K-MEANS IMPLEMENTATION

SEGMENTING CUSTOMER BASE

Nguyen Kim Anh

JCU: 13138914

Bachelor of Information Technology

James Cook University, Singapore

Singapore, 09/2016

Table of Contents

List of Figures

1. Abstract

"The k-means method is an old but popular clustering algorithm known for its observed speed and its simplicity"[1]. Popular application K-means in business is to partition clients of a wholesale distributor allows managers to discover distinct groups in their customer bases, then use this knowledge to develop targeted marketing programs.

In this report, I summarize the implementation of K-means algorithm with selection: Either Zscore or MinMax[0,1] for data normalization, Either Means of equal partitions or Actual data points as Initial centroids, Euclidean distance as Distance Measure. I found out that selecting combination of Zscore and Actual data points results in smaller Within Sum Square (WSS) compared with other 3 combinations. The report shows that Zscore works well with Actual data points in 64% cases in experiment context of testing k-cluster from (1-50)

In order to have comparison between the implementation result with the output given by Weka, I also performed k-means clustering with of similar approach (MinMax [0,1] as Normalization method, and distinctly actual data points as initial centroids, Euclidean Distance). The difference of WSS between the program's output and Weka's is average of 0.06, with standard deviation of 0.400765, max of 0.91, min of minus 0.87 when testing k within 1 to 15.

Furthermore, to evaluate which approach should be better for a given k-cluster, I did testing with 4 different combinations of Normalization method (MinMax, Zscore) and Initial centroids (Actual points, Equal partitions) with k from (1-50). It turns out that to increase chance of good WSS, k-means should perform with either 2 combinations

- MinMax[0,1] for Normalization and Means of equal partitions for Initial centroids
- Zscore for Normalization and Actual data points for Initial centroids

One thing needed to consider is that I tested K-means clustering on dataset quite small and highly skewed to the right with only 440 observations. Therefore, I include the limitation of my work and intention to improve it in discussion part.

[1] http://theory.stanford.edu/~sergei/papers/kMeans-socg.pdf

2. Introduction

2.1. Theoretical aspect of K-mean

2.1.1. What is K-mean?

The k-means method is a well-known geometric clustering algorithm based on work by Lloyd in 1982 [12]. Purpose of K-mean is to partition n observations into k clusters so that it maximizes similarities within members of the same cluster (intra-cluster), but minimizes the similarity between members belonging to different clusters (inter-cluster). Finally, each observation will be found in a cluster whose centroid is closest to it.

2.1.2. How it works?

Given a set of n data points, and wish to divided them into k-clusters.
A set of k initial cluster centers (centroids) is chosen randomly. Then each data point is assigned to the center closest to it. After that, centroid of each and every cluster is re-computed. Each data point is now re-calculated the distance between it and new centroids, and then that point can be assigned to the nearest center. The process of re-computation of new centroids, assigning elements for each cluster will be stop until there is no change in centers of all k-groups

2.1.3. Advantage/Disadvantage[2]

Advantages: First, K-means is relatively efficient, could achieve linear time when the data size is big. Second, it is easy to understand and implement
Disadvantages: First, it requires to specify k, the number of clusters in advance, which is sometimes hard to predict especially under external constraints. Second, quality of result is sensitive to noise points and outliers as they directly affect mean value. Third, randomly select initial centroids might result in no point assigned to certain clusters. Forth, K-means is only applicable for numerical data, not for categorical data

2.1.4. Time and space complexity

“For a data-set with m objects, each with n attributes, running l iterations with k cluster, the k-means clustering algorithm has:
Time complexity: O(l*k*m*n). For large data-sets where k << m & n << m, the complexity is approximately O(m)
Space-Complexity: The algorithm only needs to store the data points and centroids, so the complexity is O ((m+ k) * n).”[3]

2.2. Report Overview

2.2.1. Purpose of report

The report concerns about 3 aspects:

[2] http://www.cs.princeton.edu/courses/archive/spr08/cos435/Class_notes/clustering2_toPost.pdf

[3] https://www.researchgate.net/post/What_is_the_time_complexity_of_clustering_algorithms

First, the implementation of k-means using Python programming language to segment wholesales customer dataset, by performing 4 different approaches, then selecting the better with the lowest Within Sum Square Error. In the program, Euclidean distance is used as Distance Measure.
Second, the analysis the clustering results from 4 different approaches. The result also refers to Output from Weka as source of comparison references.
Third, analysis on result of clustering in terms of business values.

2.2.2. Why is K-means chosen?

By selecting dataset about clients of a wholesale distribution from UCI, I would like find out the insight in which customers are potential, giving high profit, and likely to buy which products, and relationship between products bought in customers' baskets. The high quality of partitioning customer base assists managers to develop appropriate marketing plan to target interested groups. Therefore, the business makes use of resources, high chance of success and increase revenue and profit.
Main attraction of K-mean to accomplish this task is its simplicity and intuitive algorithm, "its observe speed[4]". It will be beneficial if selecting right number of clusters and distance calculation which reflect true nature or dissimilar of data set

2.2.3. Implementation results

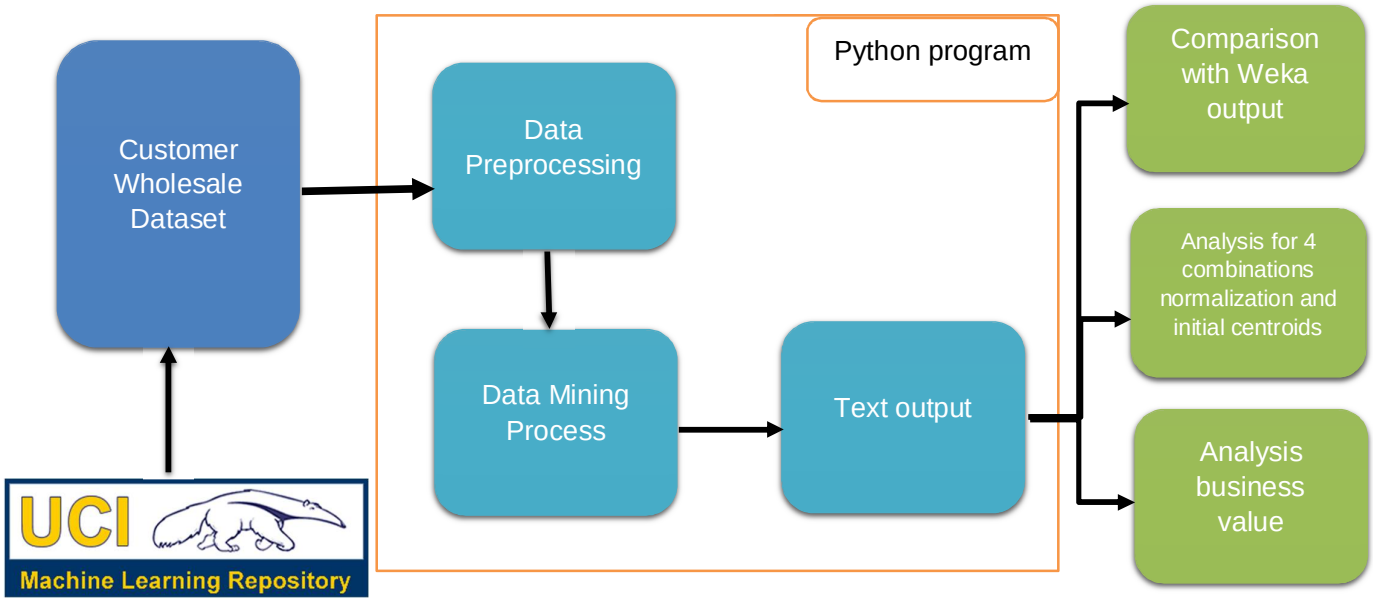
First, working software which is done by running main.py, produces output with 1 of 4 approaches at a time
Second, eval_enhance_kmeans.py, the module to generates the report when performing k-means clustering with different selection of Normalization (either Zscore or MinMax), Initial centroids (either randomly actual data points or means of equal partitions), and Euclidean distance used by default. The program automatically produces results (k, number of iteration, running time, WSS, BSS, ESS, %WSS/ESS) from k=1 to given number of k-cluster (inclusive).
Third, analysis the business value derived from clustering output with spreadsheet and chart.

[4] http://theory.stanford.edu/~sergei/papers/kMeans-socg.pdf

3. Implementation summary

3.1. Data mining flowchart

FIGURE 1: DATA MINING FLOWCHART



The chart above briefly describes workflow of data mining workflow used in this project. It includes 3 main parts: Dataset selection, Data processing, and Outcome Analysis as follows:

Dataset Selection: Data was retrieved from Machine Learning Repository (UCI) with a dataset about clients of a wholesale distributor without missing values

Data Processing consists of 3 sub-tasks:

- **Preprocess,** deals with cleaning up data: remove unwanted attributes, and normalizing data, ready for data mining process. When data is already clean and valid, then it is ready to move to second phase.
- **Data mining process**, applies K-means algorithm to partition observations. Each cluster has its own cluster center, called centroid which is used as representative for the cluster. The purpose is to separate observations into k clusters or groups which is predefined by user. An observation assigned to a certain cluster is decided by its closest distance among the observation to all k centroids.
- **Text Output** is to generate 2 csv files, one for summary clustering output, another for detail instances of each cluster. All tasks in this part is taken care by a program written in Python.

Outcome Analysis mentions about 3 perspectives:

- Comparison between the results of the program with output given by Weka in the same approach.
- Output analysis with 4 different combinations of option Normalization and Initial centroids to select optimal choice for different dataset
- Analysis on business value given by the program output with 5 clusters

3.2. Dataset and Programming Language

3.2.1. Dataset

Dataset is found on Machine Learning Repository (UCI) under Business Category, with name as Wholesale customers Data Set. It is given by Margarida G. M. S. Cardoso, margarida.cardoso '@' iscte.pt, ISCTE-IUL, Lisbon, Portugal. The data set refers to clients of a wholesale distributor. It includes the annual spending in monetary units (m.u.) on diverse product categories. Dataset has 440 instances with 8 attributes, no missing values. Attributes information is given below:

#	Attribute name	Description	Type
1	FRESH	Annual spending (m.u.) on fresh products	Continuous
2	MILK	Annual spending (m.u) on milk products	Continuous
3	GROCERY	Annual spending (m.u) on grocery products	Continuous
4	FROZEN	Annual spending (m.u) on frozen products	Continuous
5	DETERGENTS_PAPER	Annual spending (m.u) on detergents and paper products	Continuous
6	DELICATESSEN	Annual spending (m.u) on delicatessen products	Continuous
7	CHANNEL	Customer Channel - Horeca (Hotel/Restaurant/Café) or Retail Channel	Nominal
8	REGION	Customer Region in Lisnon, Oporto or Other	Nominal

Source: https://archive.ics.uci.edu/ml/datasets/Wholesale+customers

Some statistics provided as well

Attributes	Minimum	Maximum	Mean	Std. Deviation
FRESH	3	112,151	12,000.30	12,647.329
MILK	55	73,498	5,796.27	7,380.377
GROCERY	3	92,780	7,951.28	9,503.163
FROZEN	25	60,869	3,071.93	4,854.673
DETERGENTS_PAPER	3	40,827	2,881.49	4,767.854
DELICATESSEN	3	47,943	1,524.87	2,820.106

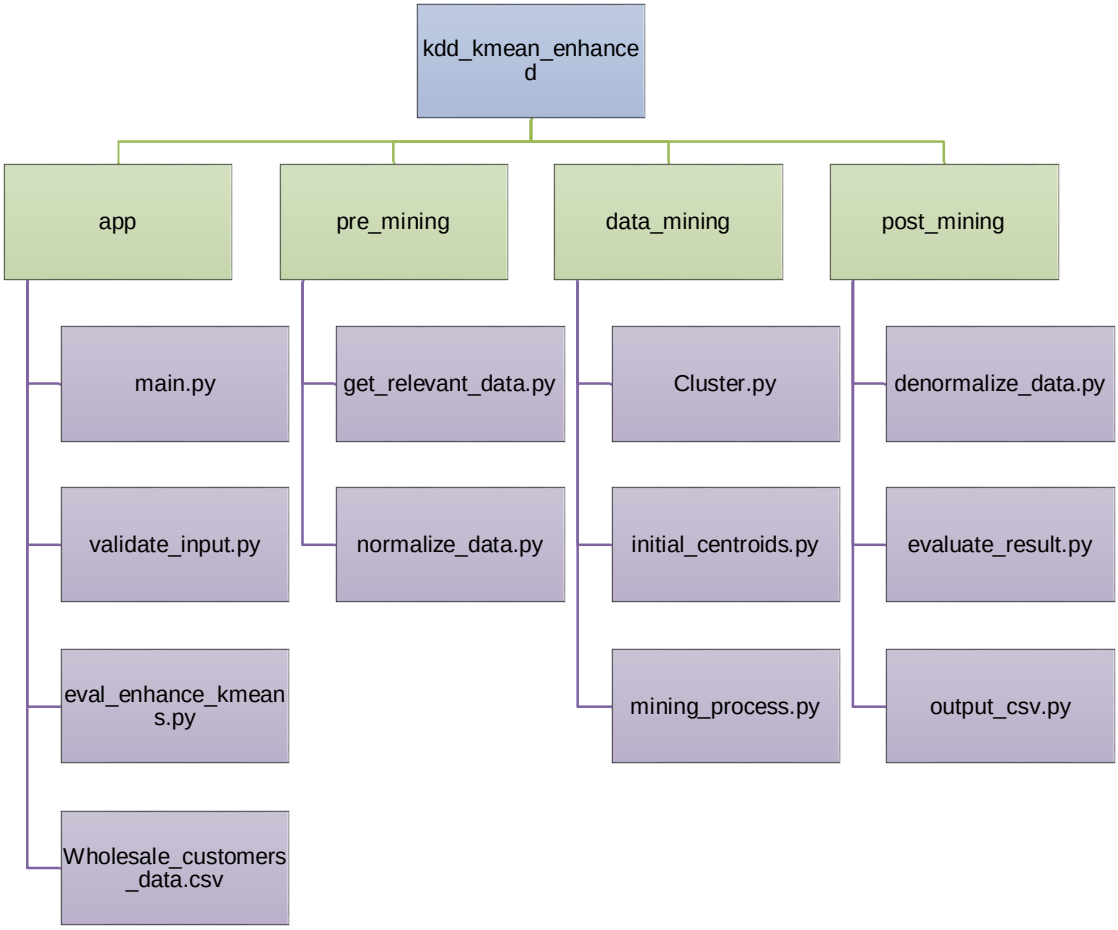
3.2.2. Python programming language

“Python was created by Guido Van Rossem in 1991 and emphasizes productivity and code readability. It is a great tool for programmers that want to do data analysis or apply statistical techniques. Python is very useful when the data analysis tasks need to be integrated with web apps or if statistics code needs to be incorporated into a production database.
Being a fully-fledged programming language, it’s a great tool to implement algorithms for production use. Python has a great deal of package to support programmers to get their jobs done such as NumPy /SciPy (scientific computing) and pandas (data manipulation) which is usable for data analysis. Also have a look at matplotlib to make graphics, and scikit-learn for machine learning[5].”
In this project, all processing part including Data cleaning, Mining, and Generate text output is accomplished on Python 3.4 through IDE PyCharm 2016.2.2. It really helps reduce development time significantly.

[5] http://www.kdnuggets.com/2015/05/r-vs-python-data-science.html

3.3. Program structure

FIGURE 2: FILE ORGANIZATION IN THE PROGRAM



The program, package kdd_kmeans_enhanced, is organized into 03 main modules and 01 app for controlling program flow.

Pre_mining: to perform cleaning tasks, make data ready for mining process, including:

- Get_relevant_data.py: read data from Wholesale_customers_data.csv file, remove 2 unnecessary fields (Channel, Region)
- Normalize_data.py: perform data normalization either using Zscore or MinMax[0,1] as user input

Data_mining: to implement k-means clustering algorithm, it consists of 3 separate files

- Cluster.py, takes care of attributes and methods working on a cluster: its centroid, its members, and related methods.
- Initial_centroids.py, consists of different interrelated functions working together to produce each starting midpoint for each cluster in k-cluster. There are 2 options for initial centroids: either means of equal partitions or actual data points
- mining_process.py, the meat of program, performs k-means clustering: Initializing centroids, assigning data points to clusters, updating centroids for each cluster. It loops until either centroids of all k-clusters unchanged or reach the cut-off value (currently default as 100).

