Sprint 2: Review

Project Group 8: Yao-Fu Yang, Qiushi Li, and Rong Chang
Trello board: https://trello.com/b/61ULBOVX/grocery-store-data-support
Git: https://github.ccs.neu.edu/yangyaof/cs5500_sum2020_group8

User Stories and Delivery

Definition of User Stories

At this stage, our store manager wants to know how the factors such as holiday, discount hours, lunch/dinner rush hours, and weather may change patterns of customers shopping, so that he can plan the operations more efficiently. The user stories were then defined as below.

The store manager needs to:

1. ~ know **the traffic of customers** in **the top 3 hours of a day** over a week, and in addition to answer the questions such as,
    - which day has the most shoppers and which day has the fewest shoppers;
    - how this traffic may change on the different days of week;
    - how this traffic may change due to the weather, holiday, and discount event.
2. ~ understand how the **weather** can affect the shopping volume and entry time, by exploring
    - ~ whether a bad weather on a Saturday may result in less shoppers than a Tuesday with a special senior discount event or even a normal weekday?
3. ~ observe how much **the senior discount event** may affect the traffic of customers and shopping duration, for example, with the senior shopping hours moved to Wednesday 1-3pm, how the customers will change on Tuesday and Wednesday?
4. ~ observe how the traffic of customers and shopping duration change if the dinner **rush hours** shift to different times in a day, such as 4-5 pm.

In addition, our store manager also wants the data to be stored into a suitable **database** to support more sophisticated data analysis.

Product Delivery

To satisfy the user requirements, we revised our techniques in generating the data. The generation of the product is partitioned into two phases.

(1) The first phase is to define the normal daily traffic and pattern over a week without the consideration of any special events and effects including those lunch and dinner peak

hours, senior discount hours, holiday (the holiday, day before holiday and week before holiday), and weather conditions.

(2) If the day is identified that involves any special events, the second phase is triggered to add a set of customer visit data of this identified special event.

The customer visit data of every special event is defined separately. For example, if a day is identified to be a weekday with lunch and dinner peak hours. The shopping traffic as well as the associated shopping patterns of lunch and dinner peak hours will be implemented. We defined lunch and dinner peak hours could bring 30% more customers for that weekday. Therefore, if lunch and dinner peak event is applied, a second phase will add 30% more customer (e.g., a Monday normally is assumed to have 620 customers, with 30% more is approximate 30% * 620 = 186) data with entry time strictly limited to lunch hour and dinner hour the user set up. In addition, the shopping durations of all these 30% more customers are strictly limited within 6-20 mins as described by the user. Considering another example on how weather applies to the data. Let's say it is a normal Saturday, then there are approximately 4000 customers in total. The data generation of this day follows the shopping entry time and duration distribution designed for a weekend. The random generation process is the same as what we did in last product delivery. If this day is documented to be a nice weather day, then we expect 40% more customers coming mostly from close to noon to early evening for a relatively short shopping experience, as described by the user. The design of shopping duration and entry time distribution of normal day and special events is documented in an excel spreadsheet titled as *visitPattern_v2.201* in Git.

In addition to these lower level logic and coding changes, we switched the dinner peak hours and senior discount hours to the new time slots as user designed. In the current product, we have dinner rush hour moved to 4 - 5 pm during the weekday, and senior discount hours to 1 - 3 pm on Wednesdays. This version also supports easier modification of the lunch/dinner rush hours, as well as the start and end time of the senior discount hours. With the specified date and time, the user is able to discover the traffic of the top 3 hours of a day and compare the differences between weekday and weekend. The user could also compare the first product and current one to find how shopping traffic changes on different senior shopping hours.

Functionality of Delivery

The basic features of this product allow the user to observe (1) the customer visiting distribution over a day and over a week, and (2) the length of shopping time at different times on different days.

Parameters

1. Visit ID: prefix letter identifying if the data is generated for any special events. "N" refers to those customers who are included as normal daily traffic.

2. Entry time: the date and time that a customer enters the store.
3. Leave time: the date and time that a customer leaves the store.
4. Shopping duration: shopping time (in minutes).
5. Day of the Week: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday.
6. Holiday type: Indicates any holiday modifiers that may change the daily customer volume. Values are set to be none, holiday, day before holiday, week before holiday.
7. Day description: Additional descriptions of the day of this visit. Values are set to be regular weekday, regular weekend, senior discount weekday, nice weather weekend.

In addition, with the new data generation logic, the current product allows the user to identify how the special events and conditions (i.e., holiday, discount hours, lunch/dinner rush hours, and very bad/nice weather) affect the shopping traffic by modifying/removing the events per analysis needs. Therefore, comparing the last version, this version of product has more features that allow the users to:

1. ~ explore the shopping traffic and duration in any specific hours, and compare the differences between days of week, and days with special events.
2. ~ modify/shift the event time or remove the event from a day and see the changes the event brings to the shopping traffic.
3. ~ define the event effects by analyzing the shopping patterns from the customers coming due to the events.

