University of Arkansas – CSCE Department
Capstone II – Final Report – Spring 2022

# Walmart: Predictive Planning, Ordering, and Monitoring

Kyle Orman, George Romano, Abigail Tee, Joshua Thornburgh, and Margaret Turner

## Abstract

Our main goal is to create a program that assists Walmart associates when ordering store supplies. To help associates order supplies more efficiently, we will be using machine learning (ML) to forecast the quantity needed of store supplies that are ordered regularly. This will simplify the process of ordering these supplies, so associates can save time using the forecasting as a guide. Associates will still have to verify that orders are correct and have the choice of removing or adding to them as the need arises. Overall, this will increase efficiency, minimize human error, and avoid costly emergency supply orders when associates are ordering store supplies.

## 1.0 Problem

Current Walmart associates order store supplies manually by planning what they need ahead of time on a week-to-week basis. This includes handheld scanners, office equipment, plastic bags, light bulbs, software, servers, etc. By planning these supplies manually week-to-week, it raises opportunities for mistakes and reduces the efficiency of the associates by spending more time on supply planning instead of their other duties. If the supplies needed were to be pre-generated, then this could improve efficiency and reduce mistakes for employees because it reduces the steps needed for assembling a supply order.

If a solution were to never be in place, then continuous mistakes, such as ordering too much or too little, are likely to be made by associates in stores. Some associates can be pressed for time and place their orders too fast and cause the orders to be inaccurate. This can also lead to associates not having their required supplies, such as a light bulb missing in a section of the store, or a few associates not having a handheld scanner. These mistakes are human error and resort to more expensive emergency ordering. Solutions to this problem have been tried and failed. This is likely because they do not keep stock numbers on store items designated for store use.

## 2.0 Objective

The objective of this project is to help associates order supplies more efficiently by forecasting what items will need to be ordered and of what quantity. Currently employees must evaluate the need for store supplies and manually purchase the needed supplies through a catalog available to Walmart retail store locations. Our goal is to capture the demand signal of supplies as well as foresee delays on the orders to predictively create business supply purchase orders with minimal intervention on behalf of the employees. It is based on trends, patterns, historical purchases, and logistics along with the increased demand of holiday and seasonal purchasing. While real store data is out of our reach, we will instead simulate a real store.

To achieve our objective, we trained a machine learning model in Python based on our generated data. In total we used six years of historical weather data to generate six years of simulated store item use. The machine learning model produces predictive versus actual order comparisons as line graphed PDFs. Our simple web interface allows a user to navigate to the data generator and witness the data generation process. Once this process is complete, the user can then see the Machine Learning Predictions made.

To properly develop a machine learning model, there is first a need for data to build around. To accurately reflect a real store, the development of a program to simulate one is needed. Variables that affect item use would be holidays and weather, as well as the day of the week. Weather can include extreme fluctuations in temperature, how much it rains, and if it snows. The aim is to generate consistent data, with random variations.

## 3.0 Background

### 3.1 Key Concepts

For the front-end development, we built the graphical user-interface of a website using HTML (Hyper Text Markup Language) and Django in Python. HTML is the backbone of any website development process. Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. These implementations will allow the user to interact with the website.

The back-end development will focus on how our website functions. It will lay the foundational code that will enable the website to process the actions of the user on the front-end and deliver the correct information in return. The technology of the back-end is a combination of servers, applications, and databases. Our programming in this area included writing APIs, creating libraries, working on data architecture, and writing code to interact with our database. Additionally, we will be generating our own data for use as training data. This, among most of the other pieces, will be done using python.

Lastly, we will use machine learning to implement our project. Machine learning is the study of computer algorithms that can automatically improve through experience and the use of data. We will create our own dummy data for the ML to take place.

### 3.2 Related Work

Many companies are using machine learning for automated ordering. [5] In an IRJET article, the proposed design of accomplishing an inventory management system was tagging the warehouse components with a RFID (Radio-Frequency Identification) tag. The data these tags hold is then backed up to the cloud for future use. From there, you can see what is and is not in stock. This implementation method will be different than ours as we will not have stock numbers for the items being reordered.

[1] Another article looks at artificial intelligence for inventory management. It mentions that Amazon implemented artificial intelligence throughout their inventory operations. The article then mentions two key implementations of artificial intelligence for inventory. These are Demand Prediction for Inventory Management and Reinforcement Learning systems for full-inventory management. The method that would be a better approach for what we are doing is Demand Prediction for Inventory Management. The general idea of this method is to build a time series prediction model that can estimate what demand will be like for the coming days across all items in your inventory. This is what we aim to accomplish.

In the implementation of the predictive planning and ordering, Walmart representatives mentioned they wanted a "Did you forget" pop-up when finalizing the stock order. [4] An article describes this as a recommendation engine. This article defines a recommendation engine as, "information filtering tools that use algorithms and data to recommend that most relevant items to a particular user in a given context." This will be beneficial in implementation as it will decrease human error and increase order efficiency in checkout.