Post_mining to generate output of program in form of 2 csv files

- denormalize_data.py converts output received from data_mining module into original form of data.
- Evaluate_result.py calculates Sum of Square Error (Within, Between, Total) to evaluate quality of clustering implementation

- Output_csv.py generates 2 csv files.
    - Summary file contains initial centroids, final centroids, number of elements within each cluster, number of iteration taken, running time, sum of square error. Its format is: {K}clusters_summary_{Norm}_{Init}.csv
    - Detail file contains observations assigned for each cluster. Its format is: {K}clusters_detail_{Norm}_{Init}.csv
    - Note: {K}- Number of clusters, {Norm}-Normalization method, {Init}-Initial centroids type

App to control how program runs by collaborating 3 other remaining modules to accomplish the clustering task. It accommodates 4 files

- Validate_input.py performs validation of user input including:
    - Number of cluster: must be integer value within 1 to number of observations inclusive
    - Normalization method: must be either Z for Zscore or M for MinMax[0,1]
    - Initial centroid method: must be either P for Means of equal partitions or A for Actual data points
- Wholesale_customers_data.csv: dataset including 8 attributes, but only 6 attributes are used for data mining process
- main.py control the flow of whole program (Refer to Figure 3 below)
- eval_enhance_kmeans.py performs clustering with k from 1 to_num_cluster inclusive, given combination of user input (Normalization method, Initial centroids). It provides the output which can be used to evaluate which combination should be picked to achieve lowest WSS by given k

3.4. Workflow of program and Clustering algorithm (Figure 3)

3.4.1. Workflow of program

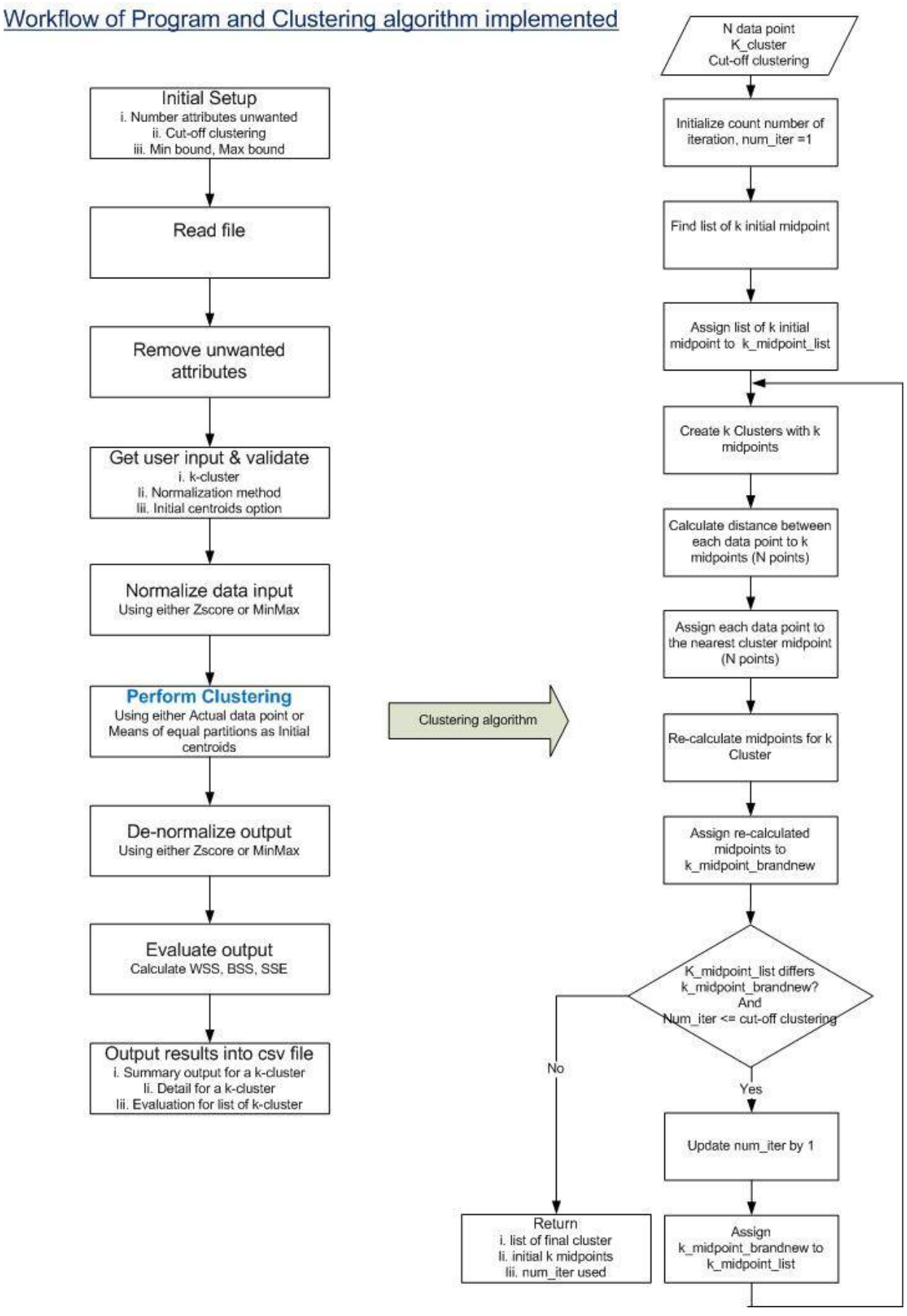
The program runs through all these steps below:

- Initial setup with default parameters
- Read csv file into program and remove 02 unnecessary fields
- Get user input and validate them
- Perform normalization
- Perform Clustering
- Perform de-normalization
- Evaluate output by calculating SSE, WSS, BSS
- Output results into csv file

3.4.2. Clustering algorithm

The algorithm is implemented with 3 stages: i) Initialize centroids; ii) Assign points for clusters; iii) Update or Re-calculate centroids for each cluster. Figure 3 explains how they collaborate

FIGURE 3: PROGRAM WORKFLOW AND CLUSTERING ALGORITHM



3.5. Implementation details

Detail of implementation in Python codes can be found in the source codes accompanied with this report. Below is to introduce the main steps in the program

3.5.1. Initial setup (Refer to main.py)

```
# 1. INITIAL SETUPS WITH DEFAULT PARAMETERS
num_attributes_unwanted = 2
cut_off_clustering = 100
min_bound = 0
max_bound = 1
normalization_dict = {"Z":"Zscore","M":"MinMax"}
inital_centroids_dict = {"P": "Partition", "A": "Actual"}
```

3.5.2. Remove unwanted fields (Refer to get_relevant_data.py)

```
# 2. READ CSV FILE INTO PROGRAM AND REMOVE UNWANTED ATTRIBUTES
file_object = open("Wholesale_customers_data.csv", "r")

# Read point into all_point_list, Remove unwanted attributes from file input
# Note: Explicitly pass all_point_list to functions (pass object ref)
list_points_interested_attr = []
list_points_interested_attr=\
   get_relevant_data.remove_unwanted_attributes(list_points_interested_attr, file_object, num_attributes_unwanted)
num_records = len(list_points_interested_attr)
```

3.5.3. Get user input and validate (Refer to validate_input.py)

```
# 3. GET USER 2 INPUTS AND VALIDATE THEM (Number of cluster, Method for Normalization, Initial centroids)
# Number of cluster must be integer within 1 to num_records
message= "Enter number of cluster: "
k_cluster = validate_input.validate_int_in_range_inc(1, num_records, message)
print("You chose {} clusters".format(k_cluster))

# Method for normalization: MinMax(0,1) or Zscore
valid_input_method_bool = False
while not valid_input_method_bool:
    input_str = input("Normalization method(Z for Zscore, M for MinMax[0,1]): ").upper()
    if input_str == "Z":
        norm_method = normalization_dict.get(input_str)
        valid_input_method_bool = True
    elif input_str == "M":
        norm_method = normalization_dict.get(input_str)
        valid_input_method_bool = True
    else:
        print("Bad input, it must be Z or M")
print("You chose method {} for Normalization".format(norm_method))
```

3.5.4. Normalization (Refer to normalize_data.py)

```
#4. NORMALIZATION: Z for Zscore, M: MinMax[min_bound, max_bound]
#list_norm = []
if norm_method == "Zscore":
    list_norm, mean_list, std_list = normalize_data.normalize_by_zscore(list_points_interested_attr)
elif norm_method == "MinMax":
    list_norm, min_list, max_list = \
        normalize_data.normalize_by_lower_upper_bound(list_points_interested_attr, min_bound, max_bound)
```

3.5.5. Clustering process (Refer to mining_process.py)

```
#============================================START K-MEAN CLUSTERING=========================================================
# Input: cluster_list, k_cluster, list point normalized
# Output: cluster_list, each cluster includes midpoint, points belonging to that cluster
print("\nStart clustering.............")

cluster_list_norm = []
start = time.time()
if norm_method == "Zscore":
    if option_initial_centroids == "Actual":
        print("User delibrately Clustering: Norm.Zscore, Initial centroids:Actual")
        cluster_list_norm, initial_k_midpoint_normalized, num_iter = \
            mining_process.clustering(cluster_list_norm, list_norm, k_cluster, cut_off_clustering, "Actual")
    elif option_initial_centroids == "Partition":
        try:
            print("Clustering: Norm. Zscore, Initial centroids: Partition")
            cluster_list_norm, initial_k_midpoint_normalized, num_iter =\
            mining_process.clustering(cluster_list_norm, list_norm, k_cluster, cut_off_clustering)
        except Exception:
            print("Exception occurs, Change: Norm. Zscore, Initial centroids: Partition =========>Actual")
            cluster_list_norm, initial_k_midpoint_normalized, num_iter = \
                mining_process.clustering(cluster_list_norm, list_norm, k_cluster, cut_off_clustering,"Actual")
        #!!!!!!!!!!!!!!!!Attention: To handle error occurs, can't assign values to clusters with Partion => Swtich to Actual

elif norm_method == "MinMax":
    if option_initial_centroids == "Actual":
        print("User delibrately: Norm. MinMax, Initial centroids: Actual")
        cluster_list_norm, initial_k_midpoint_normalized, num_iter = \
            mining_process.clustering(cluster_list_norm, list_norm, k_cluster, cut_off_clustering, "Actual")
    elif option_initial_centroids == "Partition":
```

3.5.6. Initialization of starting centroids (Refer to initial_centroids.py)

- Starting centroids are chosen from ***Distinct*** actual data points

```
# ====================INITIAL MIDPOINTS AS DISTINCT ACTUAL DATA POINTS=======================================
# OP2: Select starting midpoints as randomly distinct data points
def find_initial_centroids_in_actual_points(all_k_midpoint_list, all_point_list, k_cluster):
    print("Initial centroids: Actual points")
    for i in range(k_cluster):
        rand_index = random.randint(0,k_cluster-1)
        rand_point = all_point_list[rand_index]

        while rand_point in all_k_midpoint_list: # Check for distinction
            rand_index = random.randint(0, k_cluster-1)
            rand_point = all_point_list[rand_index]
        all_k_midpoint_list.append(rand_point)

    return all_k_midpoint_list
```

- Starting centroids are chosen from means of equal partitions

The idea is that all data points will be allocated into k equal partitions (***if possible)***, then each partition's midpoint will be calculated as a mean of those data points which are assigned to this partition. It could increase the chance of picking the closest of midpoints.

How to do:

Let say having n observation (ob1, ob2, … obn), each observation has m attributes (att1, att2, …, attm).
For each attribute,

Extract list of n values of each attributes

Find lower_bound, upper_bound of each partition

Assign right values to each partition

If no values assigned to one partition -> notify program to switch to Actual data points. Reason is sometimes partition so narrow that it cannot hold any data point.

Find midpoint for each partition

```
# ====================FIND MEAN FOR EACH FEATURE ONLY=========================================================
"""
    Purpose: Find midpoint of each partition.
           Divide the range_value into num_partion, number of data value in each partition equally (using percentile)
    Input: a range_values_of_a_feature, and num_partition
    Output: list_scrambled_midpoints_for_each_feature
    How to do:
    1. Find lower and upper bound of each partition
    2. Assign points belong to each partition
    3. Find midpoint of a partition
    => Use a class Partition to keep (lower bound, upper bound, list_partitions_points)
"""

def find_initial_midpoints_for_each_feature(range_values_of_a_feature, num_partition):

    list_scrambled_midpoints_for_a_feature = []

    #1.Find lower_bound, upper_bound of each partition
    list_partitions = []
    np_range_values = np.array(range_values_of_a_feature)  # convert into numpy array to get percentile

    for i in range(num_partition):
        #print("i=", i)
        lower_bound = np.percentile(np_range_values, round(100 / num_partition * i, 2))
        upper_bound = np.percentile(np_range_values, round(100 / num_partition * (i + 1), 2))
```

```
#==================================FUNCTION: Find all dimensions for k midpoint=====================================
def find_all_k_midpoint_list(all_k_midpoint_list, all_point_list, k_cluster):
    print("Inital midpoints: Equal Partitions")

    # Grab entire column
    scipy_array = scipy.array(all_point_list) #to extract values from 2 dimenstional array vertically

    print(all_point_list)

    # Add on 04/09/2016: If no values assigned to partition.
    # Let switch to randomly choose distinct datapoint to become initial midpoint
    not_empty = True
    while not_empty:
        list_initial_midpoints_combined = []
        for feature_index in range(len(all_point_list[0])):
```

3.5.7. Assignment of data points to clusters and re-calculation of new midpoints

(Refer to mining_process.py)

main.py × | eval_enhance_kmeans.py × | eval_kmeans_k_1_10clusters_MinMax_Actual.csv × | mining_process.py × | initial_centroids.py ×

```
63
64
65      def recalculate_all_k_midpoint_list(cluster_list):
66          all_k_midpoint_list_brandnew_temp = []
67
68          for cluster in cluster_list:
69              new_midpoint = cluster.recalculate_midpoint()
70              all_k_midpoint_list_brandnew_temp.append(new_midpoint)
71
72          return all_k_midpoint_list_brandnew_temp
73
74      def is_all_cluster_midpoint_changed(all_k_midpoint_list, all_k_midpoint_list_brandnew):
75          return not (all_k_midpoint_list == all_k_midpoint_list_brandnew) #when 2 values different: not false -> true
76
77
78      def clustering(cluster_list, all_point_list, k_cluster, value_cut_off_clustering, option_initial_centroids="Partition"):
79          num_iter = 1
80          print("{} round - Starting clustering----------".format(num_iter))
81
82          all_k_midpoint_list = []
83
84          # By default: starting midpoints using equal partitions (Percentile).
85          # If error occurs: switch to randomly distinct actual points
86          if(option_initial_centroids =="Partition"): #06/09/2016: MIS---SPELLING
```

3.5.8. De-normalization output of clustering process

```
#=========================================DENORMILIZATION OUTPUT=================================================
# Denormalize output to orginal values before writing result to_file
#!!! ATTENTION: Return a list of list, no longer a list of Clusters object, as using scipy

if norm_method == "Zscore":
    cluster_list_denorm = denormalize_data.denormalize_clusters_zscores(cluster_list_norm, mean_list, std_list)

    initial_k_midpoint_denormalized = \
        denormalize_data.demormalize_list_by_zscores(initial_k_midpoint_normalized, mean_list, std_list)

elif norm_method == "MinMax":
    cluster_list_denorm =\
        denormalize_data.denormalize_clusters_MinMax(cluster_list_norm, min_bound, max_bound, min_list, max_list)

    initial_k_midpoint_denormalized = \
    denormalize_data.demormalize_list_by_MinMax(initial_k_midpoint_normalized, min_bound, max_bound, min_list, max_list)
```

3.5.9. Evaluate Output

```
#=========================================EVALUTATE OUTPUT=================================================
# EVALUATE OUTPUT by Sum Square Error (SSE) = Within SSE (WSSE) + Between SSE (BSSE)

wsse = evaluate_result.calculate_within_sum_square_error(cluster_list_norm)
print("Within SSE = {}".format(wsse))

bsse = evaluate_result.calculate_between_sum_square_error(cluster_list_norm, list_norm)
print("Between SSE = {}".format(bsse))

total_sse = evaluate_result.calculate_total_sum_square_error(cluster_list_norm, list_norm)
print("Total SSE = {}".format(total_sse))
```

3.5.10. Output result to csv file

```
#=========================================OUTPUT RESULTS TO CSV FILE=============================================
# Write detail points in each cluster
output_csv.write_detail_result_to_file(cluster_list_denorm, norm_method, option_initial_centroids)

num_observations =len(list_norm)

# Write summary of output including sse
output_csv.write_summary_result_to_file(cluster_list_denorm, num_observations, \
                                        initial_k_midpoint_denormalized, num_iter, \
                                        elapsed_time, wsse, bsse, norm_method, option_initial_centroids)
#=========================================END OF OUTPUT RESULTS TO CSV FILE=======================================
```

3.6. How to use the program

Program can work in 2 modes:

i. Clustering with 01 k-cluster value, accompanied with selection of Normalization method (MinMax, Zscore), initial centroids (Actual data points, Means of equal Partitions)

ii. Clustering with a range of k-cluster values from 1 to n inclusive, n: number of observations

