny	80	true	?
3	overcast	83	false	80
4	overcast	70	false	?
5	rainy	68	false	80
6	rainy	65	true	70
7	overcast	60	false	?
8	sunny	72	false	95

Humidity is missing if Outlook = overcast and Temperature < 80

Figure 2.3: Missing At Random (MAR) example

2.1.3. Missing Not At Random

Data is *Missing Not At Random* (MNAR) when the probability that the value of a variable *x* is missing depends on the particular values that it takes on. For example, data for the Heart Rate variable that is missing in a patient's record can be MNAR if it is only missing when the Heart Rate is 190 or above. In the sample dataset, if the Humidity measuring sensor gets damaged and does not work when the Humidity is below 70, then the probability of missing values for the Humidity variable is related to its values. This is seen in Figure 2.4.

Day	Outlook	Temperature	Windy	Humidity
1	rainy	68	false	75
2	sunny	80	true	?
3	overcast	83	false	80
4	overcast	70	false	115
5	rainy	68	false	?
6	rainy	65	true	?
7	overcast	60	false	85
8	sunny	72	false	95

Humidity values missing if Humidity is below 70.

Figure 2.4: Missing Not At Random (MNAR) example

2.2. Traditional Methods

This section discusses well-known traditional data filling methods, which include Listwise Deletion, Available-case Analysis, Hot Deck Imputation, Mean Imputation, and Last Value Carried Forward.

To show the working of the methods of Sections 2.2.1 to 2.2.8 and of 2.3.1, we use the sample dataset shown in Figure 2.5. This dataset has five variables, *ID*, *Company*, *Designation*, *Permanent*, and *Pay*. Figure 2.5 (a) shows the complete dataset, and Figure

2.5 (b) shows the same dataset with a few values artificially taken out and considered to be missing values. The latter is henceforth referred to as the Sample Incomplete Dataset.

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	C_1	AreaManager	0	350
2151	C_2	Supervisor	1	870
3551	C_2	Supervisor	0	650
8713	C_2	AreaManager	0	899
2141	C_2	Supervisor	1	800
2351	C_2	Supervisor	0	690
8728	C_3	AreaManager	1	450
8748	C_3	AreaManager	0	324
8348	C_3	Supervisor	1	245
8638	C_3	Supervisor	0	200
8128	C_3	AreaManager	1	439
8564	C_3	AreaManager	0	360

(a) Complete Dataset

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	C_1	AreaManager	0	?
2151	C_2	?	1	870
3551	C_2	Supervisor	0	650
8713	C_2	?	0	?
2141	C_2	Supervisor	1	?
2351	C_2	?	0	?
8728	C_3	?	1	450
8748	C_3	AreaManager	0	?
8348	C_3	Supervisor	1	245
8638	C_3	Supervisor	0	200
8128	C_3	AreaManager	1	?
8564	C_3	?	0	360

(b) Sample incomplete dataset

Figure 2.5: Sample dataset with artificially introduced missing values

2.2.1. Listwise Deletion

An early approach to overcoming the missing value problem was to perform *Listwise Deletion*, also referred to as the *complete case method* [11]. The Listwise Deletion method requires that any instances of the data which have missing values for one or more variable be removed [12]. This approach discards all data instances with missing values, resulting in a dataset of only complete observations. This method is popular due to its simplicity and easy implementation. On the one hand, it ensures that the data does not contain missing values and that no improvised values are added, but on the other hand, it decreases the size of the dataset. By removing observations with any missing values, some information about the dataset is lost, which may lead to biased results.

Sampled Incomplete Dataset

from Figure 2.5

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3551	C_2	Supervisor	0	650
8348	C_3	Supervisor	1	245
8638	C_3	Supervisor	0	200

Figure 2.6: Listwise-Deletion example

Figure 2.6 shows an example of Listwise Deletion. The dataset with missing values shown in Figure 2.5 (b) has 14 instances, and, after the Listwise Deletion is performed, it is trimmed to just 5 instances. For datasets with many missing values, the amount of data discarded is magnified, and, thus, meaningful data is lost.

2.2.2. Available-Case Analysis

The *Available-Case Analysis* (ACA) method removes only those instances that have missing values among the variables to be analyzed [12]. Typically, it removes fewer instances than Listwise Deletion. ACA experiences a similar drawback as Listwise Deletion, specifically biased results, as shown by Ghosh and Pahwa in their research [9].

Sampled Incomplete Dataset

ID	Company	Pay
2365	C_1	460
3452	C_1	390
3562	C_1	?
2151	C_2	870
3551	C_2	650
8713	C_2	?
2141	C_2	?
2351	C_2	?
8728	C_3	450
8748	C_3	?
8348	C_3	245
8638	C_3	200
8128	C_3	?
8564	C_3	360

ID	Company	Pay
2365	C_1	460
3452	C_1	390
2151	C_2	870
3551	C_2	650
8728	C_3	450
8348	C_3	245
8638	C_3	200
8564	C_3	360

Figure 2.7: Available-Case Analysis example

Figure 2.7 illustrates the process of Available-Case Analysis. Suppose that for this example, the variables of interest are ID, Company, and Pay. The dataset (in the middle) represents the subset of Sample Incomplete Dataset with only these three variables. When Available-Case Analysis is applied, the rows with missing values are deleted, which produces the dataset shown at the right side of the figure. In this case, 8 of 14 instances remain, reducing the size of the dataset. For different analyses, the size of the final dataset will vary because it depends on the variables of interest in each analysis.

2.2.3. Hot Deck Imputation

Hot Deck Imputation (HDI) finds the record, *r*, that is most similar to the one with a missing value and replaces the missing value with the value in *r* [13]. For a simple example, similarity between records can be calculated by a distance function that adds up all of the distances between the values of all of the relevant variables other than the variable with a missing value. The distance function can be as simple as one that returns a distance of 0 if the nominal variables have the same values and returns a distance of 1 if they differ. Total distance is calculated by applying the distance function to all of the variables, which are considered to be included for the similarity measure.

ID	Company	Designation	Permanent	Pay($1000)
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	**C_1**	**AreaManager**	**0**	**?**

(a)

ID	Scores
2365	1
3452	2

(b)

Figure 2.8: Hot-Deck Imputation example

Figure 2.8 illustrates the Hot-Deck Imputation algorithm using a subset of the Sample Incomplete Dataset. Figure 2.8(a) shows all instances where the company is C_1; it also shows one instance that is missing a value for the Pay variable. Suppose that three variables, Company, Designation, and Permanent, are chosen to calculate the similarity distances between instances, and a distance value of 0 represents a match, with a 1

appearing otherwise. The scores of the first two rows with respect to their similarity to the third row are given in Figure 2.8(b). For the first row, the Company and Designation variables have the same values as does the third row and, thus, returns a value of 0. For the Permanent variable, the value returned is 1, because the values differ, making the overall score equal to 1 for the first row. In the second row, the values of two variables, Designation and Permanent, are different from those in the third row, resulting in score value of 2 for the second row. The lowest score is given to the instance in the first row with ID 2365, so the value for Pay in that instance (460) is chosen to replace the missing value of Pay.

2.2.4. Mean Imputation

Mean Imputation fills in missing data by using the mean value of the observed values of the variable. This method affects the statistical characteristics of the dataset, specifically by reducing the variance and the standard deviation [14]. On the one hand, mean imputation results in a dataset of the same size after the imputation and the same mean as the original dataset, but on the other hand, it results in an underestimation of the standard deviation.

Sample Incomplete Dataset from Figure 2.5

Mean = 453

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	C_1	AreaManager	0	**453**
2151	C_2	?	1	870
3551	C_2	Supervisor	0	650
8713	C_2	?	0	**453**
2141	C_2	Supervisor	1	**453**
2351	C_2	?	0	**453**
8728	C_3	?	1	450
8748	C_3	AreaManager	0	**453**
8348	C_3	Supervisor	1	245
8638	C_3	Supervisor	0	200
8128	C_3	AreaManager	1	**453**
8564	C_3	?	0	360

Figure 2.9: Mean Imputation example

Figure 2.9 shows how the mean imputation method fills in the missing values of the Pay variable. The mean value of the Pay variable is calculated from the observed values and missing values are imputed from it (imputed values shown in bold).

2.2.5. Last Value Carried Forward

Last Value Carried Forward (LVCF) uses the last observed value to fill in the missing values. This approach can lead to false results and an under or over estimation of the true values. Figure 2.10 shows the complete dataset after filling in the missing values in the Sample Incomplete Dataset using Last Value Carried Forward (imputed values shown in bold).

Sample Incomplete Dataset

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	C_1	AreaManager	0	**390**
2151	C_2	?	1	870
3551	C_2	Supervisor	0	650
8713	C_2	?	0	**650**
2141	C_2	Supervisor	1	**650**
2351	C_2	?	0	**650**
8728	C_3	?	1	450
8748	C_3	AreaManager	0	**450**
8348	C_3	Supervisor	1	245
8638	C_3	Supervisor	0	200
8128	C_3	AreaManager	1	**200**
8564	C_3	?	0	360

Figure 2.10: Least Value Carried Forward (LVCF) example

2.2.6. Central Moving Average

Central Moving Average (CMA) is an extension of the Simple Moving Average (SMA) method. In SMA, the calculated value is dependent on only the past data values, whereas CMA includes an equal number of values on either side of the missing value during a calculation [15]. Typically, both methods are used in data preprocessing for smoothing purposes and include the value at the center position when calculating the average. If the values in a dataset are represented as $\mathcal{D} = \{x_1, x_2, x_3, \ldots, x_n\}$, then the CMA of interval length of N can be calculated as [16]:

$$x_i = \frac{\sum_{j=i-(N-1)/2}^{i+(N-1)/2} x_j}{N}$$

To handle missing data, the value at the center is skipped, given that it is missing. The modified equation is then written as

$$x_i = \frac{\sum_{j=i-(N-1)/2}^{i+(N-1)/2} x_j}{N-1} \quad j \neq i$$

If x_2 is missing and a 3-day CMA is used to find the value, then according to the equation we take the average of $x_1, x_2,$ and x_3. However, x_2 is missing, so we only take the average of its adjacent values, resulting in an average of the values of x_1 and x_3, which makes the denominator $N - 1$.

$$x_2 = \frac{\sum_{j=2-(3-1)/2}^{2+(3-1)/2} x_j}{3-1} \quad \Leftrightarrow \quad x_2 = \frac{1}{2}(x_1 + x_3)$$

To calculate a CMA with an even value of N, half of each tail value is included to adjust the missing value to the center position. The following equation shows a 4-day CMA around the missing record (x_3) that takes half of x_1 and x_5:

$$x_3 = \frac{1}{3}\left(\frac{1}{2} * x_1 + x_2 + x_4 + \frac{1}{2} * x_5\right)$$

Sample Incomplete Dataset from Figure 2.5

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	C_1	AreaManager	0	**630**
2151	C_2	?	1	870
3551	C_2	Supervisor	0	650
8713	C_2	?	0	?
2141	C_2	Supervisor	1	?
2351	C_2	?	0	?
8728	C_3	?	1	450
8748	C_3	AreaManager	0	**347.5**
8348	C_3	Supervisor	1	245
8638	C_3	Supervisor	0	200
8128	C_3	AreaManager	1	**280**
8564	C_3	?	0	360

Figure 2.11: Central Moving Average example

Figure 2.11 illustrates the working of a 3-day CMA on the Sample Incomplete Dataset. To calculate each missing value, the adjacent two values were averaged.

2.2.7. Polynomial Interpolation

Time series datasets are rarely linear, which makes it impossible to fit a linear function. For non-linear behaviour, polynomial functions are used. A first degree polynomial would not give better results as it effectively is a linear polynomial. With higher degree polynomials, values become so smooth that the deviation of the calculated values is reduced. The formal representation of the *Polynomial Interpolation* (PI) is as follows:

When fitting a polynomial of fixed degree of *k* for point (x, y) the equation can be written as:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_k x^k \qquad (1)$$

with n data points: $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$ we require a polynomial of degree $n - 1$ to interpolate. The polynomial equations for degree $m = n - 1$ can be written as:

$$Y_1 = a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 + \ldots + a_m x_1^m$$
$$Y_2 = a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 + \ldots + a_m x_2^m$$
$$\vdots$$
$$Y_n = a_0 + a_1 x_n + a_2 x_n^2 + a_3 x_n^3 + \ldots + a_m x_n^m$$

Using matrix-vector representation, the polynomial interpolation problem can be reduced to the linear equation problem:

$$y = Va$$

where

$$y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \qquad V = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & & x_2^m \\ & \vdots & & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^m \end{bmatrix} \qquad a = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{\mathrm{m}} \end{bmatrix}$$

where y is an n×1 matrix of all $Y_1 \ldots Y_n$, V is an $n \times m$ matrix of all of the values of x, and a is an m×1 matrix of all of the $a_0 \ldots a_m$ variables. To solve this equation for a, Gaussian elimination is applied. The matrix V is known as the Vandermonde matrix and is solvable only if it is non-singular, meaning that the matrix has an inverse and its determinant is not equal to 0. One drawback of this approach is that the Vandermonde matrix has a bad *condition number*, where the condition number is the maximum ratio of the relative error of output to the relative error of inputs. A matrix with large condition number produces big changes in the output for a slight change to the input matrix [17]. The derivation for a is as follows:

$$y = Va$$

$$(V^T V)^{-1} V^T y = (V^T V)^{-1}(V^T V)a$$

$$a = (V^T V)^{-1} V^T y$$

To illustrate the behaviour of polynomial interpolation Figure 2.14(a) shows a portion of the sample dataset with three missing values. For Polynomial Interpolation the data was transformed in Figure 2.14(b) to only include ID and Pay, where ID was given new values to make them consecutive.

ID	Company	Designation	Permanent	Pay
2151	C_2	?	1	650
3551	C_2	Supervisor	0	870
8713	C_2	?	0	**?**
2141	C_2	Supervisor	1	**?**
2351	C_2	?	0	**?**
8728	C_3	?	1	450
8748	C_3	AreaManager	0	324

(a)

ID	Pay
1	650
2	870
3	**?**
4	**?**
5	**?**
6	450
7	324

(b)

Figure 2.12: Missing Data for PI example

Substituting values in equations we get matrices for *y* and *V*

$$y = \begin{bmatrix} 650 \\ 870 \\ 450 \\ 324 \end{bmatrix} \qquad V = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 16 \\ 1 & 6 & 36 & 216 \\ 1 & 7 & 49 & 343 \end{bmatrix} \qquad a = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{\mathrm{m}} \end{bmatrix}$$

Calculating *a* from equation

$$a = (V^T V)^{-1} V^T y$$

gives a polynomial of degree 3:

$$a = \begin{bmatrix} 178.4 \\ 617.6 \\ -156.2 \\ 10.1 \end{bmatrix}$$

$$Y = 178.4 + 617.6x_n - 156.2x_n^2 + 10.133x_n^3$$

ID	Pay (Interpolation)	Pay (Actual)
1	650	650
2	870	870
3	**899.171**	899
4	**798.352**	800
5	**628.325**	690
6	450	450
7	324	324

Figure 2.13: Polynomial Interpolation results

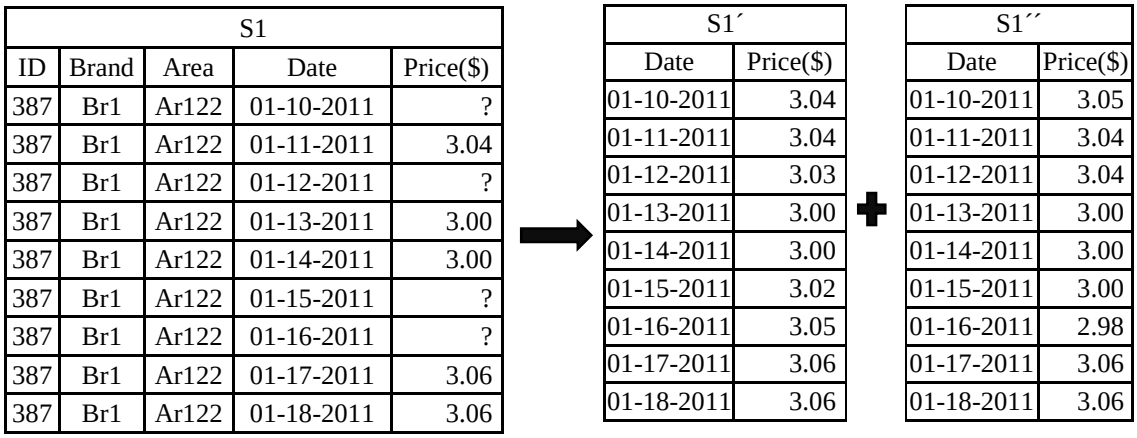
Figure 2.13 shows the missing values filled with Polynomial Interpolation of the third degree. The sample dataset is not a time series dataset that is actually needed for LVCF, CMA, and PI but to show the basic working of methods we continued using it.