[2] An Unleased Software article, "Using Machine Learning in Inventory Management." discusses reducing forecasting errors. With machine learning technology, predictions can be made using data to adjust forecasts to suit companies and account for more factors than typical forecasts. This is important to consider when using machine learning because it can predict demand in the future and allow the correct quantity to be purchased before it is needed.

Refocusing this project has allowed more research of related works. [3] A Mosaic Data Science article highlights the importance of weather and its impact. It states, "Weather has a high impact on operations in many industries, and therefore is of immense value to integrate into strategic decision making." Although not initially planned to be put into effect, the weather is a great indicator of consumer presence at a store. Weather factors that we specifically are looking at include average temperature, average daily rainfall, and average daily snowfall.

In conclusion, the research done throughout this project helped us understand the steps needed to be taken to develop the finished product. While not all these articles and methods were used during our implementation, it was key to research and take these methods into account.

## 4.0 Design

## 4.1 Requirements, Use Cases, and Design Goals

Requirements –

- Website front-end
    - Input fields
        - Store number
            - Restricts the ordering options to location
        - Employee login information
            - Restricts the ordering budget by employee tier
    - Django - Python
        - Useful for displaying our data generation tool
        - Security implementation to protect data
    - Home, data generator, machine learning predictions, about us pages
        - Necessary pages for our project
- Database back-end
    - A relational database
        - PostgreSQL or equivalent
        - Two tables needed
            - One table with supply description, ID, and price
            - A second table with order history information including date, item ID quantity, store number, and a value to flag if the order was an emergency order
            - The respective IDs will be primary keys within the relational database
    - Historical supply ordering data to be provided by Walmart
        - The data will be the basis for the machine learning model
- Machine learning model
    - We tested 3 types of ML models. For the finalized model we used a gated recurrent unit model (GRU) since it was verified through testing to be the best balance of accuracy and execution time
    - The ML model must take input. Data will be generated using item 4. Then, the ML model will read from a CSV file produced by item 4
    - The ML model must provide output. This is generated by a predictive algorithm for supply needs. In addition, the supply needs will be manually verified by an employee.
- Data Generation
    - The entire generator is coded in python. This was due in part to its flexibility as a data science tool and ease of use. We collected six years of weather data to generate the necessary data for store use. What is output is an item order history from the beginning of 2016 to the end of 2021.

Use Cases –

The primary use case of this project is for restocking and reordering Walmart store locations with business supplies that employees/customers extensively use. The website front-end interface will allow the user to see the data generation tool work and display machine learning predicted results. In addition to streamlining the supply reordering process and minimizing the amount of time that it takes for employees to evaluate, and order needed supplies for day-to-day operation of Walmart retail locations, the tool we will develop will allow Walmart stores to avoid emergency supply orders for urgently needed supplies that come with increased cost.

There are two possibilities for expanding the use case once the base case is achieved. First, we can expand the use case to restock and reorder the entire store inventory including products and goods meant for sale to Walmart customers. A second possibility is that we can apply the machine learning model to assist with management of corporate office supplies—this would require different logic, but overall holidays and weather can also help predict consumption of office products.

Design Goals –

Our design goal was to develop a system that allows the user to efficiently execute our primary use case; to predict orders. Past weather models and holidays will supply data that will drive a machine learning model that yields a predictive algorithm that will be used for supply ordering and restocking. This includes a minimalist website front-end that accepts login credentials and a database back-end that stores the weather and holiday data. The data generator tool will produce the simulated store order history for the ML module.