3.6.1. Clustering with 01 k-cluster value

- Running main.py
- *Get user inputs in console window and perform input validation task*

```
C:\Python34\python.exe D:/PythonPyCharm/practice_computing/kdd_kmean_enhanced/app/main.py
Enter number of cluster: 0
You must select an integer in range (1:440) inclusive
Enter number of cluster: a
Bad input, it must be an integer
Enter number of cluster: 3
You chose 3 clusters
Normalization method(Z for Zscore, M for MinMax[0,1]): zsc
Bad input, it must be Z or M
Normalization method(Z for Zscore, M for MinMax[0,1]): z
You chose method Zscore for Normalization
Input option initial centroids(P for Partition, A for Actual: pae
Bad input, it must be P or A
Input option initial centroids(P for Partition, A for Actual: p
You chose Partition for Initial Centroids

Start clustering.............
Clustering: Norm. Zscore, Initial centroids: Partition
1 round - Starting clustering----------
Inital midpoints: Equal Partitions
[[0.05293318980938627, 0.5235677732188025, -0.041114893436448675, -0.5893671557751019, -0.04356873190407057, -0.06
```

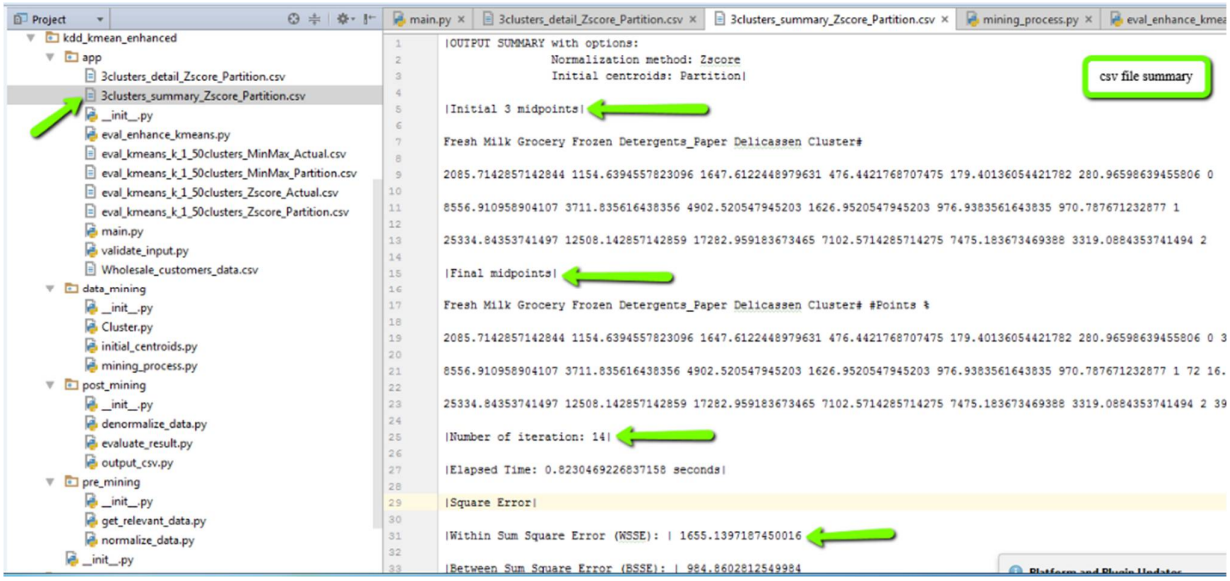
- *Observe output either in console window or in csv file*

```
12 round - Finish clustering-----------
13 round - Start clustering-----------
13 round - Finish clustering-----------
14 round - Start clustering-----------
14 round - Finish clustering-----------
Elapsed time: 0.8230469226837158 seconds
Number of iteration: 14

Finish clustering.............
Within SSE = 1655.1397187450016
Between SSE = 984.8602812549984
Total SSE = 2640.0
Start writing detail of points in clusters to file
Successful writting details to a file
Start writing a summary of clustering result
Successful writting a summary of clustering result in file

Process finished with exit code 0
```

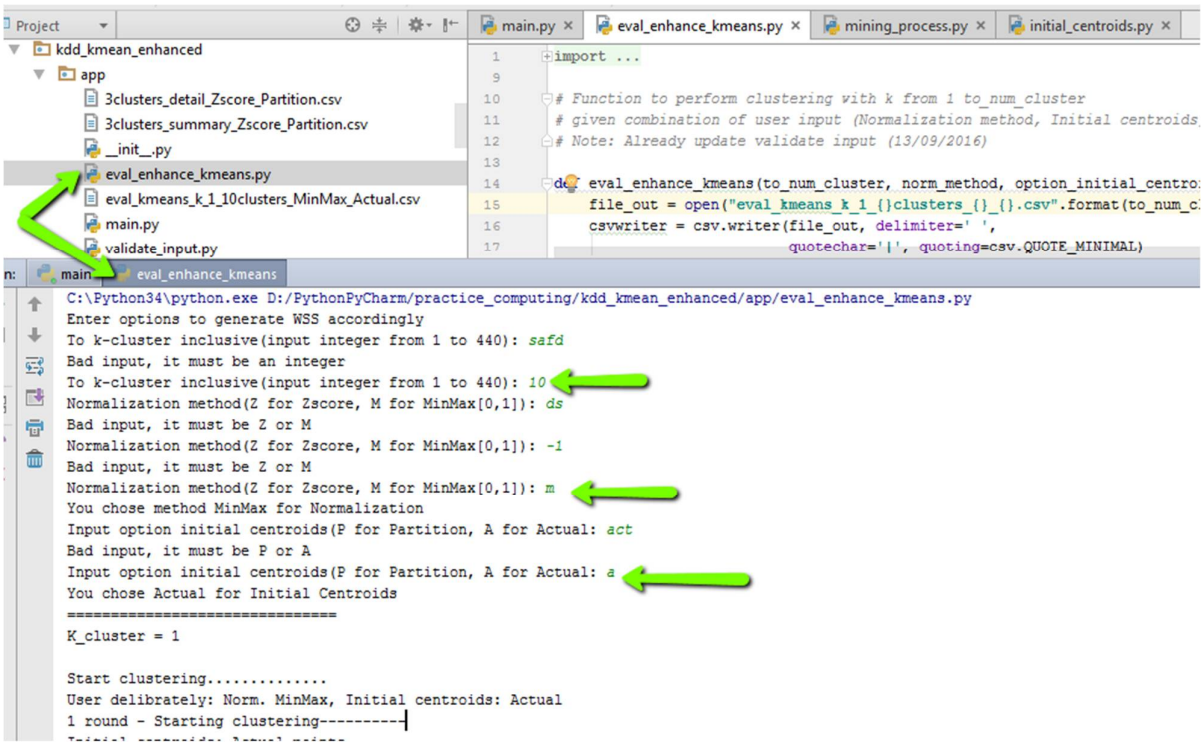
In console windown



3.6.2. Clustering with a range of k-cluster values from 1 to n inclusive

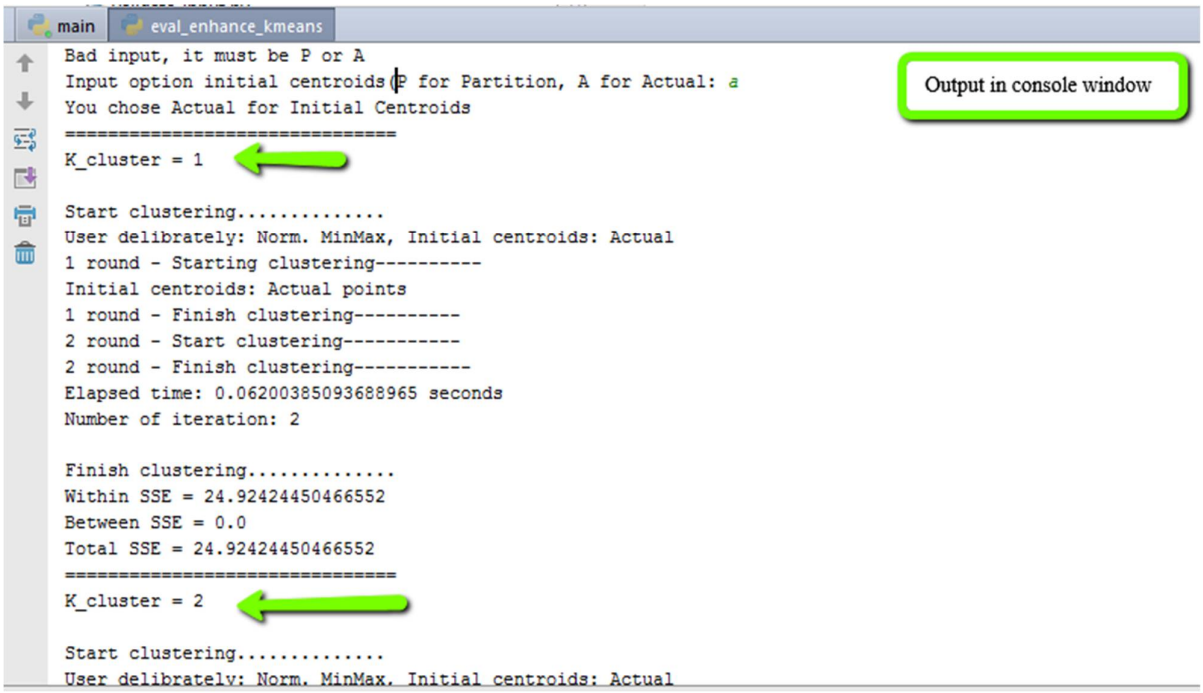
This option should be useful when user wants to compare among 04 combination approaches with the same range of k-cluster values

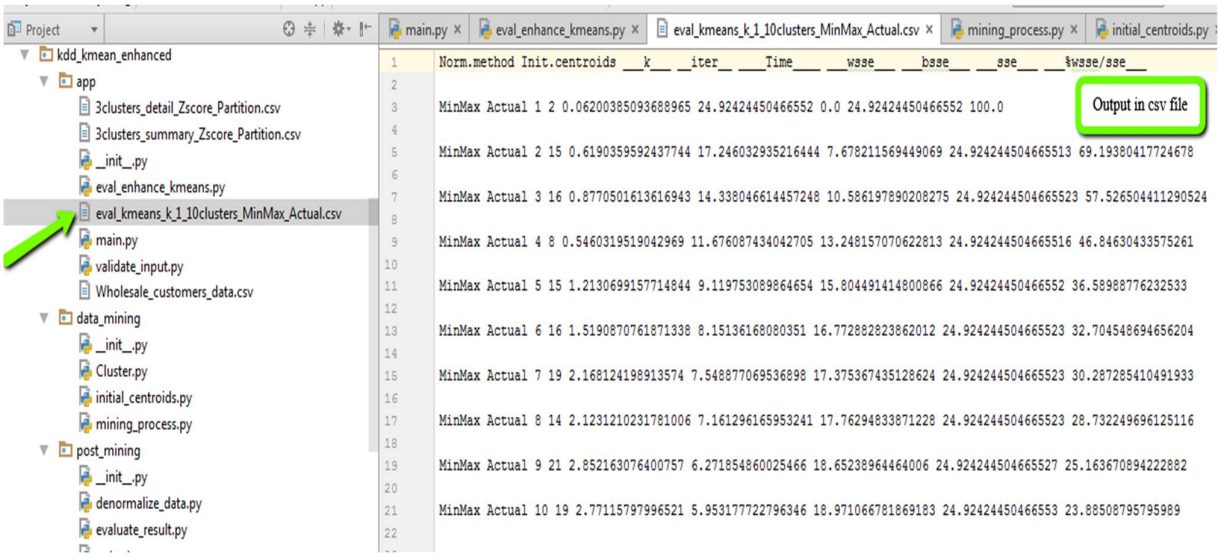
- Running eval_enhance_kmeans.py
- Get user input



- Generate output result either in console or in csv file

Note that: File output is named according to the input options from user





4. Comparison with Weka's output

4.1. Metrics to evaluate program quality using (SSE)

Clustering Validation: Cohesion (how closely related objects are) and Separation (how distinct a cluster from others). The formulas are used to calculate as below [6]

Example: Squared Error

- Cohesion is measured by the within cluster sum of squares (SSE)

$$WSS = \sum_{i} \sum_{x \in C_i} (x - m_i)^2$$

- Separation is measured by the between cluster sum of squares

$$BSS = \sum |C_i| (m - m_i)^2$$

  - Where $|C_i|$ is the size of cluster i

[6] http://www.cs.kent.edu/~jin/DM08/ClusterValidation.pdf

4.2. Output using Weka

In order to obtain the result from Weka application, following steps were done for testing purpose with the same procedures, with k from 1 to 15 and k = 440

FIGURE 4: PERFORM SIMPLEKMEANS ON WEKA APPLICATION

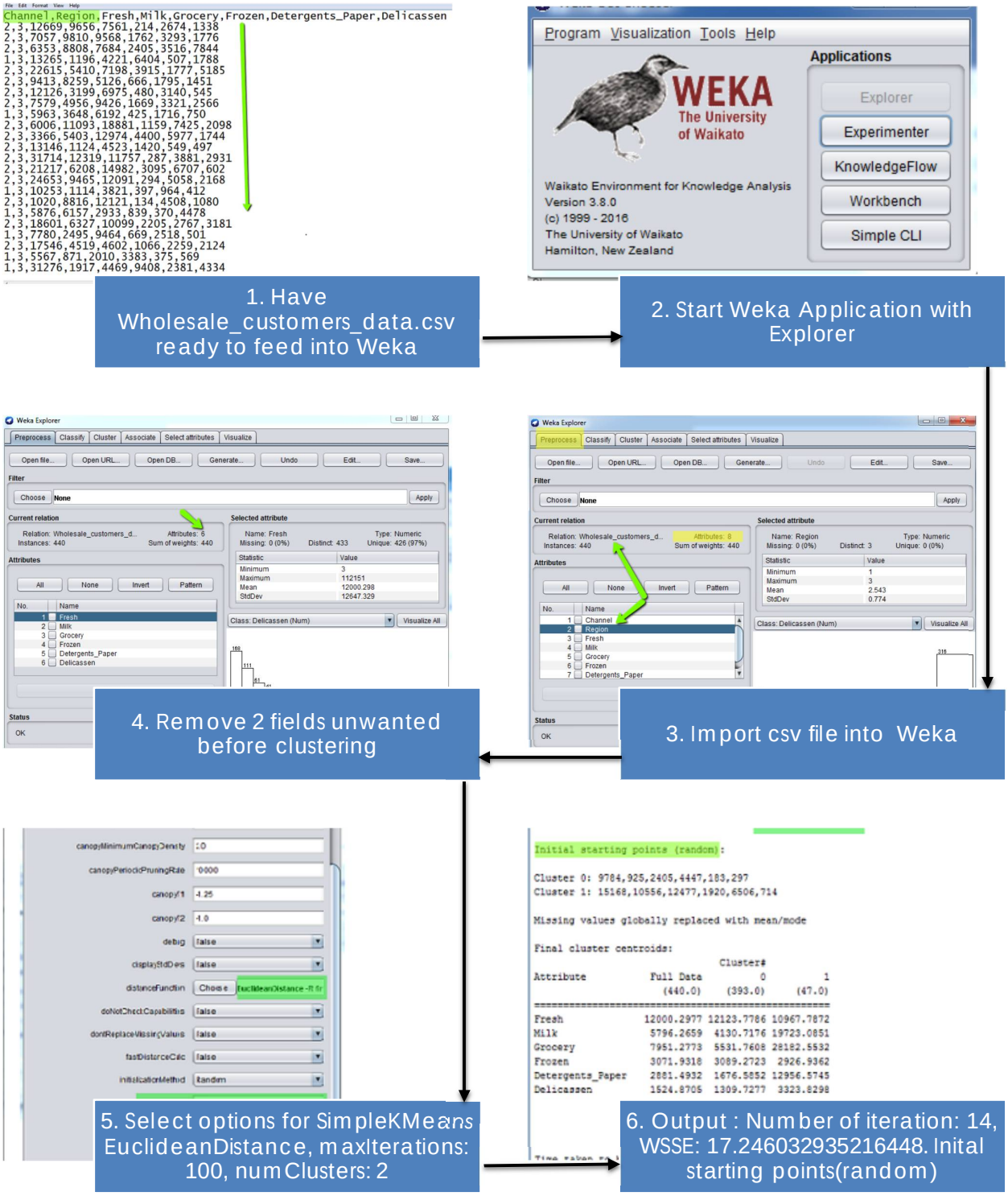
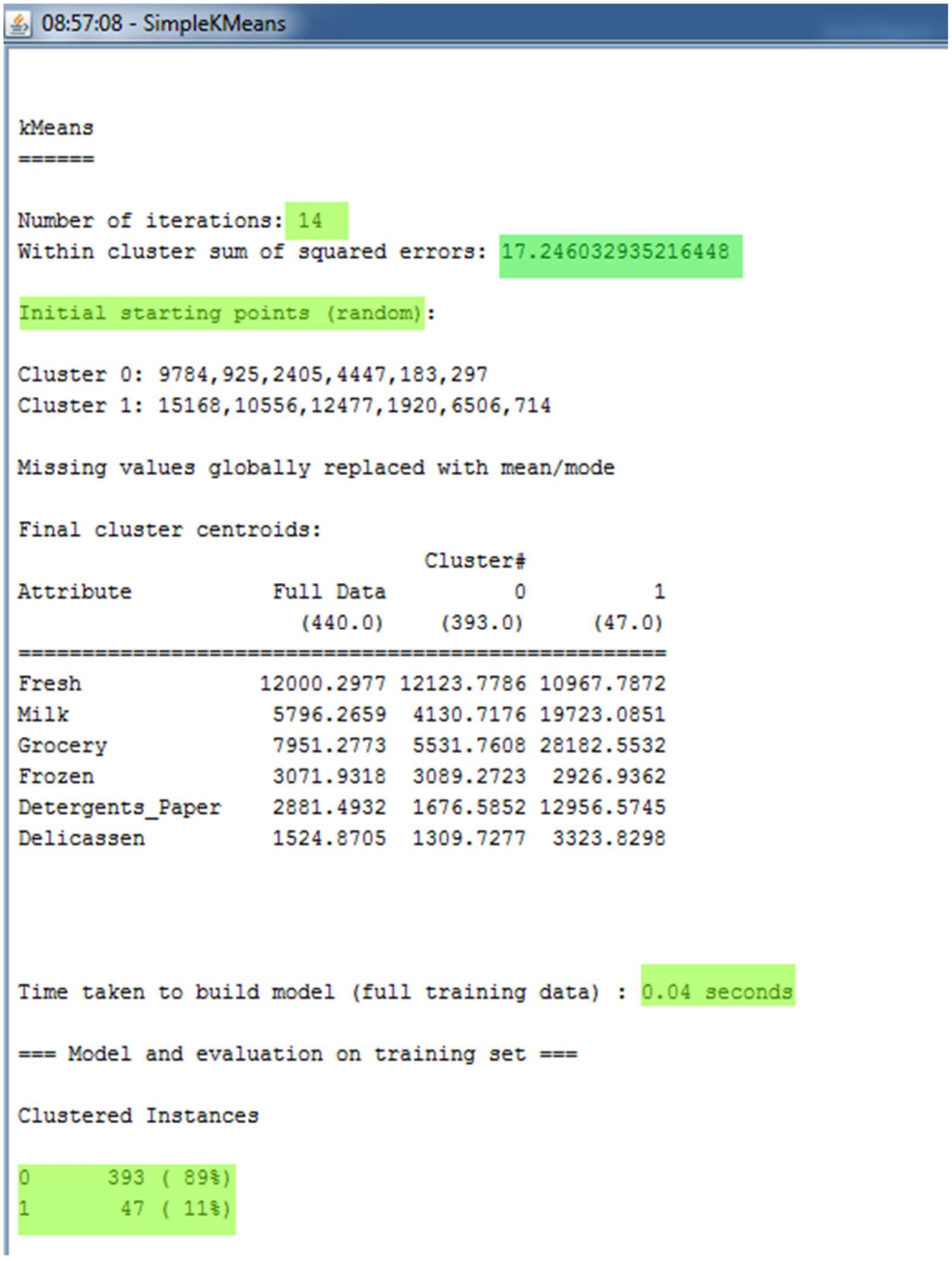


