Indian Institute of Technology, Roorkee

A mini project on

"BUS STATION MANAGEMENT"

Under the subject of

DATA MINING AND WAREHOUSING

Submitted by: Group 5

Abhishek S Adhikari	**20535002**
Anunay Katare	**20535005**
Ayush Kasara	**20535009**
Devyanshu Sengal	**20535012**
Shweta	**20535027**

Contribution from Members

Enrollment Number	Contribution	Page No.
20535005	Bus delay prediction implementation	5-6
20535027	Writing code for combining data from various files and organizing them. Analyzing Bus delay prediction result via graph	4 7
20535002	Clustering Based Analysis on Bus Transit System	8-10
20535012	Mining Association Rules with Apriori Algorithm	10-11
20535009	Implementation of Apriori Algorithm and exploring business utility of our result.	12-13

Problem Statement

Bus Stations Management is required to explore possible areas where data mining techniques could be applied on collected data in order to gain some insights into the data and expects to find some patterns which might be of use to the Bus Stations Authorities across cities or even states.

Introduction to the topic

Data mining and warehousing can be used to predict many patterns and values which can then be used to improve one's business. Here we try to do the same for Bus Stations Management.
There are several aspects on which a transit system authority may utilize data mining approaches based on the historical data collected from time to time. First, bus delay prediction is one of the areas where bus transportation authorities may utilize past data to predict bus delay on various bus stops along the route, this can be beneficial for even the commuters, as they can be provided with the possible arrival timings of buses. Secondly, some data mining techniques can help bus authorities to re-organise their bus routes, in order to serve the areas of the cities where more passengers are likely to wait, considering the spatial attributes of those bus stops as well. This will help bus station authorities to direct their buses to gather more and more numbers of passengers and gain more profit. Lastly, finding frequently occurring stops along all the routes may give bus station authorities the opportunity to lend buses for advertisements to advertisement companies which will actually benefit the bus authority financially.

The data collection on Bus Transit System is still in an infant stage, but we manage to somehow find datasets on transportation systems of cities like Delhi and Bangalore, where some data has been collected as an initiative towards Digital India.

We tried to utilize algorithms like apriori for frequent bus stops, K-nearest neighbour for bus delay prediction and finally, kmeans and kmedoids clustering algorithms to map each bus stop in a certain cluster that may be of use to bus station authorities.

Combining Data from various Files into one Python dataframe

The data was initially scattered into different files. To process the algorithm on different files would not have been possible. So, combining data from different files into one file was a very important and crucial step. I combined data from various files into a common python dataframe so that we can run our algorithms on that.
The code used for this purpose was as follows:

```
import pandas as pd
import numpy as np
from datetime import datetime
```

```
import glob
glob.glob("/home/papi/Desktop/DWM_Project/stops*.xlsx")
all_data = pd.DataFrame()
for f in glob.glob("/home/papi/Desktop/DWM_Project/stops*.xlsx"):
    df = pd.read_excel(f)
    all_data = all_data.append(df,ignore_index=True)
df=all_data
```

```
df.describe()
```

	Unnamed: 0	c+label	diff_of_arrival_time	stop_id	stop_sequence
count	48979.000000	48979.000000	48979.000000	48979.000000	48979.000000
mean	3498.000000	1.484269	1.649573	1718.887237	18.214949
std	2019.880516	0.998147	7.339901	1018.702233	12.950443
min	0.000000	0.000000	-15.566667	47.000000	0.000000
25%	1749.000000	1.000000	-3.333333	707.000000	7.000000
50%	3498.000000	2.000000	-0.033333	1736.000000	16.000000
75%	5247.000000	2.000000	7.016667	2629.000000	29.000000
max	6996.000000	3.000000	15.566667	4393.000000	43.000000

