University of British Columbia

Social-Ecological Economic Development Studies (SEEDS) Sustainability Program

Student Research Report

CLIMATE-FRIENDLY FOOD SYSTEMS (CFFS) LABELLING PROJECT

Development of the Evaluation Framework for UBC's first Climate-Friendly Food Label (Pilot Phase 1,2,3)

Prepared by: Silvia Huang, Undergraduate Student and Climate-Friendly Food Systems Data Analyst

Supervised by: Juan Diego Martinez, Ph.D. Candidate at the Institute for Resources, Environment and Sustainability (IRES)

Prepared for: Campus & Community Planning (Sustainability & Engineering), UBC Food Services, UBC Botanical Garden

University of British Columbia

May 2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

- CAP: Climate Action Plan
- CFFS: Climate-Friendly Food Systems
- CO2eq: Carbon Dioxide equivalents
- DV: Daily allowance Value
- GHG: Greenhouse Gas
- OC: Optimum Control
- UBCFS: UBC Food Services
- UBCFSP: UBC Food System Project

EXECUTIVE SUMMARY

How is the University of British Columbia (UBC) able to offer a system that helps consumers choose more Climate-Friendly menu items? From a global perspective, food systems are an enormous driver of climate change and contribute to more than one-third (34%) of global greenhouse gas (GHG) emissions, which represent 17.9 billion tonnes of carbon dioxide equivalents (CO2eq) (Crippa et al., 2021). Other estimates suggest that the food system is responsible for one-quarter (26%) of global GHG emissions, representing 13.6 billion tonnes of CO2eq (Poore & Nemecek, 2018). This brings a range of opportunities for actions to mitigate the effect of food systems on the climate.

The Climate-Friendly Food Systems (CFFS) labelling project at UBC takes action to inform the UBC community of the climate impact information of menu items they purchase every day at UBC Food Services (UBCFS). The label provides an opportunity for the campus community to make informed purchasing decisions that can promote a Climate-Friendly Food System. This research report was prepared by the CFFS data analyst, supervised by a member of the CFFS Action Team. This report is focused on the data analysis and the back-end implementation of the CFFS Labelling pilot and is complementary to the report on the communication and definition side prepared by the CFFS communication and engagement coordinator.

The CFFS Labelling pilot is part of the bold actions taken by UBC in response to the Climate Action Plan (CAP) 2030 scope 3 emission reduction goal. The CFFS Action Team has been formed to accelerate transitions toward a Climate-Friendly Food System and advance the CAP 2030 food-related actions and priorities. This project is part of the SEEDS Sustainability Program research collaboration to develop, pilot, and evaluate UBC's first Climate-Friendly Food Label that aims to evaluate the climate impact of menu items sold at UBCFS outlets and operationalize the CFFS food label to inform Climate-Friendly menu choices. The research includes developing a methodology and framework that assesses GHG emissions and other CFFS attributes for menu items at UBCFS. It also evaluates perceptions and the impacts of the Climate-Friendly Food Label on awareness, knowledge, and purchasing decisions.

This project utilized a combination of literature review, discussion with peer institutions, and assessment of the feasibility in the UBC's context to decide the methodology. The primary data sources (recipes and sales data) were extracted from the UBCFS inventory management system, Optimum Control (OC). The data on carbon, nitrogen, and water footprint factors came from external secondary data sources.

The main deliverable of the project is the evaluation framework that conducts the evaluation process of recipes automatically once GHG emission factors have been assigned to each ingredient, and is updated to incorporate additional attributes and adapt to the expansion of the CFFS Label. The evaluation framework is able to read the primary data automatically and output the total GHG emissions, nitrogen footprint, and water footprint of each menu item. To determine the cut-offs for the levels of the label according to GHG emissions, we established a 2019 UBCFS GHG emission baseline and set cut-offs in accordance with the CAP 2030 GHG scope 3 50% reduction goal for food systems.

To help the transition to a Climate-Friendly Food System, we suggest that one way to mitigate the total food system emissions is to reduce the amount of meat and dairy consumption and replace them with plant-based protein products without compromising nutritional value. In addition, to improve the accuracy and specificity of current labels, we recommend UBC lead the engagement process and the establishment of a Pacific Northwest/Canadian-specific footprint factors database by conducting research collaboratively with peer institutions.

Keywords: climate label, Climate-Friendly, reproducible data analysis, GHG, nitrogen, water use, food systems

References

Crippa, M., Solazzo, E., Guizzardi, D., Monforti-Ferrario, F., Tubiello, F. N., & Leip, A. (2021). Food systems are responsible for a third of global anthropogenic GHG emissions. *Nature Food*, *2*(3), 198–209. https://doi.org/10.1038/s43016-021-00225-9

Poore, J., & Nemecek, T. (2018). Reducing food's environmental impacts through producers and consumers. *Science*, *360*(6392), 987–992. https://doi.org/10.1126/science.aaq0216

1. INTRODUCTION

1.1 RESEARCH CONTEXT & TOPIC

Roughly 26% of global total GHG emissions (13.7 billion tons of CO2eq) generated by human activities were contributed by the food supply chain (Poore & Newecek, 2018). A range from 10.8 and 19.1 billion tonnes of CO2-equivalent (CO2e) emissions per year representing between 21% to 37% of global total emissions has been reported by The Intergovernmental Panel on Climate Change (IPCC) Special Report on Climate Change and Land report.[1] According to Hannah Ritchie[2]: "Food emissions are around 25% to 30% from food. Around one-third, if we include all agricultural products."

This brings a range of opportunities for climate action to mitigate the effect of food systems on the environment. In December 2019, UBC joined organizations and governments around the world to declare a climate emergency and renewed its commitment to sustainability, including a commitment to the CAP 2030 (an update from a 2020 plan) to accelerate UBC's climate actions. As part of the CAP 2030, food was identified as an area of opportunity under scope 3 (indirect) emissions.

The purpose of the CFFS Action Team is to serve as engaged experts from the existing UBC Food System Project (UBCFSP) Steering Committee. The CFFS Action Team is responsible for the ideation, coordination, and development of student-led research, initiatives, and interdisciplinary collaborations that can accelerate transitions towards a climate-friendly food system and advance UBCFSP's mission and priorities. In response to UBC's CAP 2030, the CFFS Action Team aims to achieve a 50% GHG emission reduction associated with food systems by 2030 compared to 2019, starting with the development of a Food System Resilience & Climate Action Strategy, with support for campus-wide climate food labelling, and a toolkit to encourage more sustainable dietary choices and habits.

This project researches how to implement and operationalize the CFFS Label across campus by developing a back-end evaluation framework for the climate impact of menu items and implementing a label that indicates the impact of food sold at UBCFS. The main objective is

---

[1] Mbow, C. et al. Food Security in Climate Change and Land: an IPCC Special Report on Climate Change, Desertification, Land Degradation, Sustainable Land Management, Food Security, and Greenhouse Gas Fluxes in Terrestrial Ecosystems (IPCC, 2019). Rosenzweig, C., Mbow, C., Barioni, L. G., Benton, T. G., Herrero, M., Krishnapillai, M., … & Portugal-Pereira, J. (2020). Climate change responses benefit from a global food system approach. *Nature Food*, 1(2), 94-97.

[2] Ritchie, H., & al. (2021). How much of global greenhouse gas emissions come from food? Our world in data (2021).

constructing an evaluation framework for analyzing the recipes and ingredients to provide a composite metric that informs customers about the food's climate impact. The evaluation framework will incorporate a range of attributes that indicate aspects of the definition of CFFS for food products. The definition work and the additional attributes can be found in the complementary report developed by the CFFS Communications and Engagement Coordinator. Along with other education and engagement materials, the label will indicate and incorporate a range of CFFS attributes to give a comprehensive view of the food's climate impact that students purchase at UBCFS.

1.2 RESEARCH RELEVANCE

In order to mitigate GHG emissions and other climate impacts of the food system, various actions from the food production and consumption side are necessary. As a major food provider at the UBC campus, UBCFS contributed to a large proportion of the total GHG emissions from food systems through students' daily meals. The action of providing students with the GHG emission information of the menu items they purchase every day could help to educate and influence their purchasing behavior in a more climate-friendly way (Brunner et al., 2018). The CFFS Label is a clear and efficient presentation to indicate the climate impact information of menu items, thus helping students make purchasing choices that take the climate impacts into consideration.

1.3 PROJECT PURPOSE, GOALS, AND OBJECTIVES

This project aims to operationalize the CFFS Label by constructing an evaluation framework for analyzing the climate impact of menu items sold at UBCFS outlets. This includes creating a reproducible data analysis framework for calculating recipes' GHG emissions; establishing a food GHG emissions baseline for the UBC Vancouver campus; deciding cut-offs for the CFFS Label; and further integrating additional CFFS attributes into the framework and establishing corresponding UBCFS baselines to decide label cut-offs.

2. METHODOLOGY

2.1 RESEARCH METHODOLOGY AND METHODS

This project utilized a combination of literature review, discussion with peer institutions, and assessment of the feasibility in the UBC's context, such as available data and department support, to decide the methodology that best met the goals and objectives of this research. Methods were also determined through discussion with researchers from the University of Michigan, Université Laval, and the University of Victoria who are working on similar climate food labelling projects.

The research methods include primary and secondary data collection; evaluating recipes' GHG emissions; developing an external data analysis framework; constructing a UBC GHG emission baseline; deciding label cut-offs; and incorporating additional attributes. Detailed explanations are provided below.

2.2 DATA COLLECTION

2.2.1 Primary Data Collection

The raw recipe data for menu items sold at UBC food venues was extracted from the inventory management system Optimum Control (OC) of UBC by the FMIS Administrator of UBCFS. Due to system and administration restrictions, the data extraction was conducted manually instead of using database queries. Recipe data was extracted in XML file format, and each file contained one aspect of the recipe information, such as raw ingredients, preprocessed recipes used, and unit conversion information. The evaluation framework was designed in accordance with this data structure.

In Phase 1 at Mercante, in order to establish a 2019 GHG emissions baseline, the sales and recipes data for all products sold at Mercante between January 1 and December 31, 2019 were extracted from OC. Then, in the next two iterations of the CFFS Labelling pilot which expands to Open Kitchen, the sales and recipes data for all products sold at UBC's three major student residence dining (i.e., Open Kitchen, Feast, and Gather) between September 1, 2019 and February 28, 2020 were extracted to calculate the UBCFS GHG emission, nitrogen, and water footprint baseline.

2.2.2 Secondary Data Collection

The GHG emission factor data comes from three main sources, in the following order of preference:

- First, we used the World Resources Institute (WRI)'s Cool Food Calculator emission factors for most of the food groups. It provides GHG emission data based on life-cycle assessments for major food categories in the North American region from research conducted from January 2015 to December 2018. These represented the factors used in the large majority of our ingredients[3] (94.76%).
- Second, we used the GHG emission data from The Big Climate Database, published by CONCITO (Denmark's green think tank), as a supplementary data source for food categories that are not in the Cool Food Calculator such as salt and vinegar. It provides GHG emission data based on life-cycle assessments for major food categories in Denmark. These represented 1.68% of total ingredients.
- Last, for some items that don't have emission factors available, we calculated their emission factors manually by approximating their ingredients using recipes stored in OC or recipes found online. These represented 3.56% of total ingredients.

The GHG emission factors for the rest of the ingredients extracted from OC such as water, beverage, sauce, packaged food, kitchen supplies etc. was assumed to be zero either by the research assumption or the unavailability to estimate. See Appendix B for detailed source of data on GHG emission factors for each food category.

Note that the food groups were slightly adjusted from the Cool Food Calculator for better assignment of GHG emission factors on ingredients procured by UBCFS. For example, the GHG emission factors for more general-level food groups (i.e., fruits) were used for assigning ingredients that were not specified as less general food groups (i.e., apples, bananas, berries) and were renamed as "other" (i.e., other fruits) in the GHG emission factors list. See Appendix B for detailed food categories and emission factors.

[3] Edible ingredients that have been evaluated only, excluding water, beverages, sauces whose emission factors are assumed zero and packaged food whose emission factors are unable to be estimated. Also excluding non-edible items that show up in the ingredients list extraction, such as human labor and kitchen supplies.

During Phase 3 of the CFFS Labeling implementation at Open Kitchen between March and April 2022, the label also incorporates the nitrogen and water footprint for each menu item to produce a composite metric. The nitrogen footprint factor data was provided by the Food Label Toolkit from Leach et al. (2016) and the water footprint factors (including freshwater and stress-weighed water use) were sourced from Poore and Newecek (2018). The footprint factors for nitrogen and water were also slightly adjusted and calculated for each food category present in the GHG emission factors list to avoid assigning footprint factors multiple times, thus improving the efficiency of the evaluation framework and label implementation. See Appendix B and C for detailed nitrogen and water footprint factors.

2.3 ASSUMPTIONS

To make the process of recipe evaluation consistent, accurate, and structured, several assumptions were made when evaluating the environmental footprints.

- The same GHG emission factor was assigned to different forms (puree, sliced, chopped, etc.) of the same raw ingredient.
- The same GHG emission factors were applied to different varieties of the same ingredient (i.e., red and yellow onions).
- GHG emissions are for the raw ingredients, and the final weight of serving takes into account loss or addition of weight during the cooking process (e.g. water evaporation in beef vs. water absorption in pasta) or loss from cutting out inedible parts (prepping stage).
- GHG emissions from the cooking process were ignored, because we had no knowledge of a standard, feasible method to calculate them.
- The GHG emission factor for water is zero.
- GHG emissions from the cooking process were ignored, because they are captured under Scope 1 (natural gas) & Scope 2 (electricity) emissions - i.e. for building energy supply. So distinct from the food (Scope 3) item
- The GHG emission factor for water is assumed zero.

- We ignored the water use in the prepping and cooking process for the water footprint calculation because we have no reliable way of estimating it for different dishes.

- We excluded the GHG emissions from sauces and dressings with unknown recipes (bought in bulk) that have no dominant ingredients.

- For prep recipes that have no standard unit information, the weight of the preps are the sum of the weight of all ingredients that they used.

- The nitrogen footprint and water footprint factors of some dairy categories (i.e. butter, yogurt, and cream) that are not specified in the source footprint data, are estimated as the nitrogen/water footprint factor of milk multiplied by the corresponding ratio of their GHG factor to milk. For example, the GHG emission factor ratio of butter to milk is 5.12. Then the nitrogen footprint factor of butter is estimated by multiplying 5.12 by the nitrogen footprint factor of milk.

- For vegan food categories whose nitrogen and water footprint factors are not specified in the source footprint data, their nitrogen and water footprint factors are the average of the footprint factors of all available vegan food categories.

- For animal food categories whose nitrogen and water footprint factors are not specified in the source footprint data, their nitrogen and water footprint factors are the average of the footprint factors of all available animal food categories.

2.4 EVALUATION OF MENU ITEMS

The GHG emissions of each menu item are calculated by summing up the weight of every raw ingredient multiplied by their respective emission factors. Ingredients' emission factors are assigned according to their category in the Cool Food Calculator, which provides data about the amount of GHGs emitted to the environment during the entire life cycle of a menu item.

For example, the process flowchart for calculating the GHG emissions of a bacon sandwich is shown in *Figure 1* below. First, we get the raw ingredient (item) information and then categorize each item into the food categories in the GHG Emission Factors List. See Appendix B for all food categories and associated GHG emission factors. Next, we assign the GHG emission factors based on the food category for each item and calculate the amount of GHG emissions in grams for each item used in this recipe. For recipes that use pre-processed recipes (preps), such as the garlic butter made of garlic and butter in this example, we calculate a GHG emission factor for this prep based on the items used and then calculate the total amount of GHG emissions in grams for this prep.

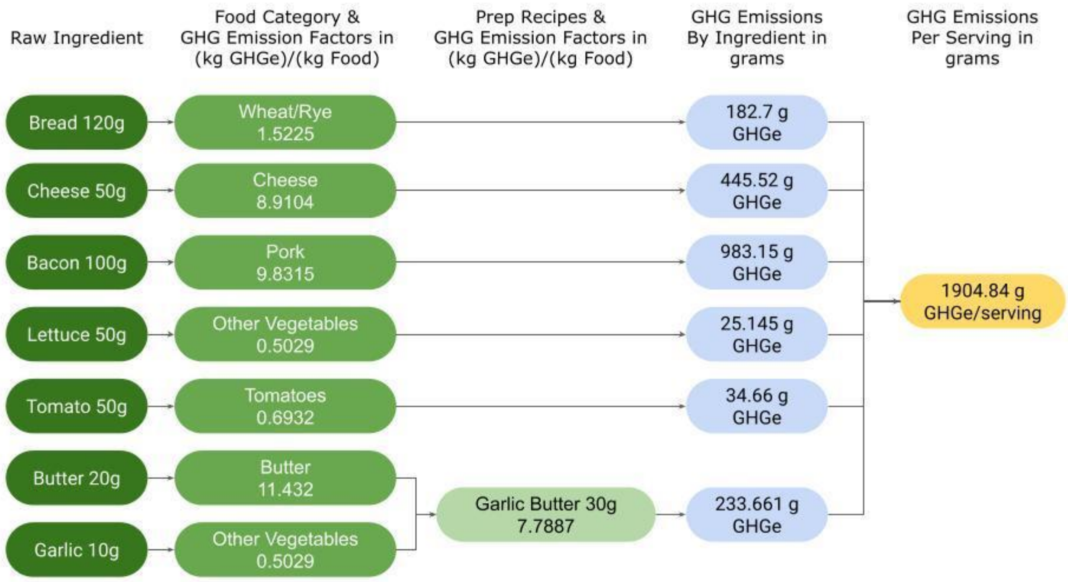


Figure 1: Flowchart for Calculating the GHG Emissions of a Bacon Sandwich

Lastly, we sum up all the GHG emissions of each item or prep and use this sum and the food group (i.e., lunch/dinner, breakfast, or desserts/snacks) to determine the label color for the Summer Pilot (Phase 1). For Phase 2 and 3 of the CFFS Labelling pilot which evaluated menu items based on per 100g standard, the weight and total GHG emissions for each dish were also calculated by summing up the weight and GHG emissions for each ingredient and then calculated the GHG emissions per 100g of food. The calculation of the nitrogen footprint and water footprint per 100g for each menu item follows the same process with the resective footprints.

2.5 EVALUATION FRAMEWORK

The evaluation of menu items is an automatic process that is conducted by an evaluation framework, a workflow documented in Python on Jupyter notebooks that calculates the GHG emission of menu items in an efficient and structured way. It reads the .xml files exported from OC and does most of the calculating process. See Appendix A for the code that constructed the evaluation framework. The process flowchart for the whole evaluation process is shown in *Figure 2*:

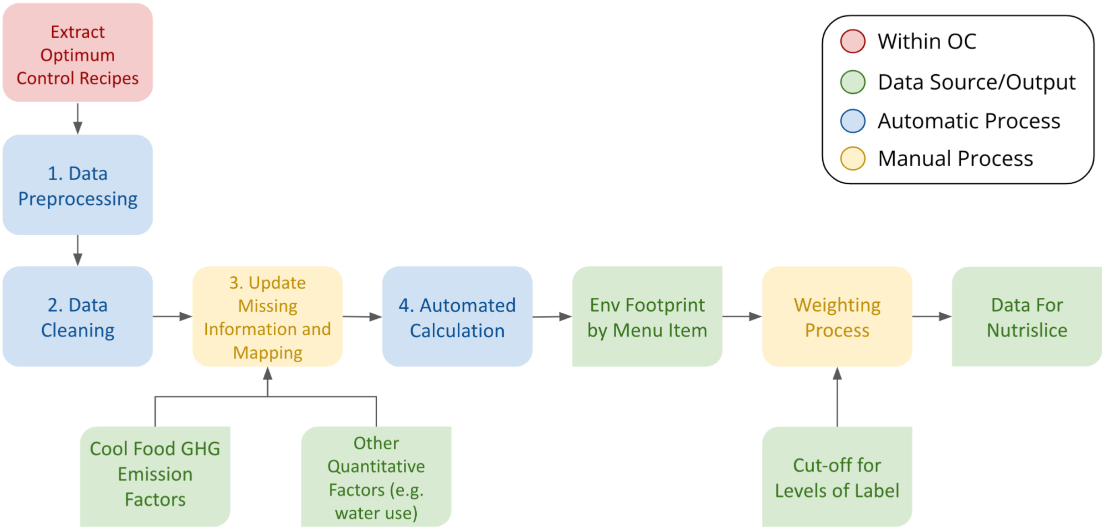


Figure 2: Evaluation Framework Flowchart

This flowchart presents the main steps and components that make up the whole evaluation process. And the color of each box indicates where this step takes place, or which system or software is associated with it. For a box that has two colors, it means it is associated with two systems or can happen in either place.