FIGURE 5: RESULT GIVEN OUT BY WEKA APPLICATION



08:57:08 - SimpleKMeans

```
kMeans
======

Number of iterations: 14
Within cluster sum of squared errors: 17.246032935216448

Initial starting points (random):

Cluster 0: 9784,925,2405,4447,183,297
Cluster 1: 15168,10556,12477,1920,6506,714

Missing values globally replaced with mean/mode

Final cluster centroids:
                                Cluster#
Attribute            Full Data          0           1
                       (440.0)    (393.0)     (47.0)
=================================================
Fresh               12000.2977 12123.7786 10967.7872
Milk                 5796.2659  4130.7176 19723.0851
Grocery              7951.2773  5531.7608 28182.5532
Frozen               3071.9318  3089.2723  2926.9362
Detergents_Paper     2881.4932  1676.5852 12956.5745
Delicassen           1524.8705  1309.7277  3323.8298



Time taken to build model (full training data) : 0.04 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       393 ( 89%)
1        47 ( 11%)
```

4.3. Output using the Application

FIGURE 6: OBTAIN OUTPUT USING THE PROGRAM WITH SIMILAR INPUT

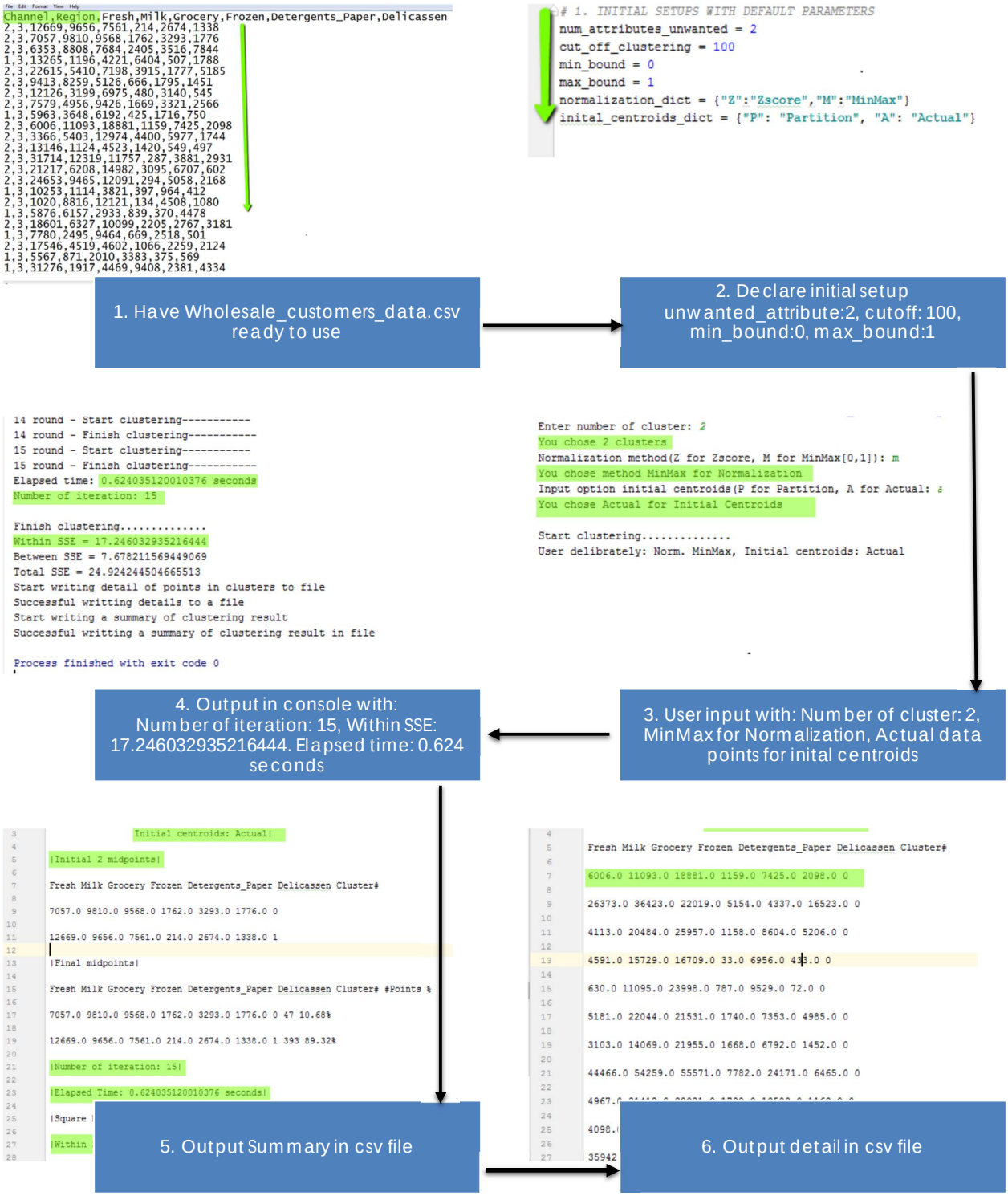
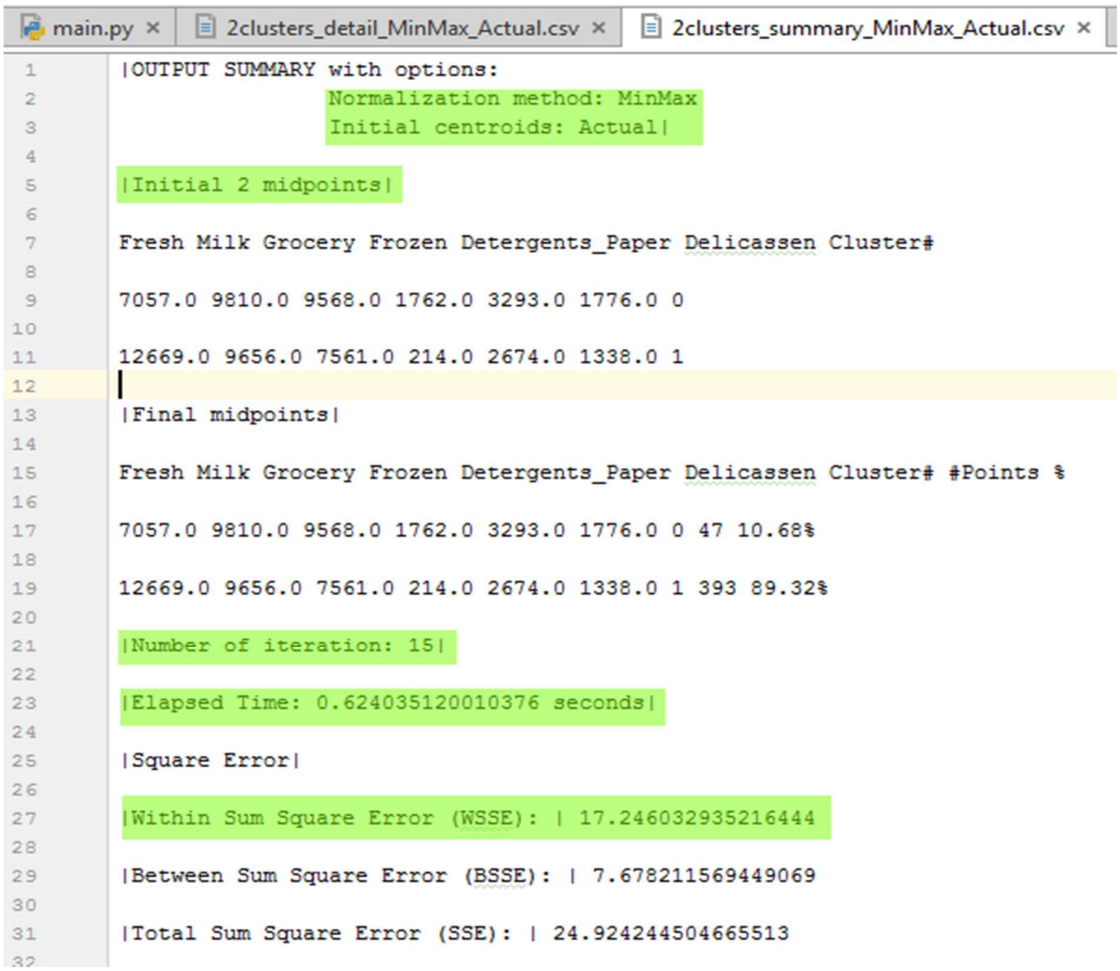


FIGURE 7: OUTPUT GENERATED BY THE PROGRAM WITH SIMILAR INPUT



4.4. Comparison output between Weka and Program

4.4.1. Analysis output with example k=2

Through the example above with the similar input given between Weka and the program, the output given is summarized as

FIGURE 8: COMPARE OUTPUT BETWEEN WEKA AND PROGRAM WHEN K=2

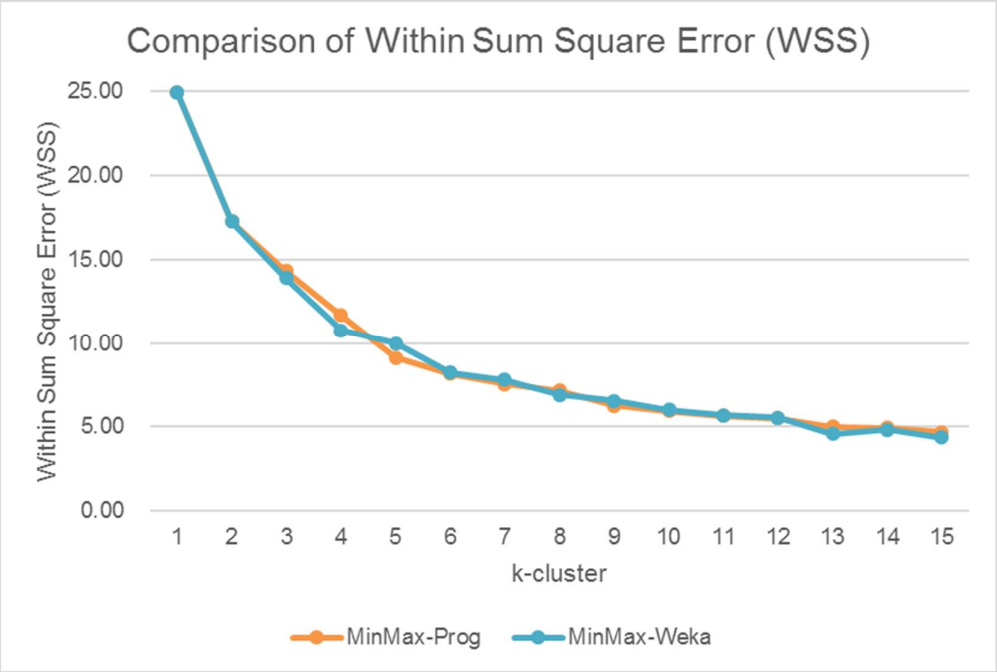
Input	Weka	The program
Number of cluster: 2 **Euclidean Distance** **Max iteration: 100** **(Program currently implicitly using Euclidean as Distance Measure)**	Number of iterations: 14 Within SSE: 17.246032935216448 Cluster 0: 393 (89%) Cluster 1: 47 (11%) Time taken to build model (full training data): 0.04 seconds	Number of iterations: 15 Within SSE: 17.246032935216444 Cluster 0: 47 (10.68%) Cluster 1: 393 (89.32%) Elapsed Time: 0.624035120010376 seconds

Figure 8 shows that Within SSE has nearly similar result, number of instances within each cluster has same result. However, number of iterations Weka manages better by 1, and time taken to run clustering extremely better 0.04 seconds vs 0.624 seconds

4.4.2. Analysis with k>2

When testing within k greater than 2, the result shows difference between output from Weka and the program. Figure 9 shows that the difference of WSS between the program's output and Weka's is average of 0.06, with standard deviation of 0.400765, max of 0.91, min of minus 0.87 when testing k within 1 to 15. Hence, the result in this context is nearly the same.
But Weka appears much better in number of iterations (Figure 10), actual running time (Figure 11) However, when dealing with extreme values such as: k-cluster: 100, 200, 300, 400, 440, the program seems to give slightly smaller SSE, number of iterations when k-cluster bigger (Figure 12).