48959	6977	15:36:17	1	15:36:17	14.750000	15:21:32	1368	36	0_13_50	su
48960	6978	15:39:58	1	15:39:58	14.750000	15:25:13	2573	37	0_13_50	su
48961	6979	15:41:55	1	15:41:55	14.750000	15:27:10	422	38	0_13_50	su
48962	6980	15:43:40	1	15:43:40	14.750000	15:28:55	2837	39	0_13_50	su
48963	6981	15:44:55	1	15:44:55	14.750000	15:30:10	47	40	0_13_50	su
48964	6982	15:45:34	1	15:45:34	14.750000	15:30:49	1886	41	0_13_50	su
48965	6983	15:47:30	1	15:47:30	14.750000	15:32:45	952	42	0_13_50	su
48966	6984	15:50:21	1	15:50:21	14.750000	15:35:36	2968	43	0_13_50	su
48967	6985	13:04:00	0	13:04:00	4.000000	13:00:00	1763	0	1_13_0	su
48968	6986	13:10:42	1	13:10:42	8.016667	13:02:41	3094	1	1_13_0	su
48969	6987	13:17:00	1	13:17:00	13.000000	13:04:00	2316	2	1_13_0	su
48970	6988	13:18:54	1	13:18:54	13.016667	13:05:53	681	3	1_13_0	su
48971	6989	13:22:58	1	13:22:58	14.016667	13:08:57	1517	4	1_13_0	su
48972	6990	13:25:12	1	13:25:12	14.016667	13:11:11	2028	5	1_13_0	su
48973	6991	13:27:59	1	13:27:59	14.016667	13:13:58	487	6	1_13_0	su
48974	6992	13:30:44	1	13:30:44	14.016667	13:16:43	2541	7	1_13_0	su
48975	6993	13:32:20	1	13:32:20	14.016667	13:18:19	2036	8	1_13_0	su
48976	6994	13:35:05	1	13:35:05	14.016667	13:21:04	2945	9	1_13_0	su
48977	6995	13:38:20	1	13:38:20	14.016667	13:24:19	1022	10	1_13_0	su
48978	6996	13:39:42	1	13:39:42	14.016667	13:25:41	2231	11	1_13_0	su

48979 rows × 10 columns

The above image shows total data rows and columns after combining the data files.

Implemented a model that would predict the amount of delay that would happen for a bus to arrive at a particular stop.

For this we first of all need Historical Data for the same so we have found a dataset that tells when a bus arrives at a particular stop its scheduled arrival time along with the day and its departure time. The historical data was in seven different excel sheets, one for each day of week. So, had to add an attribute day so that the machine can understand which data belongs where.

To predict the delay we observed that for a particular day of a week delays can be the same. Like if we consider Friday evening it has more delay and correctly so as more people get out of their house at that time.
Hence we use the day of the week and the time of day as attributes for classification using KNN, as we want to classify delays in four categories which are delayed and early arrival of the bus by upto 5 or more than 5 minutes. The historical data is divided into classes based on the difference between scheduled arrival time and the actual arrival time.

For this, first of all we need to perform data transformation. This is because attributes like “day” having values -”m, t, w, th, f, sa, su” can not be understood by the machine so we need to convert it into numbers hence we have converted days into -”1, 2, 3, 4, 5, 6, 0’ respectively.

actual_arrival_time	c+label	departure_time	diff_of_arrival_time	scheduled_arrival_time	stop_id	stop_sequence	trip_id	day
07:57:55	2	07:57:55	-2.083333	08:00:00	1763	0	3_8_0	m
07:59:56	2	07:59:56	-2.750000	08:02:41	3094	1	3_8_0	m
07:56:47	3	07:56:47	-7.216667	08:04:00	2316	2	3_8_0	m
08:02:34	2	08:02:34	-3.316667	08:05:53	681	3	3_8_0	m
08:06:31	2	08:06:31	-2.433333	08:08:57	1517	4	3_8_0	m
...	...	...	...	...	...	...	...	...
13:30:44	1	13:30:44	14.016667	13:16:43	2541	7	1_13_0	su
13:32:20	1	13:32:20	14.016667	13:18:19	2036	8	1_13_0	su
13:35:05	1	13:35:05	14.016667	13:21:04	2945	9	1_13_0	su
13:38:20	1	13:38:20	14.016667	13:24:19	1022	10	1_13_0	su
13:39:42	1	13:39:42	14.016667	13:25:41	2231	11	1_13_0	su