2.3. Multiple Imputation

Rubin (1987) proposed *Multiple Imputation* [18], which creates multiple datasets using different imputed values, rather than a single dataset imputing a single value. This approach produces multiple instances of complete datasets with different values imputed in different versions of the dataset. However, it poses another problem, namely how should analyses be performed, and which imputed dataset should be chosen? Rubin suggested performing analyses on each of the imputed datasets, and later combining the results [18]. Multiple Imputation is more easily carried out in a distributed environment as each imputed dataset can be independently analyzed and, later, combined with the others.

Each of the previous methods discussed in this chapter produces single imputation or results in a single complete dataset, whereas Multiple Imputation produces multiple complete datasets. Figure 2.14 gives an example of Multiple Imputation where $m = 2$,

which generates two complete datasets. Figure 2.15 shows the single complete dataset generated after single imputation.



S1				
ID	Brand	Area	Date	Price($)
387	Br1	Ar122	01-10-2011	?
387	Br1	Ar122	01-11-2011	3.04
387	Br1	Ar122	01-12-2011	?
387	Br1	Ar122	01-13-2011	3.00
387	Br1	Ar122	01-14-2011	3.00
387	Br1	Ar122	01-15-2011	?
387	Br1	Ar122	01-16-2011	?
387	Br1	Ar122	01-17-2011	3.06
387	Br1	Ar122	01-18-2011	3.06

S1´	
Date	Price($)
01-10-2011	3.04
01-11-2011	3.04
01-12-2011	3.03
01-13-2011	3.00
01-14-2011	3.00
01-15-2011	3.02
01-16-2011	3.05
01-17-2011	3.06
01-18-2011	3.06

S1´´	
Date	Price($)
01-10-2011	3.05
01-11-2011	3.04
01-12-2011	3.04
01-13-2011	3.00
01-14-2011	3.00
01-15-2011	3.00
01-16-2011	2.98
01-17-2011	3.06
01-18-2011	3.06

Figure 2.14: Multiple Imputation example, where m = 2

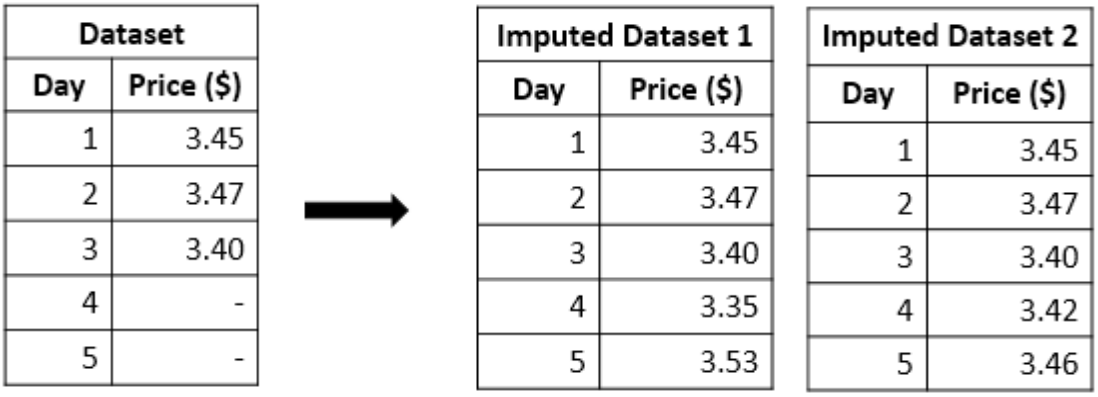
S1				
ID	Brand	Area	Date	Price($)
387	Br1	Ar122	01-10-2011	?
387	Br1	Ar122	01-11-2011	3.04
387	Br1	Ar122	01-12-2011	?
387	Br1	Ar122	01-13-2011	3.00
387	Br1	Ar122	01-14-2011	3.00
387	Br1	Ar122	01-15-2011	?
387	Br1	Ar122	01-16-2011	?
387	Br1	Ar122	01-17-2011	3.06
387	Br1	Ar122	01-18-2011	3.06

S1´	
Date	Price($)
01-10-2011	3.04
01-11-2011	3.04
01-12-2011	3.03
01-13-2011	3.00
01-14-2011	3.00
01-15-2011	3.02
01-16-2011	3.05
01-17-2011	3.06
01-18-2011	3.06

Figure 2.15: Single Imputation example

Multiple Imputation generates a set of possible values for each missing value based on the values of the other variables. The predicted values are referred to as *imputes*. When individual imputes are substituted for each of the missing values in a given set, the result is a complete dataset, called an *imputed dataset*. This operation is performed multiple

times, resulting in multiple complete datasets. Each imputed dataset is independently analyzed. Later, by combining them, one obtains an overall analysis of the data. This approach not only addresses the variability in the data, but also accounts for any uncertainty. An effort is made to model the variability in the original data source by creating imputes for missing values using a model based on the correlation of the other variables with the missing value variable. Uncertainty is handled by creating multiple datasets, each of which has a different impute for the particular missing value.



Dataset	
Day	Price ($)
1	3.45
2	3.47
3	3.40
4	-
5	-

Imputed Dataset 1	
Day	Price ($)
1	3.45
2	3.47
3	3.40
4	3.35
5	3.53

Imputed Dataset 2	
Day	Price ($)
1	3.45
2	3.47
3	3.40
4	3.42
5	3.46

Figure 2.16: Multiple Imputation example

For example, the table of Figure 2.16 is missing two values at the end. Multiple Imputation fills in those values to generate two complete datasets. In this simple example, the imputation model was based on the mean of the observed dataset. While calculating imputations, the mean for the generated datasets was the same as for the original dataset.

To understand the basic model of Multiple Imputation, the following example is included. This Multiple Imputation method uses the regression technique. It first generates multiple regression line equations using the observed dataset. It then uses those equations to create imputes. This enables one to create multiple complete datasets, even

though only one complete dataset is required. In this example, the average of the imputed values is calculated to obtain the final imputation.

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	C_1	AreaManager	0	?
2151	C_2	Supervisor	1	870
3551	C_2	Supervisor	0	?
8713	C_2	AreaManager	0	899
2141	C_2	Supervisor	1	?
2351	C_2	Supervisor	0	?
8728	C_3	AreaManager	1	?
8748	C_3	AreaManager	0	124
8348	C_3	Supervisor	1	245
8638	C_3	Supervisor	0	?
8128	C_3	AreaManager	1	239
8564	C_3	AreaManager	0	560

Equation 1.
Pay = 498.077 - 0.01756 (**ID**) + 545.717 (**C_2**) + 93.51871(**C_1**) – 24.62024 (**DesignationSupervisor**) - 103.21804(**Permanent**) + error

Equation 2.
Pay = 501.047 - 0.01756 (**ID**) + 555.717 (**C_2**) + 92.511(**C_1**) – 25.604 (**DesignationSupervisor**) - 101.214(**Permanent**) + error

Figure 2.17: Multiple Imputation step-by-step: defining equations

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	**C_1**	**AreaManager**	**0**	**?**
2151	C_2	Supervisor	1	870
3551	C_2	Supervisor	0	?
8713	C_2	AreaManager	0	899
2141	C_2	Supervisor	1	?
2351	C_2	Supervisor	0	?
8728	C_3	AreaManager	1	?
8748	C_3	AreaManager	0	124
8348	C_3	Supervisor	1	245
8638	C_3	Supervisor	0	?
8128	C_3	AreaManager	1	239
8564	C_3	AreaManager	0	560

Equation 1.
Pay = 498.077 - 0.01756 (3562) + 545.717 (0) + 93.51871(1) – 24.62024 (0) -103.21804(0) + 71.1
Pay = 600.147

Equation 2.
Pay = 501.047 - 0.01756 (3526) + 555.717 (0) + 92.511(1) – 25.604 (0) -101.214(0) + 76.3
Pay = 607.941

Figure 2.18: Multiple Imputation step-by-step: substituting values in equations

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	**C_1**	AreaManager	0	600.147
2365	C_1	Supervisor	1	870

Imputation with Equation 1.

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	**C_1**	AreaManager	0	607.941
2365	C_1	Supervisor	1	870

Imputation with Equation 2.

Figure 2.19: Multiple Imputation step by step: generating multiple complete datasets

Avg = (600.147 + 607.941) / 2

Avg = 604.044

ID	Company	Pay
2365	C_1	460
3452	C_1	390
3562	C_1	600.147
2151	C_2	870

ID	Company	Pay
2365	C_1	460
3452	C_1	390
3562	C_1	607.941
2151	C_2	870

ID	Company	Designation	Permanent	Pay
2365	C_1	AreaManager	1	460
3452	C_1	Supervisor	1	390
3562	**C_1**	**AreaManager**	**0**	**604.044**
2151	C_2	Supervisor	1	870

Figure 2.20: Multiple Imputation step-by-step: combining the imputed values

2.4. Dynamic Time Warping

Dynamic Time Warping (DTW) was originally proposed in the early 1970s as a means of finding similarities between speech patterns. It has been recently used by Li, Wang, Fang, and Liu to fill in missing data in time series [19]. Their work is based on the curve similarity principle, which suggests that if a feature of interest is similar in the associated time series, then the curves of the time series themselves will also be similar. Given two time series $X = \{x_1, x_2, \ldots, x_N\}$ and $Y = \{y_1, y_2, \ldots, y_M\}$, of lengths N and M, respectively, DTW is used to compare the similarities between them. Comparing two time series using typical distance methods, such as the Euclidean distance, which compares the i^{th} point of X with the i^{th} point of Y produces unsatisfactory similarity scores, as shown in Figure 2.21. DTW, on the other hand, uses a non-linear alignment strategy. This allows similar shapes in two times series to be matched, even when they are out of phase, as is the case in Figure 2.22.

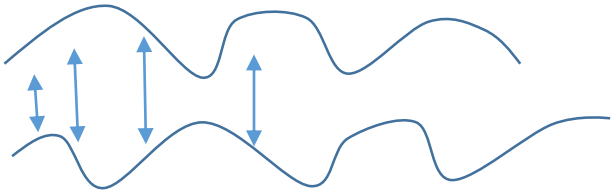


Figure 2.21: Time Series comparison with Euclidean distance

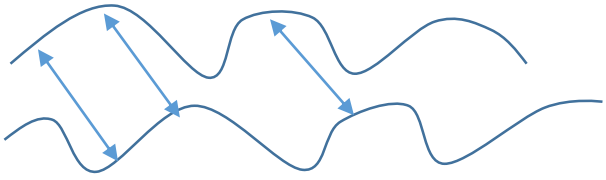


Figure 2.22: Time Series comparison with Dynamic Time Warping

DTW begins by evaluating the cost value of each pair of the two time series, X and Y. The result is a cost matrix. Cost values can be evaluated via a cost function, like the Euclidean distance, though any other cost function will work as well. The goal is to find the best alignment between X and Y, which is calculated by finding a lowest cost path through the cost matrix. The path P is denoted as $P = p_1, p_2, .., p_s, .., p_l$ where $p_s = (x_s, y_s)$. The number of possible warping paths through a grid is generally very large. To reduce the size of the search space, a number of restrictions are added to the warping function [20]:

i) Boundary condition: $p_1 = (1,1)$ and $p_l = (N, M)$.

ii) Monotonicity condition: $n_1 \leq n_2 \leq \ldots \leq n_l$ and $m_1 \leq m_2 \leq \ldots \leq m_l$

iii) Step size condition: $p_{l+1} - p_l \in \{(1,0), (0,1), (1,1)\}$ for $l \in [a: L-1]$

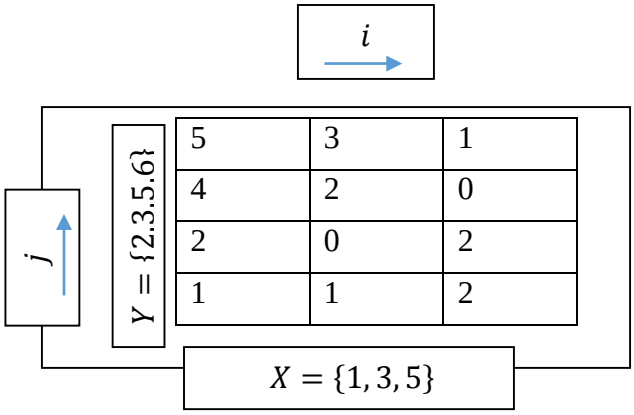
The minimal cost path, between X and Y is called the *optimal warping* path and is denoted by p^*. To find the optimal path, dynamic programming is carried out using a grid search based on the following equation:

$$\text{Initial condition } g(1,1) = d(1,1)$$

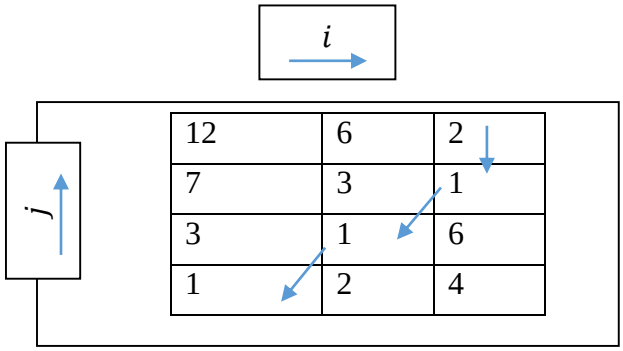
$$g(i,j) = \min \begin{pmatrix} g(i, j-1) + d(i,j) \\ g(i-1, j) + d(i,j) \\ g(i-1, j-1) + 2d(i,j) \end{pmatrix}$$

where $g(i,j)$ is the accumulated cost of the i^{th} point for X and the j^{th} point for Y in the grid, and $d(i,j)$ is the local cost obtained using the cost function of the i^{th} point of X and the j^{th} point of Y. For an example, consider two time series $X = \{1, 3, 5\}$ and $Y = \{2, 3, 5, 6\}$, in which one time series is placed along the *x*-axis and the other along the *y*-

axis, as shown in Figure 2.23. Assume, as well, that the Euclidean distance is used as the local cost function.



(a)



(b)

Figure 2.23: DTW working example

In Figure 2.23, the optimal path p^* is obtained using the cost values (1, 1, 1, 2); that is, this path results in the best alignment of the two time series. Using dynamic programming, the complexity of this algorithm is $O(nm)$. Following this approach, Li, Wang, Fang, and Liu (2012) divide a dataset into two subsets, s_c and s_m, according to

the completeness or incompleteness of the data [19]. Subset s_c consists of all observations with complete data and s_m contains each of the observations in which there is at least one missing variable value. To determine the value of a missing variable, they select one observation from s_m and fill in the corresponding value from an observation in s_c, and, then, calculate the DTW distance between the two time series. For each missing value in s_m, each of the observations in s_c are considered. The time series with the smallest DTW distance is selected. The missing value is then replaced by the corresponding value from the selected time series. Through their experiments, they identified one drawback of their method specifically that it does not work well when missing values affect more than half of the total data. As well, this method is suitable for datasets in which the observations are represented by vectors, but less so among those composed of single variable.

2.5. Discussion

Given that there are several types of time series datasets, as discussed in Chapter 1, the distribution of missing data varies. When handling missing data, most data filling methods make the assumption that the data are MAR or MCAR [13], [21], and [22]. In reality, this assumption does not hold for most datasets. Schafer and Graham considered this problem and suggested that when the missing data are from an unknown distribution and the data is assumed to be MAR, there is no way to test if the assumption holds, other than by gathering data from less reliable secondary resources or filling in values using unreliable models [23]. When a large amount of data is missing and the MAR assumption holds, the Multiple Imputation [18] method is generally preferred over the usual Listwise Deletion or other traditional methods. The precise meaning of the term “large amount” of data is debateable; normally it is thought that when more than 20% of the data

instances have missing values for at least one variable, traditional methods will produce highly biased results [13]. Few experiments have suggested, even when data is not MAR, that the application of modern data filling algorithms produce better estimates with a smaller standard error. The traditional methods, which are simple and easy to apply, may work for the third type, namely MNAR. Typically Missing Not At Random (MNAR) requires complex models to fill in data because there is a lack of information in the dataset in terms of the relevant variables or values. Traditional methods are easy to implement, but do not allow for the extraction and utilization of relations within variables. For this reason, recent research has focused on the first two types, specifically those in which relations do exist among the variables. Recent work involving MNAR datasets by Jansen, Hens, Molenberghs, Aerts, Verbeke and Kenward [24] and Enders [25] has focused on longitudinal data. Good results have been obtained on such data using curve analysis. However, time series datasets are different than longitudinal datasets in that time series datasets have many observations over time from a single source, whereas longitudinal datasets have fewer observations over time from many sources [26].