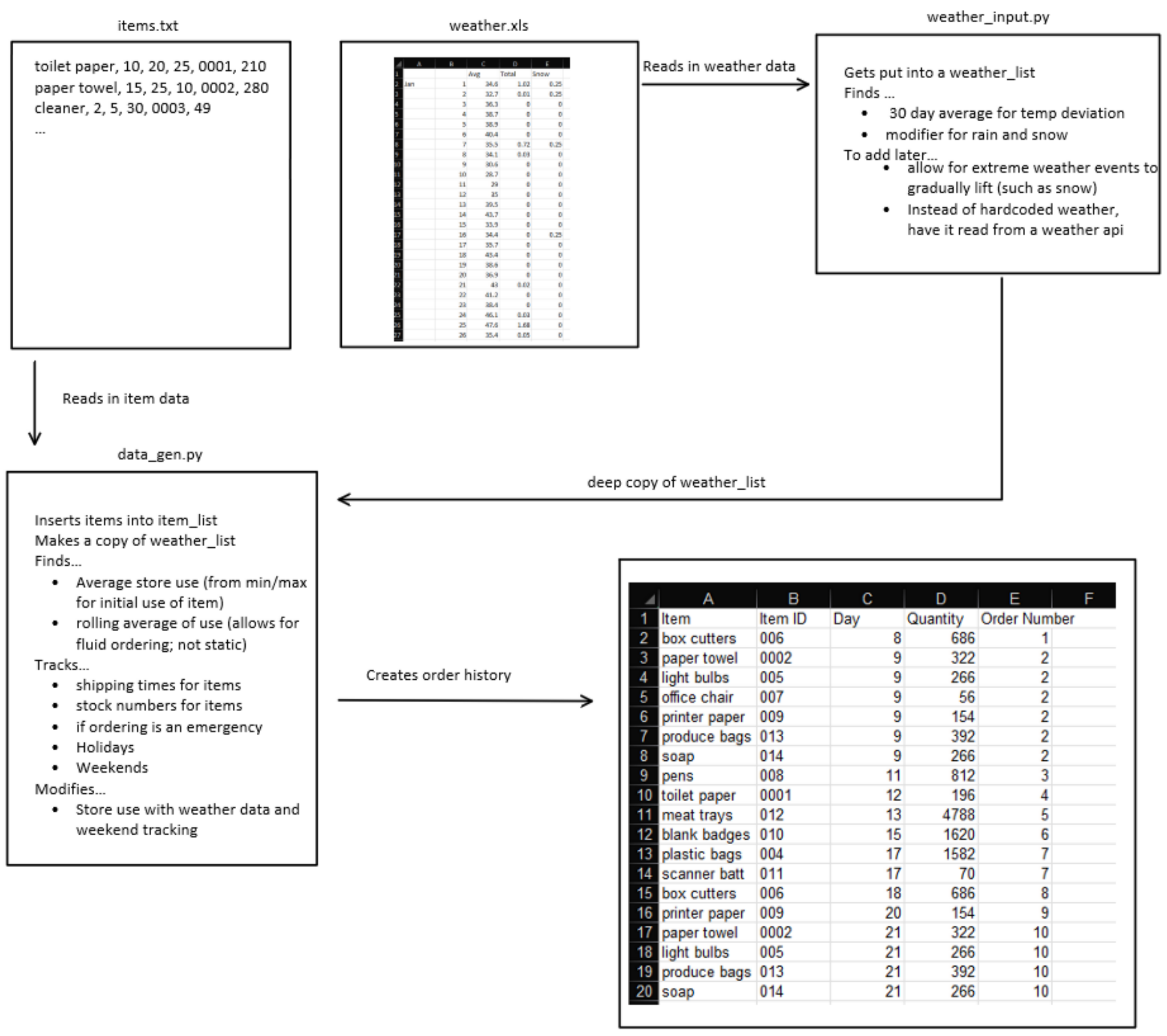
## 4.2 Detailed Architecture

Walmart provided us with data from their test system, but it was deemed necessary to create our own dummy data. Due to this, the focus of our project has shifted to data generation followed by machine learning assisted predictive ordering. Now that data generation is a factor, we have more control over what the data will look like. We can also limit the list of items available to order and have more common descriptions rather than the repetitive and uncommon descriptions found in Walmart's test system data (for instance, one-hundred different battery bundles). Simplifying our data will allow our results to make more sense to anyone observing our output, while still having the capability to apply our solution to Walmart's dataset. However, there must remain common identifiers in the data. The data we are generating will include an item number, item description, date ordered, quantity ordered, order number, and store number.

A design focus for this project is to auto-generate a list of items to be ordered for the store to function. Items such as toilet paper, cleaners, and paper towels that employees will use throughout the year need consistent and reliable replenishment. We are currently designing the process for single store use. The process will be scalable so that after we successfully implement our single store project design it will be simple to expand the project to handle multiple store locations. Since some of the details of our project have changed since inception, the metric for success has changed as well. We are no longer concerned with making an interface for the store employees to access, review, and confirm orders. Instead, data generation followed by machine learning implementation is our primary goal. The secondary goal of the project is to provide a clean user-interface for the data generation tool and provide organized output of the results of our predictive ordering process. Another change in our project scope is that we are no longer concerned with "emergency orders" since, assuming our predictive ordering is correct, the occurrences of orders of this type should be reduced and there would not be a way to avoid true emergency need for certain items.

## Data Generation:

The data generation tool was designed to account for many variables that may affect the quantity of any item that needs replenishment. The tool accounts for the fact that there may be more store traffic on weekends and more store traffic on holidays throughout the year. We have collected six years of weather data for the Fayetteville, AR area to modify item use with. The data includes average temperature, amount of rain and snow. Currently, the tool is operated on the command-line interface and does not take input. While the hosted webpage does operate the data generator, the running of the machine learning is still needed to run locally as the runtime is more than four hours and requires a great deal of calculations.


items.txt
toilet paper, 10, 20, 25, 0001, 210
paper towel, 15, 25, 10, 0002, 280
cleaner, 2, 5, 30, 0003, 49
...
weather.xls
Reads in weather data
weather_input.py
Gets put into a weather_list
Finds ...
30 day average for temp deviation
modifier for rain and snow
To add later...
allow for extreme weather events to gradually lift (such as snow)
Instead of hardcoded weather, have it read from a weather api
Reads in item data
data_gen.py
deep copy of weather_list
Inserts items into item_list
Makes a copy of weather_list
Finds...
Average store use (from min/max for initial use of item)
rolling average of use (allows for fluid ordering; not static)
Tracks...
shipping times for items
stock numbers for items
if ordering is an emergency
Holidays
Weekends
Modifies...
Store use with weather data and weekend tracking
Creates order history