Image shows dataset before conversion of day

After conversion:

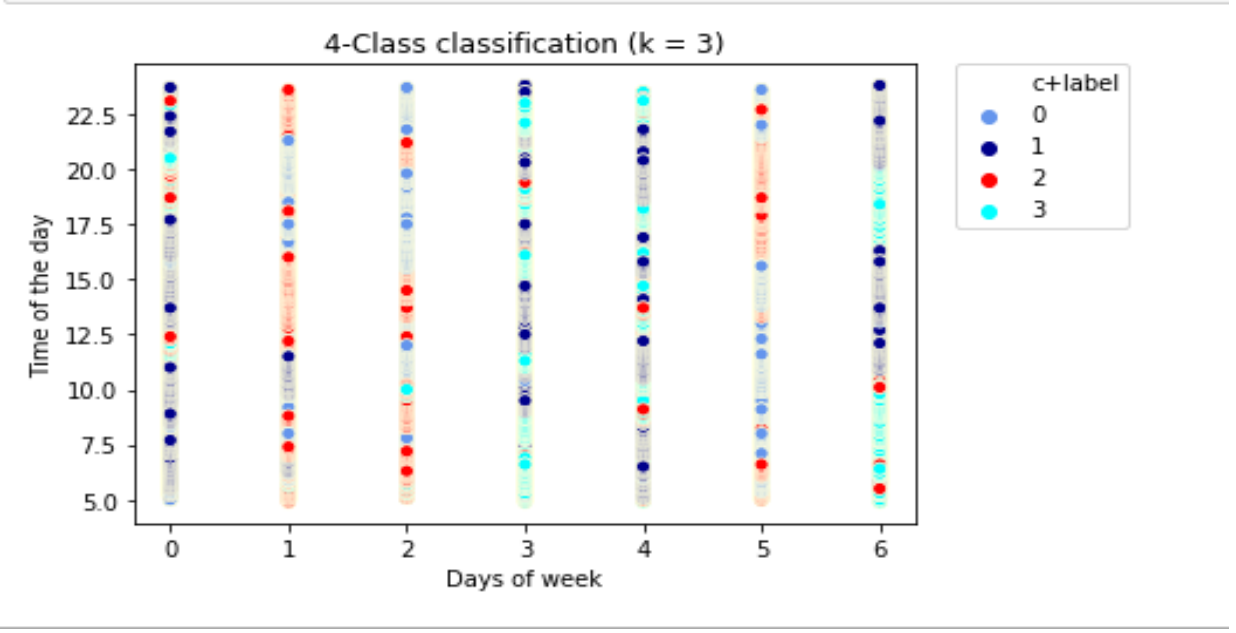
actual_arrival_time	c+label	departure_time	diff_of_arrival_time	scheduled_arrival_time	stop_id	stop_sequence	trip_id	day
07:57:55	2	07:57:55	-2.083333	08:00:00	1763	0	3_8_0	0
07:59:56	2	07:59:56	-2.750000	08:02:41	3094	1	3_8_0	0
07:56:47	3	07:56:47	-7.216667	08:04:00	2316	2	3_8_0	0
08:02:34	2	08:02:34	-3.316667	08:05:53	681	3	3_8_0	0
08:06:31	2	08:06:31	-2.433333	08:08:57	1517	4	3_8_0	0

After this we convert the actual_ arrival_time attribute into a floating point value for this we take the hour component from the string and the minute component then divide the minute component by 60 and add the result with hour component and store the value in the new column arrival_time as shown.