The first step is extracting raw ingredients and recipe data from the UBCFS inventory management system. Before feeding these raw data into the automated calculation process, it requires preprocessing and cleaning these data by listing and adjusting units for all ingredients and assigning them with associated GHG emission factors, which are from several external data sources such as the Cool Food Calculator. Data extraction from OC and preprocessing represent the largest time requirements every time new recipes need evaluation. Besides GHG emissions, the framework is also able to calculate the total nitrogen and water footprint and per 100g of food for each menu item. After these data gets processed in the automated calculation step/evaluation framework, it will output the environmental footprint of each menu item, and then we weigh these results with other qualitative attributes to have a weighted metric of the overall climate impact of each menu item. Lastly, we use the baseline data to decide the cut-offs for the three levels of labels, and the results can be shown on the Nutrislice, which is the online platform where students can see nutrition facts and also the climate label of the food they buy at UBCFS.

2.6 BASELINE AND LABEL CUT-OFFS

For the Summer Pilot (Phase 1) of the CFFS Label launched at Mercante and the following Fall Pilot (Phase 2) launched at the Open Kitchen, we decided to use the traffic light system to categorize foods by their climate impact into high, medium, and low levels, corresponding to the colors of red, yellow, and green. It would allow easy interpretation for customers to see the food's emission level by looking at the colors. See *Figure 3* for the design and meaning of the labels implemented during the Summer Pilot (Phase 1).

To determine the cut-off levels of the label according to the GHG emissions of menu items, we decided to establish a UBCFS GHG emission baseline that represents the average GHG emissions per dish before the label is launched. In this way, we can set cut-offs in accordance with the 50% UBC CAP 2030 GHG reduction goals for food systems. This requires utilizing the sales and recipe data during a period and then calculating the average GHG emissions per dish. For Phase 1, we decided to have separate sets of cut-offs for different meal groups (i.e., lunch/dinner, breakfast, desserts/snacks) due to the disparity in serving size and main ingredients inspired by the methodology by WRI.

The methods for determining cut-offs for the three levels of the label for the Summer Pilot (Phase 1) are shown below:

- Green: These food items have below-average GHG emissions compared to other food items sold within the same meal category (i.e., lunch/dinner, breakfast, or desserts/snacks) and have low enough emissions to achieve UBC's 50% reduction target in food-related GHG emissions.

- Yellow: These food items have below-average GHG emissions compared to other food items sold within the same meal category (i.e., lunch/dinner, breakfast, or desserts/snacks) but higher emissions than what is necessary to achieve UBC's 50% reduction target in food-related GHG emissions.

- Red: These food items have above-average GHG emissions compared to other food items sold within the same meal category (i.e., lunch/dinner, breakfast, or desserts/snacks). Food with red labels would drive the average GHG emissions higher, thus impeding the process for UBC in achieving the 50% reduction target in food-related GHG emissions.

Figure 3: Phase 1 Traffic Light Labelling System

For Phases 2 and 3, due to the large variety and quantity of meal categories offered by the Open Kitchen, we calculated the GHG emissions per 100g of food for each menu item so that their environmental impacts are comparable between different meal categories no matter the serving size. Correspondingly, the UBCFS GHG emission baseline was also calculated based on per 100g of food to decide label cut-offs for Phase 2 and 3. The methods for determining cut-offs for the three levels of the label for the Fall Pilot (Phase 2) are shown below:

- Green: These food items have below-average GHG emissions per 100g and have low enough emissions to achieve UBC's 50% reduction target in food-related GHG emissions.
- Yellow: These food items have below-average GHG emissions per 100g but have higher emissions than what is necessary to achieve UBC's 50% reduction target in food-related GHG emissions.
- Red: These food items have above-average GHG emissions per 100g of food. Food with red labels would drive the average GHG emissions per 100g higher, thus impeding the process for UBC in achieving the 50% reduction target in food-related GHG emissions.

See *Figure 4* below for the design and meaning of the labels during the Fall Pilot (Phase 2).

Figure 4: Phase 2 Traffic Light Labelling System

For Phase 3 of the CFFS Label, we decided to shift from the traffic-light system to a single label indicating foods that are Climate-Friendly based on a composite metric including GHG emissions, nitrogen footprint, and stress-weighted water use and weigh each attribute equally. See *Figure 5* below for the design of the label during the Spring Pilot (Phase 3).

Figure 5: Phase 3 Single Icon Labelling System

The new methodology includes the calculation of a baseline for nitrogen and water footprint per 100g for UBCFS using 2019 recipes and sales data similar to the GHG emissions baseline. To get a composite metric that measures and weighs the three attributes equally, we divided the GHG emissions, nitrogen footprint, and stress-weighed water use per 100g of food by their corresponding baseline for each menu item. Then we average the three ratios of footprint to baseline to get the standardized metric. For the cut-off deciding which menu item gets the CFFS Label, we chose the threshold of 0.5 that is in accordance with the 50% UBC CAP 2030 GHG reduction goals for food systems. A menu item with the CFFS Label means that this menu item has at least a 50% lower environmental footprint per 100 grams than other items.

2.7 SENSITIVITY ANALYSIS AND DAILY ALLOWANCE VALUE (DV)

To decide between the GHG-only metric and the composite metric on evaluating menu items, we conducted a sensitivity analysis by comparing the labelling results based on the GHG-only metric versus combining GHG, nitrogen, and water use (freshwater or stress-weighed water use) per 100g of food. It allowed us to see whether the incorporation of nitrogen and water footprints substantially changed which items got the label. Specifically, we compared the labeling results in the following steps:

1. The number of items that all 4 attributes (GHG, Nitrogen, Freshwater, Stress-weighed water) result in the same "color" of impact.

2. The number of items that all 3 attributes (GHG, Nitrogen, and freshwater) result in the same "color" of impact.

3. The number of items that all 3 attributes (GHG, Nitrogen, and stress-weighed water) result in the same "color" of impact

4. The number of green, yellow, and red items using the GHG-only metric and the composite metric (GHG, nitrogen, and stress-weighed water use)

5. The number of items that changed label color after incorporating nitrogen and water footprints (i.e., red in GHG-only -> yellow in composite metric)

In addition, to provide students with more CFFS attribute information behind the composite label and the food they purchase at UBCFS for suggesting climate-friendly menu choices, we estimated a Daily Allowance Value (DV) of GHG emissions, nitrogen, and water footprints based on a "climate-friendly'' healthy diet for a day (Leach et al., 2016). Then we can show students the percentage of footprints compared to the DV by consuming 100g of food they purchase. This information can be added to the Nutrislice menu item descriptions in the future.

2.8 ADDITIONAL ATTRIBUTES

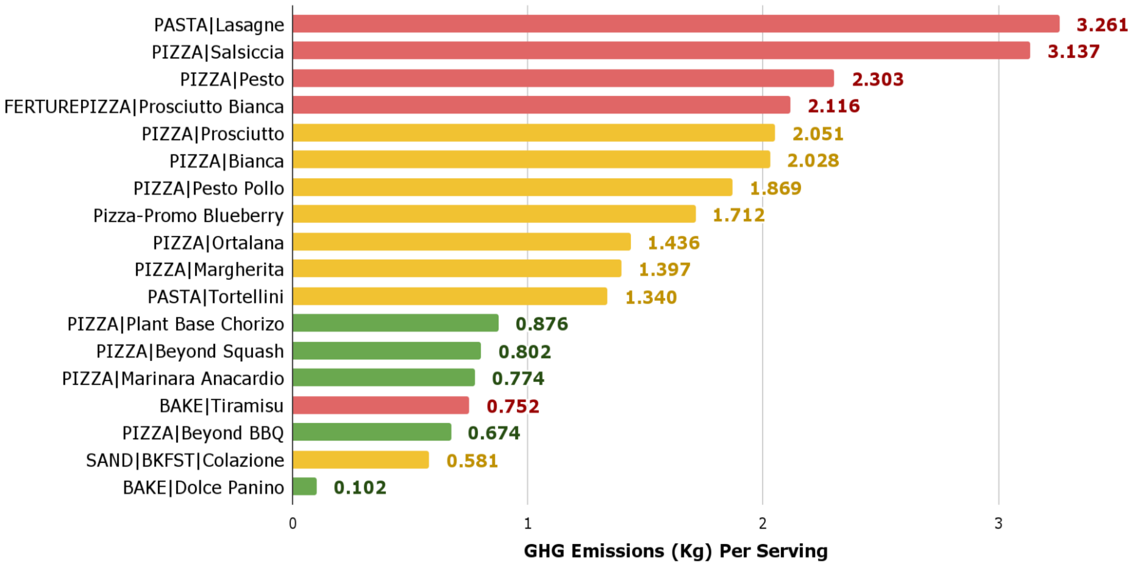
Besides GHG emissions, the evaluation framework also considers the incorporation of new additional attributes for Phase 2 and 3 to produce a more comprehensive CFFS Label. The additional attributes were the metrics to define a Climate-Friendly Food System by the CFFS Action Team, which were developed based on aspects of climate change mitigation and adaptation.

The potential additional attributes are land use, nitrogen pollution, water use, and local, which were developed from the CFFS definition research conducted by the CFFS Communication and Engagement Coordinator. To decide which additional attribute should be incorporated, we evaluated these attributes based on the availability of data, UBCFS's tracking ability/capacity for qualitative attributes, their impact on climate change mitigation and adaptation strategies, and evaluation survey results.

3. RESULTS

3.1 SUMMER PILOT (PHASE 1)

The Summer Pilot (Phase 1) for the operationalization of the CFFS Label took place at the Mercante, one of the UBCFS retail venues that remained open during the summer of 2021. The evaluation only focused on the GHG emissions of the menu items, most of which are pizzas that have almost the same serving size. The total GHG emission for each menu item, calculated by the evaluation framework, is shown in *Figure 6*:



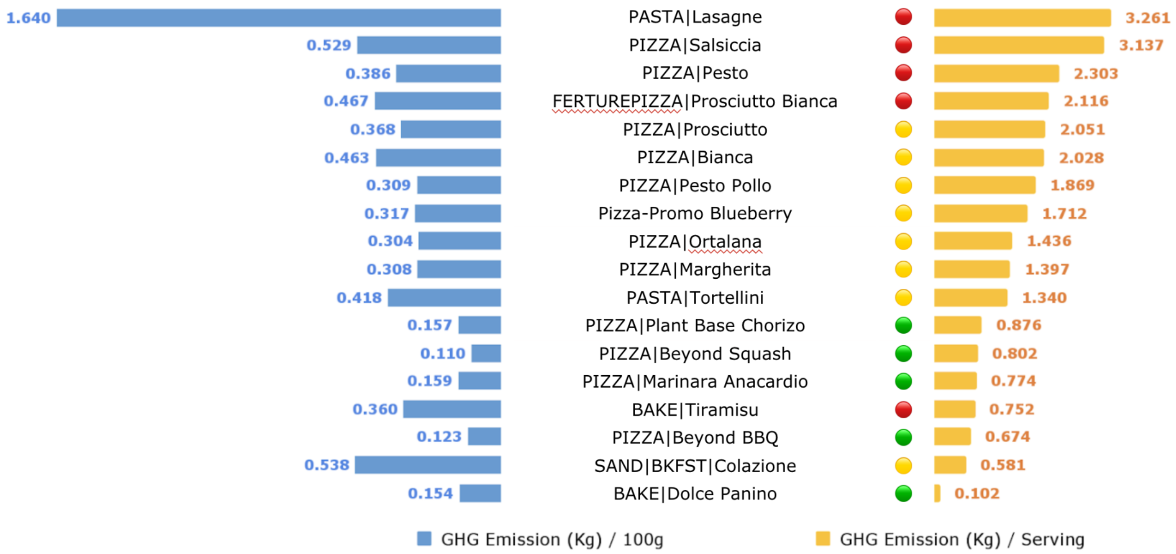
Note: the GHG emission results are based on 2021 data

Figure 6: GHG Emissions (Kg) Per Serving (Phase 1)

The corresponding CFFS Label is available to students on the menu boards and also on Nutrislice. See *Figure 7* for the actual look of labels on display.

Figure 7: CFFS Label on Menu Board (Phase 1)

The evaluation framework also calculated the GHG emissions per 100g of the product for each item. This gives another point of view for comparing the climate impact of the recipes. Although there are a few products that have high per 100 gram GHG emissions, which indicates that they may use a lot of high-emission ingredients, their respective total emissions are low due to the small serving size. For example, the GHG emission per 100g of Colazione is 538g, which is higher than the Salsiccia pizza (529g/100g), but the total GHG emission per serving of Colazione is about only ⅙ of the GHG emissions of a Salsiccia pizza. To make the label easier for interpretation by the customer and align with the goal of reducing total GHG emissions, we chose to assign labels based on total GHG emissions per serving of the products instead of per 100g, see *Figure 8*.



Note: the GHG emission results are based on 2021 data

Figure 8: GHG Emissions (Kg) Per Serving vs. Per 100g (Phase 1)

The label cut-offs for the Summer Pilot (Phase 1) are shown in *Figure 9*. GHGs are evaluated based on meal categories (lunch/dinner, breakfast, or desserts/snacks). Menu items are categorized as green, yellow, or red, depending on whether they have below or above average GHG emissions compared to other food items sold at Mercante within the same meal category. The categories also consider if food items have low enough emissions to achieve UBC's food emissions targets.

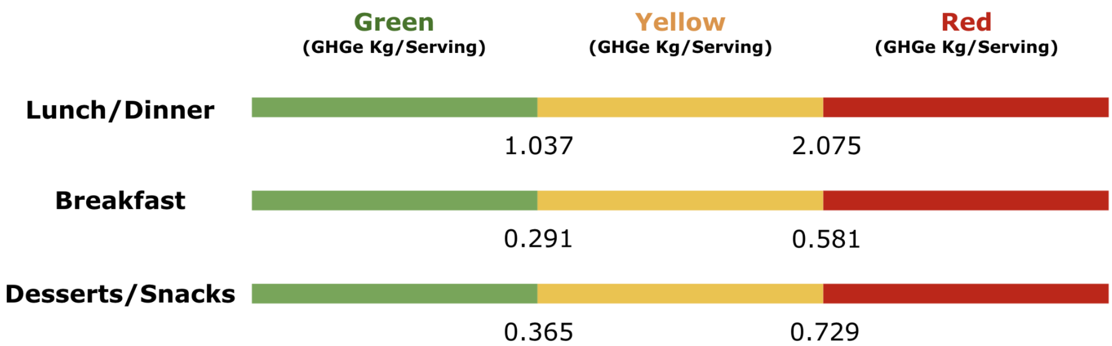


Figure 9: Phase 1 Label Cut-offs

3.2 FALL PILOT (PHASE 2)

The Fall Pilot (Phase 2) for the operationalization of the CFFS Label continued at Mercante and also expanded to Open Kitchen, which is one of the three major UBCFS residence dining halls open during the 2021-2022 academic year. Due to the increased variety and quantity of food groups offered by Open Kitchen, we decided to assign the CFFS Label based on the amount of GHG emissions per 100 grams of food. This allows us to compare the GHG emissions of foods with different serving sizes and ingredient components using a single set of thresholds. The weight, GHG emissions per serving of food, and GHG emissions per 100 grams of food were calculated by the evaluation framework for assigning labels.

To determine the label cut-offs, we updated the UBCFS GHG emissions baseline using 2019 sales and recipes data. See *Table 1* for the estimated baseline for each student residence dining room and UBCFS in total:

Location	19-20 Baseline GHG Emissions	UBC 19-20 GHG Baseline
Open Kitchen	462.81g/per 100g	360.25g/per 100g
Totem	311.25g/per 100g	
Gather	306.96g/per 100g	
Mercante	416.71g/per 100g	

Table 1: UBC 19-20 GHG Emissions Baseline (Phase 2)

The label cut-offs for the Fall Pilot (Phase 2) are shown in *Figure 10* below. By comparing the mean and medium GHG emissions per 100g of menu items offered at Open Kitchen during the 2021-2022 academic year, we can see that the mean GHG emissions per 100g (369.3g CO_2eq) is above the baseline GHG emissions. Therefore, applying the CFFS Label on menu items would likely reduce the average GHG emissions if the label is effective in reducing the purchase of red label menu items. The median GHG emissions per 100g is 232.1g CO_2eq, roughly halfway in between the 2 cut-off points that lie in the yellow range. Thus, the label cut-offs would make the quantity of menu items that are labeled as green, yellow, and red approximately the same.

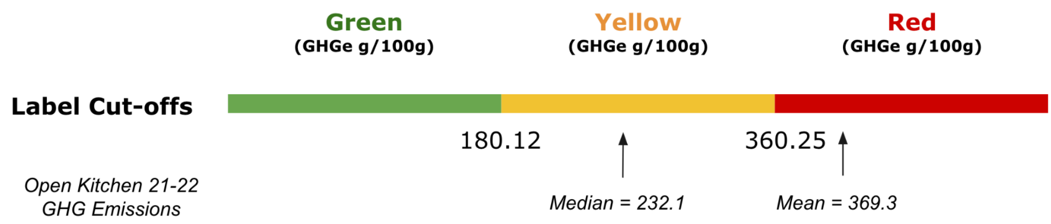
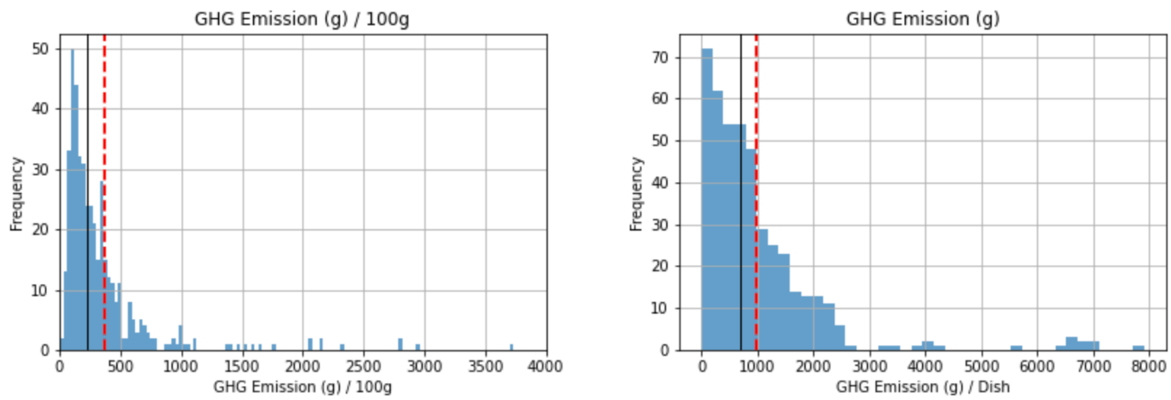


Figure 10: Phase 2 Label Cut-offs

The plots in *Figure 11* show the distribution of total GHG emissions and GHG emissions per 100g of menu items at Open Kitchen during the 2021-2022 academic year. The plots also indicate the mean and median GHG emissions per dish and per 100g of food.



Red line (mean): 369.3g/100g
Black line (median): 232.1g/100g

Red line (mean): 986.2g/dish
Black line (median): 693.7g/dish

Figure 11: GHG Emissions (g) Per Serving vs. Per 100g (Phase 2)

3.3 SPRING PILOT (PHASE 3)

The Spring Pilot (Phase 3) for the operationalization of the CFFS Label continued at Open Kitchen after evaluating the effects of the CFFS Label on students' behavior during the Fall Pilot (Phase 2). This time, the CFFS research teams decided to simplify the label system by only applying the CFFS Label to menu items that fit the criteria of climate-friendliness. While we keep evaluating the footprints of menu items on a per 100g basis, we incorporated the nitrogen and water footprints into consideration after the sensitivity analysis and produced a composite label metric to decide which menu item gets the CFFS Label assigned. The steps 1-3 of the sensitivity analysis allow us to see which attribute should be incorporated by checking the number of items that the combination of attributes (GHG, Nitrogen, and Water) results in the same "color" of impact. See *Table 2* for the results of the sensitivity analysis steps 1-3:

Attributes Combination	Count	Percentage
GHG, Nitrogen, Freshwater, Stress-weighed Water	175	28.69%
GHG, Nitrogen, Freshwater	215	35.25%
GHG, Nitrogen, Stress-weighed Water	214	35.08%

Table 2: Sensitivity Analysis 1-3 Results

The above results suggest that choosing between freshwater and stress-weighed water use results in more consistency in the "color" of impact. And we chose the stress-weighed water use because it takes water scarcity into account. Therefore, the composite metric includes the three attributes: GHG, nitrogen, and stress-weighed water use. The results of the sensitivity analysis steps 4-5 are shown in *Table 3* below:

Metric-Color	Count
GHG-Red	158
GHG-Yellow	204
GHG-Green	248
Composite-Red	189
Composite-Yellow	175
Composite-Green	246

Table 3: Label Counts in GHG-only and Composite Metric

According to *Table 3*, the distribution of label colors is balanced under both the GHG-only and the composite metric. And *Table 4* indicates that the total number of menu items that changed color after incorporating nitrogen and water footprints is small (13.93%).