Farhangfar, Kurgan, and Pedrycz claim that unsupervised methods are more stable and perform better than supervised methods when a large number of values are missing from a dataset [13]. The performance of unsupervised methods decreases more slowly than do those of the supervised methods. Unsupervised methods also tend to perform better in cases where the size of the dataset is not large, with respect to the number of variables and the number of instances in the dataset [13]. The traditional methods discussed in Section 2.2 belong to the category of unsupervised methods.

According to Rubin, the complete observations in MCAR data represent the original set of observations in the data [10]. Any inferences based on the complete observations of MCAR sample data can be generalized to the original dataset, with a possibility of estimates being less precise because the number of observations involving complete cases are smaller. MCAR or MAR data are termed as *ignorable*. This means that during an analysis, the particular reasons for missingness can be ignored [27].

The most common approach for estimating values without biasing the results involves curve fitting [28] , [29]. Using the DTW (Dynamic Time Warping) distance to find the similarity between the complete dataset curve and the incomplete dataset curve, Li et al., 2012 managed to achieve a 66% correctness rate over their dataset involving variables with discrete integer values [19]. However, in our dataset, the percentage of missing values is high and the focus is only one variable. However, the observations were made on three related variables.

The probability of a variable missing a value in our time series dataset is neither related to other variables values nor its own values. This characteristic of our dataset led us to categorize the dataset as MCAR. The multivariate time series dataset falls into the MAR category as different product prices at the same store depend on each other. After conducting a careful data analysis and applying multiple existing algorithms, we concluded that none of the methods described in this section would be effective for our dataset. The traditional methods do not work since our dataset is typically missing more than 20% of the data. Traditional methods also suffer from additional limitations. Either they reduce the size of the dataset, or the values imputed do not represent the true nature of the data, which will ultimately result in biased findings. For example, in mean

imputation, an underestimated standard deviation or the presence of “forced” values towards the mean value may occur. The idea of imputing values, instead of removing instances, is definitely a sound approach. Note, however, that the values imputed using the traditional methods are very “stiff” because they do not account for variability and uncertainty. Gelman and Hill have shown that by introducing an uncertainty error into the imputed values softens the “stiffness”. Their approach suggests that this can only be done when a person knows the true value of a variable and when it is possible to add, to an imputed value, the correct error percentage [12]. To handle missing data in the case of imputation, Multiple Imputation is normally employed. However, this does not typically work well seeing that its effectiveness depends on the presence of values of the other variables in the data. Since we had few such variables, the use of Multiple Imputation resulted in a high degree of error.

3. Retail Price Time Series Imputation (RPTSI) Methods

In order to solve the problem laid out in Chapter 1, an efficient data filling method was required to impute the values for the missing data in a regular univariate time series. We propose the Retail Price Time Series Imputation (RPTSI) method, which is a relatively simple technique for imputing missing values in a regular univariate time series. We also describe variations of the RPTSI method intended to increase the accuracy of the data filling method by mining information about the competitors, and, in the case of multivariate time series, information about other related products.

3.1. Constituent Methods of RPTSI

The RPTSI algorithm applies a combination of three simpler methods to fill in data. They are: (1) Price Change Lookup, (2) Central Moving Average (CMA), and (3) Polynomial Interpolation (PI). Each of these methods is discussed below.

Price Change Lookup (*PCL*): The Price Change Lookup method is applicable when the *k* consecutive data points occurring after the missing data instance have the same value *v*. The value *v* is chosen to fill in the missing data instance. Figure 3.1 and 3.2 show examples where $k = 3$ and the PCL method applies.

Day	Value
D_1	3.12
D_2	3.12
D_3	3.12
D_4	?
D_5	3.12
D_6	3.12
D_7	3.12

Day	Value
D_1	3.12
D_2	3.12
D_3	3.12
D_4	**3.12**
D_5	3.12
D_6	3.12
D_7	3.12

Figure 3.1: Price Change Lookup example (with $k = 3$)

Day	Value
D_1	3.12
D_2	3.12
D_3	3.14
D_4	?
D_5	3.18
D_6	3.18
D_7	3.18

Day	Value
D_1	3.12
D_2	3.12
D_3	3.14
D_4	**3.18**
D_5	3.18
D_6	3.18
D_7	3.18

Figure 3.2: Price Change Lookup example (with $k = 3$)

Central Moving Average (CMA): The Central Moving Average method is applied when the missing data instance is both preceded and followed by *m/2* data instances, and when the PCL method is not applicable. In other words, it must be the case that at least two of the following *m/2* values differ. Figure 3.3 demonstrates an example in which CMA is applicable. This method is discussed in detail in Section 2.2.6.

Day	Value
D_1	3.07
D_2	3.10
D_3	3.12
D_4	?
D_5	3.18
D_6	3.19
D_7	3.22

Day	Value
D_1	3.07
D_2	3.10
D_3	3.12
D_4	**3.15**
D_5	3.18
D_6	3.19
D_7	3.22

Figure 3.3: Central Moving Average example (with order = 5)

Polynomial Interpolation (*PI*): Polynomial Interpolation, which is discussed in Section 2.2.8, is applied when neither PCL nor CMA is applicable. For example, it is applicable when there are two or more consecutive missing data points with at least two

values occurring before the missing data points and two values after. Figure 3.4 shows an example of PI in which there are two consecutive missing values.

Day	Value
D_1	3.56
D_2	3.61
D_3	?
D_4	?
D_5	3.71
D_6	3.71
D_7	3.82

Day	Value
D_1	3.56
D_2	3.61
D_3	**3.66**
D_4	**3.69**
D_5	3.71
D_6	3.71
D_7	3.82

Figure 3.4: Polynomial Interpolation example with missing values

The Price Change Lookup (PCL) method is derived from the Last Value Carried Forward method. In some sense, they work in an opposite way as Price Change Lookup brings the next value to the previous position. Central Moving Average is preferred when the values surrounding the missing value vary. Less biased results are generated when the mean of a small window of changing values is calculated to impute rather than using the mean of a complete dataset. Polynomial Interpolation is preferred because of its ability to produce varying values, similar to the fluctuation of price values.

3.2. Algorithms

This section discusses the five algorithms proposed in this thesis. The simplest version, which uses only the data from a single store, is named Retail Price Time Series Imputation. Two of the other four algorithms depend on information about the competitors of a store. One relies on information about related products obtained using a

multivariate time series, and the other one uses information about other products and competitors. The algorithms are:

1. Retail Price Time Series Imputation (RPTSI).
2. Retail Price Time Series Imputation with Competitors (RPTSI-C).
3. Retail Price Time Series Imputation with Competitors sorted by Range (RPTSI-CR).
4. Retail Price Time Series Imputation with Products (RPTSI-P).
5. Retail Price Time Series Imputation with Products and Competitors sorted by range (RPTSI-PCR).

The RPTSI algorithm is discussed in Section 3.2.1, the method for extracting competitors is described in Section 3.2.2, and the RPTSI-C, RPTSI-CR, RPTSI-P, and RPTSI-PCR algorithms are discussed in Sections 3.2.3, 3.2.4, 3.2.5, and 3.2.6, respectively.

3.2.1. The Retail Price Time Series Imputation (RPTSI) Method

The algorithm was implemented in Java. Weka.core.Instances was used to read all of the records from the input file and store them as a list of Weka.core.Instances; the lists used, *prevDaysList*, *nextDaysList*, and *imputedPrices*, are each ArrayLists with Instance templates.

Algorithm *Imputation*

1. Call function *Initialization*
2. Read input file and store all observations in *Instances*
3. FOREACH instance *I* in *Instances*
4. Get the date value in *currentDate* from *I*
5. Get the price value in *currentPrice* from *I*
6. IF *currentDate* has a missing value
7. Create a local snapshot *S* of the relevant part of *Instances* near *I* by calling function *PopulateLists(h)* //where *h* is the higher value between *k* and *m*/2
8. Call function *RPTSI* on *S*
9. END FOREACH

Figure 3.5: The Imputation algorithm

Figure 3.5 outlines the Imputation algorithm. In line 1, the *Initialization* function is called. This function declares and initializes all required variables and lists. In line 2, the algorithm reads the dataset into an ArrayList. The loop in lines 3 to 9 traverses each record. In each iteration, lines 4 and 5 get the date and price value of the current instance, respectively. In line 6, the algorithm checks for a missing value. If a value is missing, then line 7 takes a local snapshot of the data by calling the PopulateLists function. In line 8, the RPTSI function is called. It imputes values wherever data is missing. The *Intializaiton, PopulateLists,* and *RPTSI* functions are described in further detail below.

Function *Initialization*

1. Set the value for *k* to the number of consecutive days to keep track of after the current day.
2. Set value for *m* variable for the number of instances to take the average of.
3. Create an Arraylist for *imputedPrices* which will contain the potential values to be imputed for missing value(s).

Figure 3.6: The Initialization function

In line 1 of the algorithm of Figure 3.6, the value of the variable *k* is set. It determines the number of following days that will be looked at when applying the Price Change Lookup methods. Line 2 sets the value of the variable *m*, used for the Central Moving Average. The value of *m* determines the number of days around the missing value to consider when applying CMA. Line 3 declares a list that will hold input values and the imputed value (the length of the list depends on the number of many days that are missing between the last reported day and the current day).

Function *RPTSI*

1. IF the first *k* values in *nextDaysList* have the same values
2. Assign imputed variable the first value from *nextDaysList*
3. ELSE IF only one day is missing AND there are *m*/2 values in both, *prevDaysList* and *nextDaysList*
4. Assign imputed variable a value by calculating CMA
5. ELSE IF gap between reported values is 2 or greater AND missing instances are both preceded by and followed by at least two consecutive reported values
6. Assign imputed list values from PI for all missing values

Figure 3.7: RPTSI algorithm

The RPTSI algorithm is represented in Figure 3.7. In line 1, the algorithm checks the applicability condition for Price Change Lookup, as discussed in Section 3.1, by comparing the *k* consecutive following days. If this condition is not met, then, in line 3, the algorithm checks whether the value for imputation can be calculated by Central Moving Average, as discussed in Section 3.1. This condition states that when there is only one value missing and there are *k* preceding and following days with different values, Central Moving Average can be applied. If this condition is not met, then, in line 5, the algorithm checks the applicability condition for Polynomial Interpolation with

degree 3, as discussed in Section 3.1. This condition is satisfied when the dataset has at least two consecutive reported values before and after the gap of missing values. If none of these conditions are satisfied, then there is no candidate value for imputation and the missing value in the dataset is left unchanged. Due to the design of the algorithm, such cases occur only at the beginning and end of the dataset and in portions where there are high concentrations of missing values.

Function *PopulateLists(h)*

1. Create empty snapshot *S*
2. Add up to *h* previous days to snapshot *S* by calling function *PopulatePrevDaysList*
3. Add up to *h* following days to snapshot *S* by calling function *PopulateNextDaysList*
4. Return snapshot *S*

Figure 3.8: RPTSI PopulateLists function

When it comes to the RPTSI method, it is important to keep track of the preceding and following values in the dataset. We have to perform a "lookahead" step, and at the same time, keep a record of the previous values after each iteration. The PopulateLists function in Figure 3.8 checks to see whether there exist *h* days before and after the missing value and then populates the lists of preceding and following values.

S1	
Date	Price($)
01-10-2011	3.04
01-11-2011	3.04
01-12-2011	?
01-13-2011	3.00
01-14-2011	3.00
01-15-2011	?
01-16-2011	?
01-17-2011	3.06
01-18-2011	3.06

S1 ($)	RPTSI		Imputed value ($)
	Method	Imputation	
3.04			
3.04			
?	PCL	3.00	3.00
3.00			
3.00			
?	PI	3.02	3.02
?	PI	3.03	3.03
3.06			
3.06			

Figure 3.9: RPTSI example

Figure 3.9 shows a working example of the RPTSI method. Store S1 has three missing values in the sample dataset. For each missing value, conditions are checked using all three constituent methods of RPTSI. According to the pattern of values surrounding the missing value, one of the constituent methods is applied (i.e. the first one for which the conditions are met). In this example, the first missing value occurs where there aretwo identical following values (3.00). With a value of $k = 2$, the conditions needed for PCL are satisfied. The value following the missing value is selected as the imputation value. The second and third missing value occur together, which causes us to

select the last method, namely PI. There are two reported preceding values as well as two reported following values, thereby satisfying the conditions for PI. The PI determines 3.02 and 3.03 to be the imputation values.

To show the values imputed by these methods on a real dataset, a few samples that match the applicable conditions have been selected from different stores and shown in Table 3.1.

Table 3.1: Missing values imputed with RPTSI for City1 dataset

PCL	CMA	PI
05/01/2011, 3.05	13/05/2011, 4.07	01/18/2011, 3.09
06/01/2011, 3.05	15/05/2011, 3.97	01/19/2011, 3.15
07/01/2011, 3.05	17/05/2011, 3.97	01/20/2011, 3.12
08/01/2011, 3.03	**18/05/2011, 3.925**	**01/21/2011, 3.14**
09/01/2011, 3.03	19/05/2011, 3.89	**01/22/2011, 3.10**
10/01/2011, 3.03	20/05/2011, 3.87	01/23/2011, 3.09
11/01/2011, 3.03	21/05/2011, 3.77	01/24/2011, 3.11
		01/25/2011, 2.99

3.2.2. Extracting Competitors

As discussed in Chapter 1, information concerning the competitors of a store is identified as potentially relevant when choosing values to impute for a store. One of the challenges of this research is to identify the competitors before developing techniques or methods that use their information. The competitor's information is not stored or identified in the dataset. We develop an algorithm to extract the competitors of a store from the dataset using the price values reported for each store. The process of determining competitors is applied to the original dataset before calculating any imputations. It is called a "profile building" process, because, at the end of this process, each store has a profile holding the information of its competitors. To reduce the memory requirements,

the number of stores which are considered to be in the same range is restricted. In this research, the threshold is set to 5, which allows for a maximum of five competitors in a single range for a given store. Figure 3.10 illustrates the process of this algorithm in pictorial form.