actual_arrival_time	c+label	departure_time	diff_of_arrival_time	scheduled_arrival_time	stop_id	stop_sequence	trip_id	day	arrival_time
07:57:55	2	07:57:55	-2.083333	08:00:00	1763	0	3_8_0	0	7.9
07:59:56	2	07:59:56	-2.750000	08:02:41	3094	1	3_8_0	0	8.0
07:56:47	3	07:56:47	-7.216667	08:04:00	2316	2	3_8_0	0	7.9
08:02:34	2	08:02:34	-3.316667	08:05:53	681	3	3_8_0	0	8.0
08:06:31	2	08:06:31	-2.433333	08:08:57	1517	4	3_8_0	0	8.1
...	...	...	...	...	...	...	...	...	...
13:30:44	1	13:30:44	14.016667	13:16:43	2541	7	1_13_0	6	13.5
13:32:20	1	13:32:20	14.016667	13:18:19	2036	8	1_13_0	6	13.5
13:35:05	1	13:35:05	14.016667	13:21:04	2945	9	1_13_0	6	13.6
13:38:20	1	13:38:20	14.016667	13:24:19	1022	10	1_13_0	6	13.6
13:39:42	1	13:39:42	14.016667	13:25:41	2231	11	1_13_0	6	13.7

Atlast we take the two columns arrival_time and day as features to put into knn and c+label for class. We use sklearn library for knn. At last we take as input the day and time and predict the delay using it. The program is made in python.

Analyzing Bus delay Prediction result via graph



- The above graph shows how we classify our data using KNN algorithm and classifying it into 4 classes.
- The four classes are as follows:
    - Class 0: bus can be upto 5 min late.
    - Class 1: bus can be more than 5 min late.
    - Class 2: bus can be upto 5 min early.
    - Class 3: bus can be more than 5 min early.
- **Analysis:**
    - Buses are getting delay more than 5 min mostly on monday, Friday and sunday.
    - The obvious reasons can be more traffic, more candidate passengers.
    - More passengers also means the bus may need to stop on multiple stops to let passengers come in and go out of the bus.
    - This inturn increases the traffic and hence delay.
    - Buses are able to arrive on time or before time, by max 5 min, mostly on tuesday and wednesday.
- **Management:**
    - By predicting the bus delay on a particular timing of the particular day of the week we help passengers to utilize their time efficiently and reduce unnecessary chaos on Bus station.
    - For the passengers, who can postpone their work by a day or two, we advise them to go out on tuesdays, wednesdays and saturdays to avoid bus delays and traffic.
    - The more traffic on Mondays and Fridays is also an indication of more private vehicles. So an advisory for people to take public transport if possible.

Clustering Based Analysis on Bus Transit System

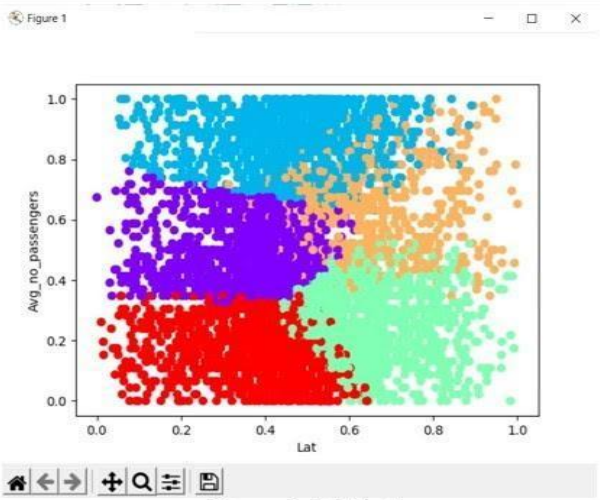
Clustering is one type of data mining methods. Clustering is the process of making a group of abstract objects into classes of similar objects. Similar objects are grouped in one cluster and dissimilar objects are grouped in another cluster.

I worked on Clustering bus stops on every route of transit-oriented public transportation routes in order to meet the customer demand and suit the planned routes with the actual condition in its operation. Using K-Means and K-Medoids clustering algorithms, I found that the optimal cluster is 5 clusters with several bus stops contained in them for the Bangalore Bus Transit System. The Elbow method was used to determine the optimal number of clusters. As we increase the number of clusters, the error would decrease and when the number of clusters are equal to the number of datapoints the error will be 0 and that we do not want.