GHG -> Composite	Count
Red -> Yellow	7

Red -> Green	2
Yellow -> Green	16
Green -> Yellow	20
Green -> Red	0
Yellow -> Red	40
Total Changed	**85 (13.93%)**

Table 4: Label Change in GHG-only and Composite Metric

The estimated UBC 19-20 baseline for GHG emissions, nitrogen footprint, and water footprints per 100g of food for the composite metric is shown in *Table 5* below:

Attribute	UBC 19-20 Baseline
GHG Emissions	381.13 g/per 100g[4]
Nitrogen	4.21 g/per 100g
Fresh Water	47.16 L/per 100g
Stress-weighed Water	1501.2 L/per 100g

Table 5: UBC 19-20 GHG, Nitrogen, Water Footprints Baselines (Phase 3)

Using the estimated baseline above and calculating a composite metric for each menu item, there are about one-third (246) of the menu items classified as Climate-Friendly (green in the composite label). Almost the same proportion of green labels as in the Fall Pilot (Phase 2).

The estimation of the Daily Allowance Value (DV) for each attribute in the composite metric is calculated using the 50% of UBC 19-20 Baseline to define as Climate-Friendly. The estimated DV is based on a standard healthy diet from the Food Label Toolkit by Leach et al. (2016), which is set as 2200 calories per day, see *Table 6* for detailed food components of the healthy diet and *Table 7* for corresponding DV for each CFFS attribute calculated:

[4] The estimated UBC 19-20 GHG emissions baseline in Phase 3 is updated as many preps get units estimated after analyzing the newly-extracted open kitchen data for phase 3, thus the units information database expanded. Many old recipes in 2019 also used these preps so they were included in the Phase 3 baseline estimation but excluded in Phase 2 (Phase 2's baseline is a rough estimate that only included menu items that didn't use preps with odd units).

Food Component	Grams Per Day
Chicken	40
Pork	20
Beef	20
Milk	280
Cheese	30
Eggs	30
Fish	30
Grains	120
Rice	40
Fruits	220
Beans	40
Potatoes	100
Vegetables	200
Nuts	10
Oils	20
Total	**1200**

Table 6: Standard Healthy Diet (2200 Calories Per Day). Source: Food Label Toolkit by Leach et al. (2016)

Attribute	Daily Allowance Value (DV)
GHG Emissions	3037.98g
Nitrogen	41.73g
Fresh Water	434.11L
Stress-weighed Water	11730.57L

Table 7: Daily Allowance Value (DV)

4. DISCUSSION

From the above analysis, we can see that foods that contain ruminant meat and dairy products (i.e. beef, lamb, cheese, etc.) tend to have high GHG emissions, nitrogen and also water footprints per serving, and in particular per 100g. This suggests one way to lower the climate impact of the food system is by reducing the amount of ruminant meat and dairy consumption and switching to plant-based protein products (i.e., beans, tofu, etc.). For example, the difference between the Salsiccia Pizza (the pizza with the highest GHG emissions at Mercante with chorizo, tomato, basil, oregano, and mozzarella) and the Beyond BBQ Pizza (the pizza with the lowest GHG emissions at Mercante with beyond meat crumble, chipotle BBQ sauce, arugula, and mushrooms) is 2,463 grams of CO2eq, which is equivalent to the emissions from an 11.96-kilometer drive in an average passenger vehicle (average of 206g CO2 emissions per km driven, Canada Energy Regulator, 2019).

The evaluation framework is tailored for what UBC has in place, such as the inventory management system OC and Nutrislice. It maximized the efficiency of implementing and evaluating the CFFS Label at UBCFS outlets and various associated displaying platforms to utilize the opportunity of reaching thousands of customers daily with reliable information collected for analysis. The systemized workflow from implementation to evaluation allows the rapid adjustment of the label for better influence on the Climate-Friendly purchasing behavior of customers. On the other hand, the specificity of the evaluation framework to UBCFS's organization and systems may limit the expansion of the CFFS Label to other institutions and places whose recipe management systems are different or there's no systemized way of storing the recipes' data. While the basic methodologies and supporting data can be transferred, the efficiency of expanding the CFFS Label to those places is limited and adjustments to the framework should be considered based on the amount and data structure of the recipes for evaluation.

In addition, the manual process involved in the evaluation process allows for better accuracy and flexibility in adjusting the labelling methodologies and incorporating more comprehensive considerations in evaluating the climate impact of some menu items. While it sacrifices the efficiency and automaticity of the framework and may limit the expansion to larger scale implementation of the CFFS Label.

There are additional limitations of the evaluation framework. First, there are several processed products and packaged foods that are directly purchased from external suppliers, such as sauces, dressings, snacks, etc. Therefore, the evaluation can only approximate their GHG emission factors by estimating the proportions of the ingredients contained in these products.

Secondly, emissions from bucket items such as "parfait," "salad bar," and "build your own" represent an average with a lot of variances since they are customized by the client at the retail venue. The recipes for these products recorded in the system use the estimated average amount for each composition that customers may choose.

Thirdly, there is human dependence on matching items with associated emission factors. Although manually matching takes less time and is more accurate, this may raise some problems if the label is expanded to more food venues and thus human work will take more time. Besides, the information for ingredients stored in OC is incomplete for some items, such as the unit information and conversion data, which needs to be adjusted and inserted manually.

Lastly, the relevance of this methodology and effort rests on the effectiveness of the label in educating and ultimately changing consumer choice among the UBC population eating food sold on campus. Establishing these methods has been the result of countless meetings and engagement with stakeholders and experts from the UBC community and beyond. Implementation and fine-tuning of the methodology has taken more than 1000 hours of work and does not take into account the time committed by the stakeholders who gave feedback and provided valuable data. These aspects need to be taken into account and compared to the effectiveness of improving the climate friendliness of food eaten at UBC primarily through supply-side solutions, i.e., the food procured and offered at UBC—which includes UBCFS and multiple other venues.

5. RECOMMENDATIONS

The steps outlined below could be taken for future development and expansion of the CFFS Labelling pilot to improve the evaluation framework and make it more resilient and suitable for expanded operations.

5.1 SHORT-TERM RECOMMENDATIONS (< 3 MONTHS)

Data Analysis Aspect:

- Integrate the CFFS attribute weighing process into the evaluation framework to automate the process.
- Conduct research on introducing biodiversity as a new CFFS attribute for evaluating menu items.

Operational Work Aspect:

- Display the carbon, nitrogen, and water footprint DV and/or percentile per dish and/or per 100g of food on Nutrislice menu item descriptions.
- Evaluate the benefit of displaying the CFFS Label on the Nutrislice Platform with the all-inclusive dining services.
- Evaluate the possibility and interest of adding new attributes to the CFFS Label calculation.

5.2 MID-TERM RECOMMENDATIONS (< 6 MONTHS)

Data Analysis Aspect:

- Improve the recording and tracking of food information stored in the inventory management system and reduce the amount of missing data for ingredients and recipes.
- Incorporate the climate footprint data for ingredients into the inventory management system if feasible to embed the calculation process within the system.
- Streamline the process of data extraction from UBCFS and data output to display CFFS Label information on Nutrislice.
- Conduct research on the GHG emissions from the cooking process for improved reporting and evaluation of menu items.

- Conduct research on the water use footprint from the prepping and cooking process for improved reporting and evaluation of menu items.

Operational Work Aspect:

- Implement the CFFS Label at all student residence dining halls managed by UBCFS.
- Have other research to support the expansion of the CFFS Label to other food outlets at UBC

5.3 LONG-TERM RECOMMENDATIONS (< 1 YEAR)

Data Analysis Aspect:

- UBC can lead the engagement process to build a Pacific Northwest/Canadian specific GHGe, nitrogen, and water footprint factors database by conducting research together with peer institutions. This can also help to improve the accuracy and specificity of current labels.

- Collaborate closely with peer institutions to have a standardized method of menu item evaluation and labeling criteria across different universities and institutions in North America (e.g, Oxford's Health Behaviors team at the Livestock, Environment and People (LEAP) project).

Operational Work Aspect:

- Evaluate the possibility of adding the CFFS Label to UBC delivery menus.
- Expand the CFFS Label implementation to other UBC campuses (i.e., UBC Okanagan).

6. CONCLUSION

In conclusion, the CFFS Label evaluation framework is a resilient approach to conducting the evaluation process in an efficient and structured way that meets the needs for the future expansion of the CFFS Label across UBCFS. However, there are a few limitations in the current framework due to the tailored design of the framework to UBCFS and the manual reliance on cleaning, assigning, and extracting data. The recommendation for the next steps is to streamline the extraction process and improve the tracking of ingredient information in the systems. It will require more time, resourcing, and close coordination between associated departments to produce a comprehensive CFFS Label that indicates all-around information on the climate impact of menu items sold by UBCFS.

REFERENCES

Brunner, F., Kurz, V., Bryngelsson, D., & Hedenus, F. (2018). Carbon Label at a University Restaurant – Label Implementation and Evaluation. *Ecological Economics*, *146*, 658–667. https://doi.org/10.1016/j.ecolecon.2017.12.012

Crippa, M., Solazzo, E., Guizzardi, D., Monforti-Ferrario, F., Tubiello, F. N., & Leip, A. (2021). Food systems are responsible for a third of global anthropogenic GHG emissions. *Nature Food*, *2*(3), 198–209. https://doi.org/10.1038/s43016-021-00225-9

Leach, A. M., Emery, K. A., Gephart, J., Davis, K. F., Erisman, J. W., Leip, A., Pace, M. L., D'Odorico, P., Carr, J., Noll, L. C., Castner, E., & Galloway, J. N. (2016). Environmental impact food labels combining carbon, nitrogen, and water footprints. *Food Policy*, *61*, 213–223. https://doi.org/10.1016/j.foodpol.2016.03.006

Poore, J., & Nemecek, T. (2018). Reducing food's environmental impacts through producers and consumers. *Science*, *360*(6392), 987–992. https://doi.org/10.1126/science.aaq0216

Searchinger, T., Waite, R., Hanson, C., Ranganathan, J., & Matthews, E. (2019). Creating a Sustainable Food Future—A Menu of Solutions to Feed Nearly 10 Billion People by 2050 (Final Report). *World Resources Institute*. https://www.wri.org/research/creating-sustainable-food-future

Waite, R., Vennard, D., & Pozzi, G. (2019). Tracking Progress Toward the Cool Food Pledge: Setting Climate Targets, Tracking Metrics, Using the Cool Food Calculator, and Related Guidance for Pledge Signatories (Technical Note). *World Resources Institute*. https://www.wri.org/research/tracking-progress-toward-cool-food-pledge

APPENDICES

APPENDIX A [CODE FOR EVALUATION FRAMEWORK]

Climate-Friendly Food Systems (CFFS) Labelling Project

The University of British Columbia

Created by Silvia Huang, CFFS Data Analyst

Part I: Data Preprocessing

Set up and Import Libraries

In [1]:
```
#pip install -r requirements.txt
```

In [2]:
```
import numpy as np
import pandas as pd
import pdpipe as pdp
import matplotlib.pyplot as plt
import glob
import os
import csv
from itertools import islice
from decimal import Decimal
import xml.etree.ElementTree as et
from xml.etree.ElementTree import parse
import openpyxl
import pytest
```

In [3]:
```
# Set the root path, change the the current working directory into the project folder
path = "/Users/silvia/cffs-label"
os.chdir(path)
```

In [4]:
```
# Enable reading data table in the scrolling window if you prefer
#pd.set_option("display.max_rows", None, "display.max_columns", None)
```

Load Data Files

Set Data File Path

In [5]:
```
# Select data file path for the chosen venue and time range where the recipes data stored
filepath_list = glob.glob(os.path.join(os.getcwd(), "data", "raw", "OK 21-22", "*.oc"))
filepath_list
```

Out[5]:
```
['/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06182021_0938.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06232021_0918.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06182021_0918.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06232021_1141.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06182021_1001.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06232021_1155.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06182021_0927.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06232021_0956.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06182021_1202.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06232021_1111.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06182021_0933.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06232021_1150.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06232021_0951.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06182021_1033.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06182021_1007.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06182021_0944.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/OK Oct 22 Request.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06182021_1026.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22/IPR_Export_06232021_0944.oc']
```

Import Items List

In [6]:
```
# Read items.xml files in the filepath_list and construct a dataframe
ItemId = []
Description = []
CaseQty = []
CaseUOM = []
PakQty = []
PakUOM = []
InventoryGroup = []

for filepath in filepath_list:
```

```
    path = filepath + '/items.xml'
    if os.path.isfile(path):
        xtree = et.parse(path)
        xroot = xtree.getroot()
        for item in xtree.iterfind('Item'):
            ItemId.append(item.attrib['id'])
            Description.append(item.findtext('Description'))
            CaseQty.append(item.findtext('CaseQty'))
            CaseUOM.append(item.findtext('CaseUOM'))
            PakQty.append(item.findtext('PakQty'))
            PakUOM.append(item.findtext('PakUOM'))
            InventoryGroup.append(item.findtext('InventoryGroup'))


Items = pd.DataFrame({'ItemId': ItemId, 'Description': Description, 'CaseQty': CaseQty,
                      'CaseUOM': CaseUOM, 'PakQty': PakQty, 'PakUOM': PakUOM, 'InventoryGroup': InventoryGroup}
                    ).drop_duplicates()

Items.reset_index(drop=True, inplace=True)
```

In [7]:
```
Items
```

Out[7]:

	ItemId	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup
0	I-4271	APPLES GRANNY SMITH	113.000	ea	1.000	CT	PRODUCE
1	I-4971	ARTICHOKE 1/4 SALAD CUT TFC	6.000	LG CAN	2.500	Kg	PRODUCE
2	I-2305	BACON PANCETTA	1.000	Kg	1.000	Kg	MEAT
3	I-1207	BAGUETTE FRENCH	24.000	each	1.000	CT	BREAD
4	I-17203	BALSAMIC GLAZE	2.000	bottle	2.000	L	FOOD - GROCERY
...	...	...	...	...	...	...	...
593	I-18915	SPRING ROLL VEG	48.000	ea	1.000	ea	FOOD - GROCERY
594	I-4903	SQUASH SPAGHETTI 36 LBS US	35.000	lb	1.000	lb	PRODUCE
595	I-28907	STEAMED BUN BBQ PORK	60.000	ea	1.000	each	FOOD - GROCERY
596	I-28920	STICKY RICE WRAP	16.000	bag	3.000	each	FOOD - GROCERY
597	I-61420	TOFU FIRM MEDIUM	12.000	pak	454.000	g	FOOD - GROCERY

598 rows × 7 columns

In [8]:
```
Items.shape
```

Out[8]:
```
(598, 7)
```

In [9]:
```
Items.dtypes
```

Out[9]:
```
ItemId            object
Description       object
CaseQty           object
CaseUOM           object
PakQty            object
PakUOM            object
InventoryGroup    object
dtype: object
```

In [10]:
```
# Save the dataframe to csv
path = os.path.join(os.getcwd(), "data", "preprocessed", "Items_List.csv")
Items.to_csv(path, index = False, header = True)
```

Import Ingredients List

In [11]:
```
# Read ingredients.xml files in the filepath_list and construct a dataframe
IngredientId = []
Conversion = []
InvFactor = []
Qty = []
Recipe = []
Uom = []

for filepath in filepath_list:
    path = filepath + '/Ingredients.xml'
    if os.path.isfile(path):
        xtree = et.parse(path)
        xroot = xtree.getroot()
        for x in xtree.iterfind('Ingredient'):
            IngredientId.append(x.attrib['ingredient'])
            Conversion.append(x.attrib['conversion'])
            InvFactor.append(x.attrib['invFactor'])
            Qty.append(x.attrib['qty'])
```

```
            Recipe.append(x.attrib['recipe'])
            Uom.append(x.attrib['uom'])

Ingredients = pd.DataFrame({'IngredientId': IngredientId, 'Qty': Qty,'Uom': Uom, 'Conversion': Conversion,
                      'InvFactor': InvFactor,'Recipe': Recipe}).drop_duplicates()

Ingredients.reset_index(drop=True, inplace=True)
```

In [12]:
```
Ingredients
```

Out[12]:

	IngredientId	Qty	Uom	Conversion	InvFactor	Recipe
0	P-18746	1.000	Kg	1.00000000	1.0000	P-10241
1	I-3388	1.000	L	1.00000000	0.3058	P-10496
2	I-4660	2.270	Kg	2.20462000	0.6942	P-10496
3	I-3451	2.560	L	1.00000000	1.2800	P-13933
4	I-4679	1.000	BUNCH	1.00000000	0.0063	P-18318
...	...	...	...	...	...	...
5377	P-26143	170.000	g	0.00100000	1.0000	R-62022
5378	P-26225	140.000	g	0.00220462	1.0000	R-62022
5379	P-50428	3.000	g	1.00000000	1.0000	R-62022
5380	P-56712	180.000	g	0.00100000	1.0000	R-62022
5381	P-56927	90.000	g	0.00100000	1.0000	R-62022

5382 rows × 6 columns

In [13]:
```
Ingredients.shape
```

Out[13]:
```
(5382, 6)
```

In [14]:
```
Ingredients.dtypes
```

Out[14]:
```
IngredientId    object
Qty             object
Uom             object
Conversion      object
InvFactor       object
Recipe          object
dtype: object
```

In [15]:
```
# Save the dataframe to csv
path = os.path.join(os.getcwd(), "data", "preprocessed", "Ingredients_List.csv")
Ingredients.to_csv(path, index = False, header = True)
```

Import Preps List

In [16]:
```
# Read preps.xml files in the filepath_list and construct a dataframe
PrepId = []
Description = []
PakQty = []
PakUOM = []
InventoryGroup = []

for filepath in filepath_list:
    path = filepath + '/Preps.xml'
    if os.path.isfile(path):
        xtree = et.parse(path)
        xroot = xtree.getroot()
        for x in xtree.iterfind('Prep'):
            PrepId.append(x.attrib['id'])
            Description.append(x.findtext('Description'))
            PakQty.append(x.findtext('PakQty'))
            PakUOM.append(x.findtext('PakUOM'))
            InventoryGroup.append(x.findtext('InventoryGroup'))

Preps = pd.DataFrame({'PrepId': PrepId, 'Description': Description,
                  'PakQty': PakQty, 'PakUOM':PakUOM, 'InventoryGroup': InventoryGroup}).drop_duplicates()

Preps.reset_index(drop=True, inplace=True)
```

In [17]:
```
Preps
```

Out[17]:

	PrepId	Description	PakQty	PakUOM	InventoryGroup
0	P-55516	BAKED|Lasagna|Spin Mushroom	5.550	Kg	
1	P-54666	BAKED|Pasta|Chicken Alfredo	6.176	Kg	

	PrepId	Description	PakQty	PakUOM	InventoryGroup
2	P-54664	BAKED|Pasta|Chorizo Penne	7.360	Kg	
3	P-56502	BAKED|Pasta|Shrimp Pesto	5.760	Kg	
4	P-56433	BATCH|Shrimp Remoulade	1.600	Kg	
...	...	...	...	...	...
748	P-47418	MIX|Cheese	2.000	Kg	PREP
749	P-42317	ROASTED|Spaghetti Squash	1.400	Kg	
750	P-56927	SAUTE|Cauliflower Rice	1.000	Kg	
751	P-56887	YIELD|Grated Pear	800.000	g	
752	P-46509	YIELD|Lettuce bun	3.000	PTN	

753 rows × 5 columns

In [18]:
```
Preps.shape
```

Out[18]:
```
(753, 5)
```

In [19]:
```
Preps.dtypes
```

Out[19]:
```
PrepId            object
Description       object
PakQty            object
PakUOM            object
InventoryGroup    object
dtype: object
```