Lastly, per request, this product is now stored in MongoDB. We have generated two sets of data from May and June. It allows the query of data by specifying the time.

Code Quality

Our assessment of software quality is divided into two portions - verification and validation.

Verification: After completing the classes in accordance with our design, we iteratively compared each stage of completeness with the new project requirements in sprint2, our initial requirements in sprint0/sprint1, and other requirements that were obtained or clarified during the requirements gathering process and subsequent requirements verification meetings with our customer. At each stage, we also weighed the cost of our design choices in terms of extensibility, overhead, and time complexity.

Validation: For this assessment phase, we primarily split our tests into black-box tests (unit testing) and white-box tests (integrated testing). We focused on unit testing for most of our model classes such as Weather, Datetime, Visit, Day, and VisitParameters classes to ensure accurate behavior for the building blocks of our system. We then created white-box tests for our six static utility classes that operated mainly at the unit and integrated level to ensure that the

arithmetic and logical aspects of our system were working properly. At this time, we have not yet tested our software at a system level (e.g. the PilotSim class) but have been using the outputs from our software to determine correctness (csv file and external data sets on MongoDB).

In addition, we ran some statistical model analysis and were able to find out the top 3 hours of a sunny weekend are mostly between 2 to 7 pm, with a few varieties between different weeks. However, on a weekday we mostly have lunch and dinner meal hours to be the top 3 hours of a day, with the third top hour after the dinner peak hour. The fewest hours usually are early morning throughout a week. We also found if we changed the senior discount shopping hours from Tuesday morning to Wednesday afternoon, we may expect more shoppers on Wednesdays. Our analysis showed that our produced data would be able to provide sufficient information for the user to test and discover more shopping patterns even with some arrangements. In other words, this product is ensured to be able to meet the user needs and requirements.

We also assessed the code quality of our entire system using CodeMR and found that the majority of our code are within expected quality levels. We noted a medium-high level of coupling in the PilotSim class, which was expected due to its nature as the controller class and its need to access all of our model classes, most static utility classes, and the view class in order to deliver the output of our system. PilotSim also triggered low-medium levels of complexity and size which were within our expectations; we are aware that PilotSim currently has some code redundancy and plan to refactor this during the next sprint.

We observed that the HolidayRelatedDates class triggered a low-medium level of size and lack of cohesion, which we felt may be improved by removing obsolete methods that were initially created in Sprint1 for functional testing purposes. After removal, we expect the size to drastically reduce and this should also improve its cohesiveness.

All of the other classes in our system also stayed within the low levels across the board, or triggered (at most) low-medium levels by CodeMR assessment. Of note, Visit, VisitParameters, DateTime, Weather, and Day classes had a low-medium level of lack of cohesion, which we feel is attributed to their intrinsic nature as a model class. We felt that these classes should be responsible for representing data and mainly included getter and some setter methods to retrieve their encapsulated information. We also noted a low-medium level of complexity for our enum classes, HolidayType and WeatherType, which was somewhat appalling as they were intuitively simple both in implementation and usage.

Progress Summary

The majority of our work is on expanding the flexibility of the code and class methods. We have developed two phases for data generation. The first part generates the initial visiting data for a normal day without any consideration of other factors; and the second part modifies the data (adds more visiting data or reduces customers) once a day is identified with any special events. Functionally, it allows the user to observe how the shopping patterns change in association with the changes in holiday, discount hours, lunch/dinner rush hours, and weather. Technically, this has reduced some degree of hard-coding for special events/days, and increased the flexibility in applying these effects to different days/times of the client's choice.

In addition, we built two sets of data and stored them into a NoSQL database *MongoDB*. Although we first considered using a SQL database because our current data presents information in a structured relational (table) format, we opted for the NoSQL database to support evolving data requirements. Specifically, we imagine that there may be additional impacts of COVID-19 that are not included in the current dataset, requiring additional fields later. We also want to account for changes to the dataset when the situation with COVID-19 improves and/or resolves over time. Additionally, we feel that a key-value database may be suitable for our current dataset, where the key represents a predefined interval in hour(s) or day(s), and the value represents information for all corresponding shopper visits. Below is the detailed list of progress we made in this product.

Part I: development of methods and classes

1. We revised our assumptions on the daily amount of customers, distribution of entry time, and duration.
    a. Only keep the normal daily pattern.
    b. Create special patterns for special events/days.
    c. Allow these special patterns to be applied flexibly.
2. The work of this phase has developed a series of new methods assisting the data modification, such as applyMealRush(), applySeniorDiscount(), applyHoliday(), applyDayBeforeHoliday(), applyNiceWeather(), and applyBadWeather().
3.

Part II: maintaining code integrity

1. We revised our code and checked for erroneous inputs that might break the system. This included customized exception classes with customized messages.
2. We included additional unit tests that aim to cover edge cases. We also completed integration/functionality testing as well as unit tests.

Part III: move csv to a NoSQL database

1. We set up our environments to support MongoDB and make the necessary configurations to transform our data from the csv files into the desired structure in the database.