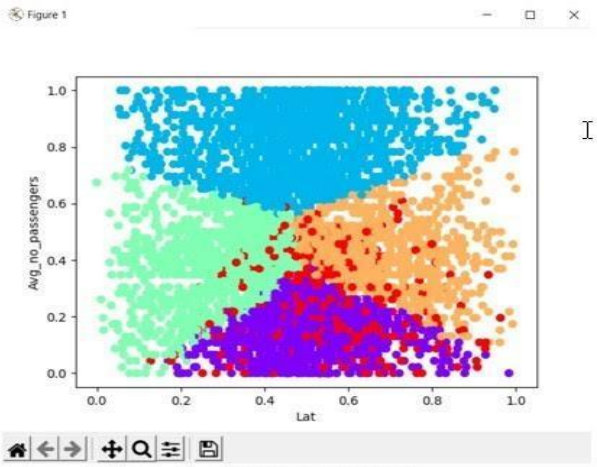
The dataset was processed to get the latitude-longitude for each bus station and their corresponding average numbers of passengers waiting at different bus stations. This file was loaded into a pickle file with proper data structure for easy access. Then, these features were normalized with the help of min-max normalization, so that they all have equal effect on the clustering result, otherwise one feature may dominate the result of analysis. The Elbow method is used to determine the optimal number of clusters in this. By Elbow Method, it is found that the optimal number of clusters is 5 clusters because there is a significant inflection point and change in slope at that point in the graph of Elbow Method.

Finally, the prepared dataset was fed to KMeans and KMedoids algorithm & results were noted. KMeans attempts to minimize the total squared error, while k-medoids minimizes the sum of dissimilarities between points labelled to be in a cluster and a point designated as the center of that cluster. Results also show that KMedoid's sum of squares comes out to be much larger as compared to that of KMeans but clustering was much more balanced in KMedoids method as it is much more robust to outliers.

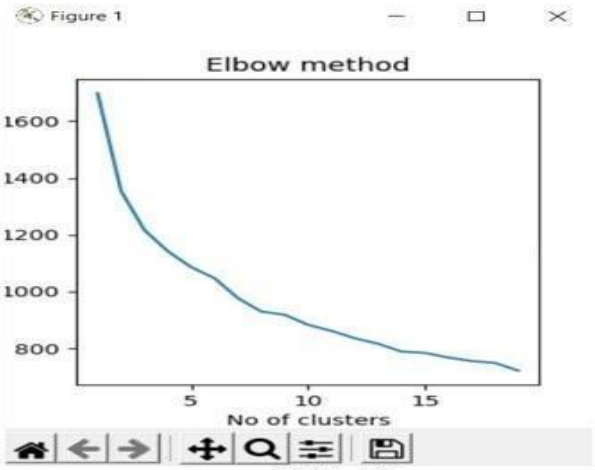
Bus stations authority may utilize this kind of analysis to know the potential passengers at each bus stop and may re-evaluate different routes to meet the customer demands and load balance at each bus stop. Using the result of KMedoids, bus station authorities may send more buses towards routes falling in the 2nd cluster while reducing buses from 5th cluster routes.