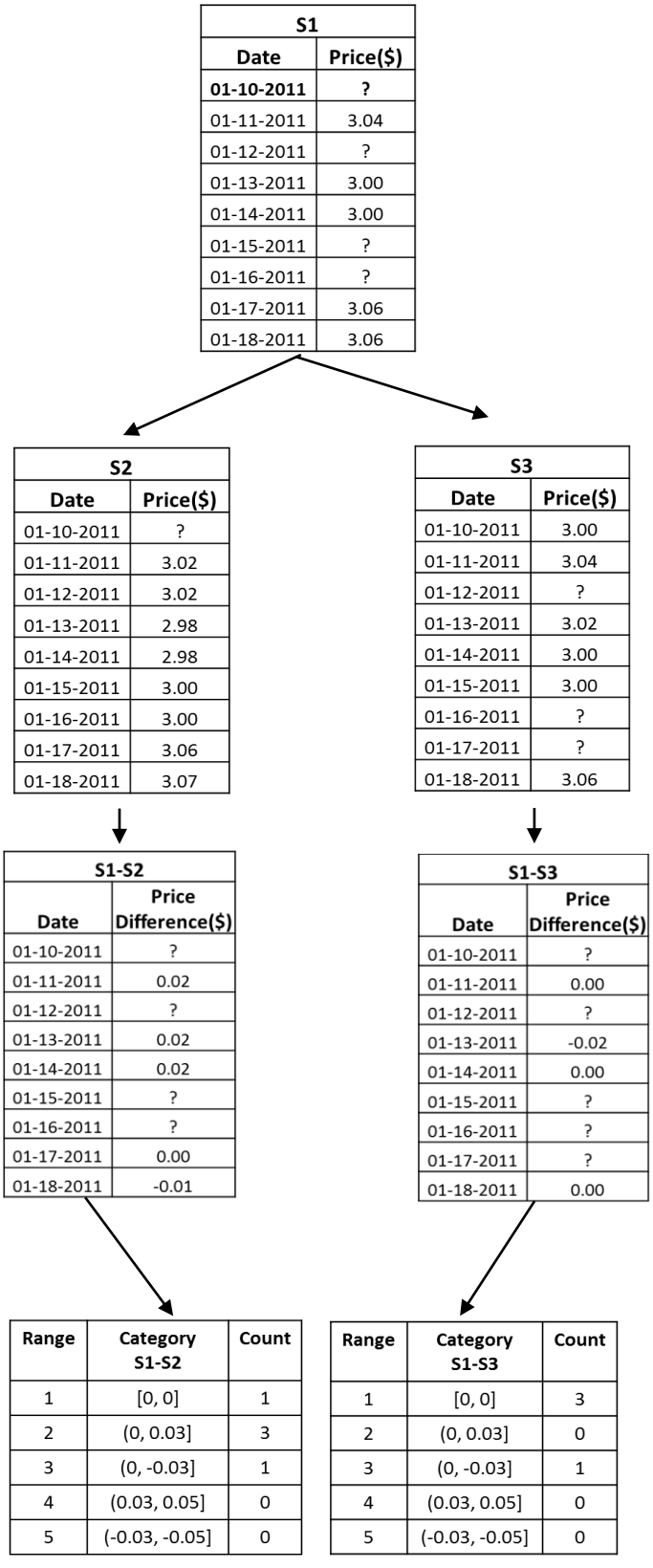


Figure 3.10: Mechanism for extracting competitors

Figure 3.10 demonstrates the process that was used to find competitors and to store that information for a store. Suppose that S_1 is the base store for which we need to perform the imputation, where the ? symbol denotes the missing values, and S_2 and S_3 are two other stores in the dataset. To find the best competitor, the process of finding a competitor for S_1 is applied to both S_2 and S_3. The first step is to find the difference of the value on each day between S_1 and S_2 and also S_1 and S_3. The information relating to the difference of the values between the two stores is stored in a separate table called the difference table. The difference table typically has a higher number of missing values than do the original stores since it is the union of the two stores and a comparison is made only when both stores have reported values on a particular day. To categorize these differences, we created five range blocks from -0.05 to 0.05. During the initial experiments, only those competitors who competed within a 5¢ ($0.05) difference were considered. This was specifically done at the request of the dataset owner. The categories and range numbers are presented in Table 3.2, which elaborates on the ranges and their boundaries. In the figure, *range1* corresponds to the same values of S_1 and S_2, *range2* corresponds to the differenced value of greater than 0¢ and less than or equal to 3¢, and so on. Once the difference is calculated, the number of days having differenced values corresponding to each range are determined and recorded separately for each store. Then, among each of the stores compared, the store having the highest count, along with its range, is recorded against the base store. Once the base store has been compared with all other stores, the store with the highest count in the list is selected as the strongest competitor.

Table 3.2: Differenced table

Range	Category S1-S2	Count
1	[0, 0]	
2	(0, 0.03]	
3	(0, -0.03]	
4	(0.03, 0.05]	
5	(-0.03, -0.05]	

Function *BuildProfile*

1. FOREACH pair of stores in dataset
2. FOREACH day *d* in data
3. IF stores Store 1 and Store 2 have values S_1 and S_2 for day *d*
4. Set *diff* to the difference $S_1 - S_2$
5. IF *diff* is equal to 0
6. Increment the counter of *range1* for Store 1 and Store 2
7. ELSE IF *diff* is greater than 0 but less than or equal to 0.03
8. Increment the counter of *range2* for Store 1 and Store 2
9. ELSE IF *diff* is less than 0 but greater than or equal to -0.03
10. Increment the counter of *range3* for Store 1 and Store 2
11. ELSE IF *diff* is greater than 0.03 but less than or equal to 0.05
12. Increment the counter of *range4* for Store 1 and Store 2
13. ELSE IF *diff* is less than -0.03 but greater than or equal to -0.05
14. Increment the counter of *range5* for Store 1 and Store 2

Figure 3.11: The BuildProfile algorithm

Figure 3.11 shows the BuildProfile algorithm used to mine information about different competitors. The algorithm first checks the difference in prices for a day, and, if the differenced value is equal to 0, then the *range1* counter is incremented; if the differenced value is between 0 and 0.03 inclusive, then the *range2* counter is incremented; and the same goes for the next three ranges. Once all of the counters have been updated for the differenced values of two gas stores, the maximum count is stored

with its range. In the example discussed in Figure 3.10, the maximum count for S_2 is 3 in *range2*. The maximum count for S_3 is also 3, but in *range1*. This information, including the maximum count and corresponding range, is recorded as the profile of S_1 under the assumption that S_2 and S_3 are competitors of S_1.

3.2.3. The Retail Price Time Series Imputation with Competitors (RPTSI-C) Method

The RPTSI-C algorithm is an extension of RPTSI that extracts competitors and uses their values as added information when imputing values (see Figure 3.13). Before the imputation process starts, profile building is performed, as described in Section 3.2.2, and a competitor list is generated for each store. In RPTSI-C, the basic imputation process of RPTSI remains unchanged. A value is generated for a potential imputation, but, before imputing that value into the dataset, the potential imputed value is compared with a competitor's value from the same day.

Function *RPTSI-C*

1. Build profile *P* for all stores, as described in Section 3.2.2.
2. Sort list of competitors for all stores in *P*, according to their *count* field.
3. FOREACH store *S* in dataset
4. FOREACH day *d* in *S*
5. IF *d* has missing value
6. Get competitor *C* with the highest *count* value
7. REPEAT UNTIL a value is found.
8. IF *C* has missing value for *d*
9. Update *C* to the next highest count competitor.
10. Get the potential imputed value v_1 from RPTSI algorithm and value v_2 from *C*.
11. Get the range r_c value of *C*
12. Take the difference *diff* of both values, v_1 and v_2
13. Find the range, r_{diff}, of *diff*.
14. If r_{diff} is equal to r_c, assign v_1, else assign v_2 to the imputed variable.

Figure 3.12: The RPTSI-C algorithm

The store for which there are missing values to be imputed is called the base store. RPTSI-C works under the assumption that the stores with higher counts are more likely to be selected as the most suitable competitor. When locating competitors, the first step is to build differenced tables of the base store involving all other stores in the dataset. From each differenced table, the range with the highest count is recorded against the store in the vector list of the base store, where each vector is a three-tuple vector of a store *identifier*, a *count*, and a *range*. Once all of the stores in the dataset have been traversed, we will have obtained a vector list for the base store in which all of the stores in the dataset are regarded as potential competitors. The RPTSI-C method determines a store to be the best possible competitor on the basis of its *count* value. To get a list of potential competitors using the RPTSI-C method, we sort the list according to these *count* values. The store with the highest count is considered to be the best competitor, the second store in the list is considered to be the second best competitor, and so on. Having several competitors is useful at times when the competitor with the highest count is also missing a value for a specific day. As illustrated in Figure 3.12, the method is designed in such a way that it will continue scanning down a list until it has found the competitor that has a value for a specific day, or the list has been exhausted. During the imputation process, a potential value is calculated by the RPTSI method and another value is taken from the best competitor. If the competitor has a missing value on the same day as well, then the RPTSI-C method scans down to the next store in the sorted list. When we have obtained values from both RPTSI and a competitor, we calculate their difference and compare the range of differenced values with the range of the competitor. If the range is the same,

then the value from the RPTSI method is selected as the final imputation. If the range differs, then the value from the competitor is selected as the final imputation.

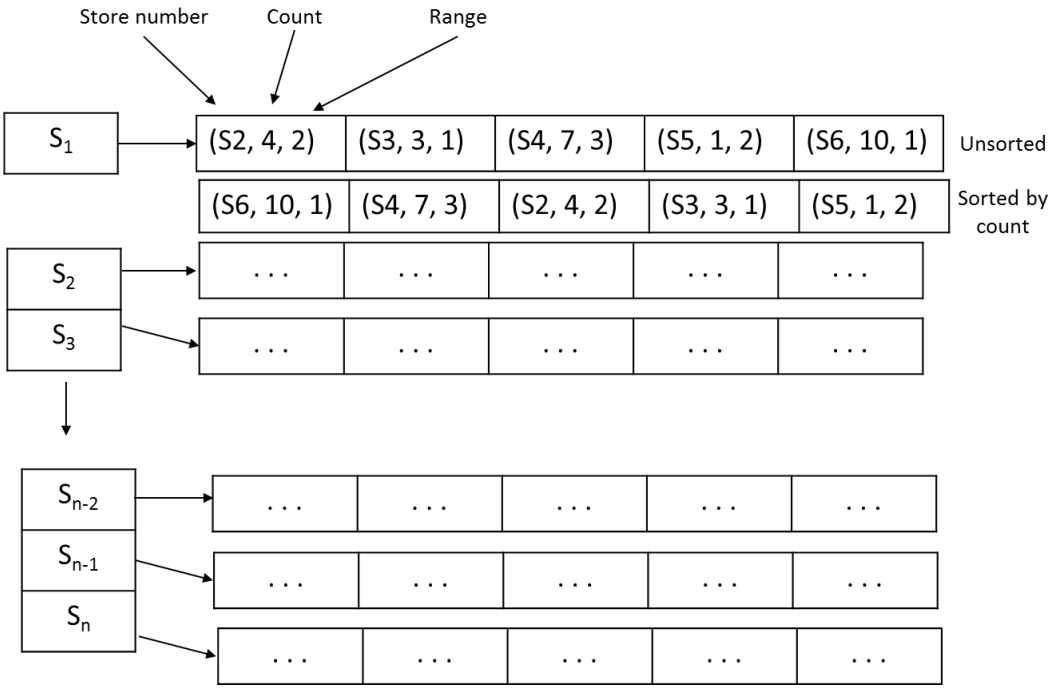


Figure 3.13: Example of RPTSI-C

Figure 3.13 gives a graphical representation of the RPTSI-C algorithm and shows how it uses competitors. S_1, S_2, … S_n are the stores in the dataset. RPTSI-C is applied to each of them individually. The list to the right of S_1 shows the five competitors, along with their counts and ranges. Each competitor is shown as a three-tuple consisting of a store *identifier*, a *count*, and a *range*. The second row shows the competitors after sorting according to the count values. Each competitor also has a range value, selected from the differenced table, associated with the count, which is used to decide whether the value from the competitor should be used for a particular imputation or ignored. For each

imputation, the search for the most suitable competitor always starts from the beginning of the list.

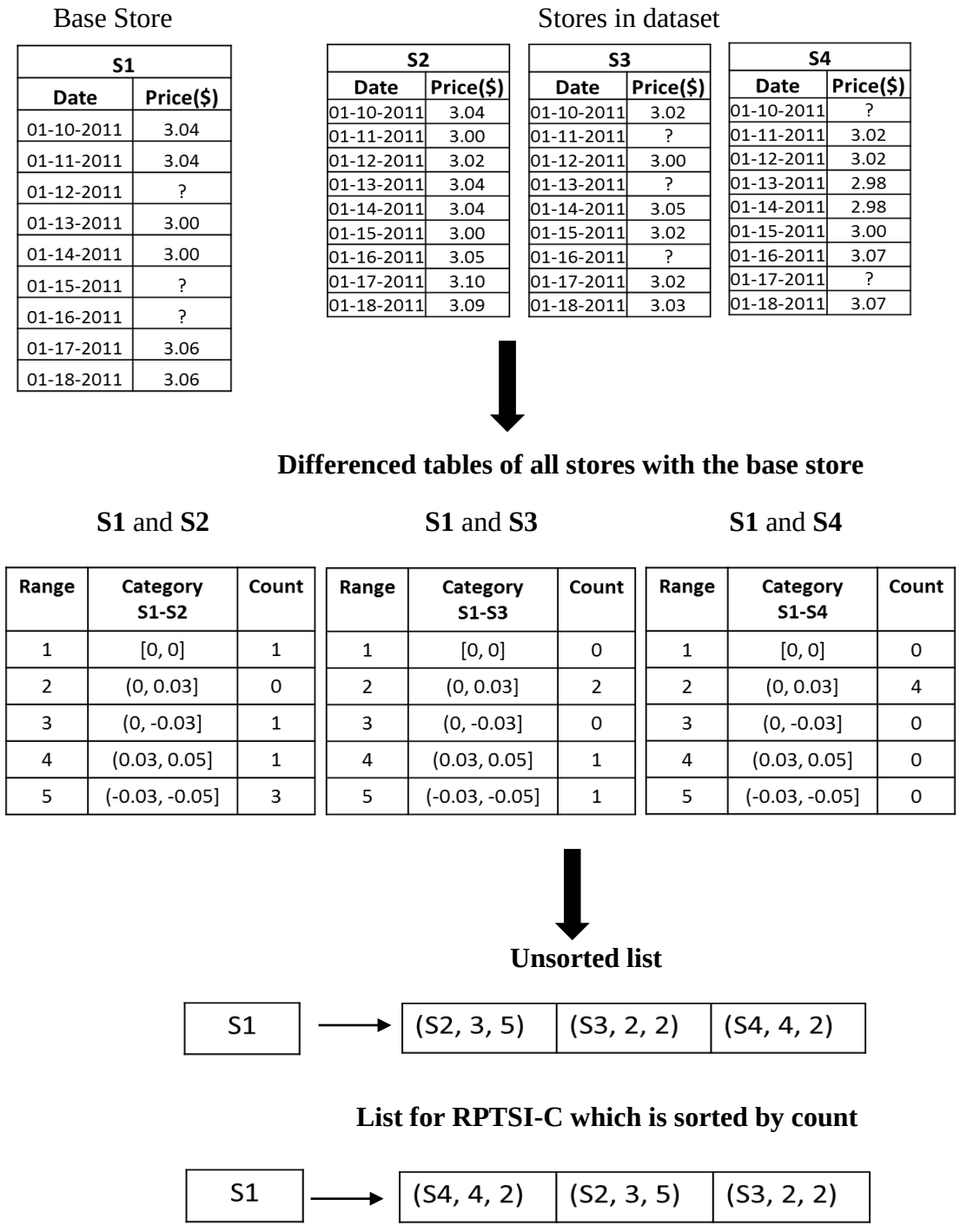


Figure 3.14: RPTSI-C example

S1 ($)	RPTSI		S4 ($) (Competitor with range 2)	Difference |RPTSI – S4|	Within same range	Imputed value
	Method	Imputation				
3.04						
3.04						
?	PCL	3.00	3.02	-0.02	No	3.02
3.00						
3.00						
?	PI	3.02	3.00	0.02	Yes	3.02
?	PI	3.03	3.07	-0.04	No	3.07
3.06						
3.06						

Figure 3.15: RPTSI-C example (continued)

Figure 3.14 shows a working example of the RPTSI-C method. In the sample dataset of this example, there are four stores, in particular S1, S2, S3, and S4, each with nine days of data. S1 is the base store and the others are the potential competitors. As the first step in RPTSI-C, we calculate the differenced tables between the base store and the potential competitors. The differenced table of S1 and S2 in Figure 3.14 shows that *range1*, *range3*, and *range4* each have a count of one, meaning that there are three days on which the difference of the values for S1 and S2 lies under those ranges. *Range5* has a count of three, which indicates that for most of the days the difference of values in both stores lies between -0.03 and -0.05. The same process is applied to the other two stores. As well, and their differenced tables are generated. From each differenced table, only the high count value is recorded, along with the range and store identifier, in a three-tuple vector and stored in a list of the base store. In this example, the *count* for S2 is 3 and range is *range5*, both of which are stored in the list of S1, represented as (S2, 3, 5), as shown in Figure 3.14.

For all of the stores in the dataset, three-tuple vectors are generated with the base store and are then sorted according to their count values. The sorted list is then used by RPTSI-C to identify the most suitable or least suitable competitors for the base store. In this example, the list comes out as (S4, 4, 2), (S2, 3, 5), and (S3, 2, 2). Figure 3.14 shows the data values of S1, including the missing values. For those missing values, the RPTSI method calculates a potential imputation value which is then compared with the value of the competitor, S4 in this example. For one of the missing values, the differenced value is within the same range as S4. Thus, the value (3.02) from the RPTSI is used as imputation. For two missing values, the differenced values are not in the same range. So, the values from a competitor (3.02 and 3.07) are used as imputation.