In [20]:
```
# Save the dataframe to csv
path = os.path.join(os.getcwd(), "data", "preprocessed", "Preps_List.csv")
Preps.to_csv(path, index = False, header = True)
```

Import Products List

In [21]:
```
# Read products.xml files in the filepath_list and construct a dataframe
ProdId = []
Description = []
SalesGroup = []

for filepath in filepath_list:
    path = filepath + '/Products.xml'
    if os.path.isfile(path):
        xtree = et.parse(path)
        xroot = xtree.getroot()
        for x in xtree.iterfind('Prod'):
            ProdId.append(x.attrib['id'])
            Description.append(x.findtext('Description'))
            SalesGroup.append(x.findtext('SalesGroup'))

Products = pd.DataFrame({'ProdId': ProdId, 'Description': Description, 'SalesGroup': SalesGroup}).drop_duplicates()

Products.reset_index(drop=True, inplace=True)
```

In [22]:
```
Products
```

Out[22]:

	ProdId	Description	SalesGroup
0	R-61778	ALF|Flatbread|4 Cheese	OK - AL FORNO
1	R-61780	ALF|Flatbread|Apple & Pancetta	OK - AL FORNO
2	R-61749	ALF|Flatbread|BBQ Chicken	OK - AL FORNO
3	R-50859	ALF|Flatbread|Bruschetta	OK - AL FORNO
4	R-50788	ALF|Flatbread|Caprese	OK - AL FORNO
...	...	...	...
453	R-57815	SQR|Tofu Sofrito Quesadilla +1	OK - SQUARE MEAL
454	R-61679	SQR|Tofu Sofrito Quesadilla +2	OK - SQUARE MEAL
455	R-56902	SQR|Vegan Lettuce Wrap	OK - SQUARE MEAL
456	R-57810	SQR|Vegan Lettuce Wrap +1	OK - SQUARE MEAL
457	R-57811	SQR|Vegan Lettuce Wrap +2	OK - SQUARE MEAL

458 rows × 3 columns

In [23]:
```
Products.shape
```
Out[23]:
```
(458, 3)
```
In [24]:
```
Products.dtypes
```
Out[24]:
```
ProdId         object
Description    object
SalesGroup     object
dtype: object
```
In [25]:
```
# Save the dataframe to csv
path = os.path.join(os.getcwd(), "data", "preprocessed", "Products_List.csv")
Products.to_csv(path, index = False, header = True)
```

Import Conversions List

In [26]:
```
# Read conventions.xml files in the filepath_list and construct a dataframe
ConversionId = []
Multiplier = []
ConvertFromQty = []
ConvertFromUom = []
ConvertToQty = []
ConvertToUom = []

for filepath in filepath_list:
    path = filepath + '/Conversions.xml'
    if os.path.isfile(path):
        xtree = et.parse(path)
        xroot = xtree.getroot()
        for x in xtree.iterfind('Conversion'):
            ConversionId.append(x.attrib['id'])
            Multiplier.append(x.attrib['multiplier'])
            ConvertFromQty.append(x.find('ConvertFrom').attrib['qty'])
            ConvertFromUom.append(x.find('ConvertFrom').attrib['uom'])
            ConvertToQty.append(x.find('ConvertTo').attrib['qty'])
            ConvertToUom.append(x.find('ConvertTo').attrib['uom'])


Conversions = pd.DataFrame({'ConversionId': ConversionId, 'Multiplier': Multiplier, 'ConvertFromQty': ConvertFromQty,
                            'ConvertFromUom': ConvertFromUom, 'ConvertToQty': ConvertToQty, 'ConvertToUom': ConvertToUom}
                           ).drop_duplicates()

Conversions.reset_index(drop=True, inplace=True)
```
In [27]:
```
Conversions
```
Out[27]:

	ConversionId	Multiplier	ConvertFromQty	ConvertFromUom	ConvertToQty	ConvertToUom
0		1.00000000	1.0000	XXX	1.0000	L
1		0.87719298	1.0000	1.14L	1.1400	L
2		0.66666667	1.0000	1.5L	1.5000	L
3		0.57142857	1.0000	1.75 L	1.7500	L
4		0.50000000	1.0000	2L	2.0000	L
...	...	...	...	...	...	...
291	I-3634	0.32258065	1.0000	Tbsp	3.1000	g
292	I-3390	0.22222222	1.0000	tsp	4.5000	g
293	I-3390	0.07407407	1.0000	Tbsp	13.5000	g
294	I-3390	0.00462963	1.0000	cup	216.0000	g
295	I-25492	0.00495050	1.0000	ea	202.0000	g

296 rows × 6 columns

In [28]:
```
Conversions.shape
```
Out[28]:
```
(296, 6)
```
In [29]:
```
Conversions.dtypes
```
Out[29]:
```
ConversionId      object
Multiplier        object
ConvertFromQty    object
ConvertFromUom    object
ConvertToQty      object
ConvertToUom      object
dtype: object
```

In [30]:
```
# Save the dataframe to csv
path = os.path.join(os.getcwd(), "data", "preprocessed", "Conversions_List.csv")
Conversions.to_csv(path, index = False, header = True)
```

Data Summary

In [31]:
```
# Summary of raw data imported for evaluation
datasum = pd.DataFrame([Items.shape, Preps.shape, Ingredients.shape, Products.shape, Conversions.shape],
                       columns = ['count', 'columns'],
                       index = ['Items', 'Preps', 'Ingredients', 'Products', 'Conversions'])
datasum
```

Out[31]:

	count	columns
Items	598	7
Preps	753	5
Ingredients	5382	6
Products	458	3
Conversions	296	6

Climate-Friendly Food Systems (CFFS) Labelling Project

The University of British Columbia

Created by Silvia Huang, CFFS Data Analyst

Part II: Data Cleaning

Set up and Import Libraries

In [1]:
```
#pip install -r requirements.txt
```

In [2]:
```
import numpy as np
import pandas as pd
import pdpipe as pdp
import matplotlib.pyplot as plt
import glob
import os
import csv
from itertools import islice
from decimal import Decimal
import xml.etree.ElementTree as et
from xml.etree.ElementTree import parse
import openpyxl
import pytest
from datetime import datetime
```

In [3]:
```
# Set the root path, change the the current working directory into the project folder
path = "/Users/silvia/cffs-label"
os.chdir(path)
```

In [4]:
```
# Enable reading data table in the scrolling window if you prefer
#pd.set_option("display.max_rows", None, "display.max_columns", None)
```

Import Preprocessed Datasets

In [5]:
```
# Read Items_List.csv
Items = pd.read_csv(os.path.join(os.getcwd(), "data", "preprocessed", "Items_List.csv"))
Items.dtypes
```

Out[5]:
```
ItemId            object
Description       object
CaseQty          float64
CaseUOM           object
PakQty           float64
PakUOM            object
InventoryGroup    object
dtype: object
```

In [6]:
```
Items.head()
```

Out[6]:

	ItemId	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup
0	I-4271	APPLES GRANNY SMITH	113.0	ea	1.0	CT	PRODUCE
1	I-4971	ARTICHOKE 1/4 SALAD CUT TFC	6.0	LG CAN	2.5	Kg	PRODUCE
2	I-2305	BACON PANCETTA	1.0	Kg	1.0	Kg	MEAT
3	I-1207	BAGUETTE FRENCH	24.0	each	1.0	CT	BREAD
4	I-17203	BALSAMIC GLAZE	2.0	bottle	2.0	L	FOOD - GROCERY

In [7]:
```
Items.shape
```

Out[7]:
```
(598, 7)
```

In [8]:
```
# Read Ingredients_List.csv
Ingredients = pd.read_csv(os.path.join(os.getcwd(), "data", "preprocessed", "Ingredients_List.csv"))
Ingredients.dtypes
```

Out[8]:
```
IngredientId     object
Qty             float64
Uom              object
Conversion      float64
```

```
InvFactor        float64
Recipe            object
dtype: object
```

In [9]:
```
Ingredients.head()
```

Out[9]:

	IngredientId	Qty	Uom	Conversion	InvFactor	Recipe
0	P-18746	1.00	Kg	1.00000	1.0000	P-10241
1	I-3388	1.00	L	1.00000	0.3058	P-10496
2	I-4660	2.27	Kg	2.20462	0.6942	P-10496
3	I-3451	2.56	L	1.00000	1.2800	P-13933
4	I-4679	1.00	BUNCH	1.00000	0.0063	P-18318

In [10]:
```
Ingredients.shape
```

Out[10]:
```
(5382, 6)
```

In [11]:
```
# Read Preps_List.csv
Preps = pd.read_csv(os.path.join(os.getcwd(), "data", "preprocessed", "Preps_List.csv"))
Preps.dtypes
```

Out[11]:
```
PrepId             object
Description        object
PakQty            float64
PakUOM             object
InventoryGroup     object
dtype: object
```

In [12]:
```
Preps.head()
```

Out[12]:

	PrepId	Description	PakQty	PakUOM	InventoryGroup
0	P-55516	BAKED|Lasagna|Spin Mushroom	5.550	Kg	NaN
1	P-54666	BAKED|Pasta|Chicken Alfredo	6.176	Kg	NaN
2	P-54664	BAKED|Pasta|Chorizo Penne	7.360	Kg	NaN
3	P-56502	BAKED|Pasta|Shrimp Pesto	5.760	Kg	NaN
4	P-56433	BATCH|Shrimp Remoulade	1.600	Kg	NaN

In [13]:
```
Preps.shape
```

Out[13]:
```
(753, 5)
```

In [14]:
```
# Read Product_List.csv
Products = pd.read_csv(os.path.join(os.getcwd(), "data", "preprocessed", "Products_List.csv"))
Products.dtypes
```

Out[14]:
```
ProdId         object
Description    object
SalesGroup     object
dtype: object
```

In [15]:
```
Products.head()
```

Out[15]:

	ProdId	Description	SalesGroup
0	R-61778	ALF|Flatbread|4 Cheese	OK - AL FORNO
1	R-61780	ALF|Flatbread|Apple & Pancetta	OK - AL FORNO
2	R-61749	ALF|Flatbread|BBQ Chicken	OK - AL FORNO
3	R-50859	ALF|Flatbread|Bruschetta	OK - AL FORNO
4	R-50788	ALF|Flatbread|Caprese	OK - AL FORNO

In [16]:
```
Products.shape
```

Out[16]:
```
(458, 3)
```

In [17]:
```
Conversions = pd.read_csv(os.path.join(os.getcwd(), "data", "preprocessed", "Conversions_List.csv"))
Conversions.dtypes
```

Out[17]:
```
ConversionId       object
Multiplier        float64
ConvertFromQty    float64
ConvertFromUom     object
ConvertToQty      float64
```

```
ConvertToUom        object
dtype: object
```

In [18]:
```
Conversions.head()
```

Out[18]:

	ConversionId	Multiplier	ConvertFromQty	ConvertFromUom	ConvertToQty	ConvertToUom
0	NaN	1.000000	1.0	XXX	1.00	L
1	NaN	0.877193	1.0	1.14L	1.14	L
2	NaN	0.666667	1.0	1.5L	1.50	L
3	NaN	0.571429	1.0	1.75 L	1.75	L
4	NaN	0.500000	1.0	2L	2.00	L

In [19]:
```
Conversions.shape
```

Out[19]: (296, 6)

Update Conversion List

In [20]:
```
# Add the specific conversion info from the newly-processed data to a unit conversion database
Update_Conv = pd.read_csv(os.path.join(os.getcwd(), "data", "cleaning", "update", "Conv_UpdateConv.csv"))
Update_Conv
```

Out[20]:

	ConversionId	Multiplier	ConvertFromQty	ConvertFromUom	ConvertToQty	ConvertToUom
0	I-1028	0.008333	1.0	CT	120.0	g
1	I-1034	0.008333	1.0	CT	120.0	g
2	I-1035	0.010000	1.0	CT	100.0	g
3	I-10605	0.008850	1.0	CT	113.0	g
4	I-1126	0.006667	1.0	CT	150.0	g
...	...	...	...	...	...	...
310	P-32664	0.016393	1.0	each	61.0	g
311	P-55707	0.005405	1.0	PTN	185.0	g
312	P-55709	0.005405	1.0	PTN	185.0	g
313	P-62293	0.005405	1.0	PTN	185.0	g
314	P-62023	0.006452	1.0	ROLL	155.0	g

315 rows × 6 columns

In [21]:
```
for index, row in Update_Conv.iterrows():
    Id = Update_Conv.loc[index, 'ConversionId']
    Conversions.drop(Conversions[Conversions['ConversionId'] == Id].index, inplace = True)
```

In [22]:
```
frames = [Conversions, Update_Conv]
Conversions = pd.concat(frames).reset_index(drop=True, inplace=False).drop_duplicates()
```

In [23]:
```
Conversions
```

Out[23]:

	ConversionId	Multiplier	ConvertFromQty	ConvertFromUom	ConvertToQty	ConvertToUom
0	NaN	1.000000	1.0	XXX	1.00	L
1	NaN	0.877193	1.0	1.14L	1.14	L
2	NaN	0.666667	1.0	1.5L	1.50	L
3	NaN	0.571429	1.0	1.75 L	1.75	L
4	NaN	0.500000	1.0	2L	2.00	L
...	...	...	...	...	...	...
585	P-32664	0.016393	1.0	each	61.00	g
586	P-55707	0.005405	1.0	PTN	185.00	g
587	P-55709	0.005405	1.0	PTN	185.00	g
588	P-62293	0.005405	1.0	PTN	185.00	g
589	P-62023	0.006452	1.0	ROLL	155.00	g

590 rows × 6 columns

In [24]:
```
path = os.path.join(os.getcwd(), "data", "cleaning", "Conversions_Added.csv")
Conversions.to_csv(path, index = False, header = True)
```

Create Unit Converter

In [25]:
```
# Import standard unit conversion information and construct a dataframe
Std_Unit = pd.read_csv(os.path.join(os.getcwd(), "data", "external", "standard_conversions.csv"))
Std_Unit.head()
```

Out[25]:

	Multiplier	ConvertFromQty	ConvertFromUom	ConvertToQty	ConvertToUom
0	4.92890	1	tsp	4.92890	ml
1	14.78700	1	Tbsp	14.78700	ml
2	946.35000	1	qt	946.35000	ml
3	473.17625	1	pt	473.17625	ml
4	28.34950	1	oz	28.34950	g

In [26]:
```
# Seperate uoms that converted to 'ml' or 'g'
liquid_unit = Std_Unit.loc[Std_Unit['ConvertToUom'] == 'ml', 'ConvertFromUom'].tolist()
solid_unit = Std_Unit.loc[Std_Unit['ConvertToUom'] == 'g', 'ConvertFromUom'].tolist()
```

In [27]:
```
# Construct a standard unit converter
def std_converter(qty, uom):
    if uom in Std_Unit['ConvertFromUom'].tolist():
        multiplier = Std_Unit.loc[Std_Unit['ConvertFromUom'] == uom, 'Multiplier']
        Qty = float(qty)*float(multiplier)
        Uom = Std_Unit.loc[Std_Unit['ConvertFromUom'] == uom, 'ConvertToUom'].values[0]
    else:
        Qty = qty
        Uom = uom
    return (Qty, Uom)
```

In [28]:
```
# Test the std_converter
#assert std_converter(0.25,'lb') == (113.398, 'g')
```

In [29]:
```
# Construct a unit converter for specific ingredients
spc_cov = list(filter(None, Conversions['ConversionId'].tolist()))

def spc_converter(ingre, qty, uom):
    if uom in liquid_unit + solid_unit:
        return std_converter(qty, uom)
    elif ingre in spc_cov:
        conversion = Conversions.loc[(Conversions['ConversionId'] == ingre) & (Conversions['ConvertFromUom'] == uom)
                                    & (Conversions['ConvertToUom'] == 'g')]
        multiplier = conversion['Multiplier']
        if multiplier.empty:
            return std_converter(qty, uom)
        else:
            Qty = float(qty)/float(multiplier)
            Uom = conversion['ConvertToUom'].values[0]
            return (Qty, Uom)
    else:
        return std_converter(qty, uom)
```

In [30]:
```
# Test the spc_converter
#assert spc_converter('I-1120', 1, 'CT') == (50, 'g')
```

Items with Non-standard Units

In [31]:
```
# Filter out the items whose unit information is unknown
col_names = list(Ingredients.columns.values)
Items_Nonstd = []

for index, row in Ingredients.iterrows():
    Ingre = Ingredients.loc[index,'IngredientId']
    Uom = Ingredients.loc[index,'Uom']
    if Uom not in ['g', 'ml'] and Uom not in liquid_unit + solid_unit and Ingre.startswith('I') and Ingre not in Conversi
        Dict = {}
        Dict.update(dict(row))
        Items_Nonstd.append(Dict)

Items_Nonstd = pd.DataFrame(Items_Nonstd, columns = col_names)
Items_Nonstd.drop_duplicates(subset=['IngredientId'], inplace=True,)
Items_Nonstd
```

Out[31]:

IngredientId	Qty	Uom	Conversion	InvFactor	Recipe

In [32]:
```
path = os.path.join(os.getcwd(), "data", "cleaning", "Items_Nonstd.csv")
Items_Nonstd.to_csv(path, index = False, header = True)
```

Clean Preps Units

In [33]:
```
Preps['StdQty'] = np.nan
Preps['StdUom'] = np.nan
```

In [34]:
```
# Convert uom into 'g' or 'ml' for each prep using the unit converter
for index in Preps.index:
    PrepId = Preps.loc[index,'PrepId']
    Qty = Preps.loc[index,'PakQty']
    Uom = Preps.loc[index,'PakUOM']
    Preps.loc[index,'StdQty'] = spc_converter(PrepId, Qty, Uom)[0]
    Preps.loc[index,'StdUom'] = spc_converter(PrepId, Qty, Uom)[1]
```

In [35]:
```
Preps
```

Out[35]:

	PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom
0	P-55516	BAKED|Lasagna|Spin Mushroom	5.550	Kg	NaN	5550.000000	g
1	P-54666	BAKED|Pasta|Chicken Alfredo	6.176	Kg	NaN	6176.000000	g
2	P-54664	BAKED|Pasta|Chorizo Penne	7.360	Kg	NaN	7360.000000	g
3	P-56502	BAKED|Pasta|Shrimp Pesto	5.760	Kg	NaN	5760.000000	g
4	P-56433	BATCH|Shrimp Remoulade	1.600	Kg	NaN	1600.000000	g
...	...	...	...	...	...	...	...
748	P-47418	MIX|Cheese	2.000	Kg	PREP	2000.000000	g
749	P-42317	ROASTED|Spaghetti Squash	1.400	Kg	NaN	1400.000000	g
750	P-56927	SAUTE|Cauliflower Rice	1.000	Kg	NaN	1000.000000	g
751	P-56887	YIELD|Grated Pear	800.000	g	NaN	800.000000	g
752	P-46509	YIELD|Lettuce bun	3.000	PTN	NaN	449.999978	g

753 rows × 7 columns

In [36]:
```
# Save cleaned preps list to file
path = os.path.join(os.getcwd(), "data", "cleaning", "Preps_Unit_Cleaned.csv")
Preps.to_csv(path, index = False, header = True)
```

Get Preps with Nonstandard Unit

In [37]:
```
col_names = list(Preps.columns.values)
Preps_Nonstd = []

for index, row in Preps.iterrows():
    StdUom = Preps.loc[index,'StdUom']
    if StdUom not in ['g', 'ml']:
        Dict = {}
        Dict.update(dict(row))
        Preps_Nonstd.append(Dict)

Preps_Nonstd = pd.DataFrame(Preps_Nonstd, columns = col_names)
```

In [38]:
```
Preps_Nonstd
```

Out[38]:

	PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom
0	P-51230	WINGS|Sauce Mix|ratio	1.0	srvg	NaN	1.0	srvg

In [39]:
```
# Filter out preps with nonstandard uom but have information already
Manual_PrepU = pd.read_csv(os.path.join(os.getcwd(), "data", "cleaning", "update", "Preps_UpdateUom.csv"))

col_names = list(Preps_Nonstd.columns.values)
Preps_Nonstd_na = []

for index, row in Preps_Nonstd.iterrows():
    PrepId = Preps_Nonstd.loc[index,'PrepId']
    if PrepId not in Manual_PrepU['PrepId'].values:
        Dict = {}
        Dict.update(dict(row))
        Preps_Nonstd_na.append(Dict)
```

```
Preps_Nonstd = pd.DataFrame(Preps_Nonstd_na, columns = col_names)
Preps_Nonstd
```