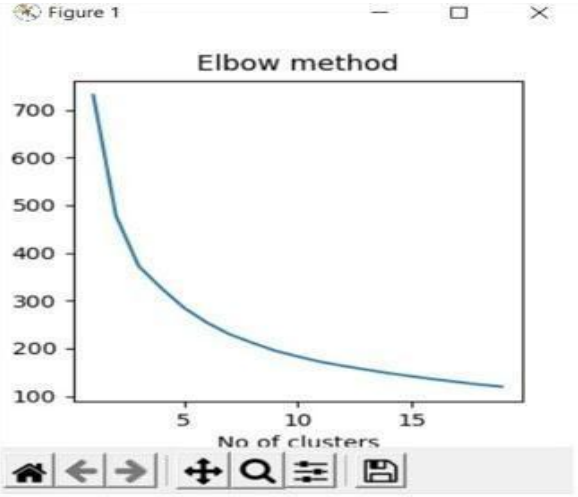
K-medoid 5-cluster



K-means 5-cluster



K-Medoids



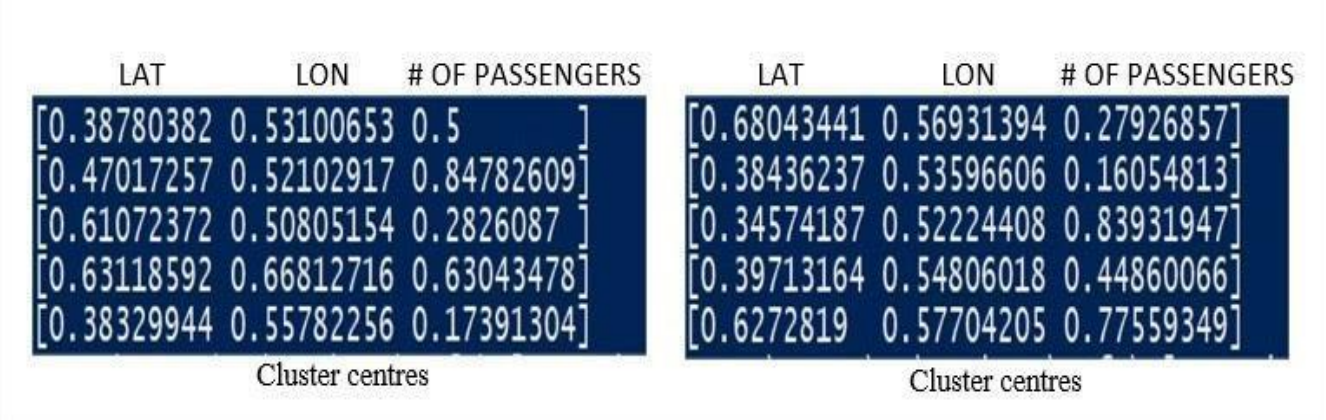
K-Means

```
No. of iterations:  10
Sum squared of error:  1084.3491897353906
cluster  4 :  1295
cluster  1 :  1040
cluster  2 :  1276
cluster  0 :  1132
cluster  3 :  609
```

Instances for each cluster (KMedoids)

```
No. of iterations:  23
Sum squared of error:  257.07297630892714
cluster  1 :  1285
cluster  2 :  736
cluster  3 :  1365
cluster  4 :  814
cluster  0 :  1152
```

Instances for each cluster (KMeans)



Mining Association Rules with Apriori Algorithm

We started the project with some literature review and landed our eyes on the paper "**Fast Algorithms for Mining Association Rules"** authored by Rakesh Agarwal and Ramakrishnan Srikant. The paper introduces us to algorithms viz. *Apriori* and *AprioriTid*. Both of these algorithms use a function named apriori-gen to determine the candidate itemsets before the first pass begins. Empirical evaluation shows that these algorithms outperform the known algorithms by factors ranging from three for small problems to more than an order of magnitude for large problems.

The paper suggests that it is not necessary to use the same algorithm in all the passes over data. We can use a mix of these two algorithms which the authors call ***AprioriHybrid***. In the earlier passes *Apriori* does better than *AprioriTid*. However in the later passes *AprioriTid* outperforms the *Apriori*. Scale-up experiments show that ***AprioriHybrid*** scales linearly with the number of transactions. ***AprioriHybrid*** also has excellent scale-up properties with respect to the transaction size and the number of items in the database.