3.2.4. The Retail Price Time Series Imputation with Competitors Sorted by Range (RPTSI-CR) Method

The RPTSI-CR algorithm is similar to the RPTSI-C algorithm, though it gives more weight to the *range* value of the competitor than to the *count*. The RPTSI-CR method chooses the most suitable competitor by range and then by count. Unlike the RPTSI-C, this algorithm first sorts the list of competitors by range, bringing all *range1* competitors to the beginning of the list, and then all *range2* competitors, and so on. Once the competitors have been arranged by ranges, the RPTSI-CR method sorts them by the count values within that range.

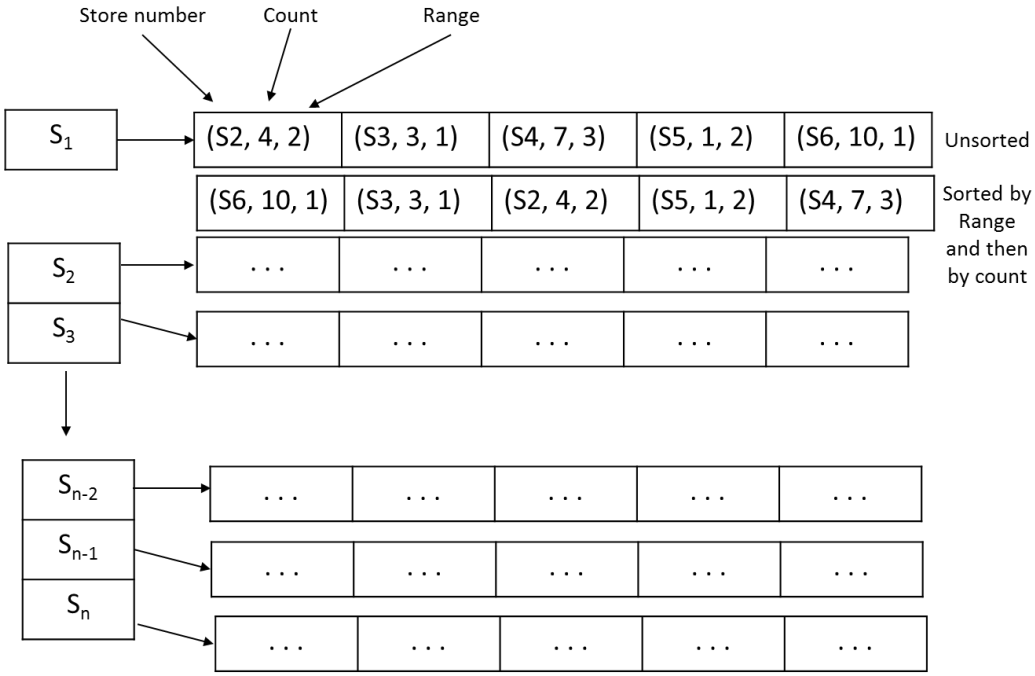
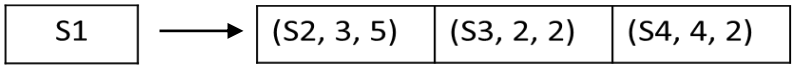


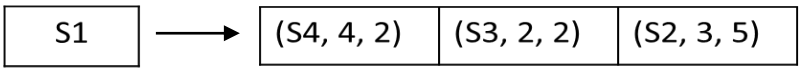
Figure 3.16: Working of the RPTSI-CR method

Figure 3.15 gives a graphical representation of the RPTSI-CR method. As in the previous example, S_1, $S_{2,\ldots}$ S_n represents the stores in the dataset. The rows beside them represent the sorted lists of their competitors. The first row of S_1 shows the same list of competitors as shown in Figure 3.13. Note, however, that the second row has the competitors sorted primarily according to the range values, which brings S_3 and S_6 to the top. Then, the competitors are sorted by count, which brings S_6 to the top of the first range as the most reliable competitor. During imputation, the difference between the potentially imputed value and the value of the competitor is calculated. If the difference value is within the same range indicated by the competitor, then the value from the RPTSI method is used as the imputation value. If the range value is different, then the competitor's value is taken as the imputation price, much like is done in the RPTSI-C method.

Unsorted list



List for RPTSI-R which is sorted by range and then by count



S1 ($)	RPTSI		S4 ($) (Competitor with range 2)	Difference (S1 – S4)	Within same range	Imputed value
	Method	Imputation				
3.04						
3.04						
?	PCL	3.00	3.02	-0.02	No	3.02
3.00						
3.00						
?	PI	3.02	3.00	0.02	Yes	3.02
?	PI	3.03	3.07	-0.04	No	3.07
3.06						
3.06						

Figure 3.17: RPTSI-CR example

In Figure 3.16, a working example of the RPTSI-CR is shown. Using the same sample dataset as in Figure 3.14, we generate the vector list for the base store, specifically S1. The sorting of the vector list differs in the RPTSI-CR method, as discussed earlier in this section. In the RPTSI-CR method, the list is first sorted by the range, which brings S3 and S4 to the top of list, because of the smallest range value, and pushes S2 to the bottom. Then, sorting is performed on each range according to the count values. This brings S4 to the top because of the count value 4. Once the list has been sorted by range and then by count, the remainder of the process, similar to the one discussed in Section 3.2.3, is executed. In this example, the best competitor remains the same as S4, while the

rest of the order changes. The imputed values in this case will also remain the same as in the previous example of Figure 3.14 because there is no instance in which both the base store and the competitor have missing values. If there were any observations in which both were having missing values, then the next competitor would have been S3, different from what we have in the example in Figure 3.14.

3.2.5. The Retail Price Time Series Imputation with Products (RPTSI-P) Method

Recall from Chapter 1 that it was hypothesized that the prices of each of the products in the dataset might be related to each other. To exploit this characteristic of the dataset, we designed the RPTSI-P method to use the prices of other products from the same store as a factor when filling in missing data. To do so, we include the prices of closely related products in our dataset. Next, we build a profile for each store to find the relationship between the prices of its products.

Date	ProductA ($)	ProductB ($)	ProductC ($)
03-04-2011	3.49	?	?
03-05-2011	3.49	3.59	3.69
03-06-2011	?	3.59	3.69
03-07-2011	3.49	3.59	3.69
03-08-2011	3.49	3.59	3.69
03-09-2011	3.49	3.59	3.69
03-10-2011	3.59	?	?
03-11-2011	?	3.69	3.79
03-12-2011	3.54	3.69	3.79

Sample data from a store

(a)

Date	ProductA ($)	ProductB ($)	ProductC ($)
03-05-2011	3.49	3.59	3.69
03-07-2011	3.49	3.59	3.69
03-08-2011	3.49	3.59	3.69
03-09-2011	3.49	3.59	3.69
03-12-2011	3.54	3.69	3.79

Temporary dataset after Listwise Deletion

(b)

Figure 3.18: BuildProfile for product types

Figure 3.17(a) shows sample data from a store, including Date, ProductA, ProductB, and ProductC variables. First, we create a temporary dataset by performing Listwise Deletion on the dataset to obtain only the observations with reported values for all three product types. This reduces the size of the dataset, but at this stage we are only interested in the relationship between the prices on a day, and we must have the prices for all of the products to get this.

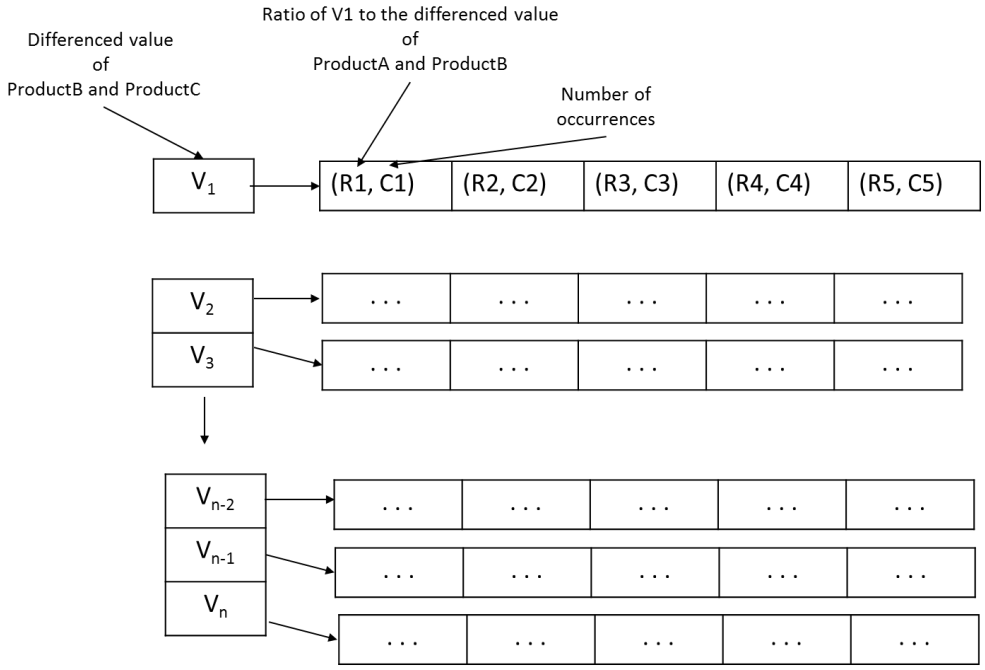
Following Listwise Deletion, we apply the RPTSI-P *BuildProfile* algorithm to the temporary dataset, as shown in Figure 3.18. The algorithm first finds the differenced

prices of ProductA and ProductB (called *diff_AB*), the differenced prices of ProductB and ProductC (called *diff_BC*), and then the ratio between these differences (called *ratio*). It also finds their occurrences in the observations (called *count*) i.e. count the occurrences of *ratio*, R, when diff_BC is D (where R is any ratio and D is any value).

Function *RPTSI-P BuildProfile*

1. FOREACH store *S* in dataset
2. Create a HashMap map_S for store *S*
3. FOREACH day *d* in data
4. IF *S* has values for ProductA, ProductB, and ProductC
5. Set *diff_AB* to the difference ProductB – ProductA
6. Set *diff_BC* to the difference ProductC – ProductB
7. Set *ratio* to the ratio of *diff_AB* to *diff_BC* rounded to the nearest 0.1
8. IF *diff_BC* $\neq 0$
9. IF *diff_BC* does not exist in *map*
10. Add *diff_BC* to *map*
11. IF (*ratio*, *count*) for some count exists in the list for *diff_BC*
12. Increment the value of *count*
13. ELSE
14. Add (*ratio*, 1) to the list for *diff_BC*

Figure 3.19: The RPTSI-P BuildProfile algorithm



Working of the RPTSI-P method

(a)

Example profile of a store

(b)

Figure 3.20: A HashMap for the RPTSI-P's BuildProfile

Figure 3.19(a) illustrates the HashMap created by the BuildProfile algorithm for one store. Each key of the HashMap stores V_i, a distinct value for the difference between ProductB and ProductC, and a list of pairs (R_j, C_j). Each pair contains R_j, which is a

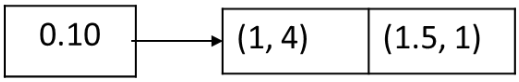
distinct value for the ratio of the differenced value of ProductA and ProductB to V_j, and C_j, which represents the number of occurrences when both V_j and R_j occur together. As an example of a profile with real values, Figure 3.19(b) shows a HashMap created by the profile building step using random data. The profile has two keys and holds all needed information in three vectors. The first pair for key 0.10 means that when the difference between ProductB and ProductC is 0.10, two occurrences have a ratio of 1. The two pairs for key 0.20 indicate that when the difference between ProductB and ProductC is 0.20, two occurrences have a ratio of 0.5, and one occurrence has a ratio of 0.75.

After a profile has been created, so as to build the relationship between products, for each key value, the ratio with highest count is selected and is used to impute values. For imputation of the missing values for ProductA, we use the values of other products to achieve a better imputation. For any missing value of ProductA, when ProductB and ProductC have values, we use those values to get better estimates.

Using the same dataset as shown in Figure 3.17, the working of the RPTSI-P method is demonstrated in Figure 3.20. It shows that all five observations have a price difference of 0.10 between ProductC and ProductB. Four have a ratio of 1 and one has a ratio of 1.5.

Date	ProductA	ProductB	ProductC	ProductB – ProductA (AB)	ProductC – ProductB (CB)	Ratio (AB:CB)
03-05-2011	3.49	3.59	3.69	0.10	0.10	1
03-07-2011	3.49	3.59	3.69	0.10	0.10	1
03-08-2011	3.49	3.59	3.69	0.10	0.10	1
03-09-2011	3.49	3.59	3.69	0.10	0.10	1
03-12-2011	3.54	3.69	3.79	0.15	0.10	1.5

(a)



(b)

Date	ProductA ($)	ProductB ($)	ProductC ($)	Imputed value
03-04-2011	3.49	?	?	
03-05-2011	3.49	3.59	3.69	
03-06-2011	?	3.59	3.69	3.49
03-07-2011	3.49	3.59	3.69	
03-08-2011	?	3.59	3.69	3.49
03-09-2011	3.49	3.59	3.69	
03-10-2011	3.59	?	?	
03-11-2011	?	3.69	3.79	3.59

(c)

Figure 3.21: The RPTSI-P method example

Figure 3.20(b) shows the profile built for this sample data. The stronger relationship is the one with a 1 to 1 ratio, as its count value is highest. During imputation, when ProductA has a missing value, and ProductB and ProductC have values such that their difference is 0.10, the imputation value is calculated by multiplying the differenced value of ProductC and ProductB with the ratio value and then subtracting it from the value of ProductB. Figure 3.20(c) shows the imputed values of ProductA calculated by

multiplying 1 with the differenced value of ProductB and ProductC, which is 0.10, in all cases.

3.2.6. The Retail Price Time Series Imputation with Products and Competitors (RPTSI-PCR) Method

The RPTSI-PCR method is a combination of the RPTSI-CR and the RPTSI-P methods, as described in Sections 3.2.4 and 3.2.5, respectively. This method works in a similar manner as RPTSI-CR. However, when an observation has ProductB and ProductC values, it calculates the possible imputation value using the RPTSI-P method rather than the RPTSI method. Once a possible imputation value is calculated, the rest of the method is executed just as it is in RPTSI-CR, i.e. it builds a profile for competitors and uses it after sorting according to range and then by count.

S1 ($)	ProductB ($)	ProdcutC ($)	Method	Possible imputation	S4 ($) (Competitor with range 2)	Difference (S1 – S4)	Within same range	Imputed value
3.04	?	?						
3.04	3.24	3.34						
?	?	?	PCL	3.00	3.02	0.02	Yes	3.00
3.00	3.10	3.20						
3.00	3.10	?						
?	3.10	3.20	RPTSI-P	**3.00**	3.00	0.00	No	3.00
?	3.13	3.23	RPTSI-P	**3.03**	3.07	0.04	No	3.07
3.06	?	3.26						
3.06	3.16	3.26						

Figure 3.22: The RPTSI-PCR method example

For the store in Figure 3.21, the profile built by the RPTSI-P method indicates that when ProductB and ProductC have a difference of 0.10, the relationship between the product values is of a 1 to 1 ratio because it has the highest count. In Figure 3.21, the first

observation with a missing value for ProductA also has a missing value for ProductB. In this case, only the RPTSI method is applied. The condition for PCL, as documented in Section 3.1, is found to be applicable. A value of 3.00 is imputed. For the second and third missing values, as ProductB and ProductC have values, the RPTSI-P method is used to impute 3.00 and 3.03 (shown in bold). Then, just as in the RPTSI-CR method, several competitor's values are compared with possible imputation values and the same conditions are applied as in RPTSI-CR to get the final imputation values, as discussed in Section 3.2.4.

4. Experimental Results

The Retail Price Time Series Imputation (RPTSI) algorithm and its variants were implemented in Java and tested on the datasets of petroleum product stores in four North American cities. City1, City2, City3, and City4 were selected by a domain expert as sample datasets that would exhibit the diverse nature of different markets. Statistics concerning missing values in the data are given in Chapter 1.

To evaluate the accuracy and imputation percentage of the algorithms, random days for which prices were known were chosen to act as missing days. Of course, there were other missing days in the datasets. Several existing methods and the proposed methods were applied to the datasets. The results are summarized in this chapter.

4.1. Evaluation Measures

To systematically evaluate the effectiveness of the algorithms, the following measures were used: (1) *mean imputed error*, (2) *mean absolute deviation*, and (3) *bracket range*. As the goal of each of the algorithms is to impute values as close as possible to the original values, the traditional statistical properties and measures of the dataset were of less concern, which ruled out the need to study the effect of the imputed values on the mean, variance, and standard deviations of the datasets.