| Item | Item ID | Day | Quantity | Order Number |
|---|---|---|---|---|
| box cutters | 006 | 8 | 686 | 1 |
| paper towel | 0002 | 9 | 322 | 2 |
| light bulbs | 005 | 9 | 266 | 2 |
| office chair | 007 | 9 | 56 | 2 |
| printer paper | 009 | 9 | 154 | 2 |
| produce bags | 013 | 9 | 392 | 2 |
| soap | 014 | 9 | 266 | 2 |
| pens | 008 | 11 | 812 | 3 |
| toilet paper | 0001 | 12 | 196 | 4 |
| meat trays | 012 | 13 | 4788 | 5 |
| blank badges | 010 | 15 | 1620 | 6 |
| plastic bags | 004 | 17 | 1582 | 7 |
| scanner batt | 011 | 17 | 70 | 7 |
| box cutters | 006 | 18 | 686 | 8 |
| printer paper | 009 | 20 | 154 | 9 |
| paper towel | 0002 | 21 | 322 | 10 |
| light bulbs | 005 | 21 | 266 | 10 |
| produce bags | 013 | 21 | 392 | 10 |
| soap | 014 | 21 | 266 | 10 |

A challenging task that has repeatedly come up is the question of how to simulate a real store. This question tasked us with understanding the variables that can affect an individual's shopping habits. We pose this question because our design goal was to aid associates in their ordering to keep the store operational. If there are no customers, then there is no reason to replace used products. This was our baseline for understanding item use. A large hurdle was the notion that stock numbers are not kept on these items, only how much is ordered at a given time. If stock numbers were kept, this would simplify the predictive process.

One thing we figured was that individual items have an average use within a store. This can obviously change from store to store. To rectify this, the data generator first randomly chooses from a range of values to be that stores average use. Next, we give the store enough supplies to be able to operate for around two weeks. If every day were the same, as there were no weekends, the weather was constant, and holidays no longer existed, we would more than likely see the use of these items as constant. Thus, the average use would in fact be the only use per day. However, this is not the case. The need for controlled chaos becomes apparent the more data that is generated.

This is where weather and weekend checks solve some of the data's problems. First, the weather. Using weather data collected from accuweather.com, we compiled six years of weather data to modify with. While one year six times may have worked, we really wanted these variables created by nature to influence our item use. With weather, there are few options for what is forecasted. Rain, sun, snow, or something in between. The data pulled was from AccuWeather and was for the location of Fayetteville, Arkansas. Considerations for environmental conditions will be limited to rain, snow, and extreme temperature fluctuations for our region, but can be expanded on later. For rain, we chose to use a $n\log(n)$ scale to determine the reduction of traffic to the store. Using this model for weather creates a deficit value. This way if there is little rain, traffic is unaffected, but if there is a downpour, then it is heavily affected. For snow, the same considerations are made. In its current form, the data generator reduces traffic on the day in which it snows. This reduction is that of a 75% of the normal use of an item.

For the weather conditions listed above; these are stored in an object list produced by weather_input.py. The size of the list is equal to the number of days within a year, 365. If you wanted to retrieve weather data for a certain day, you would only need to call up the index in the list and then the attribute of the object you wanted. For rain information on the sixth day of the year you would use weather_list[5].rain.

Second is weekend traffic. After a quick search we found that stores are often more frequented on the weekend compared to the weekdays. The slowest days are around the middle of the week, but the distribution of those days could be shifted around and still yield the same use data. All that matters is the number of those low days. How we solved for which day it is was a simple modulo calculation to keep up within seven days. Depending on the number, a store could see increased traffic or be a tad slower.

Item and order data is created in our data generator. The program reads from a list of items, finds the initial average for store use, gives an initial stock value, and then uses the weather data to modify item use. Holidays are also considered and are stored as integer values in a list. Since every day can be represented as an integer, this was the easiest way to find when holidays occur as they can be compared to against an arrays index. Currently, a holiday occurring is a static modifier, while it is not very realistic, this does allow us to see when major events occur. To store this information, two object lists are used. The first is an item list and the second is the order list.

Items have many attributes; these include min/max (range for average use), item id, shipping time, initial stock values, a rolling average for use list, and several more. Those listed here are the most important item attributes we take into consideration. Once an item is ordered, it then gets stored into the object list. This may be unnecessary, but it allows for the understanding and use of the information to be more straightforward. While the item list contains all the information, anyone can later just send out the order list and have it contained the same contents as the CSV's created later in the process. With the order list, it is then sorted based on the day the item is ordered, and then order numbers are placed on items that are all on similar days. All orders on day eight are a part of one order number and those from day ten are another, just as an example. Since we are concerned with representing accurate data and not the true functionality of how we got it, this worked to serve our needs. Once order numbers are attributed, the resulting order history is then saved to an Excel form to later be converted to csv.

The order history is created year by year, from 2016 to 2021. Each year is saved as its own individual file and then merged once all years are complete. Once merged, the file is then split into the respective items. If there are 19 items, then 19 csv files, with only one item to the file, are created from the merged file. This was necessary to facilitate ease of use with the machine learning step.