We tried to implement the *Apriori* algorithm on our problem set and gather some desired outputs, but we started with noisy and inconsistent data which posed some serious problems ahead of us:

- The data was filled with a significant amount of empty cells. For example, the file stops_details.csv had some stop_id which did not refer to any stations and vice-versa there were stations which had no stop_ids. This was probably due to mismanagement of the data.
- The data was also invaluable (by invaluable I mean that it was deemed unreliable or of no use to us for our analysis) and inconsistent. For instance, the file bmtc_dump_order.csv had the same stop orders for

two different stations of origin and destination which was not expected and other details like route number and distance were of no use to us.

We also encountered a binary file named transactions which apparently mapped the stop_ids with the stop_orders from the files stop_details.csv and bmtc_dump_order.csv respectively. But it was of no use to us and we also wasted some time on it.

We solved this problem with data, simply by picking the data points of our interest and gathering them into a single file named bus_data.csv.

The *Apriori* algorithm we have used, has the following simple functions at its heart:

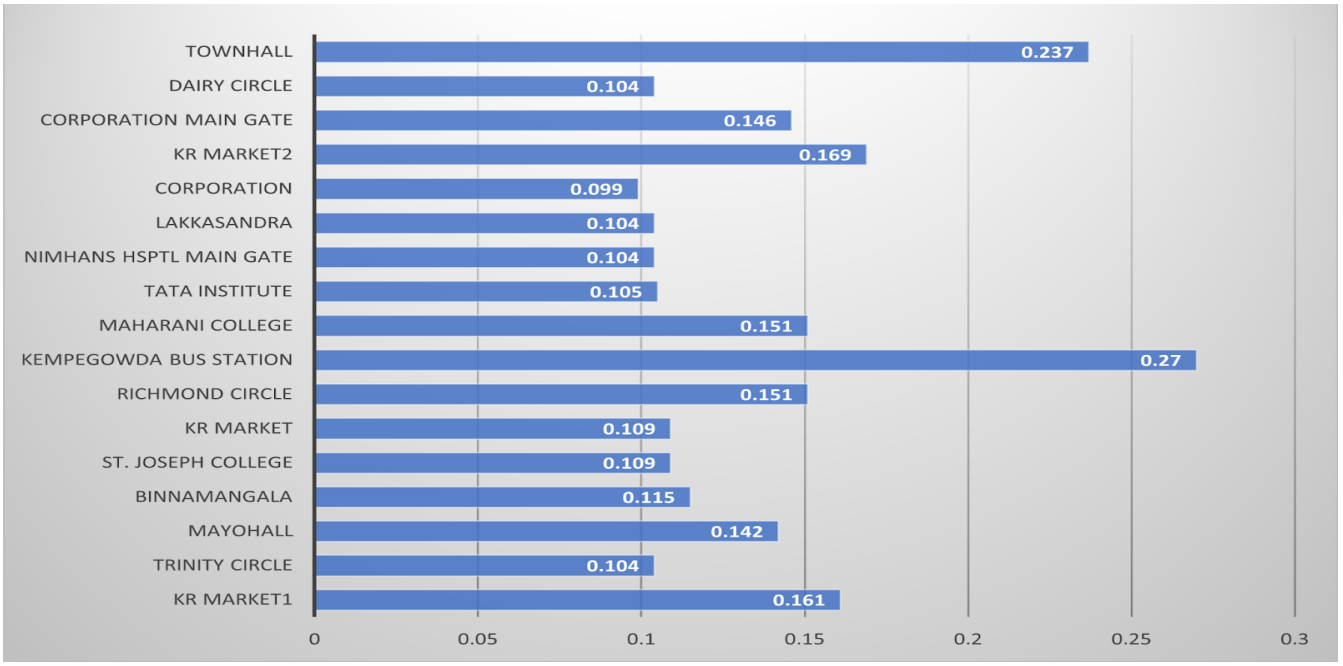
```
def runApriori(data_iter, minSupport, minConfidence):
    itemSet, transactionList = getItemSetTransactionList(data_iter)

    freqSet = defaultdict(int)
    largeSet = dict()
    assocRules = dict()

    oneCSet = returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet)

    currentLSet = oneCSet
    k = 2
    while currentLSet != set([]):
        largeSet[k - 1] = currentLSet
        currentLSet = joinSet(currentLSet, k)
        currentCSet = returnItemsWithMinSupport(
            currentLSet, transactionList, minSupport, freqSet
        )
        currentLSet = currentCSet
        k = k + 1
```