4.1.1. Mean Imputed Error (MIE)

The first measure used was the Mean Imputed Error (MIE), which measures the mean of the error. This computes the overall error of the filled-in prices, which helps to check if the model is biased. The error is calculated according to Equation (1)

$$e_j = a_j - I_j, \qquad (1)$$

where j is an index identifying an instance, a_j is the true value of observation j, I_j is the imputed value of it, and e_j is the error in the imputed value.

The formula for MIE is written as:

$$MIE = \frac{\sum_{j=1}^{n}(e_j)}{n} \qquad (2)$$

where n is the total number of missing days. If MIE > 0, then the model tends to *under-forecast*, that is, the model imputes values lower than the actual values; if MIE < 0, then the model tends to *over-forecast*, meaning that the model imputes values higher than the actual values.

4.1.2. Mean Absolute Deviation (MAD)

Mean Absolute Deviation (MAD) is the mean of the absolute value of the differences between the actual and imputed values. MAD is calculated as:

$$MAD = \frac{\sum_{j=1}^{n}|e_j|}{n} \qquad (3)$$

where n is the total number of missing days. MAD measures the model bias and indicates the absolute size of errors. For MAD, the ideal value is 0.

As an example of how the MIE and MAD measures work, see Table 4.1. The first two columns show a small time series involving six values. In this table, a is the actual value, I is the imputed value, e is the error in one value, and $|e|$ is the absolute error value. In this example, the MIE is calculated to be 0.005, which indicates that the model slightly under-forecasts. The MAD is 0.022.

Table 4.1: MIE and MAD example

Time Unit	Actual (a)	Imputed (I)	Error (e = a - I)	Absolute Error (|e|)
1	3.23	3.20	0.03	0.03
2	3.25	3.24	0.01	0.01
3	3.25	3.27	-0.02	0.02
4	3.29	3.26	0.03	0.03
5	3.30	3.33	-0.03	0.03
6	3.35	3.34	0.01	0.01
			$\sum = 0.03$	$\sum = 0.13$
			MIE = 0.03/6 = 0.005	**MAD = 0.13/6 = 0.022**

4.1.3. Bracket Range

The Bracket Range is a measure of the degree to which an error is acceptable. Four Bracket Ranges are shown in Table 4.2, where $|e_j|$ is the absolute error of the imputed price values of instance j. According to the domain expert, any of the first three ranges is acceptable, although smaller errors are preferred.

Table 4.2: Bracket Ranges

Bracket Range (BR)	Explanation
BR = 0¢	Percentage of missing values filled with $\|e_j\| = 0$
BR ≤ 3¢	Percentage of missing values filled with $\|e_j\| \leq 0.03$
BR ≤ 5¢	Percentage of missing values filled with $\|e_j\| \leq 0.05$
BR ≤ 10¢	Percentage of missing values filled with $\|e_j\| \leq 0.10$

4.2. Evaluation of Proposed Methods

Before examining the overall results of applying these various algorithms, it is useful to understand how the RPTSI algorithm works internally and what portion of these overall results is due to each constituent method. To evaluate the constituent methods,

six stores were divided into three groups by the number of days with missing data. G1, G2, and G3 represent the three groups; their percentages of missing values are shown in Table 4.3. Only stores with 35% or less missing data have been included; otherwise, the dataset would likely have so many chunks of concentrated missing values that only the PI constituent method could be employed.

Table 4.3: Quantifying missing data in groups

Group	Missing data percentage
G1	$\leq$ 5%
G2	$\leq$ 15%
G3	$\leq$ 35%

Table 4.4: Number of days filled by constituents of the RPTSI method

Store	**Number of days**				
	Missing Days	**Filled by**			
		RPTSI	**PCL**	**CMA**	**PI**
G1-1	14	12	8	3	1
G1-2	18	15	3	9	3
G2-1	75	53	21	20	12
G2-2	87	71	29	22	20
G3-1	212	174	46	10	118
G3-2	242	193	9	5	179

The results, with the number of days filled by each technique, are summarized in Table 4.4. The first column identifies the store. The second and the third columns give the number of days with missing values for a store and the number of days that were filled by the RPTSI method, respectively. The remaining three columns are placed in the order of preference the RPTSI gives to its constituent methods while imputing. PCL is most preferred and PI is least preferred. The columns give the number of days that were

filled by each of the methods individually. The results of Table 4.4 show that each of the first two constituent methods fills some of the values, and the third one fills the rest. PI was frequently employed among the group G3 stores, namely those with up to 35% missing data. The RPTSI method fills more days than its constituent methods. These results suggest that it may indeed be possible to combine the methods presented in this research. In particular, they show that the addition of the third method helps to fill in the missing values not handled by the other two methods. If the first two methods are more accurate than PI on the values they impute, then the RPTSI method can be expected to be more accurate than PI.

4.3. Dataset Generation

For each of the four cities, separate files were generated for each store using original data, as shown in Figure 4.1 (left). These files had missing values and the data-filling methods could have been applied to them, but there was no means of knowing how close the imputed values were to the original values. To evaluate the proposed methods, a dataset with missing values and where the actual values were known was required. Thus, artificial missing values were introduced.

The dataset generation process is shown in Figure 4.1. The original dataset is shown on the left. The bold and italic rows on the right identify the rows that have been randomly selected and transformed so as to have missing values (called *artificial missing values*). The accuracy of the algorithms was measured by comparing the imputed values for these artificial values with the actual values. This preprocessing was done for all of the stores in the dataset.

In this research, two types of datasets were generated, univariate time series with only ProductA values, and multivariate time series with ProductA, ProductB, and ProductC values. A separate file was created for each store in each city. The required information, in this case only the date and the one or three product prices, was stored in .arff files with suitable headers that could be interpreted by Weka.core.Instances to read them. Files were generated in this format in an attempt to simplify the I/O operations in Java. In the original dataset, if the price of a day was missing, then that day was not recorded in the data. During the process of generating the .arff files, the days with missing price values were given a value of ?, a value that will otherwise not occur.

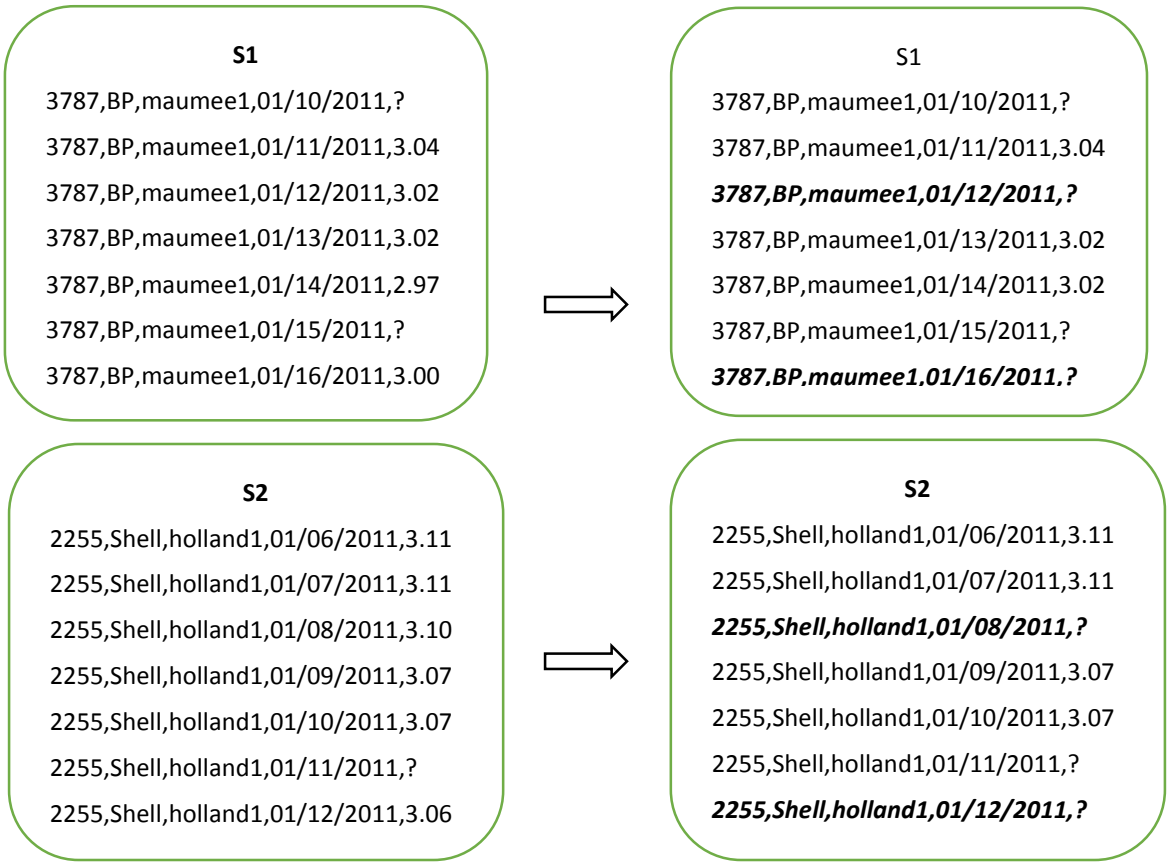


Figure 4.1: Snapshot of the original dataset and the artificial dataset

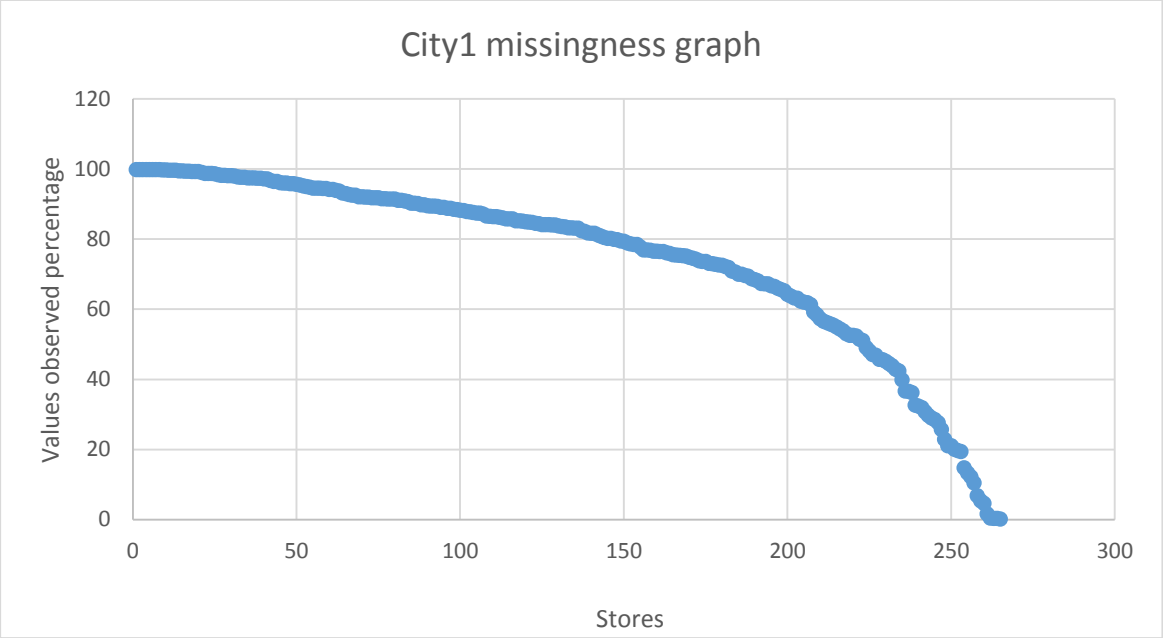


Figure 4.2: City1 missingness graph

Preliminary experiments showed that filling in values for the various stores resulted in unacceptably low performance. To select a subset of the stores for imputation, the number of missing values for each store was studied. Figure 4.2 shows a graph of the percentage of observed values of ProductA for every store in City1, with the stores sorted according to this percentage. As is apparent from the graph, there is no best number of stores to use. We somewhat arbitrarily selected the 50 stores with the highest percentages to form the sample dataset for further study. It was possible to introduce artificial missing values in this data. As well, the occurrence of long sequences of missing values was acceptably infrequent. A similar distribution of missing values was observed for the other cities. Thus, for each city, the 50 stores with the highest percentages of observed data were selected. Then, datasets with artificial missing values were constructed for each city.

Once the artificial dataset was generated, during each iteration of algorithms, the missing values, including the ones artificially introduced, were filled by the algorithm. From here, the imputed datasets were then compared with the original. As the amounts and positions of the artificial missing values were selected randomly, multiple iterations were done to determine the best and worst cases.

4.4. Experimental Results

To measure the overall effectiveness of RPTSI, RPTSI-C, and RPTSI-CR, the univariate datasets generated in Section 4.3 were used. In the case of RPTSI-P and RPTSI-PCR, the multivariate datasets were used. The results are summarized in the following tables. The measures discussed in Section 4.1, imputation percentage (percent filled), MIE, MAD, and bracket range, are also included in the tables.

4.4.1. The RPTSI and its Variants Results for City1

Table 4.5 shows that there are a total of 11,787 artificial missing days in the dataset created for City1. These missing values are distinct from the original missing days.

Table 4.5: Results for RPTSI and its variants for City1 dataset

	Days					**BR**			
Algorithm	**Missing**	**Filled**	**% Filled**	**MIE (¢)**	**MAD (¢)**	**= 0 (%)**	**≤ 3¢ (%)**	**≤ 5¢ (%)**	**≤ 10¢ (%)**
RPTSI	11,787	10,492	89.01	-0.04	5.66	31.64	49.94	64.13	80.42
RPTSI-C	11,787	11,729	99.50	-2.43	3.72	56.72	70.71	78.58	86.57
RPTSI-CR	11,787	11,787	100	-2.48	3.10	63.85	75.37	84.07	89.54
RPTSI-P	11,787	11,321	96.04	-0.07	3.15	45.91	71.18	80.18	89.21
RPTSI-PCR	11,787	11,785	99.98	-0.81	2.28	51.94	76.21	83.85	90.9

For each method in Table 4.5, the MIE value is negative, indicating that all methods tend to over-forecast. As for the MAD value, it decreased from 5.66¢ for the RPTSI method to 3.10¢ for the RPTSI-CR method, and further decreased to 2.81¢ for the RPTSI-PCR method, which indicates that the error variation is better handled by RPTSI-PCR. The percentage for bracket BR = 0 increased from 31.64% to 51.94% between the RPTSI and the RPTSI-PCR methods, which indicates that more missing days were imputed with correct values. The same increasing trend was observed for bracket BR $\leq$ 5¢ between the RPTSI and the RPTSI-PCR methods, which increased the value from 64.13% to 83.85% for the RPTSI-PCR method. Not only did the MAD value decrease and the BR = 0 range increase for the RPTSI-PCR method, but the percentage of days filled increased as well. The RPTSI method filled 89.01% of the missing days, whereas the RPTSI-PCR method filled 100%. This is evidence that the RPTSI-PCR method did not only work better in terms of the MAD value, but it also increased the imputation percentage.

Considering that the RPTSI-PCR method gave the lowest MAD values when filling in missing values, it was applied to the whole dataset for City1. After each five stores, the MAD (for all stores considered so far) value was recorded. Figure 4.3 shows a graph of the MAD for City1 when the RPTSI-PCR method is applied. The stores are sorted according to the percentage of observed values. Generally, the MAD value increases as stores with lower percentages of observed values are added. Stores close to 250 have the lowest amount of data and the highest MAD values.