FIGURE 9: COMPARISON OF WSSE BETWEEN WEKA AND PROGRAM OUTPUT

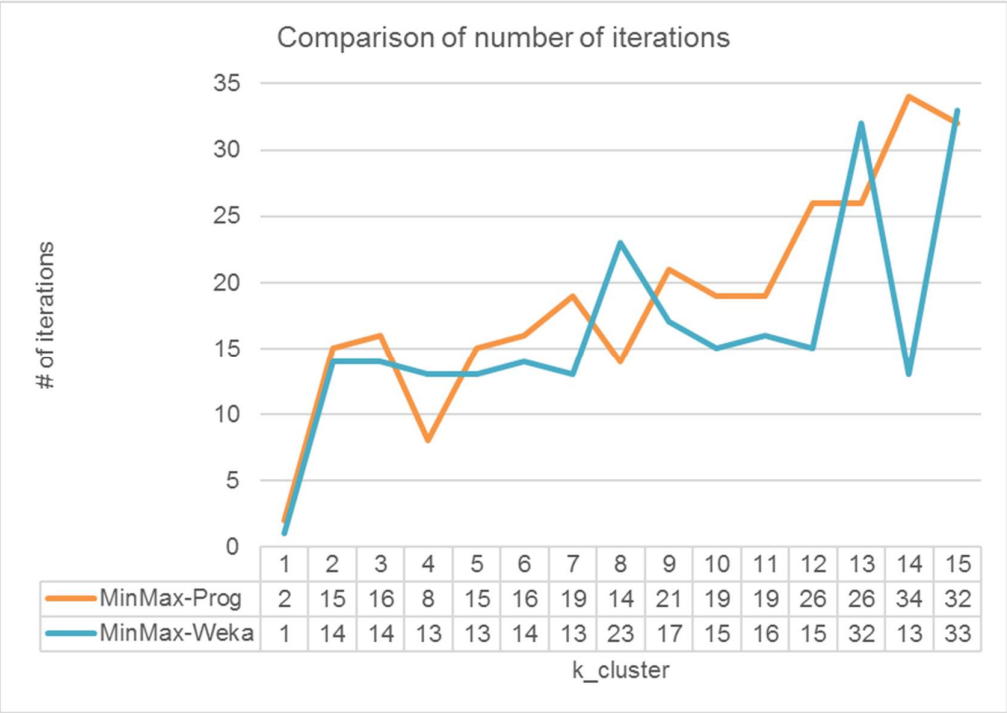


K_cluster	MinMax-Prog	MinMax-Weka	Diff_Prog_Weka
1	24.92	24.92	0.00
2	17.25	17.25	0.00
3	14.34	13.88	0.46
4	11.68	10.77	0.91
5	9.12	9.99	(0.87)
6	8.15	8.22	(0.07)
7	7.55	7.78	(0.23)
8	7.16	6.90	0.27
9	6.27	6.55	(0.28)
10	5.95	6.00	(0.04)
11	5.65	5.70	(0.05)
12	5.50	5.55	(0.05)
13	5.02	4.59	0.44
14	4.95	4.81	0.13
15	4.67	4.36	0.31

Measure the difference between Program and Weka

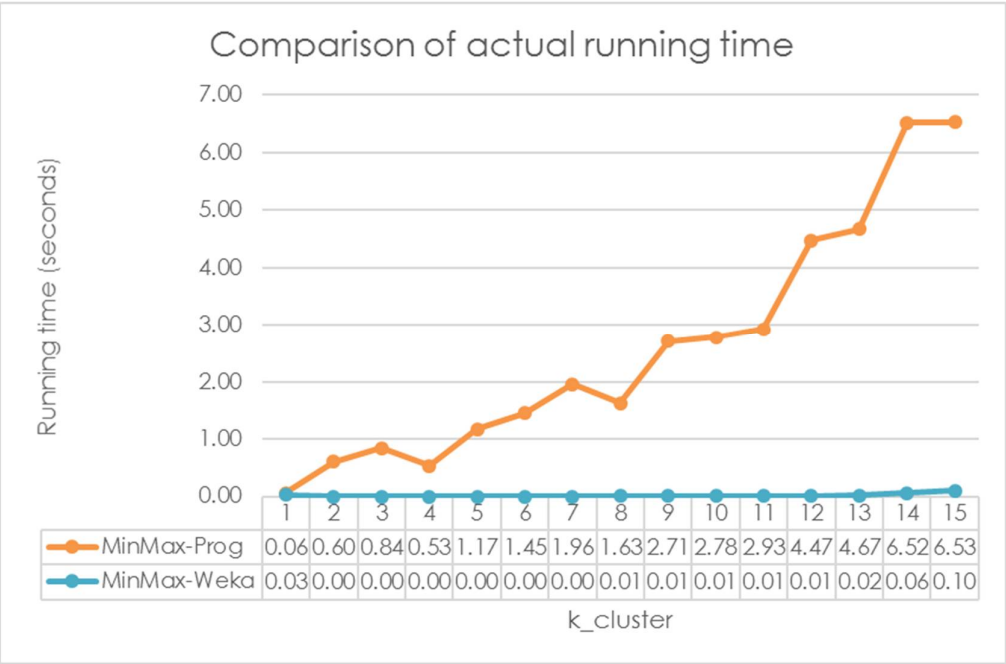
Statistics for Difference	
Mean:	0.06
Min Diff:	(0.87)
Max Diff:	0.91
Median:	0.00
Standard deviation:	0.40

FIGURE 10: COMPARISON OF NUMBER OF ITERATIONS TAKEN BETWEEN WEKA AND PROGRAM



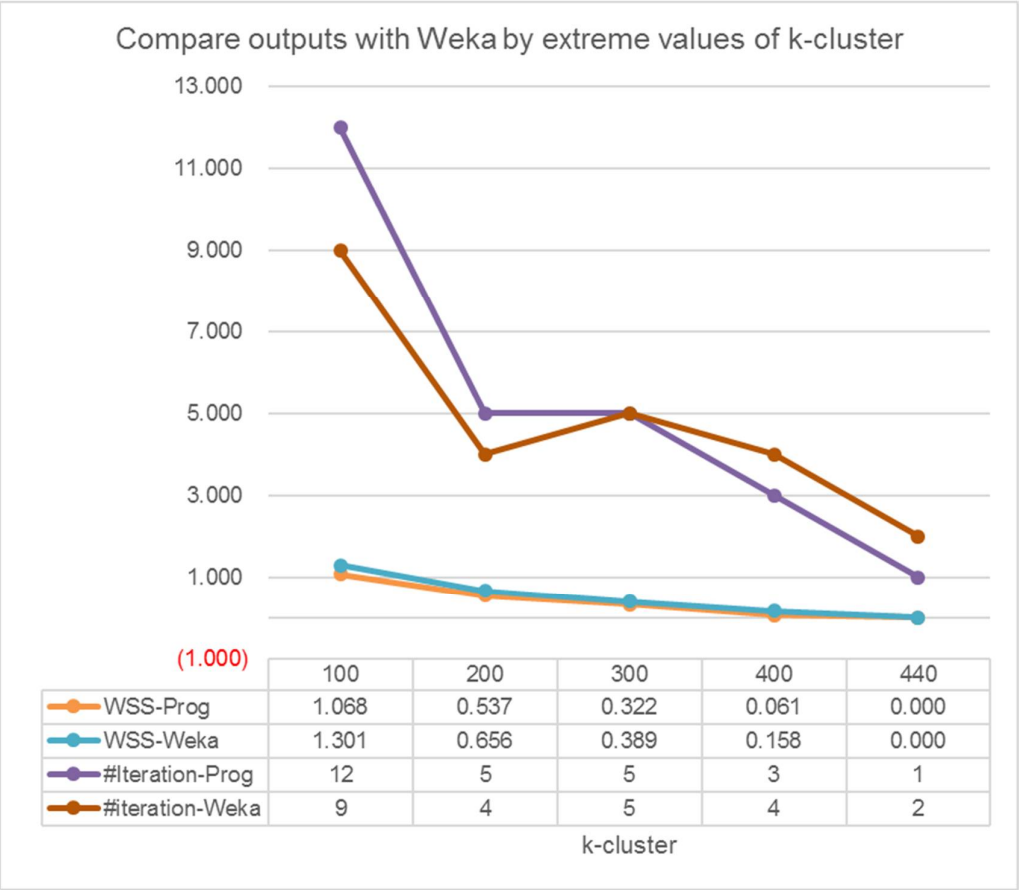
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MinMax-Prog	2	15	16	8	15	16	19	14	21	19	19	26	26	34	32
MinMax-Weka	1	14	14	13	13	14	13	23	17	15	16	15	32	13	33

FIGURE 11: COMPARISON OF ACTUAL RUNNING TIME BETWEEN WEKA AND PROGRAM



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MinMax-Prog	0.06	0.60	0.84	0.53	1.17	1.45	1.96	1.63	2.71	2.78	2.93	4.47	4.67	6.52	6.53
MinMax-Weka	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.02	0.06	0.10

FIGURE 12: COMPARE OUTPUTS BETWEEN WEKA AND PROGRAM IN CASE OF EXTREME VALUES OF K-CLUSTER



	100	200	300	400	440
WSS-Prog	1.068	0.537	0.322	0.061	0.000
WSS-Weka	1.301	0.656	0.389	0.158	0.000
#Iteration-Prog	12	5	5	3	1
#iteration-Weka	9	4	5	4	2

5. Discussion

5.1. Which combination has better chance to get lower WSS?

In order to select which combinations to give higher likelihood of having the better WSS, 4 scenarios have been tested for k-cluster from 1-50. Four combination includes

- MinMax[0,1], Actual data point
- MinMax[0,1], Means of Equal partitions
- Zscore, Actual data point
- Zscore, Means of Equal partitions

To compare across different combination, Percentage of WSS over Total Sum Square Error (SSE) is used. If the WSS/SSE is smaller, the combination indicates the better result, more closeness the data points in clusters are. The details of output for each scenario can be found on Appendix 1. Here is the summary of the result for 50 k-cluster

Figure 13 shows that there is not much difference between choosing either approach of initial centroids when MinMax normalization is selected.

FIGURE 13: PERCENTAGE OF WSS OVER SSE IN SAME NORMALIZATION AS MINMAX

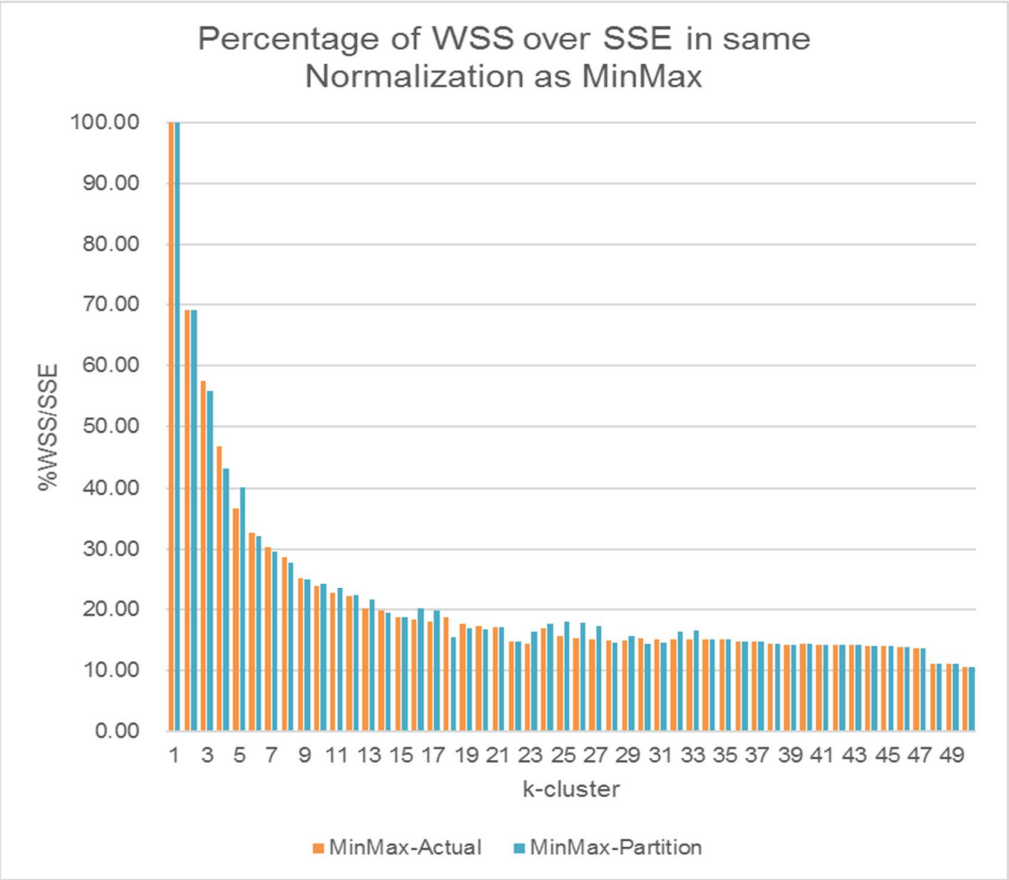


Figure 14, 15 and Figure 16 show clearly that Actual data points as initial centroids works well with Zscore as Normalization method to produce better outcome by 64% vs 34% when it goes with MinMax. Figure 15 gives a sign that Means of equal Partition probably work better with Partition rather than Zscore (56% vs 42%), though it is not so clear in Figure 14 (30% vs 32%)

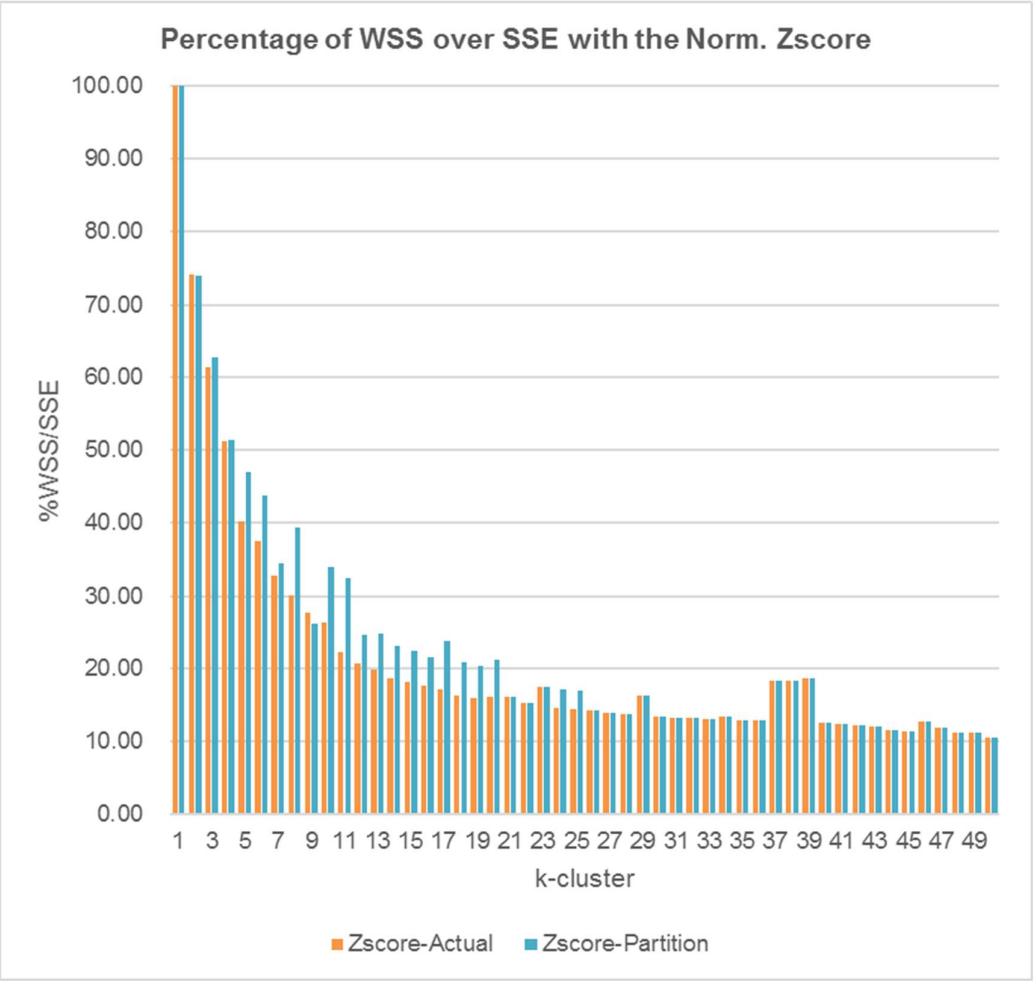
FIGURE 14: COMPARISON WITH THE SAME NORMALIZATION

	Partition better	Equal	Actual better	Total
MinMax	15 (30%)	19 (38%)	16 (32%)	50 (100%)
Zscore	2 (4%)	28 (56%)	**20 (40%)**	50 (100%)

FIGURE 15: COMPARISON WITH THE SAME INITIAL CENTROIDS

	MinMax better	Equal	Zscore better	Total
Actual	17 (34%)	1 (2%)	**32 (64%)**	50 (100%)
Partition	28 (56%)	1 (2%)	21 (42%)	50 (100%)

FIGURE 16: PERCENTAGE OF WSS OVER SSE WITH THE NORM. ZSCORE



According to the test result, there are 2 candidates, combination of (Zscore and Actual data points) and (MinMax and Partition) potentially produce higher chance of having better result. But it is very clear that Zscore works well with Actual data points. In the small datasets, it does not show the much impact, but in the bigger dataset, it would be a good choice

The reason for combination of Zscore and Actual data points as a good choice lies on the advantage of Zscore and selected K-means algorithm. Zscore normalization takes into account all values of dataset and it shows the directly relative positions of data points to means, and indirectly to other data points. By selecting actual data points as initial centroids, it is 100% guarantee of initial centroids found.