The data generator was created using the Python programming language. We chose the Python language because of the vast number of useful Python libraries and frameworks and its known use in data science. The flexibility of this language allowed us to try different solutions with ease, and because it has become one of the top languages for machine learning solutions, there was no need to switch to another language to code the second large part of our predictive ordering solution. Python libraries included in the data generation tool to achieve the desired functionality consist of random (allows pseudo-random number generation), requests (a simple HTTP library), OS (allows use of operating system dependent functionality), datetime (supplies classes for manipulation of date and time), math (provides access to mathematical functions), copy (provides shallow and deep copy operations), and xlwt (a library for writing and formatting information to Excel files).

**Machine Learning:**

The dummy data generated with our data generation tool was used as the basis for our predictive ordering implementation. Using the Python programming language along with the TensorFlow 2 library and Keras framework, we built and trained 3 predictive models used to predict the quantity needed of each individual item. TensorFlow 2 was chosen for this project as it allows for easy model building with multiple levels of abstraction. The Keras framework is an open-source deep learning API that provides a  Python interface for artificial neural networks; using it ensured our solution will be scalable as the scope of the project grows. The combination of TensorFlow 2 and Keras is the most widely adopted deep learning solution, so the number of resources and examples we were able to apply ourselves is significant. We also generated excess data that will not be used in the machine learning process to serve to check the accuracy of our predictive model. Specifically, the most recent 10 percent of  each individual item data was reserved to compare to out predicted output. One of the benefits of being able to generate our own data is that we were able to process multiple timeframes of data and see how the accuracy of our predictive ordering holds over extended periods of time. This aspect turned out to be valuable since the model required more historical data than we previously assumed to improve the accuracy of the predictions.

We tested the three predictive models we developed using Google Colab and CSV file containing data generated for one item. During the testing of the three predictive models, we opted to use the gated recurrent unit (GRU) model due to the balance of accuracy and execution time when quantitatively compared to the LSTM and Conv1D models. To assess whether the model was a good fit we used the root mean squared error (RMSE) metric which considers the value being predicted (when taking this metric into account you must decide if the value is appropriate). The results of the testing suggested that our model functioned appropriately.

Before the data could be ingested by our finalized machine learning model, it was formatted into time series data using the order date of each item we are tracking. The CSV file output of our data generation tool was read and preprocessed in a portion of Python code, then the machine learning portion of the code predicted the future order quantities of each item based on correlated input values. The Python imports we are using include TensorFlow (Keras is included with the TensorFlow library as well as the Keras sub-libraries we needed for our model), OS (to handle operating system commands in Python), Pandas (a data analysis and manipulation tool), NumPy (a Python scientific computing library), and MatPlotLib (for plotting and saving our predictions versus the actual quantity that was reserved for testing). A Keras utility imported the dataset that we previously created with the data generation tool, then the CSV paths for each individual item was used to import the data into Pandas DataFrame objects so we could manipulate the data as previously described to format our data in time series format. We created Sin and Cos functions based on the yearly trends of our data to be used as additional input to provide more complexity to our model. Since our prediction model relied on multiple inputs, we used a tensor as the input to our model. A window of 7 was used form our finalized model, meaning 7 previous orders were used to predict 1 following order. For the next predicted order, our model used the 6 previous orders plus the one previously predicted; the prediction process continued this way until all the reserved test data had a prediction to be compared to.

## Webpage Front-End:

In addition to our data-generation tool and predictive ordering implementation we are designing a front-end user-interface based on Django. This user-interface will be used to prompt for the input and display the output of the data generation tool, allowing for a user to generate data tailored to the specific inputs the user provides. The user-interface will also be utilized to display the output from the machine learning process and any visualizations that we develop to highlight the accuracy of the machine learning model.

The front-end component will be divided into four entities: the main, generate data, machine learning predictions, and about us pages. All pages will be accessible from the main page. The pages will be routed and connected using Django with Python. Django is a web framework and application that is written in Python. It is an open-source tool used for developing front-end applications. It has three key features that we will be using for this project: URL routing, template engine, and security authentication.

Each page will have a button that is to be used when the user wants to navigate back to the homepage and run the generator again or look at the machine learning predictions. The homepage will first feature the Walmart logo and log in button in the center to give the user access to all permissions for starting an order and accessing their order history. The goal is for each user to have access to the data generator and to see the machine learning predictions. The page will also feature a summary of what the project is and a button to generate data.

The first page routed when making an order is the “Generate Data” page. It is responsible for showing off our data generation tool. By selecting the button, it starts the tool and after it generates data for 1000 epochs (default value), it then outputs the predicted items that need to be ordered. This will be outputted in newly created PDF files located in the machine learning predictions section.