Out[39]:

PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom

In [40]:
```
path = os.path.join(os.getcwd(), "data", "cleaning", "Preps_NonstdUom.csv")
Preps_Nonstd.to_csv(path, index = False, header = True)
```

New Items

In [41]:
```
# Load current Items List with assigned Emission Factors Category ID
Items_Assigned = pd.read_csv(os.path.join(os.getcwd(), "data", "mapping", "Items_List_Assigned.csv"))
Items_Assigned.head()
```

Out[41]:

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup
0	I-57545	1	CHUCK FLAT BONELESS FZN	3.30	Kg	1.0	Kg	MEAT
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT

In [42]:
```
Items_Assigned.shape
```

Out[42]: (1993, 8)

Get the List of New Items

In [43]:
```
# Filter new items by itemID that not in the database and output them in a dataframe
col_names = list(Items.columns.values)
New_Items_List = []

for index, row in Items.iterrows():
    ItemId = Items.loc[index,'ItemId']
    if ItemId not in Items_Assigned['ItemId'].values:
        Dict = {}
        Dict.update(dict(row))
        New_Items_List.append(Dict)

New_Items = pd.DataFrame(New_Items_List, columns = col_names)
```

In [44]:
```
New_Items.insert(1, "CategoryID", '')
New_Items
```

Out[44]:

ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup

In [45]:
```
New_Items.shape
```

Out[45]: (0, 8)

In [46]:
```
# Store the list of new items into .csv file
if not New_Items.empty:
    path = os.path.join(os.getcwd(), "data", "mapping", "new items", str(datetime.date(datetime.now()))+"_New_Items.csv")
    New_Items.to_csv(path, index = False, header = True)
```

Data Summary

In [47]:
```
datasum = pd.DataFrame([New_Items.shape, Preps_Nonstd.shape, Items_Nonstd.shape],
                       columns = ['count', 'columns'],
                       index = ['New_Items', 'Preps_Nonstd', 'Items_Nonstd'])
datasum
```

Out[47]:

	count	columns
New_Items	0	8
Preps_Nonstd	0	7
Items_Nonstd	0	6

Climate-Friendly Food Systems (CFFS) Labelling Project

The University of British Columbia

Created by Silvia Huang, CFFS Data Analyst

Part III: Update Information and Mapping

Climate-Friendly Food Systems (CFFS) Labelling Project

The University of British Columbia

Created by Silvia Huang, CFFS Data Analyst

Part III: Update Information and Mapping

Set up and Import Libraries

In [1]:
```
#pip install -r requirements.txt
```

In [2]:
```
import numpy as np
import pandas as pd
import pdpipe as pdp
import matplotlib.pyplot as plt
import glob
import os
import csv
from itertools import islice
from decimal import Decimal
import xml.etree.ElementTree as et
from xml.etree.ElementTree import parse
import openpyxl
import pytest
from datetime import datetime
```

In [3]:
```
# Set the root path, change the the current working directory into the project folder
path = "/Users/silvia/cffs-label"
os.chdir(path)
```

In [4]:
```
# Enable reading data table in the scrolling window if you prefer
#pd.set_option("display.max_rows", None, "display.max_columns", None)
```

Import Preprocessed Datasets

In [5]:
```
Preps = pd.read_csv(os.path.join(os.getcwd(), "data", "cleaning", "Preps_Unit_Cleaned.csv"))
Preps.head()
```

Out[5]:

	PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom
0	P-55516	BAKED|Lasagna|Spin Mushroom	5.550	Kg	NaN	5550.0	g
1	P-54666	BAKED|Pasta|Chicken Alfredo	6.176	Kg	NaN	6176.0	g
2	P-54664	BAKED|Pasta|Chorizo Penne	7.360	Kg	NaN	7360.0	g
3	P-56502	BAKED|Pasta|Shrimp Pesto	5.760	Kg	NaN	5760.0	g
4	P-56433	BATCH|Shrimp Remoulade	1.600	Kg	NaN	1600.0	g

In [6]:
```
ghge_factors = pd.read_csv(os.path.join(os.getcwd(), "data", "external", "ghge_factors.csv"))
ghge_factors.head()
```

Out[6]:

	Category ID	Food Category	Active Total Supply Chain Emissions (kg CO2 / kg food)
0	1	beef & buffalo meat	41.3463
1	2	lamb/mutton & goat meat	41.6211
2	3	pork (pig meat)	9.8315
3	4	poultry (chicken, turkey)	4.3996

	Category ID	Food Category	Active Total Supply Chain Emissions (kg CO2 / kg food)
4	5	butter	11.4316

In [7]:
```
nitro_factors = pd.read_csv(os.path.join(os.getcwd(), "data", "external", "nitrogen_factors.csv"))
nitro_factors.head()
```

Out[7]:

	Category ID	Food Category	g N lost/kg product
0	1	beef & buffalo meat	329.50
1	2	lamb/mutton & goat meat	231.15
2	3	pork (pig meat)	132.80
3	4	poultry (chicken, turkey)	116.80
4	5	butter	100.35

In [8]:
```
water_factors = pd.read_csv(os.path.join(os.getcwd(), "data", "external", "water_factors.csv"))
water_factors.head()
```

Out[8]:

	Category ID	Food Category	Freshwater Withdrawals (L/FU)	Stress-Weighted Water Use (L/FU)
0	1	beef & buffalo meat	1677.200	61309.000
1	2	lamb/mutton & goat meat	461.200	258.900
2	3	pork (pig meat)	1810.300	54242.700
3	4	poultry (chicken, turkey)	370.300	333.500
4	5	butter	1010.176	50055.168

In [9]:
```
# Load current Items List with assigned Emission Factors Category ID
Items_Assigned = pd.read_csv(os.path.join(os.getcwd(), "data", "mapping", "Items_List_Assigned.csv"))
Items_Assigned.head()
```

Out[9]:

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup
0	I-57545	1	CHUCK FLAT BONELESS FZN	3.30	Kg	1.0	Kg	MEAT
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT

Import Update Info

In [10]:
```
# Import list of prep that need convert uom to standard uom manually
Manual_PrepU = pd.read_csv(os.path.join(os.getcwd(), "data", "cleaning", "update", "Preps_UpdateUom.csv"))
Manual_PrepU.head()
```

Out[10]:

	PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom
0	P-54697	LEMON|Wedge 1/8	8.0	each	PREP	84.0	g
1	P-35132	MARINATED|Lemon & Herb Chx	185.0	ea	PREP	24050.0	g
2	P-51992	YIELD|Bread|Sourdough 5/8	36.0	slice	NaN	1620.0	g
3	P-26234	BATCH|Roasted Garlic Bread	16.0	ea	PREP	1280.0	g
4	P-26170	GRILLED|NaanBread	1.0	ea	PREP	125.0	g

In [11]:
```
# Select the file path for new items list with category id
New_Items_Added = pd.read_csv(os.path.join(os.getcwd(), "data", "mapping", "new items added", "New_Items_Added_10.csv"))
New_Items_Added
```

Out[11]:

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup
0	I-13791	24	BAR GF NANAIMO 3X3 WRAPPED	12	CT	1.00	CT	BAKED GOODS
1	I-63679	24	BUTTERHORNS EACH	1	ea	1.00	ea	BAKED GOODS
2	I-31545	24	CAKE CARROT CARM STACK	48	slice	1.00	slice	BAKED GOODS
3	I-1057	24	CAKE CHOC TRIPLE TIGER 12X16IN	2	SHEET	1.00	SHEET	BAKED GOODS
4	I-60871	24	CAKE SHEET CHOC LAYER 12X16IN	2	SHEET	1.00	SHEET	BAKED GOODS
5	I-29698	24	CHEESECAKE 2" MINI	40	ea	1.00	ea	BAKED GOODS

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup
6	I-1392	24	COOKIE GF GINGER VEGAN	12	CT	1.00	CT	BAKED GOODS
7	I-63680	24	POPPYHORNS	1	ea	1.00	ea	BAKED GOODS
8	I-1250	24	PRETZEL PIZZA EACH	1	ea	1.00	CT	BAKED GOODS
9	I-3771	55	YERBA MATE TEA ORG TRADITIONAL	6	lb	1.00	lb	BEVERAGE
10	I-1790	55	JUICE APPLE 100% UNSWT TETRA	40	PTN	1.00	PTN	BEVERAGE
11	I-19634	24	LOAF FRENCH BREAD	1	ea	1.00	LOAF	BREAD
12	I-2309	3	HAM FESTIVE	1	Kg	1.00	Kg	DELI & PREPARED MEAT
13	I-3125	21	CEREAL RICE KRISPIE SQ	6	ea	1.00	ea	FOOD - GROCERY
14	I-37470	22	CORN HOMINY	6	LG CAN	2835.00	g	FOOD - GROCERY
15	I-19540	54	FENUGREEK LEAVES DRY	454	g	1.00	g	FOOD - GROCERY
16	I-3397	31	OLIVE RIPE BLACK SLCD	6	LG CAN	2.84	L	FOOD - GROCERY
17	I-3506	58	SAUCE BBQ SMOKEY TFC	2	JUG	3.78	L	FOOD - GROCERY
18	I-3540	58	SAUCE SAMBAL OELEK	3	jar	3500.00	L	FOOD - GROCERY
19	I-3110	20	TOFU EXTRA FIRM GREEN	12	pak	350.00	g	FOOD - GROCERY
20	I-2949	37	TOMATO SAN MARZANO LA REGINA	6	LG CAN	100.00	fl oz	FOOD - GROCERY
21	I-37475	40	PASTE ACHIOTE	110	g	1.00	g	FOOD - GROCERY
22	I-36996	59	SANDWICH BACON, CHEDD & TOMATO	1	ea	1.00	ea	FOOD - GROCERY
23	I-36999	59	SANDWICH CASATA FALAFEL	1	ea	1.00	ea	FOOD - GROCERY
24	I-36998	59	SANDWICH CHICKEN CRANCHERRY	1	ea	1.00	ea	FOOD - GROCERY
25	I-36995	59	SANDWICH EGG SALAD	1	ea	1.00	ea	FOOD - GROCERY
26	I-37000	59	SANDWICH HAM & CHEESE	1	ea	1.00	ea	FOOD - GROCERY
27	I-36997	59	SANDWICH SALMON SALAD	1	ea	1.00	ea	FOOD - GROCERY
28	I-38987	59	WRAP BREAKFAST HAM & CHEESE	1	ea	1.00	ea	FOOD - GROCERY
29	I-38988	59	WRAP BREAKFAST MEDITERRANEAN	1	ea	1.00	ea	FOOD - GROCERY
30	I-3912	18	BURGER VEGGIE 44CT FZN	44	CT	1.00	CT	MEAT
31	I-3852	3	PORK BUTT SHLDR BNLS N/NFR	1	Kg	1.00	Kg	MEAT
32	I-22915	3	PORK FEET CUT	1	Kg	1.00	Kg	MEAT
33	I-64876	18	TMRW BURGER PATTIES VEGAN	40	each	1.00	each	MEAT
34	I-64877	3	TMRW SAUSAGE BREAKFAST PATTY	100	each	1.00	ea	MEAT
35	I-55331	4	CHICK BREAST BL/SO HAL TENDOUT	1	Kg	1.00	Kg	POULTRY
36	I-3999	4	CHICK DRUMSTICK HALAL	1	Kg	1.00	Kg	POULTRY
37	I-4465	36	ASPARAGUS (LARGE) MX	11	lb	1.00	lb	PRODUCE
38	I-22443	40	BAMBOO SHOOTS STRIP	6	LG CAN	2.84	L	PRODUCE
39	I-10616	17	BEANS ROMANO	1	lb	1.00	lb	PRODUCE
40	I-4582	38	CARROTS BABY BUNCHED BC	1	each	1.00	CT	PRODUCE
41	I-11670	40	COLESLAW MIX CABBAGE&CARROT	5	lb	1.00	lb	PRODUCE
42	I-10265	37	TOMATOES HEIRLOOM BC	1	lb	1.00	lb	PRODUCE
43	I-4849	36	SALAD MIX ARTISAN	3	bag	2.00	lb	PRODUCE
44	I-62863	59	CK|G&G|HMR|Meat Ball & Pasta.	1	ea	1.00	ea	PRODUCTION FOOD
45	I-19923	12	HALIBUT STEAK 4OZ.OW	1	lb	1.00	lb	SEAFOOD
46	I-8105	12	SAL LOX SMK SLC OW	1	lb	1.00	lb	SEAFOOD
47	I-3094	54	PEPPERCORN BLACK WHOLE	3	Kg	1.00	Kg	SPICES
48	I-16060	54	SUMAC GROUND	1	each	454.00	g	SPICES

In [12]:

```
# Import list of items that adjusted GHGe factor manually
Manual_Factor = pd.read_csv(os.path.join(os.getcwd(), "data", "mapping", "Manual_Adjust_Factors.csv"))
Manual_Factor.head()
```

Out[12]:

ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup	Active Total Supply Chain Emissions (kg CO2 / kg food)	g N lost/kg product	Freshwater Withdrawals (L/FU)	Stress-Weighted Water Use (L/FU)

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup	Active Total Supply Chain Emissions (kg CO2 / kg food)	g N lost/kg product	Freshwater Withdrawals (L/FU)	Stress-Weighted Water Use (L/FU)
0	I-52090	59	BURGER BEEF & MUSHROOM HALAL	1.0	cs	48.00	CT	MEAT	25.00894	200.86	1038.84	37961.2
1	I-45558	59	Prep-Vegan Parmesan	1000.0	g	1.00	g	PRODUCTION FOOD	3.85686	0.00	0.00	0.0
2	I-3352	59	MAYONNAISE PAIL TFC 4L	2.0	each	4.00	L	FOOD - GROCERY	3.55000	0.00	0.00	0.0
3	I-3223	59	COCONUT MILK 17/19% MILK FAT	6.0	LG CAN	2.84	L	FOOD - GROCERY	3.50000	0.00	1.00	1.0
4	I-2898	59	MUSTARD DIJON WINE FLEUR	6.0	jar	1.00	Kg	FOOD - GROCERY	3.32600	0.00	0.00	0.0

Update Correct Uom for Preps

In [13]:
```
# Update prep list with manully adjusted uom
for index, row in Manual_PrepU.iterrows():
    PrepId = Manual_PrepU.loc[index, 'PrepId']
    qty = Manual_PrepU.loc[index, 'StdQty']
    uom = Manual_PrepU.loc[index, 'StdUom']
    Preps.loc[Preps['PrepId'] == PrepId, 'StdQty'] = qty
    Preps.loc[Preps['PrepId'] == PrepId, 'StdUom'] = uom
```

In [14]:
```
Preps.drop_duplicates(subset=['PrepId'], inplace=True,)
```

In [15]:
```
Preps.head()
```

Out[15]:

	PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom
0	P-55516	BAKED|Lasagna|Spin Mushroom	5.550	Kg	NaN	5550.0	g
1	P-54666	BAKED|Pasta|Chicken Alfredo	6.176	Kg	NaN	6176.0	g
2	P-54664	BAKED|Pasta|Chorizo Penne	7.360	Kg	NaN	7360.0	g
3	P-56502	BAKED|Pasta|Shrimp Pesto	5.760	Kg	NaN	5760.0	g
4	P-56433	BATCH|Shrimp Remoulade	1.600	Kg	NaN	1600.0	g

In [16]:
```
Preps.shape
```

Out[16]: (752, 7)

In [17]:
```
path = os.path.join(os.getcwd(), "data", "cleaning", "Preps_List_Cleaned.csv")
Preps.to_csv(path, index = False, header = True)
```

Import List of New Items with Emission Factors Category ID Assigned

In [18]:
```
frames = [Items_Assigned, New_Items_Added]
Items_Assigned_Updated = pd.concat(frames).reset_index(drop=True, inplace=False).drop_duplicates()
Items_Assigned_Updated.head()
```

Out[18]:

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup
0	I-57545	1	CHUCK FLAT BONELESS FZN	3.30	Kg	1.0	Kg	MEAT
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT

In [19]:
```
Items_Assigned_Updated.shape
```

Out[19]: (1993, 8)

In [20]:
```
Items_Assigned_Updated[['CategoryID']] = Items_Assigned_Updated[['CategoryID']].apply(pd.to_numeric)
```

In [21]:
```
path = os.path.join(os.getcwd(), "data", "mapping", "Items_List_Assigned.csv")
Items_Assigned_Updated.to_csv(path, index = False, header = True)
```

Mapping Items to Footprint Factors

In [22]:
```
# Map GHG footprint factors
mapping = pd.merge(Items_Assigned_Updated, ghge_factors.loc[:,['Category ID','Food Category','Active Total Supply Chain E
                   how = 'left',
                   left_on = 'CategoryID',
                   right_on = 'Category ID')
for index in mapping.index:
    if np.isnan(mapping.loc[index,'Category ID']):
        mapping.loc[index,'Active Total Supply Chain Emissions (kg CO2 / kg food)'] = 0

mapping = mapping.drop(columns=['Category ID', 'Food Category'])
mapping
```

Out[22]:

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup	Active Total Supply Chain Emissions (kg CO2 / kg food)
0	I-57545	1	CHUCK FLAT BONELESS FZN	3.30	Kg	1.0	Kg	MEAT	41.3463
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT	41.3463
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT	41.3463
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT	41.3463
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT	41.3463
...	...	...	...	...	...	...	...	...	...
1988	I-62863	59	CK|G&G|HMR|Meat Ball & Pasta.	1.00	ea	1.0	ea	PRODUCTION FOOD	0.0000
1989	I-19923	12	HALIBUT STEAK 4OZ.OW	1.00	lb	1.0	lb	SEAFOOD	4.9798
1990	I-8105	12	SAL LOX SMK SLC OW	1.00	lb	1.0	lb	SEAFOOD	4.9798
1991	I-3094	54	PEPPERCORN BLACK WHOLE	3.00	Kg	1.0	Kg	SPICES	9.3703
1992	I-16060	54	SUMAC GROUND	1.00	each	454.0	g	SPICES	9.3703

1993 rows × 9 columns

In [23]:
```
# Map nitrogen footprint factors
mapping = pd.merge(mapping, nitro_factors.loc[:,['Category ID','Food Category','g N lost/kg product']],
                   how = 'left',
                   left_on = 'CategoryID',
                   right_on = 'Category ID')

for index in mapping.index:
    if np.isnan(mapping.loc[index,'Category ID']):
        mapping.loc[index,'g N lost/kg product'] = 0

mapping = mapping.drop(columns=['Category ID', 'Food Category'])
mapping
```

Out[23]:

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup	Active Total Supply Chain Emissions (kg CO2 / kg food)	g N lost/kg product
0	I-57545	1	CHUCK FLAT BONELESS FZN	3.30	Kg	1.0	Kg	MEAT	41.3463	329.50
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT	41.3463	329.50
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT	41.3463	329.50
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT	41.3463	329.50
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT	41.3463	329.50
...	...	...	...	...	...	...	...	...	...	...
1988	I-62863	59	CK|G&G|HMR|Meat Ball & Pasta.	1.00	ea	1.0	ea	PRODUCTION FOOD	0.0000	0.00
1989	I-19923	12	HALIBUT STEAK 4OZ.OW	1.00	lb	1.0	lb	SEAFOOD	4.9798	70.30
1990	I-8105	12	SAL LOX SMK SLC OW	1.00	lb	1.0	lb	SEAFOOD	4.9798	70.30

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup	Active Total Supply Chain Emissions (kg CO2 / kg food)	g N lost/kg product
1991	I-3094	54	PEPPERCORN BLACK WHOLE	3.00	Kg	1.0	Kg	SPICES	9.3703	6.75
1992	I-16060	54	SUMAC GROUND	1.00	each	454.0	g	SPICES	9.3703	6.75

1993 rows × 10 columns

In [24]:
```
# Map water footprint factors
mapping = pd.merge(mapping, water_factors.loc[:,['Category ID','Food Category','Freshwater Withdrawals (L/FU)', 'Stress-W
                   how = 'left',
                   left_on = 'CategoryID',
                   right_on = 'Category ID')

for index in mapping.index:
    if np.isnan(mapping.loc[index,'Category ID']):
        mapping.loc[index,'Freshwater Withdrawals (L/FU)'] = 0
        mapping.loc[index,'Stress-Weighted Water Use (L/FU)'] = 0

mapping = mapping.drop(columns=['Category ID', 'Food Category'])
mapping
```