It is not always true for Means of equal partitions, especially for such small dataset with highly skewed. It is due to the fact that it might not have any data value assigned to a particular partition in some case of k-cluster. For example, in the program by choosing k-cluster = 50, Partition, either MinMax or Zscore, it could not find out data point assigned to some partitions. However, it could be more suitable for dataset with nearly normally distributed, it promises having good initial centroids, and decrease number of iterations and small WSS as well.

When it comes to business, it needs to be evaluated by decision makers to pick the best number of segments. It should have variety of options to choose from, and pick the best one, which matches best the requirements.

5.2. Limitation of current program

When performing descriptive analysis on the dataset, it turns out to be highly skewed to the right, and contain extreme values. However, the program has not taken into account this aspect in order to choose which approach should be more appropriate. The better way to deal with such dataset is

- Perform 2 test cases with extreme values included and excluded. Compare the results, and examine the outliners if necessary.
- Only tested on small dataset with highly skewed.
- Consider to use City Block Distance as Distance Measure and Median as center of cluster in K-means clustering to reduce the skewness effect of dataset

6. Analysis business value aspects

6.1. Customers partitioned into segments

Limited with 5 clusters, I performed testing with 4 different approaches in the program and in Weka. the result comes out quite positive (Figure 17). I picked the result with lowest WSS/SSE of 36.59%

FIGURE 17: OUTPUT CLUSTERING WITH K-CLUSTER = 5

Norm.method	Init.centroids	k	#iter	Time	wsse	bsse	sse	%wsse/sse
MinMax-Weka	Actual	5	13	-	9.99	14.94	24.92	40.06
MinMax-Prog	Actual	5	15	1.17	9.12	15.80	24.92	36.59
MinMax	Partition	5	23	1.84	9.99	14.94	24.92	40.06
Zscore	Actual	5	18	1.42	1,062.68	1,577.32	2,640.00	40.25
Zscore	Partition	5	17	1.34	1,241.49	1,398.51	2,640.00	47.03

Cluster	# instances	%
0	3	0.68%
1	10	2.27%
2	55	12.50%
3	279	63.41%
4	93	21.14%
Total	440	100.00%

6.2. Customer behavior analysis

6.2.1. Overall sales from customers

Figure 18 describes that 76% of sales comes from 3 items including Fresh, Milk, Grocery. Other products only contribute 24% of revenues.

Moreover, Figure 19 & 20 illustrates that there is highly positive, strong correlation between sales of Grocery and Detergents Paper (0.92). In other words, Customers tends to buy these 2 products together in the trip to the shop. We also can find positive, moderate relationship between Milk and Grocery and Detergents Paper as well. (Figure 19).

By understanding the relationship between Grocery and Detergents Paper and Milk, manager can use to organize products display so that it helps customers easily to find products, and boost the sale in the end.

FIGURE 18: PROPORTION OF SALES BY ITEMS

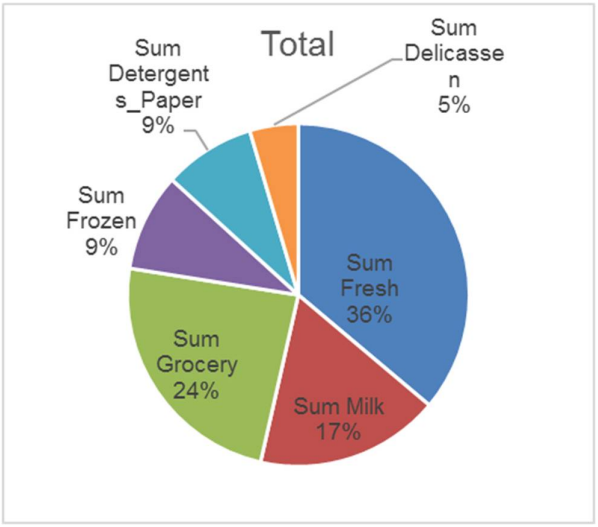
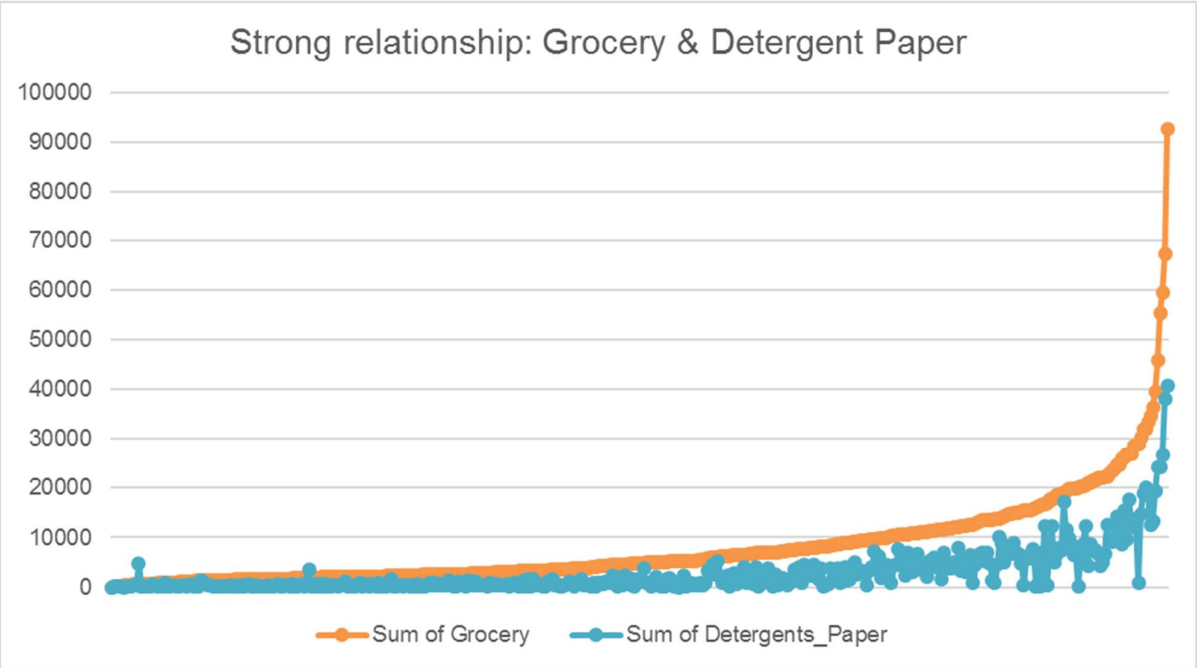


FIGURE 19: TABLE OF CORRELATION COEFFICIENT BETWEEN SALES OF ITEMS

All customers data	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Fresh	1.00					
Milk	0.10	1.00				
Grocery	(0.01)	0.73	1.00			
Frozen	0.35	0.12	(0.04)	1.00		
Detergents_Paper	(0.10)	0.66	0.92	(0.13)	1.00	
Delicassen	0.24	0.41	0.21	0.39	0.07	1.00

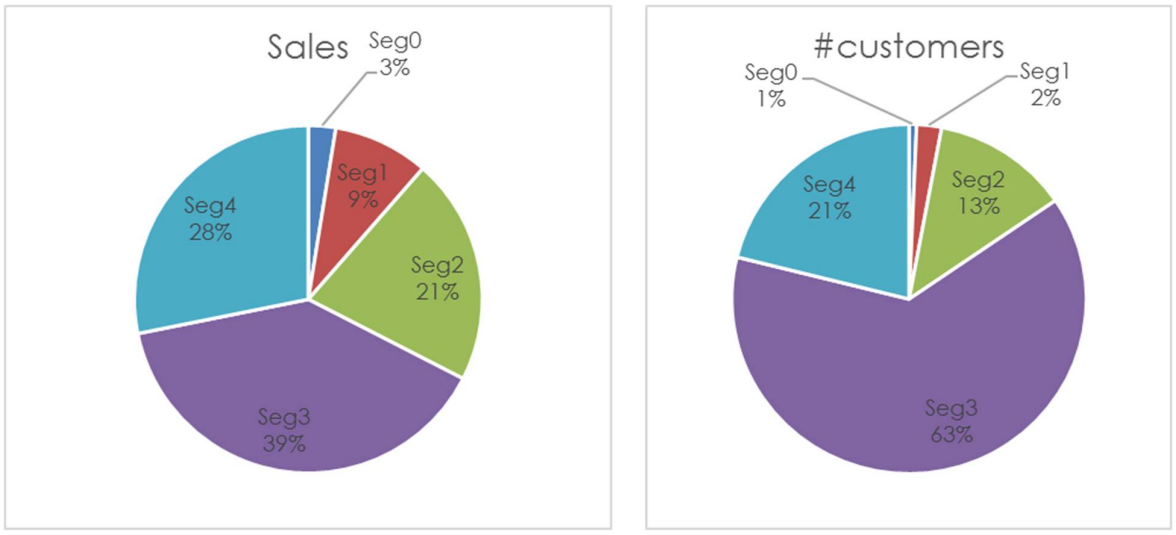
FIGURE 20: POSITIVE STRONG RELATIONSHIP BETWEEN GROCERY AND DETERGENT PAPER



6.2.2. Customers' behavior broken down by clusters

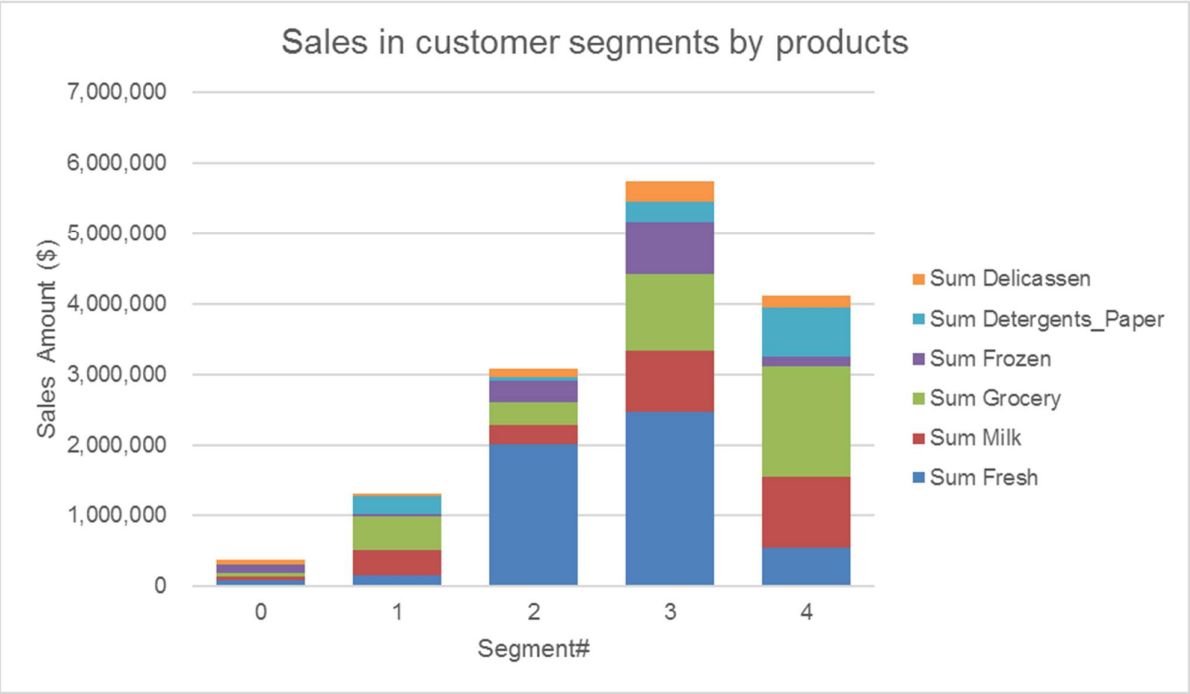
First, by segmenting customer base into different groups, it turns out that Group 0 and 1, which contain only 3% total customers, bring 12% sales, that's 4 times in term of magnitude. Also, group 3 with 63% customers but only generates 39% sales, one half value (Figure 20)

FIGURE 21: PROPORTION OF CUSTOMERS AND SALES



Second, it is clearly that customers in each segment are not interested in the same type of products, they tend to buy more than certain products than others. Figure 21 indicates that customers in Segment3 and Segments4 buy most of Fresh products, about 4,500,000$ in sales, or 85.2% of Fresh Sales. Group 3 and 4 consume the largest grocery about 2,600,000$ in sales, or 76.07% of Grocery sales. Customers in segment 1 are much interested in Grocery over Fresh products, and nearly haven't interest at all in Delicassen.

FIGURE 22: SALES IN CUSTOMER SEGMENTS BROKEN DOWN BY PRODUCTS

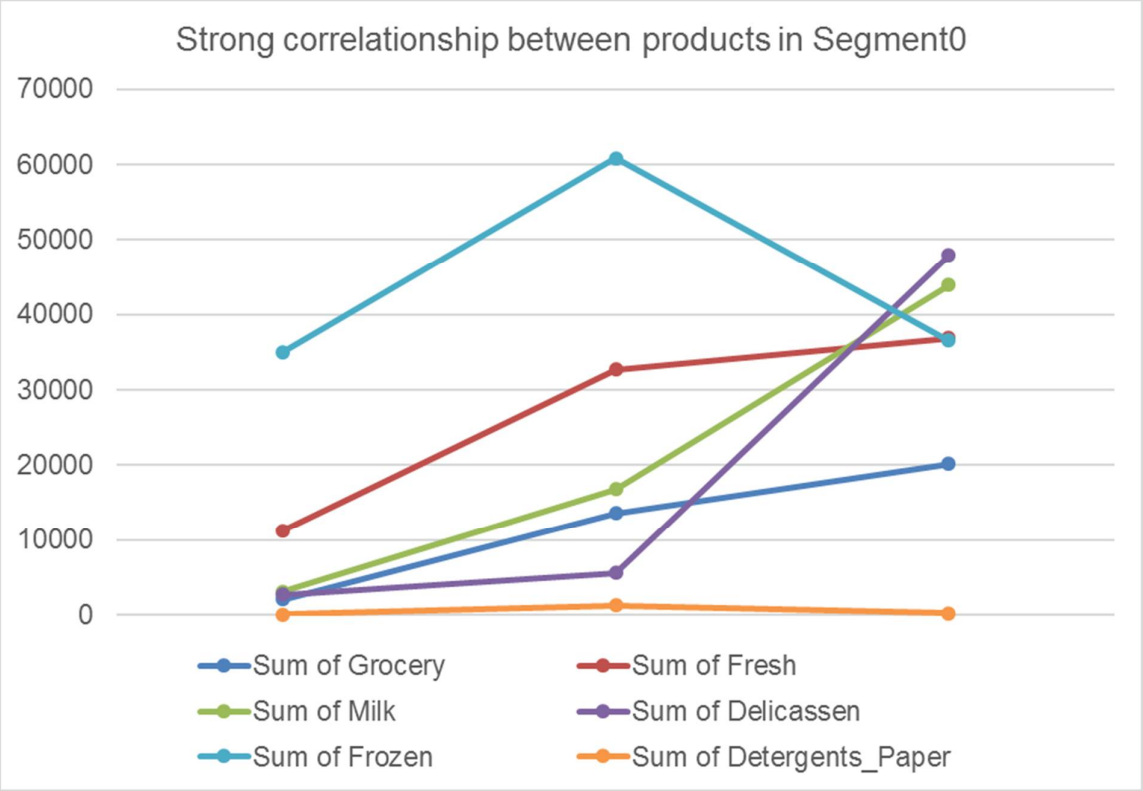


Third, the very interesting in Segment0 is that customers buying Detergents Paper guarantees buying Frozen products. So do the couple of Grocery and Fresh with positive strong correlation of 0.98 and those of Delicassen and Milk with correlation of 0.96. Figure 23 proves for statements above

Surprisingly, these customers have not the same behavior as general customer, such that buying Grocery and Detergents Paper together (correlation only by 0.28). We could predict that these customers in the segment might have high income, consume cooked meats, diary product, and unusual or foreign prepared foods, prefer grocery and fresh products.