Implementation of Apriori Algorithm and exploring business utility of our result

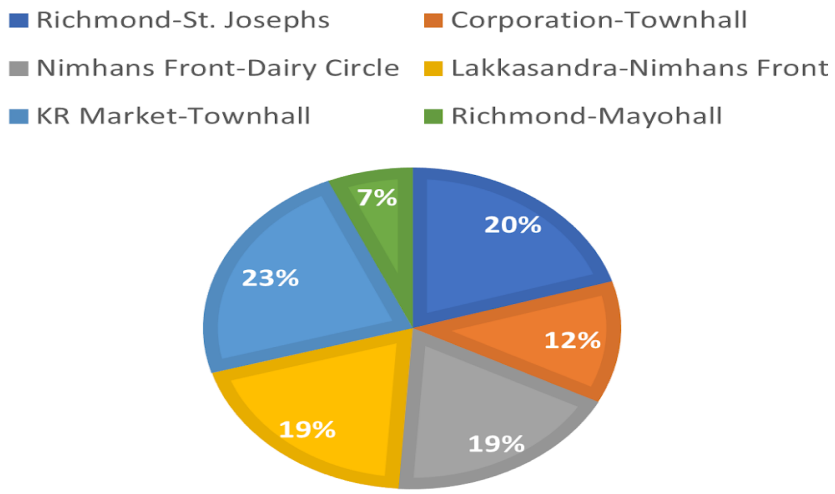
The dataset named bmtc_dump_order.csv was our main data set. It has a column named stops_order of our interest. As a Preprocessing step for further analysis, I extract that column and save it to the file named bus_data.csv. It served as my main dataset. Then I use it for analysis using the Apriori Algorithm. I try to implement the Apriori Algorithm as explained in the paper named "Fast Algorithms for Mining Association Rule" by Rakesh Agarwal and Ramakrishnan Srikant. This paper divides the problem into two subproblems:

- Find all itemsets that have transaction support value above minimum support value. These itemsets are called Large Itemsets. The others are called Small Itemsets.
- Use the large itemsets to generate the desired rules. Here is a straightforward algorithm for this task. For every large itemset *l*, find all non-empty subsets of *l*. For every such subset *a*, output a rule of the form *a* => (*l* - *a*) if the ratio of support(*l*) to support(*a*) is at least equal to the value of minimum confidence value. We need to consider all subsets of *l* to generate rules with multiple consequents.

So, we generate all Large Itemsets and output them along with their support count value. We output strong association rules using the strategy explained above along with their confidence value. The output was in stop numbers which was mapped with the corresponding stop name from the file named stops_detail.csv.
Using the result, my objective was to find the utility of this result as a business model. So I performed various analyses like finding the bus stops mostly frequent in terms of traffic. We can find it by having the count values of stations in large itemsets and one with a high support value is most busiest.



Using this it is clear that Kempegowda Bus Station deals with most traffic. This is also helpful in analysing what resources a bus station needs in order to service properly with respect to large masses. Another one is to find frequently used routes.



This pie chart shows that in a set of routes in a specific region, route from KR Market to Townhall is most frequently used and Richmond to St. Josephs College is second most frequent. This is important as we can increase or decrease the number of buses for those routes depending on the amount of people travelling in that particular route. We can also use this opportunity to lend our buses for advertisements to advertisement companies which will actually benefit the bus authority financially.

Zhang, Wang, Xiong in their paper “An Automatic Approach for Transit Advertising in Public Transportation Systems” discussed this idea.

This most frequent routes analysis is also important in exploiting trip patterns of passengers. There are various analyses like bus traffic management, re-routing analysis and so on that could be used to make a good revenue generation from it.