# Automated Wardrobe

Aarya Kagalwala

December 12, 2024

## 1 Project Description and Motivation

This project is centered around simplifying the daily routine of selecting outfits and relieving decision fatigue. Everyday people wake up and have to search through their closet to put together an outfit they are confident enough to wear. This decision-making process contributes to the idea of decision fatigue where the brain can become increasingly exhausted throughout the day making decisions. After learning about this concept, some of the most famous CEOs and brilliant minds of our times have spoken about simplifying their wardrobe and what they wear in an effort to reduce their decision fatigue. Steve Jobs and Albert Einstein are some notable people who believed that having simple outfits allowed them to use their brains for more pressing matters. They both wore the same clothes every day to maximize their mental energy to what they were passionate about (Grohol 2017). This project aims to simplify outfit decisions so that you can save your mental energy while still having fashionable and different outfits that you can be confident in every day.

## 2 Data Description

For this project, data consists of the author's personal wardrobe, such as t-shirts, formal clothing, pants, and outerwear. The data is stored in a database which can be found in the GitHub repository. The database in GitHub contains enough clothing items so that every feature of the program has sufficient clothing items for the user to test out the numerous combinations of options to get an outfit and see how the program works.

The data is structured in the database with nine columns. Each clothing item object has an item_number, clothing_name, category, color, style, material, temperature_suitability, precipitation_suitability, wear_limit, and wear_count. All of the variables are text type except for item_number (integer), precipitation_suitability (Boolean), and wear_count/wear_limit (integer). The program has numerous functions which draw from the tabular database based on what the function is looking for.

The program will prompt entry of clothing items and as the user enters all the information about the clothing the details are entered into a database where all the data is stored. There is a capability in the program which allows for the user to see all the clothing item data that has been entered and there is the ability to remove items from the database. The data itself can be modified in a variety of ways through the program if the user decides to do so. The database is created through the sqlite3 library.

Database Diagram

| my_wardrobe |
| --- |
| **item_number (INT)** |
| clothing_name (TEXT) |
| category (TEXT) |
| color (TEXT) |
| style (TEXT) |
| material (TEXT) |
| temperature_suitablity (TEXT) |
| precipitation_suitablity (BOOLEAN) |
| wear_count (INT) |
| wear_limit (INT) |

# 3 Methods

This project works with a combination of four files: data_handler.py, functions.py, outfit_selector.py, and main.py. The Data_handler file is the backbone of the program. It is where the functions related to the interactions with the database are contained and is often referenced by the other files to perform certain tasks related to the database. The file also enables storage of clothing item inputs and general database management. It works using a SQLite3 library to create the database (my_wardrobe.db) which holds all of the items and their details. The file begins by defining the ClothingItem class, which creates an object for each row of the database, representing each clothing item. Each object contains attributes corresponding to the database columns, allowing for functions across the project to easily access and manipulate a piece of clothing's properties in a simple manner. After ClothingItem, the connect_db() function is defined, which works to establish a connection to the database and create the database if necessary. Add_item_from_db() is the next function defined. It utilizes connect_db() to establish connection to the database and then works by taking a ClothingItem object and then pulling the attributes of that item and using the 'INSERT INTO' statement from SQL to enter a new item into the database. To remove items, delete_item_from_db() was defined. The function utilizes

connect_db() similarly to add item function and then takes an input of the item_id which is just a reference to the item_number of a clothing item. The id number, in conjunction with a 'DELETE' SQL statement is used identify the clothing item being referenced by the user and deletes the item from the database. The next function defined is the load_wardrobe_from_db() function which first selects all from the database and then converts each row in the database into a ClothingItem object by iterating through each row with a for loop and appending each row as an item of a list called wardrobe which the function returns. The next function is do_laundry(), which focuses on resetting the wear count of all clothing items. It does this utilizing the 'UPDATE' statement in SQL and then setting the wear_count to 0. The function then prompts a QMesssageBox which alerts the user that the laundry has been completed. The last function is the update_wear_count() function which pulls from an clothing item object's wear count and updates it into the database. This function is referenced in the outfit_selector function to update the counts.