Out[24]:

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup	Active Total Supply Chain Emissions (kg CO2 / kg food)	g N lost/kg product	Freshwater Withdrawals (L/FU)	Stress-Weighted Water Use (L/FU)
0	I-57545	1	CHUCK FLAT BONELESS FZN	3.30	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.2	61309.0
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.2	61309.0
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.2	61309.0
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT	41.3463	329.50	1677.2	61309.0
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT	41.3463	329.50	1677.2	61309.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1988	I-62863	59	CK|G&G|HMR|Meat Ball & Pasta.	1.00	ea	1.0	ea	PRODUCTION FOOD	0.0000	0.00	0.0	0.0
1989	I-19923	12	HALIBUT STEAK 4OZ.OW	1.00	lb	1.0	lb	SEAFOOD	4.9798	70.30	1580.5	8483.4
1990	I-8105	12	SAL LOX SMK SLC OW	1.00	lb	1.0	lb	SEAFOOD	4.9798	70.30	1580.5	8483.4
1991	I-3094	54	PEPPERCORN BLACK WHOLE	3.00	Kg	1.0	Kg	SPICES	9.3703	6.75	24.9	220.3
1992	I-16060	54	SUMAC GROUND	1.00	each	454.0	g	SPICES	9.3703	6.75	24.9	220.3

1993 rows × 12 columns

Manully Adjust Footprint Factor for Specific Items

In [25]:
```
for index, row in Manual_Factor.iterrows():
    itemId = Manual_Factor.loc[index, 'ItemId']
    ghge = Manual_Factor.loc[index, 'Active Total Supply Chain Emissions (kg CO2 / kg food)']
    nitro = Manual_Factor.loc[index, 'g N lost/kg product']
    water = Manual_Factor.loc[index, 'Freshwater Withdrawals (L/FU)']
    str_water = Manual_Factor.loc[index, 'Stress-Weighted Water Use (L/FU)']
    mapping.loc[mapping['ItemId'] == itemId, 'Active Total Supply Chain Emissions (kg CO2 / kg food)'] = ghge
    mapping.loc[mapping['ItemId'] == itemId, 'g N lost/kg product'] = nitro
    mapping.loc[mapping['ItemId'] == itemId, 'Freshwater Withdrawals (L/FU)'] = water
    mapping.loc[mapping['ItemId'] == itemId, 'Stress-Weighted Water Use (L/FU)'] = str_water
```

In [26]:
```
mapping.drop_duplicates(subset = ['ItemId'], inplace=True)
mapping.dtypes
```

Out[26]:
```
ItemId                                                        object
CategoryID                                                     int64
Description                                                   object
CaseQty                                                      float64
```

```
CaseUOM                                                  object
PakQty                                                  float64
PakUOM                                                   object
InventoryGroup                                           object
Active Total Supply Chain Emissions (kg CO2 / kg food)  float64
g N lost/kg product                                     float64
Freshwater Withdrawals (L/FU)                           float64
Stress-Weighted Water Use (L/FU)                        float64
dtype: object
```

In [27]:
```
mapping.shape
```

Out[27]:
```
(1993, 12)
```

In [28]:
```
mapping
```

Out[28]:

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup	Active Total Supply Chain Emissions (kg CO2 / kg food)	g N lost/kg product	Freshwater Withdrawals (L/FU)	Stress-Weighted Water Use (L/FU)
0	I-57545	1	CHUCK FLAT BONELESS FZN	3.30	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.2	61309.0
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.2	61309.0
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.2	61309.0
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT	41.3463	329.50	1677.2	61309.0
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT	41.3463	329.50	1677.2	61309.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1988	I-62863	59	CK|G&G|HMR|Meat Ball & Pasta.	1.00	ea	1.0	ea	PRODUCTION FOOD	0.0000	0.00	0.0	0.0
1989	I-19923	12	HALIBUT STEAK 4OZ.OW	1.00	lb	1.0	lb	SEAFOOD	4.9798	70.30	1580.5	8483.4
1990	I-8105	12	SAL LOX SMK SLC OW	1.00	lb	1.0	lb	SEAFOOD	4.9798	70.30	1580.5	8483.4
1991	I-3094	54	PEPPERCORN BLACK WHOLE	3.00	Kg	1.0	Kg	SPICES	9.3703	6.75	24.9	220.3
1992	I-16060	54	SUMAC GROUND	1.00	each	454.0	g	SPICES	9.3703	6.75	24.9	220.3

1993 rows × 12 columns

In [29]:
```
path = os.path.join(os.getcwd(), "data", "mapping", "Mapping.csv")
mapping.to_csv(path, index = False, header = True)
```

Climate-Friendly Food Systems (CFFS) Labelling Project

The University of British Columbia

Created by Silvia Huang, CFFS Data Analyst

Part IV: Data Analysis

Set up and Import Libraries

In [1]:
```
#pip install -r requirements.txt
```

In [2]:
```
import numpy as np
import pandas as pd
import pdpipe as pdp
import matplotlib.pyplot as plt
import glob
import os
import csv
from itertools import islice
from decimal import Decimal
import xml.etree.ElementTree as et
from xml.etree.ElementTree import parse
import openpyxl
import pytest
pd.set_option('mode.chained_assignment', None)
```

In [3]:
```
# Set the root path, change the the current working directory into the project folder
path = "/Users/silvia/cffs-label"
os.chdir(path)
```

In [4]:
```
# Enable reading data table in the scrolling window if you prefer
#pd.set_option("display.max_rows", None, "display.max_columns", None)
```

Import Cleaned Datasets

In [5]:
```
Items = pd.read_csv(os.path.join(os.getcwd(), "data", "preprocessed", "Items_List.csv"))
Items.dtypes
```

Out[5]:
```
ItemId            object
Description       object
CaseQty          float64
CaseUOM           object
PakQty           float64
PakUOM            object
InventoryGroup    object
dtype: object
```

In [6]:
```
Items.head()
```

Out[6]:

	ItemId	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup
0	I-4271	APPLES GRANNY SMITH	113.0	ea	1.0	CT	PRODUCE
1	I-4971	ARTICHOKE 1/4 SALAD CUT TFC	6.0	LG CAN	2.5	Kg	PRODUCE
2	I-2305	BACON PANCETTA	1.0	Kg	1.0	Kg	MEAT
3	I-1207	BAGUETTE FRENCH	24.0	each	1.0	CT	BREAD
4	I-17203	BALSAMIC GLAZE	2.0	bottle	2.0	L	FOOD - GROCERY

In [7]:
```
Ingredients = pd.read_csv(os.path.join(os.getcwd(), "data", "preprocessed", "Ingredients_List.csv"))
Ingredients.dtypes
```

Out[7]:
```
IngredientId     object
Qty             float64
Uom              object
Conversion      float64
InvFactor       float64
Recipe           object
dtype: object
```

In [8]:
```
Ingredients.head()
```

Out[8]:

IngredientId	Qty	Uom	Conversion	InvFactor	Recipe

	IngredientId	Qty	Uom	Conversion	InvFactor	Recipe
0	P-18746	1.00	Kg	1.00000	1.0000	P-10241
1	I-3388	1.00	L	1.00000	0.3058	P-10496
2	I-4660	2.27	Kg	2.20462	0.6942	P-10496
3	I-3451	2.56	L	1.00000	1.2800	P-13933
4	I-4679	1.00	BUNCH	1.00000	0.0063	P-18318

In [9]:
```
Preps = pd.read_csv(os.path.join(os.getcwd(), "data", "cleaning", "Preps_List_Cleaned.csv"))
Preps.dtypes
```

Out[9]:
```
PrepId             object
Description        object
PakQty            float64
PakUOM             object
InventoryGroup     object
StdQty            float64
StdUom             object
dtype: object
```

In [10]:
```
Preps.head()
Preps.shape
```

Out[10]:
```
(752, 7)
```

In [11]:
```
Products = pd.read_csv(os.path.join(os.getcwd(), "data", "preprocessed", "Products_List.csv"))
Products.dtypes
```

Out[11]:
```
ProdId         object
Description    object
SalesGroup     object
dtype: object
```

In [12]:
```
Products.head()
```

Out[12]:

	ProdId	Description	SalesGroup
0	R-61778	ALF|Flatbread|4 Cheese	OK - AL FORNO
1	R-61780	ALF|Flatbread|Apple & Pancetta	OK - AL FORNO
2	R-61749	ALF|Flatbread|BBQ Chicken	OK - AL FORNO
3	R-50859	ALF|Flatbread|Bruschetta	OK - AL FORNO
4	R-50788	ALF|Flatbread|Caprese	OK - AL FORNO

In [13]:
```
Conversions = pd.read_csv(os.path.join(os.getcwd(), "data", "cleaning", "Conversions_Added.csv"))
Conversions.dtypes
```

Out[13]:
```
ConversionId       object
Multiplier        float64
ConvertFromQty    float64
ConvertFromUom     object
ConvertToQty      float64
ConvertToUom       object
dtype: object
```

In [14]:
```
Conversions
```

Out[14]:

	ConversionId	Multiplier	ConvertFromQty	ConvertFromUom	ConvertToQty	ConvertToUom
0	NaN	1.000000	1.0	XXX	1.00	L
1	NaN	0.877193	1.0	1.14L	1.14	L
2	NaN	0.666667	1.0	1.5L	1.50	L
3	NaN	0.571429	1.0	1.75 L	1.75	L
4	NaN	0.500000	1.0	2L	2.00	L
...	...	...	...	...	...	...
585	P-32664	0.016393	1.0	each	61.00	g
586	P-55707	0.005405	1.0	PTN	185.00	g
587	P-55709	0.005405	1.0	PTN	185.00	g
588	P-62293	0.005405	1.0	PTN	185.00	g
589	P-62023	0.006452	1.0	ROLL	155.00	g

590 rows × 6 columns

In [15]:
```
mapping = pd.read_csv(os.path.join(os.getcwd(), "data", "mapping", "Mapping.csv"))
mapping.dtypes
```

Out[15]:
```
ItemId                                                      object
CategoryID                                                   int64
Description                                                 object
CaseQty                                                    float64
CaseUOM                                                     object
PakQty                                                     float64
PakUOM                                                      object
InventoryGroup                                              object
Active Total Supply Chain Emissions (kg CO2 / kg food)    float64
g N lost/kg product                                        float64
Freshwater Withdrawals (L/FU)                              float64
Stress-Weighted Water Use (L/FU)                           float64
dtype: object
```

In [16]:
```
mapping
```

Out[16]:

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup	Active Total Supply Chain Emissions (kg CO2 / kg food)	g N lost/kg product	Freshwater Withdrawals (L/FU)	Stress-Weighted Water Use (L/FU)
0	I-57545	1	CHUCK FLAT BONELESS FZN	3.30	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.2	61309.0
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.2	61309.0
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.2	61309.0
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT	41.3463	329.50	1677.2	61309.0
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT	41.3463	329.50	1677.2	61309.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1988	I-62863	59	CK|G&G|HMR|Meat Ball & Pasta.	1.00	ea	1.0	ea	PRODUCTION FOOD	0.0000	0.00	0.0	0.0
1989	I-19923	12	HALIBUT STEAK 4OZ.OW	1.00	lb	1.0	lb	SEAFOOD	4.9798	70.30	1580.5	8483.4
1990	I-8105	12	SAL LOX SMK SLC OW	1.00	lb	1.0	lb	SEAFOOD	4.9798	70.30	1580.5	8483.4
1991	I-3094	54	PEPPERCORN BLACK WHOLE	3.00	Kg	1.0	Kg	SPICES	9.3703	6.75	24.9	220.3
1992	I-16060	54	SUMAC GROUND	1.00	each	454.0	g	SPICES	9.3703	6.75	24.9	220.3

1993 rows × 12 columns

Unit Converter

In [17]:
```
# Import standard unit conversion information for items
Std_Unit = pd.read_csv(os.path.join(os.getcwd(), "data", "external", "standard_conversions.csv"))
Std_Unit.head()
```

Out[17]:

	Multiplier	ConvertFromQty	ConvertFromUom	ConvertToQty	ConvertToUom
0	4.92890	1	tsp	4.92890	ml
1	14.78700	1	Tbsp	14.78700	ml
2	946.35000	1	qt	946.35000	ml
3	473.17625	1	pt	473.17625	ml
4	28.34950	1	oz	28.34950	g

In [18]:
```
# Import list of prep that need convert uom to standard uom manually
Manual_PrepU = pd.read_csv(os.path.join(os.getcwd(), "data", "cleaning", "update", "Preps_UpdateUom.csv"))
Manual_PrepU.head()
```

Out[18]:

	PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom
0	P-54697	LEMON|Wedge 1/8	8.0	each	PREP	84.0	g
1	P-35132	MARINATED|Lemon & Herb Chx	185.0	ea	PREP	24050.0	g
2	P-51992	YIELD|Bread|Sourdough 5/8	36.0	slice	NaN	1620.0	g

	PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom
3	P-26234	BATCH|Roasted Garlic Bread	16.0	ea	PREP	1280.0	g
4	P-26170	GRILLED|NaanBread	1.0	ea	PREP	125.0	g

In [19]:
```
# Add unit conversion info for preps into converter
Prep_cov = Manual_PrepU[['PrepId', 'PakQty','PakUOM','StdQty','StdUom']]
Prep_cov.insert(1, "Multiplier", '')
Prep_cov.columns = Conversions.columns
Prep_cov.loc['Multiplier'] = Prep_cov['ConvertFromQty']/Prep_cov['ConvertToQty']
Prep_cov.head()
```

Out[19]:

	ConversionId	Multiplier	ConvertFromQty	ConvertFromUom	ConvertToQty	ConvertToUom
0	P-54697		8.0	each	84.0	g
1	P-35132		185.0	ea	24050.0	g
2	P-51992		36.0	slice	1620.0	g
3	P-26234		16.0	ea	1280.0	g
4	P-26170		1.0	ea	125.0	g

In [20]:
```
frames = [Conversions, Prep_cov]
Conversions = pd.concat(frames).reset_index(drop=True, inplace=False).drop_duplicates()
Conversions
```

Out[20]:

	ConversionId	Multiplier	ConvertFromQty	ConvertFromUom	ConvertToQty	ConvertToUom
0	NaN	1	1.0	XXX	1.00	L
1	NaN	0.877193	1.0	1.14L	1.14	L
2	NaN	0.666667	1.0	1.5L	1.50	L
3	NaN	0.571429	1.0	1.75 L	1.75	L
4	NaN	0.5	1.0	2L	2.00	L
...	...	...	...	...	...	...
798	P-55707		1.0	PTN	185.00	g
799	P-55709		1.0	PTN	185.00	g
800	P-62293		1.0	PTN	185.00	g
801	P-62023		1.0	ROLL	155.00	g
802	NaN	NaN	NaN	NaN	NaN	NaN

803 rows × 6 columns

In [21]:
```
# Seperate uoms that converted to 'ml' or 'g'
liquid_unit = Std_Unit.loc[Std_Unit['ConvertToUom'] == 'ml', 'ConvertFromUom'].tolist()
solid_unit = Std_Unit.loc[Std_Unit['ConvertToUom'] == 'g', 'ConvertFromUom'].tolist()
```

In [22]:
```
# Construct a standard unit converter
def std_converter(qty, uom):
    if uom in Std_Unit['ConvertFromUom'].tolist():
        multiplier = Std_Unit.loc[Std_Unit['ConvertFromUom'] == uom, 'Multiplier']
        Qty = float(qty)*float(multiplier)
        Uom = Std_Unit.loc[Std_Unit['ConvertFromUom'] == uom, 'ConvertToUom'].values[0]
    else:
        Qty = qty
        Uom = uom
    return (Qty, Uom)
```

In [23]:
```
# Test the std_converter
std_converter(0.25,'lb')
```

Out[23]: (113.398, 'g')

In [24]:
```
# Construct a unit converter for specific items
spc_cov = list(filter(None, Conversions['ConversionId'].tolist()))

def spc_converter(ingre, qty, uom):
    if uom in liquid_unit + solid_unit: #convert to std uom for ingredients has no specific convention instruction
        return std_converter(qty, uom)
    elif ingre in spc_cov: #convert to std uom for ingredients has specific convention instruction
        conversion = Conversions.loc[(Conversions['ConversionId'] == ingre) & (Conversions['ConvertFromUom'] == uom)
                                     & (Conversions['ConvertToUom'] == 'g')]
        conversion.drop_duplicates(subset=['ConversionId'], inplace = True)
        multiplier = conversion['Multiplier']
        if multiplier.empty:
```

```
                return std_converter(qty, uom)
            else:
                #print(conversion)
                Qty = float(qty)/float(multiplier)
                Uom = conversion['ConvertToUom'].values[0]
                return (Qty, Uom)
        else:
            return std_converter(qty, uom)
```

In [25]:
```
# Test the spc_converter
#spc_converter('I-1120', 1, 'CT')
```

In [26]:
```
spc_converter('P-35132', 1, 'ea')
```

Out[26]: (129.9999948000002, 'g')

GHG Factors Calculation for Preps

In [27]:
```
Preps['GHG Emission (g)'] = 0
Preps['GHG Emission(g)/StdUom'] = 0
Preps['N lost (g)'] = 0
Preps['N lost (g)/StdUom'] = 0
Preps['Freshwater Withdrawals (ml)'] = 0
Preps['Freshwater Withdrawals (ml)/StdUom'] = 0
Preps['Stress-Weighted Water Use (ml)'] = 0
Preps['Stress-Weighted Water Use (ml)/StdUom'] = 0
```

In [28]:
```
# Calculate GHG, nitro, water footprints per gram/ml of each prep for items as ingredients only
def get_items_ghge_prep(index, row):
    ingres = Ingredients.loc[Ingredients['Recipe'] == Preps.loc[index,'PrepId']]
    ghg = Preps.loc[index, 'GHG Emission (g)']
    nitro = Preps.loc[index, 'N lost (g)']
    water = Preps.loc[index, 'Freshwater Withdrawals (ml)']
    str_water = Preps.loc[index, 'Stress-Weighted Water Use (ml)']
    weight = Preps.loc[index, 'StdQty']
    #print('Index:', index, '\nIngres:\n', ingres)
    for idx, row in ingres.iterrows():
        ingre = ingres.loc[idx,'IngredientId']
        if ingre.startswith('I'):
            ghge = mapping.loc[mapping['ItemId'] == ingre, 'Active Total Supply Chain Emissions (kg CO2 / kg food)']
            nitro_fac = mapping.loc[mapping['ItemId'] == ingre, 'g N lost/kg product']
            water_fac = mapping.loc[mapping['ItemId'] == ingre, 'Freshwater Withdrawals (L/FU)']
            str_water_fac = mapping.loc[mapping['ItemId'] == ingre, 'Stress-Weighted Water Use (L/FU)']
            #print(ghge)
            Qty = float(ingres.loc[idx,'Qty'])
            Uom = ingres.loc[idx,'Uom']
            if ingre in spc_cov:
                qty = spc_converter(ingre, Qty, Uom)[0]
                ghg += qty*float(ghge)
                nitro += qty*float(nitro_fac)/1000
                water += qty*float(water_fac)
                str_water += qty*float(str_water_fac)
            else:
                qty = std_converter(Qty, Uom)[0]
                ghg += qty*float(ghge)
                nitro += qty*float(nitro_fac)/1000
                water += qty*float(water_fac)
                str_water += qty*float(str_water_fac)
            #print(ingre, Qty, Uom, qty, float(ghge), qty*float(ghge))
            #print(ghg, nitro, water, str_water)
    Preps.loc[index, 'GHG Emission (g)'] = float(ghg)
    Preps.loc[index, 'GHG Emission(g)/StdUom'] = ghg/float(weight)
    Preps.loc[index, 'N lost (g)'] = float(nitro)
    Preps.loc[index, 'N lost (g)/StdUom'] = nitro/float(weight)
    Preps.loc[index, 'Freshwater Withdrawals (ml)'] = float(water)
    Preps.loc[index, 'Freshwater Withdrawals (ml)/StdUom'] = water/float(weight)
    Preps.loc[index, 'Stress-Weighted Water Use (ml)'] = float(str_water)
    Preps.loc[index, 'Stress-Weighted Water Use (ml)/StdUom'] = str_water/float(weight)
```