FIGURE 23: STRONG CORRELATION BETWEEN PRODUCTS IN SEGMENT0

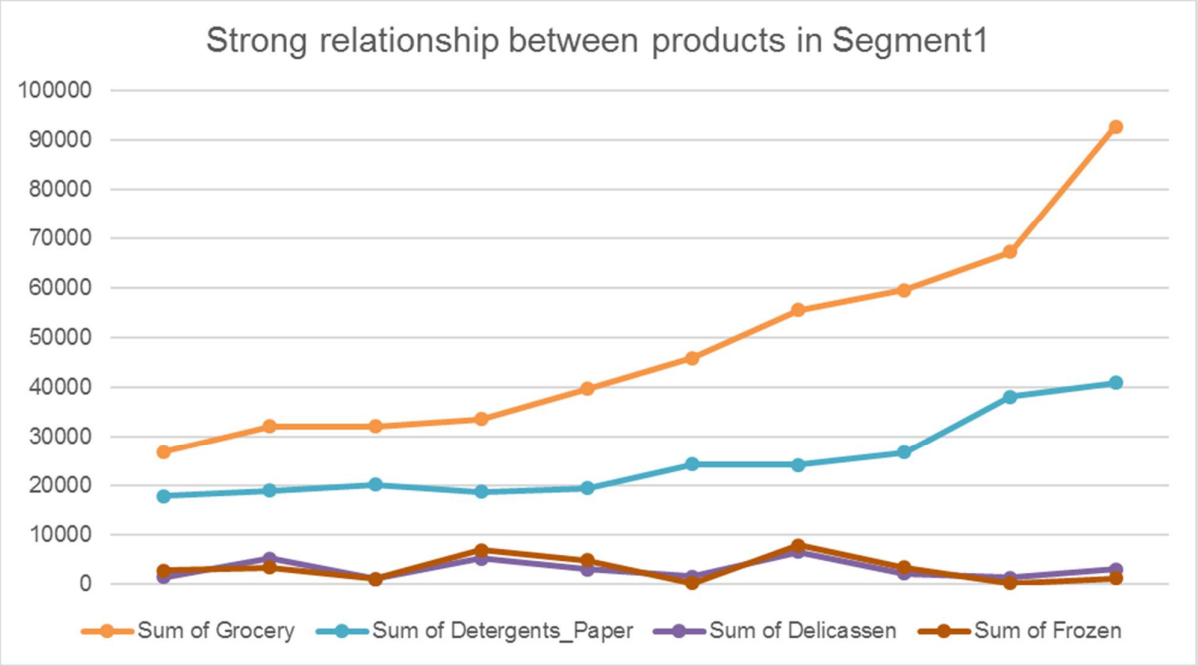
Seg0	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Fresh	1.00					
Milk	0.85	1.00				
Grocery	0.98	0.94	1.00			
Frozen	0.41	(0.14)	0.21	1.00		
Detergents_Paper	0.48	(0.06)	0.28	1.00	1.00	
Delicassen	0.67	0.96	0.81	(0.40)	(0.33)	1.00



Fourth, unlike Segment0, shoppers in Segment1 buy Grocery and Detergent Paper together high frequently. Also, when they buy Frozen product, and choose Delicassen as well. Figure 24 describes those relationships in terms of correlation co-efficient and chart

FIGURE 24: STRONG RELATIONSHIP BETWEEN PRODUCTS IN SEGMENT1

Seg1	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Fresh	1.00					
Milk	0.63	1.00				
Grocery	0.29	(0.01)	1.00			
Frozen	0.44	0.14	(0.26)	1.00		
Detergents_Paper	0.10	(0.15)	0.95	(0.46)	1.00	
Delicassen	0.34	0.08	(0.04)	0.83	(0.21)	1.00



One things should be noticed that such relationships between types of products do not appear on any 3 other segments. The comprehensive correlation coefficients calculated for 5 segments can be found in Appendix 2

To sum up, by segmenting customer base of the business, it gives a better picture about where customers behavior, whether sales come from, which customers more potential. In the case of this dataset, most of sales are generated by Fresh, Milk, Grocery. Customers in Segment0 and Segment1 are the high value customers, who should be taken care of, though there are only 3% in terms of numbers, but they bring 12% revenue. These group also could be most sensitive or responsive to promotion program.

7. Conclusion

In the report, I did implementation of K-means algorithms for clustering Wholesale customers dataset retrieved from UIC. The program uses 04 different combinations for Normalization (either Zscore or MinMax[0,1]) and selecting initial centroids (either Actual data points or Means of equal partition), by using Euclidean distance as Distance Measure. The program gives out the result of WSS nearly similar to Weka's output which is used to compare with under the similar inputs.

By examining K-means with different combinations, I figured that there are two potential candidates giving the higher probability of having good K-means result, but Zscore for Normalization and Actual Data points for initial centroids should be the better choice in general. However, with particular k-cluster, it would be good idea, to test different approach and figure which approach gives better answer.

By using K-means to segment customer database, I could separate customers into different groups, customers with behaviors highly correlated are grouped together. Therefore, it is extremely useful for decision makers to know how to react and select appropriate to target customer of interest group.

As mention above about the limitation of my work, I intend to improve it in the future so that it can be designed specifically to highly skewed dataset which is quite popular in sales, marketing, and finance.

8. List of references

M., S. A., & Wasimi, S. A. (2007). *Data* mining*: Methods and techniques*. South Melbourne, Vic.: Thomson.

Arthur, D., & Vassilvitskii, S. (2006). How Slow is the k-means Method?. Proceedings of the twenty-second annual symposium on Computational geometry. ACM.

(2006). SimpleKMeans (Documentation for extended WEKA including ... - DBS. Retrieved September 15, 2016, from http://www.dbs.ifi.lmu.de/~zimek/diplomathesis/implementations/EHNDs/doc/weka/clusterers/SimpleKMeans.html.

(2015). What is the time complexity of clustering algorithms? - ResearchGate. Retrieved September 16, 2016, from https://www.researchgate.net/post/What_is_the_time_complexity_of_clustering_algorithms

(2014). UCI Machine Learning Repository: Wholesale customers Data Set. Retrieved September 16, 2016, from https://archive.ics.uci.edu/ml/datasets/Wholesale+customers..

(2015). R vs Python for Data Science: The Winner is - KDnuggets. Retrieved September 16, 2016, from http://www.kdnuggets.com/2015/05/r-vs-python-data-science.html.

(2011). k-means clustering - Wikipedia, the free encyclopedia. Retrieved September 15, 2016, from https://en.wikipedia.org/wiki/K-means_clustering.

(2008). Cluster Validation. Retrieved September 15, 2016, from http://www.cs.kent.edu/~jin/DM08/ClusterValidation.pdf.

Normalize - Weka. Retrieved September 15, 2016, from http://weka.sourceforge.net/doc.stable/weka/filters/unsupervised/attribute/Normalize.html

(2010). SimpleKMeans - Weka. Retrieved September 15, 2016, from http://weka.sourceforge.net/doc.dev/weka/clusterers/SimpleKMeans.html.

(2014). K-Mean Evaluation in Weka Tool and Modifying It ... - Academia.edu. Retrieved September 15, 2016, from http://www.academia.edu/9326983/K-Mean_Evaluation_in_Weka_Tool_and_Modifying_It_using_Standard_Score_Method.

(2008). Clustering Algorithms: K-means. Retrieved September 16, 2016, from http://www.cs.princeton.edu/courses/archive/spr08/cos435/Class_notes/clustering2_toPost.pdf.

9. Appendix 1

FIGURE 25: TEST RESULTS IN COMBINATION OF MINMAX AND ACTUAL DATA POINTS

Norm.method	Init.centroids	k-cluster	#iteration	Time	wsse	bsse	sse	%wsse/sse
MinMax	Actual	1	2	0.06	24.92	-	24.92	100.00
MinMax	Actual	2	15	0.60	17.25	7.68	24.92	69.19
MinMax	Actual	3	16	0.84	14.34	10.59	24.92	57.53
MinMax	Actual	4	8	0.53	11.68	13.25	24.92	46.85
MinMax	Actual	5	15	1.17	9.12	15.80	24.92	36.59
MinMax	Actual	6	16	1.45	8.15	16.77	24.92	32.70
MinMax	Actual	7	19	1.96	7.55	17.38	24.92	30.29
MinMax	Actual	8	14	1.63	7.16	17.76	24.92	28.73
MinMax	Actual	9	21	2.71	6.27	18.65	24.92	25.16
MinMax	Actual	10	19	2.78	5.95	18.97	24.92	23.89
MinMax	Actual	11	19	2.93	5.65	19.28	24.92	22.66
MinMax	Actual	12	26	4.47	5.50	19.42	24.92	22.07
MinMax	Actual	13	26	4.67	5.02	19.90	24.92	20.15
MinMax	Actual	14	34	6.52	4.95	19.98	24.92	19.84
MinMax	Actual	15	32	6.53	4.67	20.25	24.92	18.74
MinMax	Actual	16	27	5.85	4.57	20.35	24.92	18.34
MinMax	Actual	17	26	6.93	4.49	20.43	24.92	18.03
MinMax	Actual	18	24	6.76	4.64	20.28	24.92	18.63
MinMax	Actual	19	38	9.68	4.41	20.51	24.92	17.70
MinMax	Actual	20	30	8.07	4.32	20.61	24.92	17.32
MinMax	Actual	21	31	8.70	4.24	20.68	24.92	17.03
MinMax	Actual	22	29	8.50	3.68	21.24	24.92	14.77
MinMax	Actual	23	27	8.92	3.59	21.33	24.92	14.40
MinMax	Actual	24	29	10.14	4.23	20.70	24.92	16.95
MinMax	Actual	25	20	6.90	3.90	21.03	24.92	15.63
MinMax	Actual	26	23	9.32	3.79	21.13	24.92	15.22
MinMax	Actual	27	14	5.58	3.77	21.16	24.92	15.12
MinMax	Actual	28	17	6.29	3.73	21.19	24.92	14.99
MinMax	Actual	29	33	13.73	3.72	21.20	24.92	14.94
MinMax	Actual	30	22	9.32	3.82	21.10	24.92	15.34
MinMax	Actual	31	21	9.84	3.78	21.15	24.92	15.16

Norm.method	Init.centroids	k-cluster	#iteration	Time	wsse	bsse	sse	%wsse/sse
MinMax	Actual	32	21	8.93	3.77	21.15	24.92	15.13
MinMax	Actual	33	22	9.52	3.78	21.15	24.92	15.16
MinMax	Actual	34	22	10.38	3.77	21.16	24.92	15.12
MinMax	Actual	35	21	9.72	3.74	21.18	24.92	15.01
MinMax	Actual	36	19	8.96	3.67	21.25	24.92	14.74
MinMax	Actual	37	21	10.19	3.67	21.26	24.92	14.70
MinMax	Actual	38	23	11.43	3.57	21.36	24.92	14.32
MinMax	Actual	39	21	10.67	3.55	21.37	24.92	14.25
MinMax	Actual	40	21	10.99	3.56	21.37	24.92	14.27
MinMax	Actual	41	21	11.22	3.54	21.38	24.92	14.22
MinMax	Actual	42	21	11.51	3.53	21.39	24.92	14.17
MinMax	Actual	43	15	8.37	3.53	21.39	24.92	14.17
MinMax	Actual	44	13	7.46	3.50	21.43	24.92	14.03
MinMax	Actual	45	13	7.61	3.49	21.44	24.92	13.99
MinMax	Actual	46	12	7.16	3.45	21.48	24.92	13.82
MinMax	Actual	47	13	7.92	3.41	21.51	24.92	13.70
MinMax	Actual	48	9	5.61	2.76	22.16	24.92	11.08
MinMax	Actual	49	9	5.72	2.75	22.18	24.92	11.02
MinMax	Actual	50	9	6.24	2.62	22.31	24.92	10.49

FIGURE 26: TEST RESULT IN COMBINATION OF MINMAX AND MEANS OF EQUAL PARTITION

Norm.method	Init.centroids	k-cluster	#iteration	Time	wsse	bsse	sse	%wsse/sse
MinMax	Partition	1	1	0.08	24.92	-	24.92	100.00
MinMax	Partition	2	14	0.58	17.25	7.68	24.92	69.19
MinMax	Partition	3	15	0.81	13.95	10.98	24.92	55.96
MinMax	Partition	4	18	1.32	10.77	14.16	24.92	43.20
MinMax	Partition	5	23	1.84	9.99	14.94	24.92	40.06
MinMax	Partition	6	46	4.18	8.00	16.93	24.92	32.08
MinMax	Partition	7	24	2.54	7.35	17.57	24.92	29.50
MinMax	Partition	8	27	3.17	6.90	18.02	24.92	27.69
MinMax	Partition	9	29	3.80	6.22	18.70	24.92	24.98
MinMax	Partition	10	44	6.18	6.06	18.86	24.92	24.32
MinMax	Partition	11	33	5.08	5.88	19.04	24.92	23.59
MinMax	Partition	12	24	3.99	5.58	19.34	24.92	22.40
MinMax	Partition	13	28	5.04	5.41	19.52	24.92	21.69
MinMax	Partition	14	34	9.21	4.85	20.08	24.92	19.45
MinMax	Partition	15	41	11.54	4.65	20.27	24.92	18.66
MinMax	Partition	16	27	8.17	5.03	19.89	24.92	20.18
MinMax	Partition	17	25	5.85	4.93	20.00	24.92	19.77
MinMax	Partition	18	43	12.76	3.87	21.06	24.92	15.51
MinMax	Partition	19	30	7.81	4.20	20.72	24.92	16.86
MinMax	Partition	20	30	8.20	4.17	20.75	24.92	16.74
MinMax	Partition	21	31	9.88	4.24	20.68	24.92	17.03
MinMax	Partition	22	29	9.63	3.68	21.24	24.92	14.77
MinMax	Partition	23	48	15.17	4.07	20.86	24.92	16.31
MinMax	Partition	24	41	13.67	4.39	20.54	24.92	17.61
MinMax	Partition	25	29	10.06	4.47	20.46	24.92	17.92
MinMax	Partition	26	36	13.59	4.45	20.47	24.92	17.87
MinMax	Partition	27	44	16.37	4.31	20.62	24.92	17.27
MinMax	Partition	28	41	15.72	3.61	21.31	24.92	14.49
MinMax	Partition	29	24	9.35	3.91	21.02	24.92	15.68
MinMax	Partition	30	37	14.99	3.58	21.34	24.92	14.37
MinMax	Partition	31	44	19.17	3.61	21.31	24.92	14.50
MinMax	Partition	32	50	21.73	4.07	20.86	24.92	16.31

Norm.method	Init.centroids	k-cluster	#iteration	Time	wsse	bsse	sse	%wsse/sse
MinMax	Partition	33	28	12.79	4.13	20.80	24.92	16.56
MinMax	Partition	34	22	10.93	3.77	21.16	24.92	15.12
MinMax	Partition	35	21	10.53	3.74	21.18	24.92	15.01
MinMax	Partition	36	19	9.83	3.67	21.25	24.92	14.74
MinMax	Partition	37	21	11.55	3.67	21.26	24.92	14.70
MinMax	Partition	38	23	13.24	3.57	21.36	24.92	14.32
MinMax	Partition	39	21	11.80	3.55	21.37	24.92	14.25
MinMax	Partition	40	21	12.22	3.56	21.37	24.92	14.27
MinMax	Partition	41	21	12.47	3.54	21.38	24.92	14.22
MinMax	Partition	42	21	12.92	3.53	21.39	24.92	14.17
MinMax	Partition	43	15	9.77	3.53	21.39	24.92	14.17
MinMax	Partition	44	13	8.65	3.50	21.43	24.92	14.03
MinMax	Partition	45	13	9.61	3.49	21.44	24.92	13.99
MinMax	Partition	46	12	8.38	3.45	21.48	24.92	13.82
MinMax	Partition	47	13	9.05	3.41	21.51	24.92	13.70
MinMax	Partition	48	9	6.69	2.76	22.16	24.92	11.08
MinMax	Partition	49	9	6.92	2.75	22.18	24.92	11.02
MinMax	Partition	50	9	7.04	2.62	22.31	24.92	10.49

FIGURE 27: TEST RESULT IN COMBINATION OF ZSCORE AND ACTUAL DATA POINT

Norm.method	Init.centroids	k-cluster	#iteration	Time	wsse	bsse	sse	%wsse/sse
Zscore	Actual	1	2	0.05	2,640.00	-	2,640.00	100.00
Zscore	Actual	2	13	0.52	1,956.12	683.88	2,640.00	74.10
Zscore	Actual	3	19	1.01	1,620.50	1,019.50	2,640.00	61.38
Zscore	Actual	4	21	1.37	1,352.78	1,287.22	2,640.00	51.24
Zscore	Actual	5	18	1.42	1,062.68	1,577.32	2,640.00	40.25
Zscore	Actual	6	17	1.55	990.43	1,649.57	2,640.00	37.52
Zscore	Actual	7	13	1.33	867.30	1,772.70	2,640.00	32.85
Zscore	Actual	8	16	1.87	792.18	1,847.82	2,640.00	30.01
Zscore	Actual	9	19	2.85	733.63	1,906.37	2,640.00	27.79
Zscore	Actual	10	21	3.38	694.66	1,945.34	2,640.00	26.31
Zscore	Actual	11	28	4.58	589.37	2,050.63	2,640.00	22.32
Zscore	Actual	12	40	7.09	546.38	2,093.62	2,640.00	20.70
Zscore	Actual	13	24	4.72	522.31	2,117.69	2,640.00	19.78
Zscore	Actual	14	25	5.28	493.18	2,146.82	2,640.00	18.68
Zscore	Actual	15	27	5.95	478.23	2,161.77	2,640.00	18.11
Zscore	Actual	16	18	4.13	465.76	2,174.24	2,640.00	17.64
Zscore	Actual	17	18	4.28	450.06	2,189.94	2,640.00	17.05
Zscore	Actual	18	24	5.99	427.09	2,212.91	2,640.00	16.18
Zscore	Actual	19	53	13.93	419.42	2,220.58	2,640.00	15.89
Zscore	Actual	20	23	6.31	425.08	2,214.92	2,640.00	16.10
Zscore	Actual	21	26	7.72	423.17	2,216.83	2,640.00	16.03
Zscore	Actual	22	38	11.66	401.42	2,238.58	2,640.00	15.21
Zscore	Actual	23	22	7.06	460.58	2,179.42	2,640.00	17.45
Zscore	Actual	24	25	8.53	386.06	2,253.94	2,640.00	14.62
Zscore	Actual	25	20	6.98	377.63	2,262.37	2,640.00	14.30
Zscore	Actual	26	24	8.70	373.24	2,266.76	2,640.00	14.14
Zscore	Actual	27	29	10.83	366.47	2,273.53	2,640.00	13.88
Zscore	Actual	28	22	8.57	362.81	2,277.19	2,640.00	13.74
Zscore	Actual	29	24	9.58	429.22	2,210.78	2,640.00	16.26
Zscore	Actual	30	24	9.90	351.79	2,288.21	2,640.00	13.33
Zscore	Actual	31	25	10.63	350.21	2,289.79	2,640.00	13.27
Zscore	Actual	32	33	14.23	348.65	2,291.35	2,640.00	13.21