The next file is functions.py. This file is the responsible for the majority of the interface and is broken down into four classes, each of which corresponds to a specific functionality from the main menu. The first class is the AddItem() class which creates a dialog with an interface which allows the user to input attributes for a clothing items name, category, color, style, material, temperature suitability, rain suitability, and wear limit. The class initializes a combination of QLineEdit, QComboBox, QCheckBox, and QListWidget when constructing the dialog interface for the different questions being asked to the user. The user inputs are a combination of typing, selecting one or more options, and checking multiple options. The interfaces are then referenced in the addItem() function within the class which pulls the user inputs by using .text() if it is a string input or .currentItem() if the input is a button pressed. The input for style, precipitation, and temperature are slightly different due to the way the database is created. For style, the selected items from the list widget are then pulled and converted into a list and then joined and converted into a string (example would be user selected 'Formal' and 'Casual' the processing would make the final output 'Formal,Casual'. For temperature the inputs are stored as a list and the inputs are pulled by using the .isChecked() method. For precipitation, an if statement uses the .currentItem() method to see if a button has been selected (this option is only for outerwears). If a button has been clicked, its text will be pulled, otherwise a default 'None' is assigned. The precipitation variable is then set to true or false based on if 'yes' has been selected or not. Each user input that gets pulled is processed and then converted to a ClothingItem object and then added to the database using the add_item_to_db() function from data_handler. Once assigned, the last line of the function prompts a message box which alerts the user that their item has been successfully added.

The next class is RemoveItem(). This class creates an interface using QDialog as well and has a place for the user to input which item id they would like to remove or if they would like to remove all the items. Within the class is the removeItem() function which works to pull the number inputted. It checks to see if the user has inputted anything and that it is a digit using two if statements. If both are true, then using the delete_item_from_db() function the database deletes the item, and the user is notified with a message box. If either of the if statements fail, there will be a message stating the removal was not successful or there needs to be a number entered. To remove all items a removeAll() function was created which uses load_wardrobe_from_db() and

then iterates through the wardrobe with a for loop and uses delete_item_from_db() to delete each item_number.

The next class is View(). This class creates the wardrobe interface by creating two tables. The top table is the general wardrobe holding all clean clothing items and the bottom table is the 'hamper' which populates as clothes get dirty. The class uses QTableWidget and sets the size of the table by loading the wardrobe using load_wardrobe_from_db() and then counting the len() of the wardrobe. A similar process was done to initialize the hamper table but the combination of an if statement and for loop were used where for each item in the wardrobe if the wear count was equal to the wear limit it would increase a counter which would then serve as the number of rows for the table. The viewCloset() function is created within this class to load in each item into the table. Once again, the load wardrobe function was called and then iterated through. For each iteration if the wear count was less than the wear limit it, the ClothingItem object would be added, and each attribute of the object would be added to its corresponding column. The same thing was done in the viewHamper() function but for items which had a greater wear count than wear limit. Both tables are sorted in ascending order based on the category column.

The final class of this file is GetOutfit(). This class worked on creating the outfit suggestion interface. Using QDialog, it sets a line edit widget for entering the temperature, and then another for if it is raining (selection between yes or no buttons). Within the class is the select_outfits() function which takes the temperature and precipitation input and runs it in the select_outfit() function from the outfit_selector file. This function will then return a top, bottom, and outerwear (when applicable). This is then set as a text in the dialog for the user to view. The function also prompts an image associated with the temperature using if statements.

The next file is the outfit_selector file which builds the entire outfit selection process. It begins by defining the select_outfit() function which takes two arguments: temperature (int) and raining (str). These arguments are from the GetOutfit() class in the functions file. The function begins by using the load wardrobe function. Then using if and elif statements based on temperature ranges and the user inputted temperature (from the argument) a temp_category is created. With this temperature category a for loop iterates through all the clothing items from the wardrobe loaded and adds all clothing items to a selected_outfits list if the outfit wear count is less than its limit and its temperature suitability attribute from the ClothingItem class object matches the temp_category determined. Next the function separates all items into tops, bottoms, and outerwear using the category clothing item attribute. Once items have been filtered by temperature, the outerwear is filtered based on weather and rain. If it is raining a for loop adds all outerwear compatible in the rain. From there an if and else statement sort the list of rain compatible outerwears by warm or hot and cold temperatures and then selects a random outerwear based on what is left of the filtered list. If it is not raining and is considered 'hot' an outerwear is just not selected, otherwise an outerwear is randomly selected based on temperature suitability by iterating through the filtered list of outerwears from selected_outfits. The function also allows for hot and warm tops to be selected if it is cold as long as a cold appropriate outerwear is selected.