In [29]:
```
# Calculate GHG, nitro, water footprints per gram/ml of each prep for other preps as ingredients
def get_preps_ghge_prep(index, row):
    ingres = Ingredients.loc[Ingredients['Recipe'] == Preps.loc[index,'PrepId']]
    ghg = Preps.loc[index, 'GHG Emission (g)']
    nitro = Preps.loc[index, 'N lost (g)']
    water = Preps.loc[index, 'Freshwater Withdrawals (ml)']
    str_water = Preps.loc[index, 'Stress-Weighted Water Use (ml)']
    weight = Preps.loc[index, 'StdQty']
    #print('Index:', index, '\nIngres:\n', ingres)
    for idx, row in ingres.iterrows():
        ingre = ingres.loc[idx,'IngredientId']
        if ingre.startswith('P') and len(ingres) > 1:
            ghge = Preps.loc[Preps['PrepId'] == ingre, 'GHG Emission(g)/StdUom']
            nitro_fac = Preps.loc[Preps['PrepId'] == ingre, 'N lost (g)/StdUom']
```

```
            water_fac = Preps.loc[Preps['PrepId'] == ingre, 'Freshwater Withdrawals (ml)/StdUom']
            str_water_fac = Preps.loc[Preps['PrepId'] == ingre, 'Stress-Weighted Water Use (ml)/StdUom']
            #print(ghge)
            Qty = float(ingres.loc[idx,'Qty'])
            Uom = ingres.loc[idx,'Uom']
            if ingre in spc_cov:
                qty = spc_converter(ingre, Qty, Uom)[0]
                ghg += qty*float(ghge)
                nitro += qty*float(nitro_fac)
                water += qty*float(water_fac)
                str_water += qty*float(str_water_fac)
            else:
                qty = std_converter(Qty, Uom)[0]
                ghg += qty*float(ghge)
                nitro += qty*float(nitro_fac)
                water += qty*float(water_fac)
                str_water += qty*float(str_water_fac)
            #print(ingre, Qty, Uom, qty, qty*float(ghge))
            #print(ghg, nitro, water, str_water)
    Preps.loc[index, 'GHG Emission (g)'] = float(ghg)
    Preps.loc[index, 'GHG Emission(g)/StdUom'] = ghg/float(weight)
    Preps.loc[index, 'N lost (g)'] = float(nitro)
    Preps.loc[index, 'N lost (g)/StdUom'] = nitro/float(weight)
    Preps.loc[index, 'Freshwater Withdrawals (ml)'] = float(water)
    Preps.loc[index, 'Freshwater Withdrawals (ml)/StdUom'] = water/float(weight)
    Preps.loc[index, 'Stress-Weighted Water Use (ml)'] = float(str_water)
    Preps.loc[index, 'Stress-Weighted Water Use (ml)/StdUom'] = str_water/float(weight)
```

In [30]:

```
# Calculate GHG, nitro, water footprints per gram/ml of each prep for linked preps
def link_preps(index, row):
    ingres = Ingredients.loc[Ingredients['Recipe'] == Preps.loc[index,'PrepId']]
    ghg = Preps.loc[index, 'GHG Emission (g)']
    nitro = Preps.loc[index, 'N lost (g)']
    water = Preps.loc[index, 'Freshwater Withdrawals (ml)']
    str_water = Preps.loc[index, 'Stress-Weighted Water Use (ml)']
    weight = Preps.loc[index, 'StdQty']
    if len(ingres) == 1:
        ingre = ingres.iloc[0]['IngredientId']
        if ingre.startswith('P'):
            #print(ingres)
            ghge = Preps.loc[Preps['PrepId'] == ingre, 'GHG Emission(g)/StdUom']
            nitro_fac = Preps.loc[Preps['PrepId'] == ingre, 'N lost (g)/StdUom']
            water_fac = Preps.loc[Preps['PrepId'] == ingre, 'Freshwater Withdrawals (ml)/StdUom']
            str_water_fac = Preps.loc[Preps['PrepId'] == ingre, 'Stress-Weighted Water Use (ml)/StdUom']
            Qty = float(ingres.iloc[0]['Qty'])
            Uom = ingres.iloc[0]['Uom']
            if ingre in spc_cov:
                qty = spc_converter(ingre, Qty, Uom)[0]
                ghg = qty*float(ghge)
                nitro = qty*float(nitro_fac)
                water = qty*float(water_fac)
                str_water = qty*float(str_water_fac)
            else:
                qty = std_converter(Qty, Uom)[0]
                ghg = qty*float(ghge)
                nitro = qty*float(nitro_fac)
                water = qty*float(water_fac)
                str_water = qty*float(str_water_fac)
            #print(ingre, ghge, Qty, Uom, qty, weight)
            #print(ghg, nitro, water, str_water)
    Preps.loc[index, 'GHG Emission (g)'] = float(ghg)
    Preps.loc[index, 'GHG Emission(g)/StdUom'] = ghg/float(weight)
    Preps.loc[index, 'N lost (g)'] = float(nitro)
    Preps.loc[index, 'N lost (g)/StdUom'] = nitro/float(weight)
    Preps.loc[index, 'Freshwater Withdrawals (ml)'] = float(water)
    Preps.loc[index, 'Freshwater Withdrawals (ml)/StdUom'] = water/float(weight)
    Preps.loc[index, 'Stress-Weighted Water Use (ml)'] = float(str_water)
    Preps.loc[index, 'Stress-Weighted Water Use (ml)/StdUom'] = str_water/float(weight)
```

In [31]:

```
for index, row in Preps.iterrows():
    get_items_ghge_prep(index , row)
```

In [32]:

```
for index, row in Preps.iterrows():
    link_preps(index, row)
```

In [33]:

```
for index, row in Preps.iterrows():
    get_preps_ghge_prep(index, row)
```

In [34]:

```
Preps
```

Out[34]:

PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom	GHG Emission (g)	GHG Emission(g)/StdUom	N lost (g)	N lo (g)/StdU

	PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom	GHG Emission (g)	GHG Emission(g)/StdUom	N lost (g)	N lo (g)/StdUo
0	P-55516	BAKED|Lasagna|Spin Mushroom	5.550	Kg	NaN	5550.0	g	19928.597446	3.590738	200.304444	0.0360
1	P-54666	BAKED|Pasta|Chicken Alfredo	6.176	Kg	NaN	6176.0	g	17657.751270	2.859092	220.725071	0.0357
2	P-54664	BAKED|Pasta|Chorizo Penne	7.360	Kg	NaN	7360.0	g	22177.046905	3.013186	263.810524	0.0358
3	P-56502	BAKED|Pasta|Shrimp Pesto	5.760	Kg	NaN	5760.0	g	29040.084386	5.041681	178.743124	0.0310
4	P-56433	BATCH|Shrimp Remoulade	1.600	Kg	NaN	1600.0	g	12750.771772	7.969232	45.416647	0.0283
...	...	...	...	...	...	...	...	...	...	...	
747	P-47418	MIX|Cheese	2.000	Kg	PREP	2000.0	g	17820.800000	8.910400	186.600000	0.0933
748	P-42317	ROASTED|Spaghetti Squash	1.400	Kg	NaN	1400.0	g	1411.767296	1.008405	18.978199	0.0135
749	P-56927	SAUTE|Cauliflower Rice	1.000	Kg	NaN	1000.0	g	809.481296	0.809481	8.311199	0.008
750	P-56887	YIELD|Grated Pear	800.000	g	NaN	800.0	g	430.600000	0.538250	2.700000	0.0033
751	P-46509	YIELD|Lettuce bun	3.000	PTN	NaN	450.0	g	469.611174	1.043580	5.964515	0.0132

752 rows × 15 columns

In [35]:
```
path = os.path.join(os.getcwd(), "data", "final", "Preps Footprints.csv")
Preps.to_csv(path, index = False, header = True)
```

GHGe Calculation for Products

In [36]:
```
Products['Weight (g)'] = 0
Products['GHG Emission (g)'] = 0
Products['N lost (g)'] = 0
Products['Freshwater Withdrawals (ml)'] = 0
Products['Stress-Weighted Water Use (ml)'] = 0
```

In [37]:
```
# Calculate GHG, nitro, water footprints per gram/ml of each product for items ingredients only
def get_items_ghge(index, row):
    ingres = Ingredients.loc[Ingredients['Recipe'] == Products.loc[index,'ProdId']]
    ghg = Products.loc[index, 'GHG Emission (g)']
    nitro = Products.loc[index, 'N lost (g)']
    water = Products.loc[index, 'Freshwater Withdrawals (ml)']
    str_water = Products.loc[index, 'Stress-Weighted Water Use (ml)']
    weight = Products.loc[index, 'Weight (g)']
    #print('Index:', index, '\nIngres:\n', ingres)
    for idx, row in ingres.iterrows():
        ingre = ingres.loc[idx,'IngredientId']
        if ingre.startswith('I'):
            ghge = mapping.loc[mapping['ItemId'] == ingre, 'Active Total Supply Chain Emissions (kg CO2 / kg food)']
            nitro_fac = mapping.loc[mapping['ItemId'] == ingre, 'g N lost/kg product']
            water_fac = mapping.loc[mapping['ItemId'] == ingre, 'Freshwater Withdrawals (L/FU)']
            str_water_fac = mapping.loc[mapping['ItemId'] == ingre, 'Stress-Weighted Water Use (L/FU)']
            Qty = float(ingres.loc[idx,'Qty'])
            Uom = ingres.loc[idx,'Uom']
            if ingre in Conversions['ConversionId'].tolist():
                qty = spc_converter(ingre, Qty, Uom)[0]
                weight += qty
                ghg += qty*float(ghge)
                nitro += qty*float(nitro_fac)/1000
                water += qty*float(water_fac)
                str_water += qty*float(str_water_fac)
            else:
                qty = std_converter(Qty, Uom)[0]
                weight += qty
                ghg += qty*float(ghge)
                nitro += qty*float(nitro_fac)/1000
                water += qty*float(water_fac)
                str_water += qty*float(str_water_fac)
            #print(ingre, Qty, Uom, qty, float(ghge), qty*float(ghge))
    Products.loc[index, 'GHG Emission (g)'] = float(ghg)
    Products.loc[index, 'Weight (g)'] = float(weight)
    Products.loc[index, 'N lost (g)'] = float(nitro)
```

```
    Products.loc[index, 'Freshwater Withdrawals (ml)'] = float(water)
    Products.loc[index, 'Stress-Weighted Water Use (ml)'] = float(str_water)
```

In [38]:
```
# Calculate GHG, nitro, water footprints per gram/ml of each product for preps ingredients only
def get_preps_ghge(index, row):
    ingres = Ingredients.loc[Ingredients['Recipe'] == Products.loc[index,'ProdId']]
    ghg = Products.loc[index, 'GHG Emission (g)']
    nitro = Products.loc[index, 'N lost (g)']
    water = Products.loc[index, 'Freshwater Withdrawals (ml)']
    str_water = Products.loc[index, 'Stress-Weighted Water Use (ml)']
    weight = Products.loc[index, 'Weight (g)']
    #print('Index:', index, '\nIngres:\n', ingres)
    for idx, row in ingres.iterrows():
        ingre = ingres.loc[idx,'IngredientId']
        if ingre.startswith('P'):
            ghge = Preps.loc[Preps['PrepId'] == ingre, 'GHG Emission(g)/StdUom']
            nitro_fac = Preps.loc[Preps['PrepId'] == ingre, 'N lost (g)/StdUom']
            water_fac = Preps.loc[Preps['PrepId'] == ingre, 'Freshwater Withdrawals (ml)/StdUom']
            str_water_fac = Preps.loc[Preps['PrepId'] == ingre, 'Stress-Weighted Water Use (ml)/StdUom']
            Qty = float(ingres.loc[idx,'Qty'])
            Uom = ingres.loc[idx,'Uom']
            if ingre in Conversions['ConversionId'].tolist():
                qty = spc_converter(ingre, Qty, Uom)[0]
                weight += qty
                ghg += qty*float(ghge)
                nitro += qty*float(nitro_fac)
                water += qty*float(water_fac)
                str_water += qty*float(str_water_fac)
            else:
                qty = std_converter(Qty, Uom)[0]
                weight += qty
                ghg += qty*float(ghge)
                nitro += qty*float(nitro_fac)
                water += qty*float(water_fac)
                str_water += qty*float(str_water_fac)
            #print(ingre, Qty, Uom, qty, float(ghge), qty*float(ghge))
    Products.loc[index, 'GHG Emission (g)'] = float(ghg)
    Products.loc[index, 'Weight (g)'] = float(weight)
    Products.loc[index, 'N lost (g)'] = float(nitro)
    Products.loc[index, 'Freshwater Withdrawals (ml)'] = float(water)
    Products.loc[index, 'Stress-Weighted Water Use (ml)'] = float(str_water)
```

In [39]:
```
# Calculate GHG, nitro, water footprints per gram/ml of each product for other products ingredients
def get_products_ghge(index, row):
    ingres = Ingredients.loc[Ingredients['Recipe'] == Products.loc[index,'ProdId']]
    ghg = Products.loc[index, 'GHG Emission (g)']
    nitro = Products.loc[index, 'N lost (g)']
    water = Products.loc[index, 'Freshwater Withdrawals (ml)']
    str_water = Products.loc[index, 'Stress-Weighted Water Use (ml)']
    weight = Products.loc[index, 'Weight (g)']
    #print('Index:', index, '\nIngres:\n', ingres)
    for idx, row in ingres.iterrows():
        ingre = ingres.loc[idx,'IngredientId']
        if ingre.startswith('R'):
            ghge = Products.loc[Products['ProdId'] == ingre, 'GHG Emission (g)']
            nitro_fac = Products.loc[Products['ProdId'] == ingre, 'N lost (g)']
            water_fac = Products.loc[Products['ProdId'] == ingre, 'Freshwater Withdrawals (ml)']
            str_water_fac = Products.loc[Products['ProdId'] == ingre, 'Stress-Weighted Water Use (ml)']
            Weight = Products.loc[Products['ProdId'] == ingre, 'Weight (g)']
            Qty = float(ingres.loc[idx,'Qty'])
            ghg += Qty*float(ghge)
            nitro += Qty*float(nitro_fac)
            water += Qty*float(water_fac)
            str_water += Qty*float(str_water_fac)
            weight += Qty*float(Weight)
            #print(ingre, Qty, float(ghge), Qty*float(ghge))
    Products.loc[index, 'GHG Emission (g)'] = float(ghg)
    Products.loc[index, 'Weight (g)'] = float(weight)
    Products.loc[index, 'N lost (g)'] = float(nitro)
    Products.loc[index, 'Freshwater Withdrawals (ml)'] = float(water)
    Products.loc[index, 'Stress-Weighted Water Use (ml)'] = float(str_water)
```

In [40]:
```
for index, row in Products.iterrows():
    get_items_ghge(index , row)
```

In [41]:
```
for index, row in Products.iterrows():
    get_preps_ghge(index, row)
```

In [42]:
```
for index, row in Products.iterrows():
    get_products_ghge(index, row)
```

In [43]:
```
# Filter out products using preps with unknown units
Preps_Nonstd = pd.read_csv(os.path.join(os.getcwd(), "data", "cleaning", "Preps_NonstdUom.csv"))
Preps_Nonstd
```

Out[43]:

PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom

In [44]:
```python
def filter_products(index, row):
    ingres = Ingredients.loc[Ingredients['Recipe'] == Products.loc[index,'ProdId']]
    #print(ingres)
    for idx, row in ingres.iterrows():
        ingre = ingres.loc[idx,'IngredientId']
        if ingre in Preps_Nonstd['PrepId'].tolist():
            print(ingre, index, Products.loc[index,'ProdId'])
            Products.drop(index, inplace=True)
            break
```

In [45]:
```python
for index, row in Products.iterrows():
    filter_products(index, row)
```

In [46]:
```python
Products['Freshwater Withdrawals (L)'] = round(Products['Freshwater Withdrawals (ml)']/1000, 2)
Products['Stress-Weighted Water Use (L)'] = round(Products['Stress-Weighted Water Use (ml)']/1000, 2)
Products = Products.drop(columns=['Freshwater Withdrawals (ml)', 'Stress-Weighted Water Use (ml)'])
```

In [47]:
```python
Products['GHG Emission (g) / 100g'] = round(100*Products['GHG Emission (g)']/Products['Weight (g)'], 2)
Products['N lost (g) / 100g'] = round(100*Products['N lost (g)']/Products['Weight (g)'], 2)
Products['Freshwater Withdrawals (L) / 100g'] = round(100*Products['Freshwater Withdrawals (L)']/Products['Weight (g)'],
Products['Stress-Weighted Water Use (L) / 100g'] = round(100*Products['Stress-Weighted Water Use (L)']/Products['Weight (
```

In [48]:
```python
Products
```

Out[48]:

	ProdId	Description	SalesGroup	Weight (g)	GHG Emission (g)	N lost (g)	Freshwater Withdrawals (L)	Stress-Weighted Water Use (L)	GHG Emission (g) / 100g	N lost (g) / 100g	Freshwater Withdrawals (L) / 100g
0	R-61778	ALF|Flatbread|4 Cheese	OK - AL FORNO	185.000000	1229.947500	11.939000	212.84	10662.59	664.84	6.45	115.05
1	R-61780	ALF|Flatbread|Apple & Pancetta	OK - AL FORNO	140.000000	756.345750	9.031250	144.99	5287.95	540.25	6.45	103.56
2	R-61749	ALF|Flatbread|BBQ Chicken	OK - AL FORNO	245.000000	1011.756923	18.958191	118.17	3196.50	412.96	7.74	48.23
3	R-50859	ALF|Flatbread|Bruschetta	OK - AL FORNO	215.000000	454.128829	4.589045	75.98	3520.38	211.22	2.13	35.34
4	R-50788	ALF|Flatbread|Caprese	OK - AL FORNO	233.000000	1010.276452	10.809234	150.95	7671.61	433.60	4.64	64.79
...	...	...	...	...	...	...	...	...	...	...	...
453	R-57815	SQR|Tofu Sofrito Quesadilla +1	OK - SQUARE MEAL	650.000002	1542.748135	11.964991	171.88	8616.10	237.35	1.84	26.44
454	R-61679	SQR|Tofu Sofrito Quesadilla +2	OK - SQUARE MEAL	1065.000002	2275.218575	16.560511	214.99	11264.83	213.64	1.55	20.19
455	R-56902	SQR|Vegan Lettuce Wrap	OK - SQUARE MEAL	399.999993	640.295633	3.987816	81.04	4682.22	160.07	1.00	20.26
456	R-57810	SQR|Vegan Lettuce Wrap +1	OK - SQUARE MEAL	544.999993	806.712203	4.920096	157.61	5354.54	148.02	0.90	28.92
457	R-57811	SQR|Vegan Lettuce Wrap +2	OK - SQUARE MEAL	689.999993	973.128774	5.852376	234.18	6026.86	141.03	0.85	33.94

458 rows × 12 columns

In [49]:
```python
Products.shape
```

Out[49]: (458, 12)

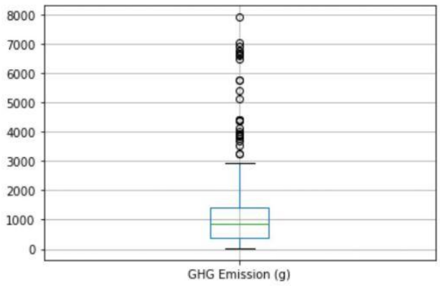
In [50]:
```python
path = os.path.join(os.getcwd(), "data", "final", "Recipes Footprints.csv")
Products.to_csv(path, index = False, header = True)
```

Data Visualization

In [51]:
```python
path = os.path.join(os.getcwd(), "reports", "figures/")
```

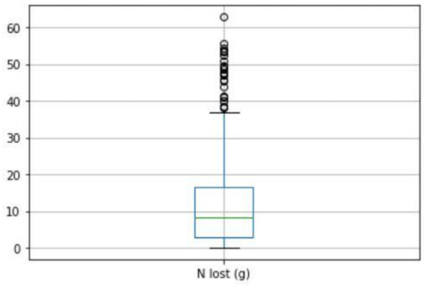
In [52]:
```python
Products.boxplot(column=['GHG Emission (g)'], return_type='axes')
```

Out[52]: <AxesSubplot:>



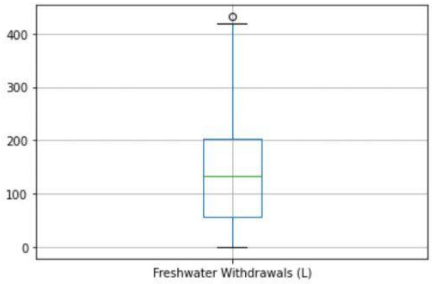
In [53]:
```
Products.boxplot(column=['N lost (g)'], return_type='axes')
```