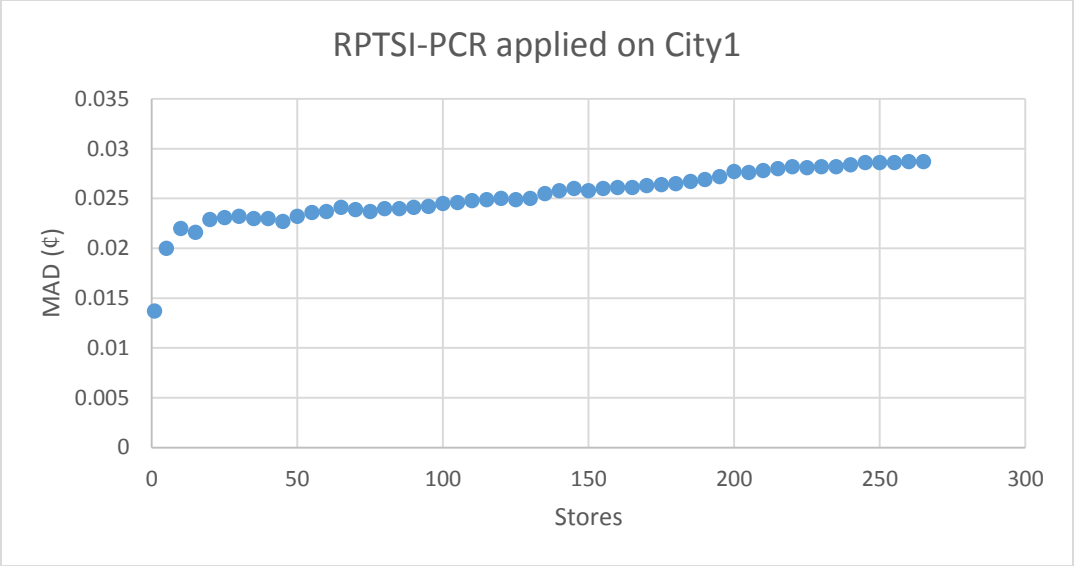


Figure 4.3: The RPTSI-PCR method applied to the whole City1

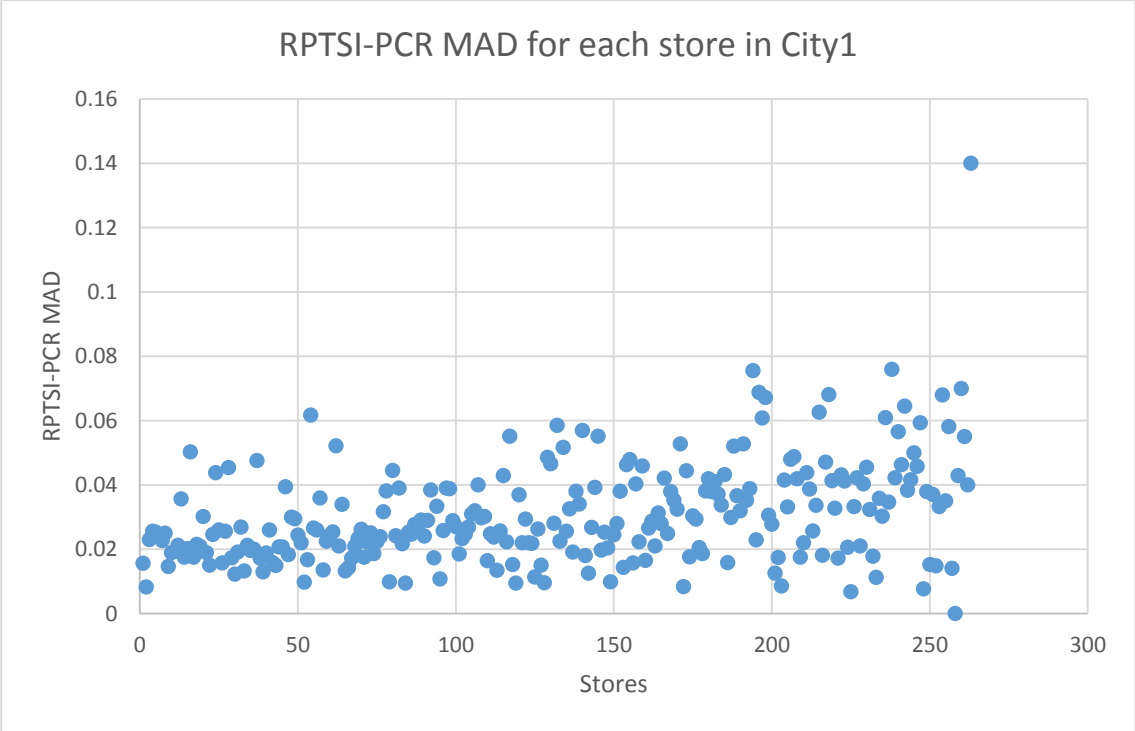


Figure 4.4: The RPTSI-PCR method on individual stores of City1

The graph in Figure 4.4 shows the results of the RPTSI-PCR method on every individual store in City1, sorted according to the percentage of observed values. The graph shows that the MAD values from the RPTSI-PCR method have higher variance, even for stores with similar percentages of observed data.

We studied the effect of the random nature of the insertions of missing values in a dataset. We generated 30 datasets of the same 50 City1 stores, all but with different values randomly selected to be missing. We applied all five methods, namely RPTSI and its variants, to these datasets. The results are shown in Table 4.6.

Table 4.6: The RPTSI and its variants for the 30 datasets

	MAD(¢)				
Datasets	**RPTSI**	**RPTSI-C**	**RPTSI-CR**	**RPTSI-P**	**RPTSI-PCR**
D1	5.54	3.69	2.85	3.29	2.38
D2	5.76	3.91	3.07	3.29	2.37
D3	5.5	3.5	2.76	3.26	2.3
D4	5.57	3.72	2.9	3.18	2.26
D5	5.65	3.75	2.95	3.29	2.38
D6	5.66	3.88	3.01	3.3	2.37
D7	5.53	3.52	2.75	3.31	2.38
D8	5.76	3.97	3.11	3.24	2.31
D9	5.62	3.76	2.97	3.4	2.49
D10	5.43	3.63	2.77	3.28	2.37
D11	5.4	3.64	2.8	3.12	2.25
D12	5.6	3.65	2.9	3.2	2.3
D13	5.5	3.66	2.85	3.28	2.43
D14	5.62	3.81	3.02	3.28	2.36
D15	5.6	3.79	2.99	3.49	2.62
D16	5.42	3.68	2.81	3.3	2.46
D17	5.36	3.49	2.71	3.35	2.32
D18	5.45	3.72	2.93	3.15	2.24
D19	5.45	3.6	2.81	3.29	2.45
D20	5.52	3.65	2.85	3.15	2.27
D21	5.66	3.66	2.82	3.4	2.4
D22	5.25	3.43	2.67	3.18	2.23
D23	5.74	3.81	3.02	3.34	2.38
D24	5.75	3.9	3.04	3.2	2.27
D25	5.43	3.63	2.89	3.2	2.38
D26	5.59	3.8	2.93	3.18	2.34
D27	5.57	3.74	3.01	3.15	2.29
D28	5.56	3.61	2.84	3.27	2.32
D29	5.41	3.57	2.77	3.44	2.49
D30	5.57	3.83	2.61	3.43	2.59

Table 4.7: Statistical measures of 30 datasets

	RPTSI	**RPTSI-C**	**RPTSI-CR**	**RPTSI-P**	**RPTSI-CRP**
Mean MAD(¢)	5.549	3.7	2.893667	3.274667	2.366667
Variance	0.01574	0.017366	0.01319	0.008909	0.00923
Standard Deviation	0.125461	0.131778	0.114846	0.094385	0.096072

The low measures of variance and standard deviation in Table 4.7 indicate that the RPTSI method and its variants behave consistently. Even though 30 datasets were tested, each having a different pattern of missing values, the results did not vary a great deal. A standard paired samples *t*-test, with a significance level of $\alpha = 0.05$, resulted in the decision to reject the null hypothesis stating that there were no significant differences between the results. As an example, the mean MAD value for the RPTSI was 3.182¢ higher than the RPTSI-PCR mean MAD value. To determine if this difference was statistically significant, we performed a *t*-test. Given that we wanted to test if the mean of one method was greater than the other, we used a one tail *t*-test. The null hypothesis (H_o) and the alternate hypothesis (H_A) defined for a test to check if the RPTSI method gave significantly higher MAD value than the RPTSI-PCR method are given as

H_o: The mean of the RPTSI method and the RPTSI-PCR method is equal.

H_A: The difference between the means is convincing enough to say that the mean of the RPTSI method is significantly higher than that of the RPTSI-PCR method.

If the calculated *t* value from *t*-test is greater than the *critical one-tail t* value, then we reject the null hypothesis and accept alternate hypothesis, H_A.

Table 4.8: Paired samples t-test between the RPSTI and the RPTSI-PCR for City1

t-Test: Two-Sample Assuming Equal Variances		
	RPTSI	**RPTSI-PCR**
Mean	5.549	2.366667
Variance	0.01574	0.00923
t Stat	116.4313	
P(T ≤ t) one-tail	1.39E-40	
t Critical one-tail	1.699127	

Table 4.8 shows the result of the paired samples *t*-test. As the calculated *t* value, 116.4313, was greater than the *critical one-tail t* value, 1.699127, we rejected the null

hypothesis and accepted the alternate hypothesis that states that the RPTSI mean is significantly higher than that of the RPTSI-PCR mean. To compare the other three methods, we applied the same methodology. The paired samples *t*-test results, critical one-tail *t* values, and the acceptance or rejection of the null hypothesis are shown in Table 4.9.

Table 4.9: Paired samples t-test on RPTSI-C, RPTSI-CR, and RPTSI-CR for City1

	t-Test values with RPTSI-PCR	**t Critical one-tail value**	**t > t Critical value**	H_o
RPTSI-C	48.92067	1.699127	Yes	Rejected
RPTSI-CR	20.87598	1.699127	Yes	Rejected
RPTSI-P	99.75578	1.699127	Yes	Rejected

Table 4.9 shows the paired samples *t*-test values for the mean of the RPTSI-PCR method with the other three methods, the RPTSI-C, RPTSI-CR, and RPTSI-P. For all three methods, the null hypothesis was rejected, indicating that there were statistically significant differences between their mean values. The result enabled us to determine that the RPTSI-PCR method had the lowest MAD value and, moreover, that it was significantly lower than the MAD values of the other methods.

4.4.2. The RPTSI Method and its Variants Results for City2, City3, and City4

The RPTSI method and its variants were also tested on datasets from three other cities, here called City2, City3, and City4, and selected by domain experts. From all three cities, the 50 stores with the highest percentage of observed values were selected; artificial missing values were also constructed. The results are shown in Tables 4.10, 4.11, and 4.12.

Table 4.10: Results for the RPTSI method and its variants for City2

	Days					BR			
Algorithm	**Missing**	**Filled**	**% Filled**	**MIE (¢)**	**MAD (¢)**	**= 0 (%)**	**≤ 3¢ (%)**	**≤ 5¢ (%)**	**≤ 10¢ (%)**
RPTSI	11,434	10,286	89.95	-0.06	1.35	62.79	82.84	91.68	97.12
RPTSI-C	11,434	10,755	94.06	-0.12	1.66	43.59	80.53	90.18	96.61
RPTSI-CR	11,434	11,434	100	-0.62	1.62	87.45	91.85	95.25	97.95
RPTSI-P	11,434	10,949	95.75	-0.57	1.67	53.21	80.58	91.39	96.67
RPTSI-PCR	11,434	11,434	100	-0.79	1.34	63.87	84.74	93.1	97.08

Table 4.11: Results for the RPTSI method and its variants for City3

	Days					BR			
Algorithm	**Missing**	**Filled**	**% Filled**	**MIE (¢)**	**MAD (¢)**	**= 0 (%)**	**≤ 3¢ (%)**	**≤ 5¢ (%)**	**≤ 10¢ (%)**
RPTSI	15,600	13,731	88.01	-0.19	4.42	29.99	58.73	71.43	84.78
RPTSI-C	15,600	15,451	99.04	-1.91	2.94	57.05	76.92	82.82	88.36
RPTSI-CR	15,600	15,600	100	-2.02	2.32	85.12	86.62	87.67	90.04
RPTSI-P	15,600	15,098	96.78	-0.04	2.38	51.22	77.88	85.51	92.71
RPTSI-PCR	15,600	15,600	100	-0.47	1.87	64.65	83.98	88.92	93.65

Table 4.12: Results for the RPTSI method and its variants for City4

	Days					BR			
Algorithm	**Missing**	**Filled**	**% Filled**	**MIE**	**MAD (¢)**	**= 0 (%)**	**≤ 3¢ (%)**	**≤ 5¢ (%)**	**≤ 10¢ (%)**
RPTSI	15,306	13,767	89.94	-0.17	1.34	67.71	81.94	90.68	97.54
RPTSI-C	15,306	14,927	97.52	-0.88	1.86	55.18	76.6	87.48	96.4
RPTSI-CR	15,306	15,306	100	-0.64	1.50	86.26	91.15	95.2	98.48
RPTSI-P	15,306	15,149	98.97	-0.74	1.32	65.05	87.37	92.57	96.56
RPTSI-PCR	15,306	15,306	100	-0.77	1.26	66.91	88.06	92.88	96.64

For all three cities, the MIE values were negative, indicating that the methods tend to over-forecast values while imputing. The rest of the results for City3 were similar to

City1 results. The MAD values decreases as one shifts from the RPTSI method to the RPTSI-PCR method, while the imputation percentages increase, leading us to select the RPTSI-PCR as the best imputation method. In terms of City2 and City4, though they had the lowest MAD values for RPTSI-PCR, they were close to the MAD values of the RPTSI method. The RPTSI-C and the RPTSI-CR methods resulted in higher MAD values than the RPTSI and the RPTSI-PCR methods. For City2 and City4, RPTSI-PCR is still considered to be the best imputation method. This is due to the fact that the bracket BR = 0 increased from 82.84% to 84.74% for City2, and from 81.94% to 88.06% for City4. The imputation percentage also increased to 100% for these cities, whereas it was around 89% for the RPTSI method.

4.5. Comparison with Existing Methods

In this section, the results of the RPTSI method and its variations are compared with several well-known techniques. For these comparisons, the Last Value Carried Forward (LVCF), Next Value Carried Backward (NVCB), Mean Imputation, Moving Average (MA), Polynomial Interpolation (PI), and Multiple Imputation methods are used. While LVCF and NVCB could be considered similar in nature, they performed differently on dataset and the result difference is shown in tables 4.13, 4.14, 4.15, and 4.16.

Section 4.5.1 compares the results for City1 for all of the methods and Section 4.5.2 compares those of City2, City3, and City4. For each city, the same dataset of 50 stores, generated as described in Section 4.3, was used in the comparison.

4.5.1. Comparison of Results for City1

In the case of City1, 11,787 days were artificially introduced as missing values. As before, the measures used were imputation percentage, MIE, MAD, and bracket ranges.

Table 4.13: Comparison of results for City1

	Days					BR			
Algorithm	**Missing**	**Filled**	**% Filled**	**MIE (¢)**	**MAD (¢)**	**= 0 (%)**	**≤ 3¢ (%)**	**≤ 5¢ (%)**	**≤ 10¢ (%)**
LVCF	11,787	11,767	99.83	0.03	6.22	31.72	47.69	60.87	77.17
NVCF	11,787	9,214	78.17	-0.19	5.48	35.3	52.11	66.24	81.18
Mean Imputation	11,787	11,783	99.96	-0.28	22.41	0	5.83	11.47	24.7
MA	11,787	11,737	99.57	0.23	9.44	3.42	18.83	31.9	60.85
PI	11,787	871	7.38	-0.30	7.72	0	29.85	45.24	72.45
Multiple Imputation*	4,824	4,824	**100**	-0.08	9.60	3.09	23.49	36.19	61.75
RPTSI	11,787	10,492	89.01	-0.04	5.66	31.64	49.94	64.13	80.42
RPTSI-C	11,787	11,729	99.50	-0.43	3.72	56.72	70.71	78.58	86.57
RPTSI-CR	11,787	11,787	**100**	-0.48	3.10	**58.85**	72.37	**84.07**	89.54
RPTSI-P	11,787	11,321	96.04	-0.07	3.15	45.91	71.18	80.18	89.21
RPTSI-PCR	11,787	**11,787**	**100**	-0.81	**2.28**	51.94	**76.21**	83.85	**90.9**

*Multiple Imputation was applied to only the top 20 stores because of memory overflow problems

The results are shown in Table 4.13 for City1. The best results in each column are highlighted in boldface. Last Value Carried Forward (LVCF) and Moving Average (MA) were the only methods with positive MIE values, suggesting that they tend to under-forecast. All other methods had negative MIE values. The LVCF method filled almost all of the missing values, and the MAD value was about 6.22¢, with BR = 0 percentage 31.72%. NVCF lowered MAD to 5.48¢, with BR = 0 percentage 35.3%. Mean Imputation performed the worst among all of the methods and generated the highest MAD value, a value of 22.41¢. Moving Average did only slightly better than Mean Imputation and Multiple Imputation. Multiple Imputation did not perform well either and

was the slowest method. Due to its slow performance, only the top 20 stores were chosen to run, due to memory overflow problems associated with larger numbers of stores. RPTSI, RPTSI-C, RPTSI-CR, RPTSI-P, and RPTSI-PCR showed trends similar to those reported above. RPTSI-CR had the highest value for BR, specifically a value of BR $\leq$ 5¢. Other than that, RPTSI-PCR outperformed RPTSI, RPTSI-C, RPTSI-CR, and RPTSI-P, as well as the existing methods in all three aspects: (1) percentage imputation, (2) MAD, and (3) percentage of correct imputations in the acceptable brackets (BR).