Once the temperature and rain elements have been sorted the function calls the do_laundry() function if there are two or less clean tops or bottoms left in the wardrobe. The do_laundry

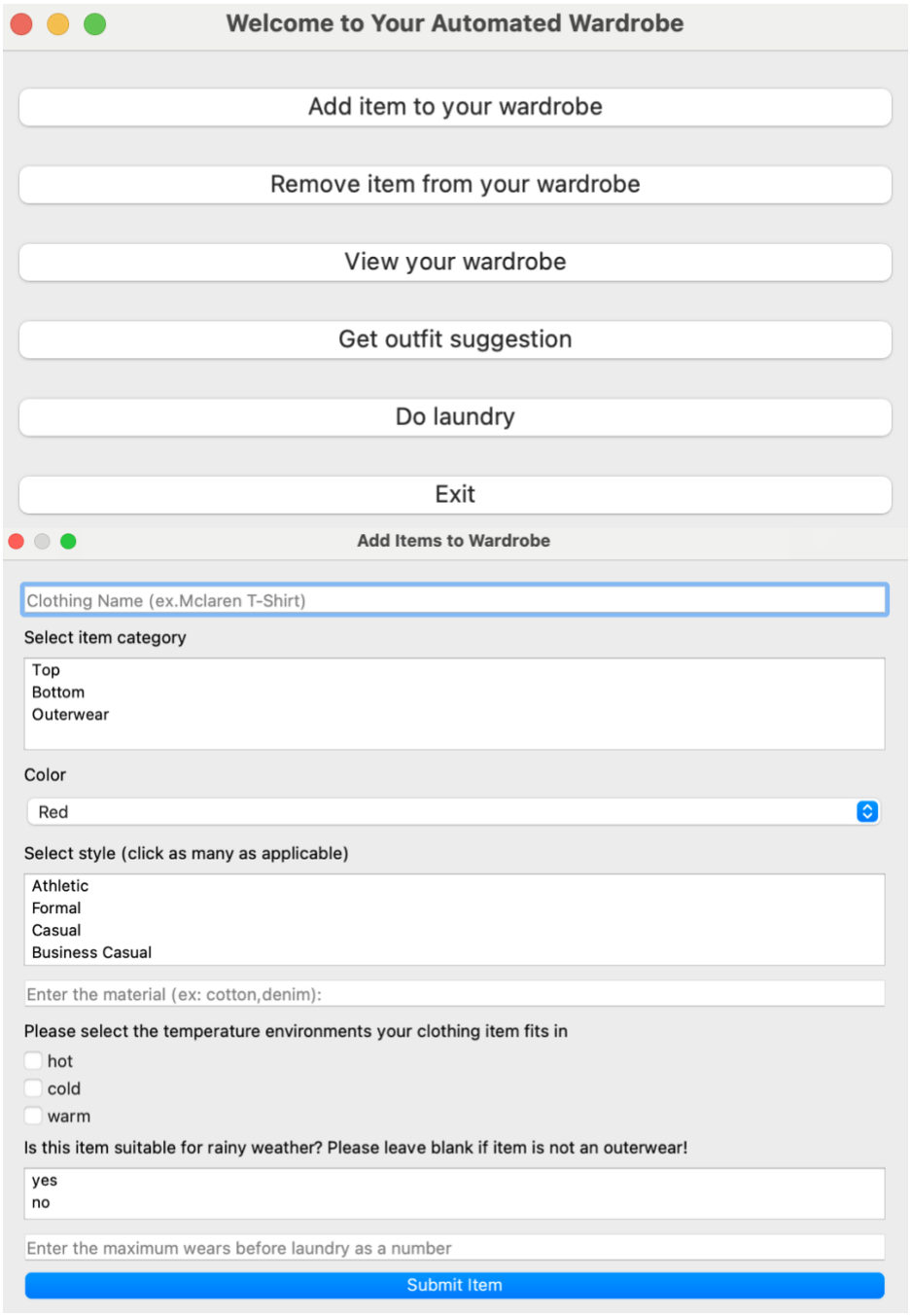
function simply pops up a message box alerting the user of the lack of clothes available. After this, the select_outfit function checks if there are enough clothes period based on available clothes after filtering. If not, it will return a message stating the issue. Once it has been established enough clothes are available the function begins filtering by fashion. There are two dictionaries which have 'banned' combinations. One dictionary is for color combinations and the other is for style combinations. A for loop runs 10 times to find a combination of tops and bottoms which do not fit into the banned combinations. If within the ten iterations a combination is considered fashionable it will break the loop and save the top and bottom. The filter works by first iterating through the different styles assigned to an item (user can make a shirt both formal and business casual for example) and then if that passes it will change a flag from false to true which will break the loop and then prompt an if statement to see if the colors match. If the loop runs 10 times, it will randomly select an item and prompt the no_style function which pops a message box urging the user to 'Upgrade their style!' Once the fashion filtering concludes, the wear count for the selected items is updated using the function from data_handler and an outfit is returned.