Out[53]: <AxesSubplot:>



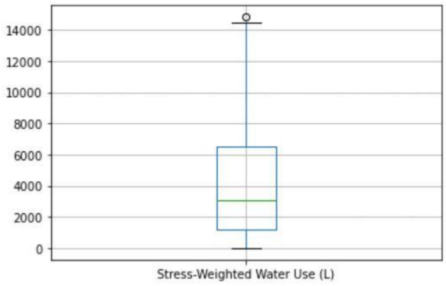
In [54]:
```
Products.boxplot(column=['Freshwater Withdrawals (L)'], return_type='axes')
```

Out[54]: <AxesSubplot:>

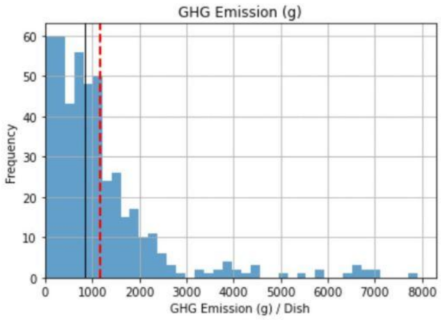


In [55]:
```
Products.boxplot(column=['Stress-Weighted Water Use (L)'], return_type='axes')
```

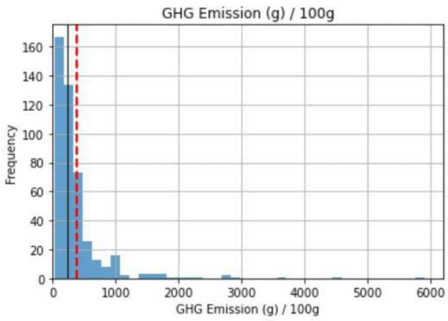
Out[55]: <AxesSubplot:>



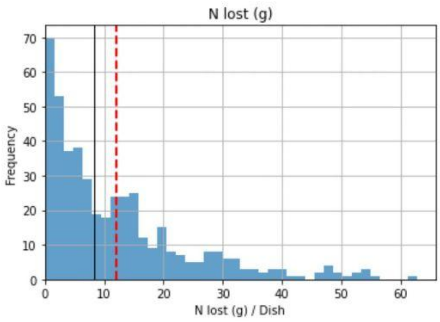
In [56]:
```
Products.hist(column=['GHG Emission (g)'], bins= 40, alpha = 0.7)
plt.axvline(Products['GHG Emission (g)'].mean(), color='r', linestyle='dashed', linewidth=2, label = 'mean' )
plt.axvline(Products['GHG Emission (g)'].median(), color='k', linewidth=1, label = 'median')
plt.xlabel('GHG Emission (g) / Dish')
plt.ylabel('Frequency')
plt.xlim(left=0)
plt.savefig(path + 'GHGe_dish.png')
```



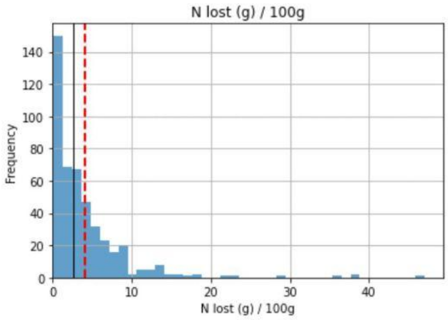
In [57]:
```
Products.hist(column=['GHG Emission (g) / 100g'], bins= 40, alpha = 0.7)
plt.axvline(Products['GHG Emission (g) / 100g'].mean(), color='r', linestyle='dashed', linewidth=2, label = 'mean' )
plt.axvline(Products['GHG Emission (g) / 100g'].median(), color='k', linewidth=1, label = 'median')
plt.xlabel('GHG Emission (g) / 100g')
plt.ylabel('Frequency')
plt.xlim(left=0)
plt.savefig(path + 'GHGe_100g.png')
```



In [58]:
```
Products.hist(column=['N lost (g)'], bins= 40, alpha = 0.7)
plt.axvline(Products['N lost (g)'].mean(), color='r', linestyle='dashed', linewidth=2, label = 'mean' )
plt.axvline(Products['N lost (g)'].median(), color='k', linewidth=1, label = 'median')
plt.xlabel('N lost (g) / Dish')
plt.ylabel('Frequency')
plt.xlim(left=0)
plt.savefig(path + 'N lost_dish.png')
```

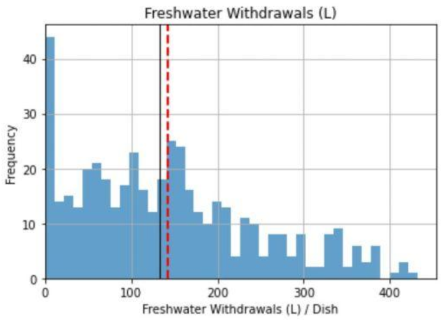


In [59]:
```
Products.hist(column=['N lost (g) / 100g'], bins= 40, alpha = 0.7)
plt.axvline(Products['N lost (g) / 100g'].mean(), color='r', linestyle='dashed', linewidth=2, label = 'mean' )
plt.axvline(Products['N lost (g) / 100g'].median(), color='k', linewidth=1, label = 'median')
plt.xlabel('N lost (g) / 100g')
plt.ylabel('Frequency')
plt.xlim(left=0)
plt.savefig(path + 'N lost_100g.png')
```



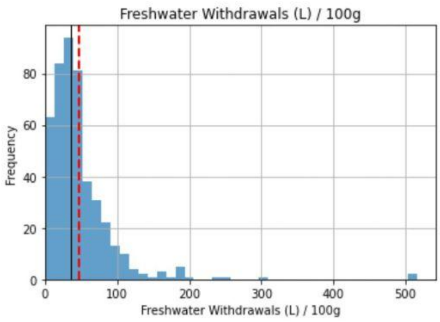
In [60]:

```
Products.hist(column=['Freshwater Withdrawals (L)'], bins= 40, alpha = 0.7)
plt.axvline(Products['Freshwater Withdrawals (L)'].mean(), color='r', linestyle='dashed', linewidth=2, label = 'mean' )
plt.axvline(Products['Freshwater Withdrawals (L)'].median(), color='k', linewidth=1, label = 'median')
plt.xlabel('Freshwater Withdrawals (L) / Dish')
plt.ylabel('Frequency')
plt.xlim(left=0)
plt.savefig(path + 'Fresh water_dish.png')
```



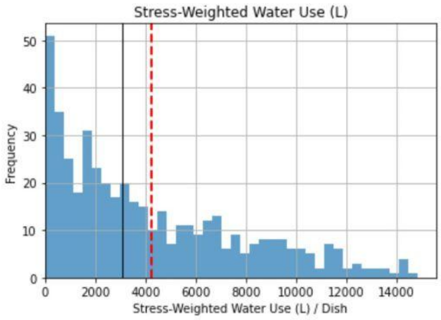
In [61]:

```
Products.hist(column=['Freshwater Withdrawals (L) / 100g'], bins= 40, alpha = 0.7)
plt.axvline(Products['Freshwater Withdrawals (L) / 100g'].mean(), color='r', linestyle='dashed', linewidth=2, label = 'me
plt.axvline(Products['Freshwater Withdrawals (L) / 100g'].median(), color='k', linewidth=1, label = 'median')
plt.xlabel('Freshwater Withdrawals (L) / 100g')
plt.ylabel('Frequency')
plt.xlim(left=0)
plt.savefig(path + 'Fresh water_100g.png')
```

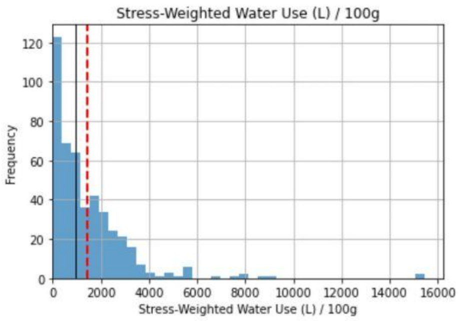


In [62]:

```
Products.hist(column=['Stress-Weighted Water Use (L)'], bins= 40, alpha = 0.7)
plt.axvline(Products['Stress-Weighted Water Use (L)'].mean(), color='r', linestyle='dashed', linewidth=2, label = 'mean'
plt.axvline(Products['Stress-Weighted Water Use (L)'].median(), color='k', linewidth=1, label = 'median')
plt.xlabel('Stress-Weighted Water Use (L) / Dish')
plt.ylabel('Frequency')
plt.xlim(left=0)
plt.savefig(path + 'Stress water_dish.png')
```




```
In [63]: Products.hist(column=['Stress-Weighted Water Use (L) / 100g'], bins= 40, alpha = 0.7)
plt.axvline(Products['Stress-Weighted Water Use (L) / 100g'].mean(), color='r', linestyle='dashed', linewidth=2, label =
plt.axvline(Products['Stress-Weighted Water Use (L) / 100g'].median(), color='k', linewidth=1, label = 'median')
plt.xlabel('Stress-Weighted Water Use (L) / 100g')
plt.ylabel('Frequency')
plt.xlim(left=0)
plt.savefig(path + 'Stress water_100g.png')
```



APPENDIX B [GHG EMISSION FACTORS LIST]

Category ID	Food Category	Active Total Supply Chain Emissions (kg CO2 / kg food)	Data Source
1	beef & buffalo meat	41.3463	Cool Food Calculator
2	lamb/mutton & goat meat	41.6211	Cool Food Calculator
3	pork (pig meat)	9.8315	Cool Food Calculator
4	poultry (chicken, turkey)	4.3996	Cool Food Calculator
5	butter	11.4316	Cool Food Calculator
6	cheese	8.9104	Cool Food Calculator
7	ice cream	4.0163	Cool Food Calculator
8	cream	6.9824	Cool Food Calculator
9	milk (cow's milk)	2.2325	Cool Food Calculator
10	yogurt	2.9782	Cool Food Calculator
11	eggs	3.6615	Cool Food Calculator
12	fish (finfish)	4.9798	Cool Food Calculator
13	crustaceans (shrimp/prawns)	21.1274	Cool Food Calculator
14	mollusks	2.4351	Cool Food Calculator
15	animal fats	6.9693	Cool Food Calculator
16	other legumes	1.6042	Cool Food Calculator
17	beans and pulses (dried)	1.6776	Cool Food Calculator
18	peas	0.6995	Cool Food Calculator
19	peanuts/groundnuts	1.692	Cool Food Calculator
20	soybeans/tofu	1.7542	Cool Food Calculator
21	other grains/cereals	1.4785	Cool Food Calculator
22	corn (maize)	0.9734	Cool Food Calculator
23	oats (oatmeal)	2.3017	Cool Food Calculator
24	wheat/rye (bread, pasta, baked goods)	1.5225	Cool Food Calculator
25	rice	2.5345	Cool Food Calculator
26	tree nuts and seeds	4.2854	Cool Food Calculator
27	almond milk	0.7021	Cool Food Calculator
28	oat milk	0.9943	Cool Food Calculator
29	rice milk	0.6972	Cool Food Calculator
30	soy milk	0.489	Cool Food Calculator
31	other fruits	0.4306	Cool Food Calculator
32	apples	0.3581	Cool Food Calculator

33	bananas	0.7115	Cool Food Calculator
34	berries	1.6547	Cool Food Calculator
35	citrus fruit	0.3942	Cool Food Calculator
36	cabbages and other brassicas (broccoli)	0.622	Cool Food Calculator
37	tomatoes	0.6932	Cool Food Calculator
38	root vegetables	0.3062	Cool Food Calculator
39	onions and leeks	0.3015	Cool Food Calculator
40	other vegetables	0.5029	Cool Food Calculator
41	potatoes	0.397	Cool Food Calculator
42	cassava and other roots	0.397	Cool Food Calculator
43	sugars and sweeteners	1.6414	Cool Food Calculator
44	other vegetable oils	3.1509	Cool Food Calculator
45	soybeans (oil)	3.0336	Cool Food Calculator
46	palm (oil)	4.2483	Cool Food Calculator
47	sunflower (oil)	3.0231	Cool Food Calculator
48	rapeseed/canola (oil)	3.2401	Cool Food Calculator
49	olives (oil)	5.6383	Cool Food Calculator
50	barley (beer)	0.9535	Cool Food Calculator
51	wine grapes (wine)	1.3776	Cool Food Calculator
52	cocoa	10.456	Cool Food Calculator
53	coffee	16.6995	Cool Food Calculator
54	stimulants & spices misc.	9.3703	Cool Food Calculator
55	water & beverages	0	By Assumption
56	salt	0.44	The Big Climate Database
57	vinegar	1.93	The Big Climate Database
58	sauces & paste	0	By Assumption
59	manually adjusted	0	Estimated Individually
60	human labor	0	By Assumption
61	kitchen supplies	0	By Assumption

APPENDIX C [NITROGEN FOOTPRINT FACTORS LIST]

Category ID	Food Category	g N lost/kg product	Data Source
1	beef & buffalo meat	329.5	Food Label Toolkit
2	lamb/mutton & goat meat	231.15	Average of beef (1) and pork (3)
3	pork (pig meat)	132.8	Food Label Toolkit
4	poultry (chicken, turkey)	116.8	Food Label Toolkit
5	butter	100.35	GHG ratio to milk (9)
6	cheese	93.3	Food Label Toolkit
7	ice cream	16.2	GHG ratio to milk (9)
8	cream	28.08	GHG ratio to milk (9)
9	milk (cow's milk)	19.6	Food Label Toolkit
10	yogurt	26.07	GHG ratio to milk (9)
11	eggs	61.2	Food Label Toolkit
12	fish (finfish)	70.3	Food Label Toolkit
13	crustaceans (shrimp/prawns)	70.3	Same as fish (12)
14	mollusks	70.3	Same as fish (12)
15	animal fats	0.2	Same as oil (44)
16	other legumes	5.9	Food Label Toolkit
17	beans and pulses (dried)	5.9	Food Label Toolkit
18	peas	5.9	Food Label Toolkit
19	peanuts/groundnuts	12.2	Food Label Toolkit
20	soybeans/tofu	5.9	Food Label Toolkit
21	other grains/cereals	5.9	Food Label Toolkit
22	corn (maize)	6.75	Average of all veg products
23	oats (oatmeal)	6.75	Average of all veg products
24	wheat/rye (bread, pasta, baked goods)	14.8	Food Label Toolkit
25	rice	5.3	Food Label Toolkit
26	tree nuts and seeds	12.2	Food Label Toolkit
27	almond milk	3.05	1/4 of nuts (19)
28	oat milk	0.68	1/10 of oats (23)
29	rice milk	1.06	1/5 of rice (25)
30	soy milk	2.37	2/5 of soybean (20)
31	other fruits	2.7	Food Label Toolkit
32	apples	2.7	Food Label Toolkit
33	bananas	2.7	Food Label Toolkit

34	berries	2.7	Food Label Toolkit
35	citrus fruit	2.7	Food Label Toolkit
36	cabbages and other brassicas (broccoli)	7.9	Food Label Toolkit
37	tomatoes	7.9	Food Label Toolkit
38	root vegetables	7.9	Food Label Toolkit
39	onions and leeks	7.9	Food Label Toolkit
40	other vegetables	7.9	Food Label Toolkit
41	potatoes	5	Food Label Toolkit
42	cassava and other roots	5	Food Label Toolkit
43	sugars and sweeteners	0	By Assumption
44	other vegetable oils	0.2	Food Label Toolkit
45	soybeans (oil)	0.2	Food Label Toolkit
46	palm (oil)	0.2	Food Label Toolkit
47	sunflower (oil)	0.2	Food Label Toolkit
48	rapeseed/canola (oil)	0.2	Food Label Toolkit
49	olives (oil)	0.2	Food Label Toolkit
50	barley (beer)	9.32	GHG ratio to wheat (24)
51	wine grapes (wine)	8.64	GHG ratio to fruits (31)
52	cocoa	2.7	Same as fruits (31)
53	coffee	2.7	Same as fruits (31)
54	stimulants & spices misc.	6.75	Average of all veg products
55	water & beverages	0	By Assumption
56	salt	0	By Assumption
57	vinegar	0	By Assumption
58	sauces & paste	6.75	Average of all veg products
59	manually adjusted	0	Estimated Individually
60	human labor	0	By Assumption
61	kitchen supplies	0	By Assumption

APPENDIX D [WATER FOOTPRINT FACTORS LIST]

Category ID	Food Category	Freshwater Withdrawals (L/FU)	Stress-Weighted Water Use (L/FU)	Data Source
1	beef & buffalo meat	1677.2	61309	Poore & Newecek
2	lamb/mutton & goat meat	461.2	258.9	Poore & Newecek
3	pork (pig meat)	1810.3	54242.7	Poore & Newecek
4	poultry (chicken, turkey)	370.3	333.5	Poore & Newecek
5	butter	1010.176	50055.168	GHG ratio to milk (9)
6	cheese	1559.3	80463.1	Poore & Newecek
7	ice cream	16.2	17597.52	GHG ratio to milk (9)
8	cream	28.08	30502.368	GHG ratio to milk (9)
9	milk (cow's milk)	197.3	9776.4	Poore & Newecek
10	yogurt	262.409	13002.612	GHG ratio to milk (9)
11	eggs	632.9	18621	Poore & Newecek
12	fish (finfish)	1580.5	8483.4	Poore & Newecek
13	crustaceans (shrimp/prawns)	1207.8	48737.6	Poore & Newecek
14	mollusks	0	0	By Assumption
15	animal fats	1810.3	54242.7	Same as pork (3)
16	other legumes	0	0	Poore & Newecek
17	beans and pulses (dried)	0	0	Poore & Newecek
18	peas	0	0	Poore & Newecek
19	peanuts/groundnuts	900.2	44352.1	Poore & Newecek
20	soybeans/tofu	6.6	32.4	Poore & Newecek
21	other grains/cereals	677.075	10563.3	Average of all grains
22	corn (maize)	43.9	349.6	Poore & Newecek
23	oats (oatmeal)	670.3	24456.3	Poore & Newecek
24	wheat/rye (bread, pasta, baked goods)	419.2	12821.7	Poore & Newecek
25	rice	1574.9	4625.6	Poore & Newecek
26	tree nuts and seeds	1823.3	129364.3	Poore & Newecek
27	almond milk	455.825	32341.075	1/4 of nuts (19)
28	oat milk	67.03	2445.63	1/10 of oats (23)
29	rice milk	314.98	925.12	1/5 of rice (25)
30	soy milk	1.3	6.2	Poore & Newecek
31	other fruits	3.5	4.7	Poore & Newecek

32	apples	114.5	1024.7	Poore & Newecek
33	bananas	1	31.3	Poore & Newecek
34	berries	403.5	16245.1	Poore & Newecek
35	citrus fruit	37.4	1345.5	Poore & Newecek
36	cabbages and other brassicas (broccoli)	54.5	2483.4	Poore & Newecek
37	tomatoes	77	4480.7	Poore & Newecek
38	root vegetables	9.9	37.9	Poore & Newecek
39	onions and leeks	1.9	57	Poore & Newecek
40	other vegetables	81.3	2939.5	Poore & Newecek
41	potatoes	2.6	78.3	Poore & Newecek
42	cassava and other roots	9.9	37.9	Poore & Newecek
43	sugars and sweeteners	10.1	65.2	Poore & Newecek
44	other vegetable oils	67.5	4937.72	Average of all veg oils
45	soybeans (oil)	1.6	7.8	Poore & Newecek
46	palm (oil)	6.4	34.8	Poore & Newecek
47	sunflower (oil)	10.2	236.7	Poore & Newecek
48	rapeseed/canola (oil)	1.4	13.6	Poore & Newecek
49	olives (oil)	317.9	24395.7	Poore & Newecek
50	barley (beer)	7	27.3	Poore & Newecek
51	wine grapes (wine)	4.5	60.4	Poore & Newecek
52	cocoa	24.9	220.3	Poore & Newecek
53	coffee	33.3	340.7	Poore & Newecek
54	stimulants & spices misc.	24.9	220.3	Same as cocoa (52)
55	water & beverages	1	1	By Assumption
56	salt	0	0	By Assumption
57	vinegar	1	1	By Assumption
58	sauces & paste	20.225	1134.925	½ water + ¼ tomato + ¼ onion
59	manually adjusted	0	0	Estimated Individually
60	human labor	0	0	By Assumption
61	kitchen supplies	0	0	By Assumption