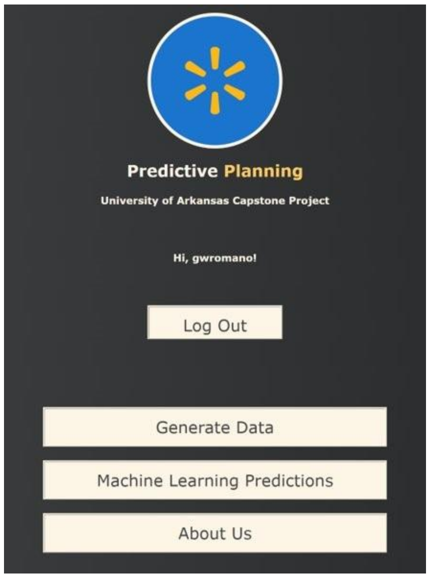

Predictive Planning
University of Arkansas Capstone Project
Hi, gwromano!
Log Out
Generate Data
Machine Learning Predictions
About Us

Fig 1. Main page

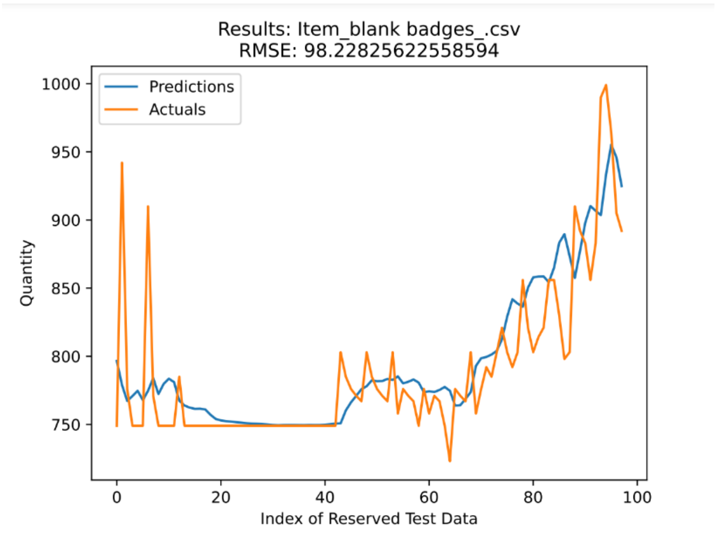


Fig 2. Example machine learning prediction output

The machine learning predictions page shows accessible buttons for our outputs. Each output is formatted as a PDF to show our actual vs. predicted items needed to order in graph form. There is a button for each item that the store needs. The example above is a sample output of blank badges ran after 1000 epochs.

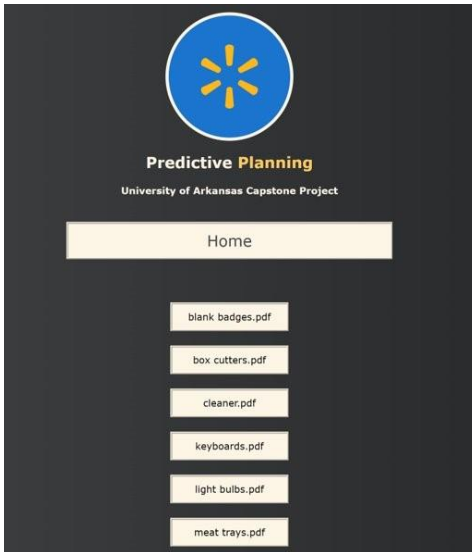


Fig 3. Machine learning predictions page

## Future Works:

Here we discuss changes that could be made to enhance our predictive planning. Given enough time, these are the recommendations and changes we would implement.

## Data Generation:

There are a few functions that did not get added to the data generator. The first being the weather API addition. An ideal use for this tool would be its flexibility in the sourced weather da ta. While the data generator works with the weather data collected, being able to specify the time period for weather as well as the location would be an ideal next step for the data generator.

Another aspect that was not fully realized was the use of the rolling average for the temperature. An initial goal was to have extreme fluctuations in temperature be a deterrent to customers visiting a store. However, this was pure speculation on our part. There were too many factors at play to make the reasonable assumption that it was legitimate. Another reason for breaking focus on this feature was how we recorded temperature. The data collected was an average for the day. This means any deviation may not have ever been large enough to ever see the scenario play out. To overcome this, one thing that could be done is take the object use from a day to an hour. This way the program can observe specific times of day as to which times are peak hours.

Second, building tension to a holiday. Currently, the program only has the day of the event as the modifier for the holiday. This may not be the case as customers shop during the days leading up to the holiday as well as the day itself. A real benefit here, though, would be actual data as to how customers shop.

Finally, being able to accept more user input. One area that we would like to see from this is the ability to take in outside variables. While weather would require a zip code, being able to request a certain number of items as well as multiple stores could be useful for variation of data. Since the scope was limited to proof of concept, the use of a single store was sufficient evidence of its validity.

**Machine Learning:**

Model hyperparameters such as the learning rate and model topology can be adjusted to refine the machine learning model. In addition to using real world data, more data could be used to improve the complexity of the model which would lead to greater accuracy of future predictions. Also, a program could be developed that utilizes the trained version of the model to specifically use the information of only the last seven orders to predict the following order; once the model is trained with a sufficient amount of data the execution time of prediction alone would be extremely fast.

**Webpage:**

Since the machine learning model currently takes about four hours to generate accurate predictions, it is unnecessary to build a user-friendly interface while the generating data tool is running. In the future, we want to make this process more efficient and to show our outputs better. Currently we show what the program is doing while generating data through the command prompt, but it is there to show that the tool is working. If future associates were to use the tool, we need to make it an efficient process. The user would click on the button then be routed to machine learning prediction outputs.