The final file is the Main.py. This file is simply creating the MainWindow in the Window class which adds the main buttons in the opening menu. Each button is linked to a specific class or function defined in functions.py. For adding items, removing items, viewing the wardrobe, and getting an outfit there are functions created in the Window class for each feature, which gets executed and opens the interface created in the functions.py. These functions are connected to each button initialized in the Window class. The laundry button utilizes the do_laundry function from data_handler and doesn't require its own function to be executed. The last button is the exit button which uses .close to close the entire interface.

# 4 Results and Outputs

This project resulted in a robust interface and accomplished the goal of automating outfit selection. Users can view their wardrobe and dirty clothes easily and get reminders for their laundry with pop up messages. Furthermore, the outfits selected have some consideration for fashion and weather. The GUI created is also smooth and the program has been tested to ensure that it should not crash but rather prompt users if they did not use the program correctly.

**Here are screenshots of the main features of the program:**


Welcome to Your Automated Wardrobe
Add item to your wardrobe
Remove item from your wardrobe
View your wardrobe
Get outfit suggestion
Do laundry
Exit
Add Items to Wardrobe
Clothing Name (ex.Mclaren T-Shirt)
Select item category
Top
Bottom
Outerwear
Color
Red
Select style (click as many as applicable)
Athletic
Formal
Casual
Business Casual
Enter the material (ex: cotton,denim):
Please select the temperature environments your clothing item fits in
hot
cold
warm
Is this item suitable for rainy weather? Please leave blank if item is not an outerwear!
yes
no
Enter the maximum wears before laundry as a number
Submit Item

Your Wardrobe

What is the temperature?

32

Is it raining?

yes
no

Get an outfit!

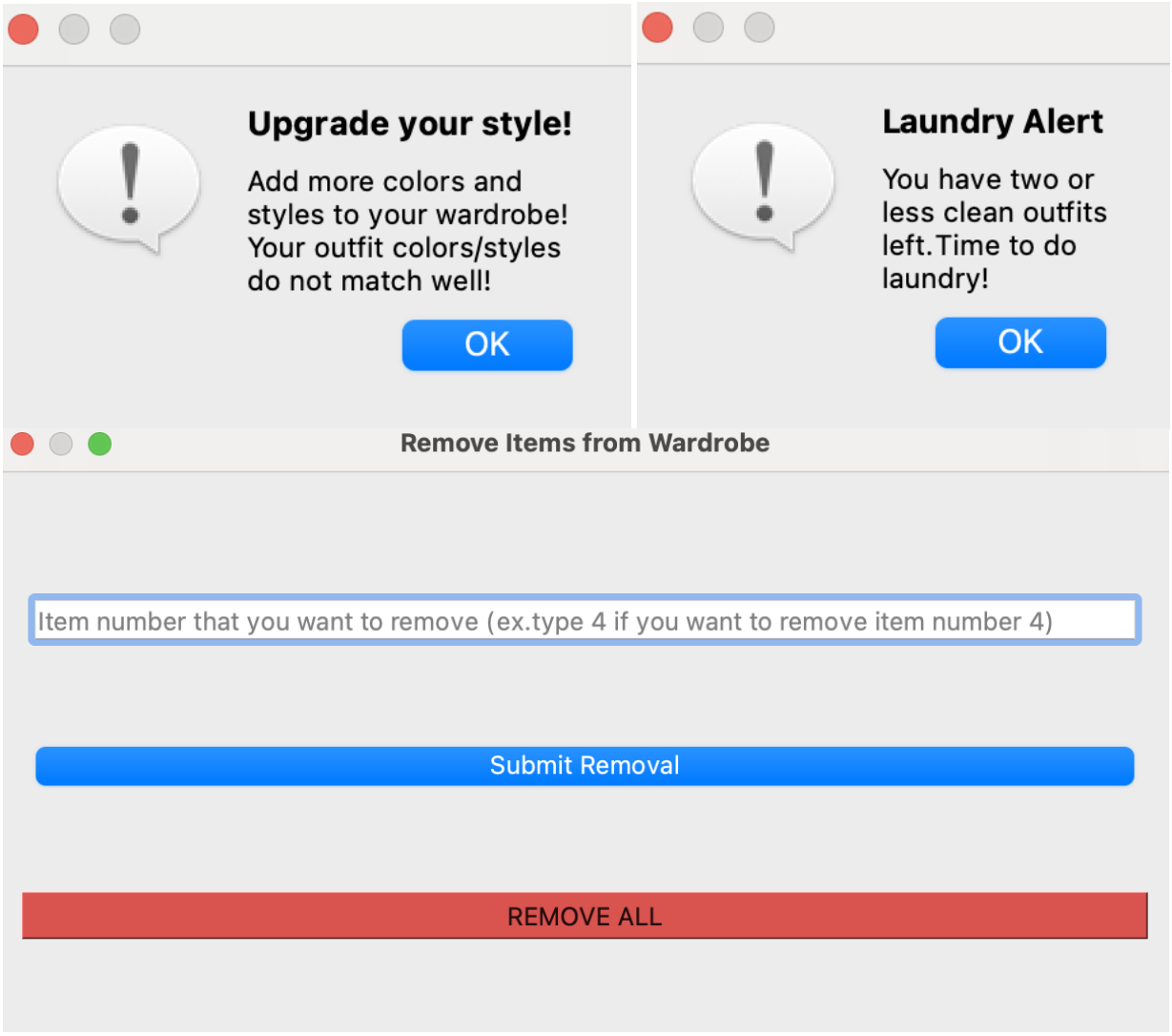
Top: Sunshine T-shirt, Bottom: baggy jeans, Outerwear: NorthFace Jacket