4.5.2. Comparison of Results for City2, City3, and City4

To evaluate the generality of the methods, we performed an evaluation on datasets of three other cities. The results for City2, City3, and City4 are presented in Tables 4.14, 4.15, and 4.16, respectively. The comparison of the methods for City2 yielded results similar to those of City1. Last Value Carry Forward had a higher percentage for BR = 0 than did RPTSI, RPTSI-C, RPTSI-CR, though the imputation percentage was less. Multiple Imputation filled in all of the missing values, but, again, with a high MAD value and very small percentage value for BR = 0.

In general, City3 and City4 both had similar results and variations to City1 and City2. City3 showed more similarity to City1 and City4 showed more similarity to City2, where LVCF worked better than RPTSI-C, but RPTSI-PCR again provided the best results. Overall, the RPTSI-PCR method performed better than any other method and resulted in the lowest MAD values among the different cities.

Table 4.14: Comparison of results for City2

	Days					BR			
Algorithm	**Missing**	**Filled**	**% Filled**	**MIE (¢)**	**MAD (¢)**	**= 0 (%)**	**≤ 3¢ (%)**	**≤ 5¢ (%)**	**≤ 10¢ (%)**
LVCF	11,434	11,418	99.86	-0.85	2.58	63.95	79.88	89	95.85
NVCF	11,434	8,565	74.90	-0.73	1.96	**68.22**	83.07	91.66	97.21
Mean Imputation	11,434	11,429	99.95	-0.55	23.57	0	7.73	15.07	29.98
MA	11,434	11,362	99.37	-0.66	3.61	24.49	66.78	82.65	94.91
PI	11,434	854	7.46	-3.27	5.29	0	74.12	86.89	96.14
Multiple Imputation*	4,808	4,808	**100**	-0.2	12.46	2.52	18.86	29.72	52.58
RPTSI	11,434	10,286	89.95	-0.06	1.35	62.79	82.84	91.68	**97.12**
RPTSI-C	11,434	10,755	94.06	-0.12	1.66	43.59	80.53	90.18	96.61
RPTSI-CR	11,434	11,434	**100**	-0.62	1.62	57.45	77.85	85.25	96.95
RPTSI-P	11,434	10,949	95.75	-0.57	1.67	53.21	80.58	91.39	96.67
RPTSI-PCR	11,434	**11,434**	**100**	-0.79	**1.34**	63.87	**84.74**	**93.1**	97.08

*Multiple Imputation was applied to only the top 20 stores because of memory overflow problems

Table 4.15: Comparison of results for City3

	Days					BR			
Algorithm	**Missing**	**Filled**	**% Filled**	**MIE (¢)**	**MAD (¢)**	**= 0 (%)**	**≤ 3¢ (%)**	**≤ 5¢ (%)**	**≤ 10¢ (%)**
LVCF	15,600	15,571	99.81	-0.17	5.08	29.91	56.34	68.52	82.36
NVCF	15,600	11,862	76.03	-0.45	4.49	34.11	61.18	73.61	85.46
Mean Imputation	15,600	15,590	99.93	-0.39	35.91	0	4.64	7.99	16.86
MA	15,600	15,520	99.48	-0.12	7.53	2.49	24.02	41.97	74.35
PI	15,600	1,216	7.79	-0.31	6.39	0	34.13	51.15	78.7
Multiple Imputation*	6,240	6,240	100	-0.15	14.15	2.34	16.33	24.68	45.71
RPTSI	15,600	13,731	88.01	-0.19	4.42	29.99	58.73	71.43	84.78
RPTSI-C	15,600	15,451	99.04	-1.91	2.94	57.05	76.92	82.82	88.36
RPTSI-CR	15,600	15,600	100	-2.02	2.32	59.12	77.62	82.67	90.04
RPTSI-P	15,600	15,098	96.78	-0.04	2.38	51.22	77.88	85.51	92.71
RPTSI-PCR	15,600	**15,600**	**100**	-0.47	**1.87**	**64.65**	**83.98**	**88.92**	**93.65**

*Multiple Imputation was applied to only the top 20 stores because of memory overflow problems

Table 4.16: Comparison of results for City4

	Days					BR			
Algorithm	**Missing**	**Filled**	**% Filled**	**MIE (¢)**	**MAD (¢)**	**= 0 (%)**	**≤ 3¢ (%)**	**≤ 5¢ (%)**	**≤ 10¢ (%)**
LVCF	15,306	15,276	99.80	-0.73	2.45	68.08	79.41	88.78	96.73
NVCF	15,306	11,538	75.38	-1.04	2.25	**71.92**	81.8	90.45	97.46
Mean Imputation	15,306	15,296	99.93	-0.97	35.92	0	5.35	9.19	19.97
MA	15,306	15,205	99.34	-0.55	3.56	32.42	67.66	82.8	94.55
PI	15,306	1,085	7.08	-0.61	2.92	0	77.51	89.68	97.51
Multiple Imputation*	6,321	6,321	**100**	-0.62	15.74	1.82	13.72	21.64	40.96
RPTSI	15,306	13,767	89.94	-0.17	1.34	67.71	81.94	90.68	97.54
RPTSI-C	15,306	14,927	97.52	-0.88	1.86	55.18	76.6	87.48	96.4
RPTSI-CR	15,306	**15,306**	**100**	-0.64	1.50	56.26	81.15	89.2	95.48
RPTSI-P	15,306	15,149	98.97	-0.74	1.32	65.05	87.37	92.57	96.56
RPTSI-PCR	15,306	**15,306**	**100**	-0.77	**1.26**	66.91	**88.06**	**92.88**	**96.64**

*Multiple Imputation was applied to only the top 20 stores because of memory overflow problems

4.6. Discussion

Although Multiple Imputation is generally held in high regard, it did not yield highly accurate results in this research problem addressed by this thesis. Wayman stated that the values imputed from the Multiple Imputation model are not merely estimates of the true values; instead, they are believed to maintain the true nature (or variability) of the dataset [30]. The generated datasets maintain the observed representation of the relationships with the other variables of a sample dataset. If Multiple Imputation is applied to our datasets, the prices often miss the mark by a few cents because the method is not intended to impute the true value. Such inaccuracies led to relatively low values for the performance measures in comparison to the methods proposed here. Multiple Imputation is best utilized by those who require a complete dataset for their analysis and are interested in preserving the characteristics (mean, variance, etc.) of the dataset for

statistical analysis. Since the goal of the current research is to impute values with the lowest possible mean error, the proposed RPTSI-PCR method is recommended.

5. Conclusions and Future Work

When filling in missing values and constructing complete datasets for forecasting retail price values for datasets generated through crowdsourcing, the results clearly indicate that the proposed method, RPTSI, performs better than the modern method, namely Multiple Imputation, and all other traditional methods. The proposed method is also simpler than Multiple Imputation because it does not require any additional or complex procedures to combine the results of multiple datasets to generate a single result, as is the case with Multiple Imputation. The modifications and additions to RPTSI, including other products and competitors in RPTSI-PCR, further reduced the error of the imputed values, giving it to the highest level of accuracy among the tested algorithms.

After achieving results accurate to within 5¢ using RPTSI and its variants, RPTSI-C, RPTSI-CR, RPTSI-P, and RPTSI-PCR, we compared their performance to other available algorithms. There was no guarantee that RPTSI would be able to fill all the missing values. It did not fill in days occurring near the very beginning or end of the dataset, or from the larger than threshold gaps between two consecutive reported prices. To permit a comparison of the accuracy of RPTSI with Multiple Imputation, the dataset had to be filled in completely, which was achieved by using competitors in RPTSI-C, RPTSI-CR, and RPTSI-PCR algorithms. Multiple Imputation techniques can also be applied to RPTSI to generate multiple complete data sets, but our focus is currently on single imputation.

In order to include competitors, a profile had to be built for each gas store. A profile holds information about the changes made in the gas store prices with respect to the changes made in another gas store. The inclusion of the competitors' data reduced the

error in imputation. The inclusion of competitors had a significant impact on the imputations, not just in terms of accuracy, but also the percentage of imputation. Using the multivariate dataset, with different product types, and by combining the competitors with the multivariate dataset, we obtained imputations even closer to the original values. With these results in mind, future research will include finding most suitable competitors. While imputing the use of information of followers and following in terms of competitors seems a sound approach to take. One piece of information in the dataset that has not yet been used is the user information, that is, all available information about the user who reported the price values. By using this information, it may be possible to reduce the dependency on competitors while imputing. The use of irregular time series may also have a considerable impact on the accuracy of data filling method. These avenues ought to be explored to see if they can increase the accuracy measures or reduce the computational load on our dataset and further to other commodity prices datasets e.g. gold, electricity, and wheat.

References

[1] U.S.GeologicalSurvey, "Chromium Statistics Lead Statistics," August 2013. [Online]. Available: http://data.is/1bmmNMD.

[2] G. E. P. Box and G. M. Jenkins, Time series analysis: forecasting and control, San Francisco: Holden-Day, 1970.

[3] P. Lévy, Collective Intelligence: Mankind's Emerging World in Cyberspace, Massachusetts: Perseus Books Cambridge, 1997.

[4] D. C. Brabham, "Crowdsourcing as a Model for Problem Solving," *Convergence: The International Journal of Research into New Media Technologies,* pp. 75-90, 2008.

[5] E. Estellés-Arolas and F. González-Ladrón-de-Guevara, "Towards an integrated crowdsourcing definition," *Journal of Information Science,* vol. 38, pp. 189-200, 2012.

[6] N. Zhang and W. Lu, "An Efficient Data Preprocessing Method for Mining Customer Survey Data," in *Industrial Informatics*, pp. 573-578, 2007.

[7] GasBuddy. [Online]. Available: http://www.toledogasprices.com/Retail_Price_Chart.aspx. [Accessed 25 September 2012].

[8] H. W. Ahmad, *Prediction of retail prices using local competitors,* Regina: University of Regina, 2014.

[9] S. Ghosh and P. Pahwa, "Assessing bias associated with missing data from joint Canada/U.S. survey of health: An application," *JSM Biometrics Section,* pp. 3394-3401, 2008.

[10] D. B. Rubin, "Inference and missing data," *Biometrika*, pp. 581-592, 1976.

[11] N. J. Horton and K. P. Kleinman, "Much Ado About Nothing: A Comparison of Missing Data Methods and Software to Fit Incomplete Data Regression Models," *American Statistical Association,* pp. Vol. 61. 79-90, 2007.

[12] A. Gelman and J. Hill, Data Analysis Using Regression and Multilevel/Hierarchical Models, Cambridge University Press, 2006.

[13] A. Farhangfar, L. A. Kurgan and W. Pedrycz, "Experimental analysis of methods for imputation of missing values in databases," in *Intelligent Computing: Theory and Applications II*, Orlando, pp. 172-182, 2004.

[14] S. Sridevi, D. S. Rajaram, C. Parthiban, S. SibiArasan and C. Swadhikar, "Imputation for the Analysis of Missing Values and Prediction of Time Series Data," in *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pp. 1158 - 1163, 2011.

[15] T. Mitsa, Temporal Data Mining, Chapman & Hall/CRC, 2010.

[16] D. Hyde, "Moving Average," 3 11 2009. [Online]. Available: http://www.dplot.com/blog/2009/11/moving-average.html.

[17] E. W. Cheney and D. R. Kincaid, Numerical Mathematics and Computing, Brooks Cole, 2007.

[18] D. B. Rubin, Multiple Imputation for Nonresponse in Surveys, Wiley-Interscience, 1987.

[19] H.-m. Li, P. Wang, L.-y. Fang and J.-w. Liu, "An algorithm based on time series similarity measurement for missing data filling," in *Control and Decision Conference (CCDC), 2012 24th Chinese*, pp. 3933 - 3935, 2012.

[20] M. Müller, Information Retrieval for Music and Motion, Springer, 2007.

[21] J. Honaker and G. King, "What to Do about Missing Values in Time-Series Cross-Section Data," *American Journal of Political Science,* vol. 54, pp. 561-581, 2010.

[22] B. AN and E. CK, "An introduction to modern missing data analyses.," *Journal of School Psychology,* pp. 5-37, 2010.

[23] J. L. Schafer and J. W. Graham, "Missing Data: our view of the state of the art," *Psychological Methods,* vol. 7, pp. 147-177, 2002.

[24] I. Jansen, N. Hens, G. Molenberghs, M. Aerts, G. Verbeke and M. Kenward, "The nature of sensitivity in monotone missing not at random models," *Computational statistics & data analysis,* vol. 50, pp. 830-858, 2006.

[25] C. K. Enders, "Missing Not at Random Models for Latent Growth Curve Analyses," *American Psychological Association,* vol. 16, pp. 1-16, 2011.

[26] C. Huber, "The Stata Blog," 18 2 2013. [Online]. Available: http://blog.stata.com/tag/longitudinal-data/. [Accessed 15 3 2014].

[27] T. D. Pigott, "A Review of Methods for Missing Data," *Educational Research and Evaluation,* vol. 7, pp. 353-383, 2001.

[28] C. Chatfield, The Analysis of Time Series An Introduction, 2003.

[29] J. D. Hamilton, Time Series Analysis, Princeton University Press, 1994.

[30] J. C. Wayman, "Multiple Imputation For Missing Data: What Is It And How Can I Use It?," in *American Educational Research Association*, Chicago, 2003.

[31] W. W. Wei, Time Series Analysis Univariate and Multivariate Methods – 2nd ed, Pearson Education, 2006.

[32] B. Mau, J. Leonard and T. I. W. Boundaries, Massive Change, London: Phaidon, 2004.

[33] J. W. Graham, P. E. Cumsille and E. Elek-Fisk, Methods for Handling Missing Data. Handbook of Psychology, 2003.

[34] P. J. Brockwell and R. A. Davis, Introduction to Time Series and Forecasting, New York: Springer, 2002.

[35] C. Chatfield, Time-Series Forecasting, Chapman and Hall/CRC, 2000.

[36] m. ucla, *Online Polynomial Interpolation calculator with graph.*

[37] D. B. Rubin, "Multiple Imputation After 18+ Years," *Journal of the American Statistical Association,* vol. 91, pp. 473-489, 1996.

[38] A. Sen and W. H. Choi, "Retail Gasoline Prices, Market Shares, and Local Competition: Evidence from Station Level Data," Waterloo, 2009.

[39] T. Yan and R. Curtin, "The Relation Between Unit Nonresponse and Item Nonresponse: A Response Continuum Perspective," *Int J Public Opin Res,* vol. 22, no. 4, pp. 535-551, 2010.

[40] D. S. Hosken, R. S. McMillan and C. T. Taylor, "Retail gasoline pricing: What do we know?," *International Journal of Industrial Organization,* vol. 26, no. 6, pp. 1425–1436, 2008.

[41] R. R. Andridge and R. J. A. Little, "A Review of Hot Deck Imputation for Survey Non-response," *International statistical review,* vol. 78, no. 1, pp. 40-64, 2010.

[42] G. Gardner, A. C. Harvey and G. D. A. Phillips, "An Algorithm for Exact Maximum Likelihood Estimation of Autoregressive-Moving Average Models by Means of Kalman Filtering," *Journal of the Royal Statistical Society. Series C (Applied Statistics),* vol. 29, no. 3, pp. 311-322, 1980.

[43] R. F. Potthoff, G. E. Tudor, K. S. Pieper and V. Hasselblad, "Can one assess whether missing data are missing at random in medical studies?," *Statistical Methods in Medical Research,* vol. 15, no. 3, pp. 15:213-234, 2006.

[44] K. Mohan, J. Pearl and J. Tian, "Graphical Models for Inference with Missing Data," in *Neural Information Processing Systems Conference*, Nevada, pp. 1277-1285, 2013.

[45] G. D. Luca and F. Peracchi, "A sample selection model for unit and item nonresponse in cross-sectional surveys," in *CEIS*, Working Paper No. 99, 2007.