## 4.3 Risks

| Risk | Risk Reduction |
|---|---|
| Potential SQL injections | Using prepared statements |
| Dropping the database | Backing up regularly |
| Potential overordering | Webpage front-end that associates can use to confirm orders |
| Simulated data is bad | Consistent assessment of produced data |

## 4.4 Tasks

Understand the given data

- Generating our own dummy data allows us to manipulate the data and make it more functional for our purposes.

Creation of Data Generator

- Tool to generate data for several stores.

- Consider weather, holiday activity, and weekend vs. weekday foot traffic
- Output in a format easily readable by the ML module
- Will eventually be able to fetch weather data for zip codes, not just a static location.

Researching machine learning

- Since our team has limited to no experience with machine learning, we will be researching what makes an accurate ML model. Are there items that have relations? Or does it only seem like they do?
- A longer time slot has been given to this as it is the area that will have the largest weight for the project.

Designing the database to pull from

- Considering the large scale of data that we will be receiving; we will need to design our database with quick queries in mind. A cluster design would be quick but would require a lot of maintenance. A sorted design with binary search should be more than sufficient.

Designing the front-end that the user interacts with

- Initial front-end design was captured through using Canva. It was necessary to be able to manipulate and emulate the overall design goal to group as a whole
- We will design a mock front-end that will show which items may need to be ordered. The user will still need to confirm before the system orders.

Designing the algorithm to predict orders

- Once the research is done into ML, we will begin designing the algorithm to predict future needs. While in the design phase, we will select a small subset of data to tweak the system. Without the data, it is hard to say what we do and do not have.

Implement designs

- Once the database structure is complete, front-end designed, and the ML model tweaked, we can then begin setting everything up.
- The database will need to be filled with the given data and tested for accuracy.
- The front-end will need to be implemented in such a way that form and function are considered. Our goal for this is swiftness.
- Once the database and webpage are complete, the ML model can then be connected and then the testing begins.

Testing

- Test the ML model on the dummy data generated in implementation
- To curb any need overordering, one concept we will consider is under performing in the prediction. This should keep costs down and still have the capabilities of being tweaked through the user page for final ordering.

Documentation

- Once everything is functioning, we will then compile a formal document that outlines how the functions operate. This will include expected input/output and possible tweaks.
- These documents will include the PHP, HTML, JavaScript, Python, and SQL code and queries that make the model operational.

## 4.5 Schedule

| | Description | Team Members | Begin | End | Status |
|---|---|---|---|---|---|
| **Frontend** | | | | | |
| Design | Involves details pertaining to looks and flow of the site. | Maggie, George, Abigail | 6-Feb | 14-Feb | Complete |
| Integrate with Heroku to Github | Connect Github root directory to Heroku to have a working build. | George | 7-Feb | 15-Feb | Complete |
| Code | Coding the design. | Maggie, George, Abigail | 17-Feb | 24-Feb | Complete |
| Subtask - Route Heroku | | George, Maggie | 15-Feb | | Complete |
| Launch | Launching the webpage to a hosting service. This may be Heroku or something like it. | Maggie, George, Abigail | 22-Feb | 28-Feb | Complete |
| Integration With DB | Once the initial site is coded and running, we will then test queries from our DB to the site. | Maggie, George, Abigail | 1-Mar | 7-Mar | Complete |
| | | | | | |
| **DB** | | | | | |
| Design | Find out what is needed and what is not needed. From there, draw what the relationships will be. | Josh | 18-Jan | 23-Jan | Complete |
| Implement | Create the DB and insert the given data. Update: We will be creating our own data. | Josh | 24-Jan | 16-Feb | Deferred |
| Testing | Testing integration with the webpage. Possible connection to the ML module. | Josh | 1-Mar | 8-Mar | Deferred |
| Trim | There may be data we do not need. May require a rework of the tables. | Josh | 9-Mar | 15-Mar | Deferred |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| **Machine Learning** | | | | | |
| Research | What make a good model. What libraries exist for us to utilize. | Kyle | 2-Feb | 25-Feb | Complete |
| Implement | Begin writing the code that will process the data from the DB. Dummy data can be used in early versions to see how it works. | Kyle | 1-Mar | 15-Mar | Complete |
| Test | Testing of the data for early validation. | Kyle | 28-Mar | 4-Apr | Complete |
| Tweak | Any changes that need to made can be done so here. | Kyle | 5-Apr | 13-Apr | Complete |
| | | | | | |
| | | | | | |
| **Data Generator** | | | | | |
| Early model | Generates data based on a range of values. | Josh | 15-Feb | 22-Feb | Complete |
| Simulate daily flow | Simulate the daily "flow" of a store. Give estimates for item use, shipping times, etc. and have the model order based on how often an item is used. | Josh, Maggie | 22-Feb | 2-Mar | Complete |
| Add weather to model | Adding weather to better simulate natural chaos to customer flow. | Maggie | 3-Mar | 10-Mar | Complete |
| Export to Excel | Adding functionality to export to excel for ML use. | Josh | 3-Mar | 10-Mar | Complete |
| Additional Chaos | More variability that holidays or weather does not affect. (Shipping delays, flash sale, etc.) | Maggie | 10-Mar | 17-Mar | Deferred |
| Graphical User Interface | Allow the user to input certain attributes to influence data gen | Josh | 28-Mar | 4-Apr | Deferred |