Your Wardrobe

| | Item Number | Clothing Name | Category | Color | Style | Material | Temperature Suitability | Precipitation Suitability | Wear Count | Wear Lim |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 26 | yellow pants | bottom | Yellow | Formal,... | cotton | warm,hot | False | 1 | 4 |
| 2 | 29 | Grey Sweatpants | bottom | Grey | Casual,... | cotton | cold | False | 0 | 3 |
| 3 | 31 | Grey Khaki Pants | bottom | Grey | Casual,... | cotton | cold,warm | False | 0 | 3 |
| 4 | 33 | Suit Pants | bottom | Blue | Business ... | cotton | cold,warm,hot | False | 0 | 3 |
| 5 | 40 | baggy jeans | bottom | Blue | Casual | denim | cold,warm | False | 0 | 5 |
| 6 | 41 | Green Cargo Pants | bottom | Green | Casual | cotton | cold,warm | False | 1 | 5 |

Your Hamper

| | Item Number | Clothing Name | Category | Color | Style | Material | Temperature Suitability | Precipitation Suitability | Wear Count | Wear Limit |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 35 | White Polo | top | White | Business ... | cotton | warm,hot | False | 2 | 1 |
| 2 | 38 | Mclaren T-shirt | top | Black | Casual | cotton | warm,hot | False | 1 | 1 |
| 3 | 45 | white button down | top | White | Formal,... | cotton | cold,warm,hot | False | 1 | 1 |
| 4 | 51 | WM T-shirt | top | Green | Casual | cotton | warm,hot | False | 1 | 1 |




# 5 Future work

Moving forward the plan for this project is to get the clothing matching making more intricate. The goal will be to have a prediction algorithm which can learn to understand fashion styles and 'rules' such as colors, materials, and occasion. In addition to taking material and occasions (such as going to a fancy dinner), I want the algorithm to scrape weather data from a weather website and utilize that data when deciding clothing. This way the user would not have to input any information when getting their outfit suggestion. The last feature to add to the project in the future would be a recommendation for new clothing to buy. Using a few select popular stores websites such as Hollister or Nike and then web scrape some new clothing items and display them whenever the "upgrade your wardrobe" message pops up on the interface would add a new element to the program.

# 6 Bibliography

John M. Grohol, Psy. D. (2017, April 17). *Decision fatigue: Does it help to wear the same clothes every day?*. Psych Central. https://psychcentral.com/blog/decision-fatigue-does-it-help-to-wear-the-same-clothes-every-day#1