Norm.method	Init.centroids	k-cluster	#iteration	Time	wsse	bsse	sse	%wsse/sse
Zscore	Actual	33	32	14.26	344.25	2,295.75	2,640.00	13.04
Zscore	Actual	34	18	8.73	352.73	2,287.27	2,640.00	13.36
Zscore	Actual	35	23	11.12	340.89	2,299.11	2,640.00	12.91
Zscore	Actual	36	22	10.88	341.07	2,298.93	2,640.00	12.92
Zscore	Actual	37	21	10.50	484.27	2,155.73	2,640.00	18.34
Zscore	Actual	38	23	11.75	481.00	2,159.00	2,640.00	18.22
Zscore	Actual	39	10	5.19	489.82	2,150.18	2,640.00	18.55
Zscore	Actual	40	17	9.18	332.29	2,307.71	2,640.00	12.59
Zscore	Actual	41	18	9.92	325.17	2,314.83	2,640.00	12.32
Zscore	Actual	42	15	8.44	320.63	2,319.37	2,640.00	12.15
Zscore	Actual	43	16	9.61	319.35	2,320.65	2,640.00	12.10
Zscore	Actual	44	13	8.27	303.89	2,336.11	2,640.00	11.51
Zscore	Actual	45	23	14.85	299.71	2,340.29	2,640.00	11.35
Zscore	Actual	46	13	8.23	334.04	2,305.96	2,640.00	12.65
Zscore	Actual	47	13	8.33	310.82	2,329.18	2,640.00	11.77
Zscore	Actual	48	13	9.19	296.66	2,343.34	2,640.00	11.24
Zscore	Actual	49	13	8.86	295.44	2,344.56	2,640.00	11.19
Zscore	Actual	50	13	8.71	274.94	2,365.06	2,640.00	10.41

FIGURE 28: TEST RESULT IN COMBINATION OF ZSCORE AND PARTITION

Norm.method	Init.centroids	k-cluster	#iteration	Time	wsse	bsse	sse	%wsse/sse
Zscore	Partition	1	1	0.09	2,640.00	-	2,640.00	100.00
Zscore	Partition	2	9	0.38	1,954.80	685.20	2,640.00	74.05
Zscore	Partition	3	14	0.76	1,655.14	984.86	2,640.00	62.69
Zscore	Partition	4	15	0.98	1,356.96	1,283.04	2,640.00	51.40
Zscore	Partition	5	17	1.34	1,241.49	1,398.51	2,640.00	47.03
Zscore	Partition	6	19	1.72	1,155.30	1,484.70	2,640.00	43.76
Zscore	Partition	7	28	2.89	909.29	1,730.71	2,640.00	34.44
Zscore	Partition	8	18	2.09	1,039.80	1,600.20	2,640.00	39.39
Zscore	Partition	9	27	3.45	692.11	1,947.89	2,640.00	26.22
Zscore	Partition	10	28	3.93	896.53	1,743.47	2,640.00	33.96
Zscore	Partition	11	33	5.01	858.33	1,781.67	2,640.00	32.51
Zscore	Partition	12	28	4.65	653.38	1,986.62	2,640.00	24.75
Zscore	Partition	13	28	5.00	654.59	1,985.41	2,640.00	24.79
Zscore	Partition	14	27	5.20	613.09	2,026.91	2,640.00	23.22
Zscore	Partition	15	37	7.50	594.75	2,045.25	2,640.00	22.53
Zscore	Partition	16	33	7.09	565.00	2,075.00	2,640.00	21.40
Zscore	Partition	17	23	5.26	629.94	2,010.06	2,640.00	23.86
Zscore	Partition	18	44	10.53	550.92	2,089.08	2,640.00	20.87
Zscore	Partition	19	31	7.83	535.55	2,104.45	2,640.00	20.29
Zscore	Partition	20	34	9.02	556.58	2,083.42	2,640.00	21.08
Zscore	Partition	21	26	7.70	423.17	2,216.83	2,640.00	16.03
Zscore	Partition	22	38	11.65	401.42	2,238.58	2,640.00	15.21
Zscore	Partition	23	22	7.16	460.58	2,179.42	2,640.00	17.45
Zscore	Partition	24	37	11.67	453.03	2,186.97	2,640.00	17.16
Zscore	Partition	25	39	12.77	444.94	2,195.06	2,640.00	16.85
Zscore	Partition	26	24	8.76	373.24	2,266.76	2,640.00	14.14
Zscore	Partition	27	29	10.86	366.47	2,273.53	2,640.00	13.88
Zscore	Partition	28	22	8.65	362.81	2,277.19	2,640.00	13.74
Zscore	Partition	29	24	9.71	429.22	2,210.78	2,640.00	16.26
Zscore	Partition	30	24	10.05	351.79	2,288.21	2,640.00	13.33
Zscore	Partition	31	25	10.80	350.21	2,289.79	2,640.00	13.27
Zscore	Partition	32	33	17.02	348.65	2,291.35	2,640.00	13.21

Norm.method	Init.centroids	k-cluster	#iteration	Time	wsse	bsse	sse	%wsse/sse
Zscore	Partition	33	32	15.25	344.25	2,295.75	2,640.00	13.04
Zscore	Partition	34	18	9.43	352.73	2,287.27	2,640.00	13.36
Zscore	Partition	35	23	12.36	340.89	2,299.11	2,640.00	12.91
Zscore	Partition	36	22	11.29	341.07	2,298.93	2,640.00	12.92
Zscore	Partition	37	21	10.91	484.27	2,155.73	2,640.00	18.34
Zscore	Partition	38	23	12.29	481.00	2,159.00	2,640.00	18.22
Zscore	Partition	39	10	5.75	489.82	2,150.18	2,640.00	18.55
Zscore	Partition	40	17	9.61	332.29	2,307.71	2,640.00	12.59
Zscore	Partition	41	18	10.34	325.17	2,314.83	2,640.00	12.32
Zscore	Partition	42	15	8.90	320.63	2,319.37	2,640.00	12.15
Zscore	Partition	43	16	9.69	319.35	2,320.65	2,640.00	12.10
Zscore	Partition	44	13	8.45	303.89	2,336.11	2,640.00	11.51
Zscore	Partition	45	23	15.75	299.71	2,340.29	2,640.00	11.35
Zscore	Partition	46	13	8.68	334.04	2,305.96	2,640.00	12.65
Zscore	Partition	47	13	8.70	310.82	2,329.18	2,640.00	11.77
Zscore	Partition	48	13	9.98	296.66	2,343.34	2,640.00	11.24
Zscore	Partition	49	13	9.44	295.44	2,344.56	2,640.00	11.19
Zscore	Partition	50	13	9.94	274.94	2,365.06	2,640.00	10.41

FIGURE 29: DIFFERENCE OF COMBINATIONS

	MinMax Compare (Parition vs Actual)		Zscore (Partition vs Actual)		(Zscore vs MinMax) Actual		(Zscore vs MinMax) Partition		(Zscore + Partition) vs (MinMax + Actual)	
k	#Iter	%WSS/ SSE	#Iter	%WSS/ SSE	Iter	%WSS /SSE	Iter	%WSS /SSE	Iter	%WSS /SSE
1	(1)	0.00	(1)	0.00	0	0.00	0	0.00	(1)	0.00
2	(1)	0.00	(4)	(0.05)	(2)	4.90	(5)	4.85	(6)	4.85
3	(1)	(1.57)	(5)	1.31	3	3.86	(1)	6.73	(2)	5.17
4	10	(3.65)	(6)	0.16	13	4.40	(3)	8.20	7	4.55
5	8	3.47	(1)	6.77	3	3.66	(6)	6.96	2	10.44
6	30	(0.63)	2	6.25	1	4.81	(27)	11.68	3	11.06
7	5	(0.78)	15	1.59	(6)	2.56	4	4.94	9	4.16
8	13	(1.04)	2	9.38	2	1.27	(9)	11.70	4	10.65
9	8	(0.19)	8	(1.57)	(2)	2.63	(2)	1.24	6	1.05
10	25	0.43	7	7.65	2	2.43	(16)	9.64	9	10.07
11	14	0.93	5	10.19	9	(0.34)	0	8.92	14	9.85
12	(2)	0.33	(12)	4.05	14	(1.37)	4	2.35	2	2.68
13	2	1.54	4	5.01	(2)	(0.37)	0	3.10	2	4.64
14	0	(0.39)	2	4.54	(9)	(1.16)	(7)	3.77	(7)	3.38
15	9	(0.08)	10	4.41	(5)	(0.62)	(4)	3.87	5	3.79
16	0	1.84	15	3.76	(9)	(0.70)	6	1.22	6	3.06
17	(1)	1.74	5	6.81	(8)	(0.98)	(2)	4.09	(3)	5.83
18	19	(3.12)	20	4.69	0	(2.45)	1	5.36	20	2.24
19	(8)	(0.84)	(22)	4.40	15	(1.81)	1	3.43	(7)	2.59
20	0	(0.58)	11	4.98	(7)	(1.22)	4	4.35	4	3.76
21	0	0.00	0	0.00	(5)	(1.00)	(5)	(1.00)	(5)	(1.00)
22	0	0.00	0	0.00	9	0.43	9	0.43	9	0.43
23	21	1.91	0	0.00	(5)	3.04	(26)	1.13	(5)	3.04
24	12	0.65	12	2.54	(4)	(2.33)	(4)	(0.45)	8	0.21
25	9	2.29	19	2.55	0	(1.32)	10	(1.06)	19	1.23
26	13	2.65	0	0.00	1	(1.08)	(12)	(3.73)	1	(1.08)
27	30	2.15	0	0.00	15	(1.24)	(15)	(3.39)	15	(1.24)
28	24	(0.50)	0	0.00	5	(1.24)	(19)	(0.74)	5	(1.24)
29	(9)	0.74	0	0.00	(9)	1.32	0	0.58	(9)	1.32
30	15	(0.96)	0	0.00	2	(2.01)	(13)	(1.05)	2	(2.01)
31	23	(0.66)	0	0.00	4	(1.89)	(19)	(1.23)	4	(1.89)
32	29	1.18	0	0.00	12	(1.93)	(17)	(3.11)	12	(1.93)
33	6	1.41	0	0.00	10	(2.12)	4	(3.53)	10	(2.12)
34	0	0.00	0	0.00	(4)	(1.76)	(4)	(1.76)	(4)	(1.76)
35	0	0.00	0	0.00	2	(2.10)	2	(2.10)	2	(2.10)
36	0	0.00	0	0.00	3	(1.82)	3	(1.82)	3	(1.82)
37	0	0.00	0	0.00	0	3.64	0	3.64	0	3.64
38	0	0.00	0	0.00	0	3.90	0	3.90	0	3.90
39	0	(0.00)	0	0.00	(11)	4.30	(11)	4.30	(11)	4.30
40	0	0.00	0	0.00	(4)	(1.69)	(4)	(1.69)	(4)	(1.69)
41	0	0.00	0	0.00	(3)	(1.90)	(3)	(1.90)	(3)	(1.90)
42	0	0.00	0	0.00	(6)	(2.03)	(6)	(2.03)	(6)	(2.03)

	MinMax Compare (Parition vs Actual)		Zscore (Partition vs Actual)		(Zscore vs MinMax) Actual		(Zscore vs MinMax) Partition		(Zscore + Partition) vs (MinMax + Actual)	
k	#Iter	%WSS/ SSE	#Iter	%WSS/ SSE	Iter	%WSS /SSE	Iter	%WSS /SSE	Iter	%WSS /SSE
43	0	0.00	0	0.00	1	(2.07)	1	(2.07)	1	(2.07)
44	0	0.00	0	0.00	0	(2.52)	0	(2.52)	0	(2.52)
45	0	0.00	0	0.00	10	(2.64)	10	(2.64)	10	(2.64)
46	0	0.00	0	0.00	1	(1.17)	1	(1.17)	1	(1.17)
47	0	0.00	0	0.00	0	(1.93)	0	(1.93)	0	(1.93)
48	0	0.00	0	0.00	4	0.16	4	0.16	4	0.16
49	0	0.00	0	0.00	4	0.17	4	0.17	4	0.17
50	0	0.00	13	10.41	4	(0.08)	13	10.41	13	10.41

FIGURE 30: SUMMARY OF SIMILAR NORMALIZATION METHOD IN 2 INITIAL CENTROIDS APPROACHES

	Absolute values				Percentage			
Norm. Method	Partition better	Equal	Actual better	Total	Partition better	Equal	Actual better	Total
MinMax	15	19	16	50	30%	38%	32%	100%
Zscore	2	28	**20**	50	4%	56%	**40%**	100%

FIGURE 31: SUMMARY OF SIMILAR INITIAL CENTROIDS IN 2 NORMALIZATION APPROACHES

	Absolute values				Percentage			
Initial centroids	MinMax better	Equal	Zscore better	Total	MinMax better	Equal	Zscore better	Total
Actual	17	1	32	50	34%	2%	64%	100%
Partition	28	1	21	50	56%	2%	42%	100%

10. Appendix 2

FIGURE 32: CORRELATION FOR ALL OBSERVATIONS

All customers data	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Fresh	1.00					
Milk	0.10	1.00				
Grocery	(0.01)	0.73	1.00			
Frozen	0.35	0.12	(0.04)	1.00		
Detergents_Paper	(0.10)	0.66	0.92	(0.13)	1.00	
Delicassen	0.24	0.41	0.21	0.39	0.07	1.00

FIGURE 33: CORRELATION FOR OBSERVATIONS IN SEGMENT0

Seg0	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Fresh	1.00					
Milk	0.85	1.00				
Grocery	0.98	0.94	1.00			
Frozen	0.41	(0.14)	0.21	1.00		
Detergents_Paper	0.48	(0.06)	0.28	1.00	1.00	
Delicassen	0.67	0.96	0.81	(0.40)	(0.33)	1.00

FIGURE 34: CORRELATION FOR OBSERVATION IN SEGMENT1

Seg1	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Fresh	1.00					
Milk	0.63	1.00				
Grocery	0.29	(0.01)	1.00			
Frozen	0.44	0.14	(0.26)	1.00		
Detergents_Paper	0.10	(0.15)	0.95	(0.46)	1.00	
Delicassen	0.34	0.08	(0.04)	0.83	(0.21)	1.00

FIGURE 35: CORRELATION FOR OBSERVATION IN SEGMENT2

Seg2	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Fresh	1.00					
Milk	0.35	1.00				
Grocery	0.29	0.67	1.00			
Frozen	0.39	0.31	0.21	1.00		
Detergents_Paper	0.32	0.47	0.55	0.08	1.00	
Delicassen	0.24	0.33	0.31	0.26	0.34	1.00

FIGURE 36: CORRELATION FOR OBSERVATION IN SEGMENT3

Seg3	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Fresh	1.00					
Milk	(0.09)	1.00				
Grocery	(0.02)	0.47	1.00			
Frozen	0.14	(0.03)	(0.11)	1.00		
Detergents_Paper	(0.08)	0.42	0.67	(0.17)	1.00	
Delicassen	0.13	0.39	0.24	0.09	0.13	1.00

FIGURE 37: CORRELATION FOR OBSERVATION IN SEGMENT4

Seg4	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Fresh	1.00					
Milk	0.18	1.00				
Grocery	0.10	0.34	1.00			
Frozen	0.31	0.31	0.04	1.00		
Detergents_Paper	(0.06)	0.07	0.61	(0.18)	1.00	
Delicassen	0.42	0.36	0.12	0.40	(0.21)	1.00