| | | | | | |
|---|---|---|---|---|---|
| **Data Gen & ML Merge** | | | | | |
| Pipeline | Automatic transfer of generated data to the machine learning model | Josh and Kyle | 15-Apr | 20-Apr | Complete |
| Cleaning | Cleaning up code and comments | Josh and Kyle | 20-Apr | 22-Apr | Complete |
| | | | | | |
| | | | | | |
| **Documentation** | | | | | |
| Website | | Maggie, George, Abigail | Apr-20 | 23-Apr | Future works |
| Database | | Josh, Maggie | | | Not in use |
| ML | | Kyle | Apr-20 | 23-Apr | Complete |
| Data Generator | | Josh | Apr-20 | 23-Apr | Complete |

## 4.6 Deliverables

a. Design document: contains a listing of each major software component and any diagrams.
    a. Early database design
    b. The evolution of the ML model
    c. Design of the web front
b. Database schema and initial data.
    a. The schema is for a SQL database.
    b. Initial data is six months of data provided by Walmart for ML model building.
    c. New data is generated by our data generator
c. Web site code: including front-end and connected back-end/database interface code.
    a. These will include the Python and HTML code for the webpages.
d. Machine Learning code and analysis.
    a. The language of choice for our ML model will be python.
e. Data Generator
    a. How to use
    b. What it produces
    c. Limitations and things that would be changed in future iterations
f. Final report

a. A detailed analysis and breakdown of the ML model, how the data generator functions, and the web front.

## 5.0 Key Personnel

**Kyle Orman** – Orman is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Programming Foundations I and II, Programming Paradigms and Software Engineering which are relevant to the software design aspects of this project. His professional experience as a broadcast engineer will be useful for creative problem solving and logistics tasks. During this project he will be responsible for machine learning research and implementation.

**Josh Thornburgh** – Thornburgh is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed the following relevant courses: Database Management Systems, Computer Security, Cryptography, Information Retrieval, Programming Paradigms, Algorithms, Software Engineering, and Programming Foundations I and II. During this project he will be responsible for the creation of the data generator.

**Abigail Tee –** Tee is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. She has completed Programming Foundations I and II, Programming Paradigms, Database Management Systems, and Software Engineering. During this project she will be responsible for front-end development.

**George Romano** – Romano is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed relevant courses such as Programming Foundations I and II, Programming Paradigms, Database Management Systems, Software Engineering, Mobile Programming, etc. He will be working on front-end development for the project.

**Margaret Turner** – Turner is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. She has completed relevant courses. Relevant courses she has completed are as follows: Database Management Systems, Programming Paradigms, Information Retrieval, and Software Engineering. She worked an Information Technology internship the summer of 2021 at Texas AirSystems. She will be working on the front-end and back-end development.

**Dipika Mohapatra** – Our industry point of contact. She works for Walmart as a software developer.

**Aneshkumar Tadi and Prasoon Anand** –Walmart technical architect and Walmart tech lead in Bangalore, respectively. They will assist our project by providing technical support and the data we will use to populate our database.

## 6.0 Facilities and Equipment

The scope of this project is primarily focused on software design and implementation. We will not need facilities and equipment apart from personal devices to research and code with. We used Heroku to host our web application with a PostgreSQL database containing our generated supply order data. Everyone in the group must have access to a computer with an internet connection so they can participate in the research, design, and coding of the project. We will be using a database containing computer generated supply chain data and ML model developed in python to produce the predictive algorithm that will be utilized to forecast business supply orders for Walmart store locations. We will utilize the task management software Trello to assign individual tasks and keep to our proposed schedule. No physical location or facility is required to achieve the goals of this project.

## 7.0 References

[1] AI, Remi. "Artificial Intelligence for Inventory Management." *Medium*, Medium, 25 Sept. 2019, https://medium.com/@RemiStudios/artificial-intelligence-for-inventory-management-c8a9c0c2a694.

[2] Chan, Melanie. "Using Machine Learning in Inventory Management." *Unleashed Software*, 20 Apr. 2021, https://www.unleashedsoftware.com/blog/using-machine-learning-inventory-management.

[3] "Fusing Weather Data into Machine Learning Predictions." *Mosaic Data Science*, 25 Oct. 2021, https://mosaicdatascience.com/2020/03/22/fusing-weather-data-into-machine-learning-predictions/.

[4] Gaspar, By: Huba, et al. "10 Product Recommendation Techniques to Improve UX & Conversions." *CXL*, 25 Sept. 2020, https://cxl.com/blog/product-recommendations/.

[5] Tambe, Prof. Dr. "Review of Inventory Management System for Warehouse." *International Journal for Research in Applied Science and Engineering Technology*, vol. 7, no. 6, 2019, pp. 1912–1915., https://doi.org/10.22214/ijraset.2019.6320.