University of British Columbia

Social-Ecological Economic Development Studies (SEEDS) Sustainability Program

Student Research Report

CLIMATE-FRIENDLY FOOD SYSTEMS (CFFS) LABELLING PROJECT

An Evaluation Framework for the Operationalization of UBC Vancouver's Climate-Friendly Food Label

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

Supervised by: Juan Diego Martinez, Institute for Resources Environment and Sustainability

Prepared for: UBC Food Services and UBC Botanical Gardens

University of British Columbia

August 2021

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

- GHG: Greenhouse Gas
- CFFS: Climate-Friendly Food Systems
- UBCFS: UBC Food Services
- OC: Optimum Control
- CAP: Climate Action Plan
- UBCFSP: UBC Food System Project

EXECUTIVE SUMMARY

How to make your daily menu choices climate-friendly? Roughly 26% of global total greenhouse gas (GHG) emissions generated by human activities were contributed by the food supply chain (Poore & Newecek, 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 the University of British Columbia (UBC) takes action to provide students with the climate impact information of menu items they purchase every day that could help to educate, bring awareness and influence their purchasing behavior in a more climate-friendly way. This research report was prepared by the CFFS data analyst, a member of the CFFS Action Team and the CFFS Labelling Project Research Group. This report is focused on the data analysis and back-end implementation of the CFFS labelling project and is complementary to the report on the communication and definition side prepared by the CFFS communication and engagement coordinator.

The CFFS labelling project is part of the actions taken by UBC in response to the Climate Action Plan (CAP) 2030 scope 3 emission reduction goal. The CFFS Action Team was formed to accelerate transitions towards a climate-friendly food system and advance the UBC Food System Project mission and priorities. This project aims to evaluate the climate impact of menu items sold at UBC Food Services (UBCFS) venues and operationalize the CFFS food label to inform climate-friendly menu choices. The goal of this project includes creating a reproducible data analysis framework for calculating recipes' greenhouse gas (GHG) emissions, establishing a food GHG emission baseline at the UBC Vancouver campus, determining cut-offs for the CFFS traffic-light label, and further integrating additional CFFS attributes into the framework for expanded impact.

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. The data on GHG emission factors came from external secondary data sources.

The main deliverable of the project is the external framework that conducts the evaluation process of recipes automatically once GHG emission factors have been assigned to each ingredient, and it will be further developed to incorporate additional attributes and adapt to the expansion of the CFFS label. The external framework is able to read the primary data automatically and output the total GHG emissions 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 reduction goals for food systems. For the initial pilot phase of the label implementation, we determined separate sets of cut-offs for different meal groups (i.e., lunch/dinner, breakfast, desserts/snacks) due to the incomparability between products from different meal groups.

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 GHGe factors database by conducting research collaboratively with peer institutions.

Keywords: climate label, climate-friendly, reproducible framework, greenhouse gas, food systems

References

Poore, J., and T. Nemecek. 2018. “Reducing Food’s Environmental Impacts through Producers and Consumers.” *Science* 360 (6392):987–92. doi:10.1126/science.aaq0216.

1. INTRODUCTION

1.1 RESEARCH CONTEXT & TOPIC

Roughly 26% of global total greenhouse gas (GHG) emissions (13.7 billion tons of carbon dioxide equivalents (CO2eq)) generated by human activities were contributed by the food supply chain (Poore & Newecek, 2018). 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 a Climate Action Plan (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 Climate-Friendly Food Systems (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 labels 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 constructing an evaluation framework for analyzing the recipes and ingredients to provide a weighted 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, UBC Food Services contributed to a large proportion of the total GHG emissions from the food systems through students' daily meals. The action of providing students with the GHG emission information of 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 UBC Food Services venues. 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.

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 UBC Food Services. 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.

For the summer pilot at Mercante, in order to establish a 2019 GHG emissions baseline, the sales data for all products between January 1 and December 31, 2019 were extracted from

Optimum Control. Future iterations will include the residence dining halls sales and recipe emissions data to calculate the UBCFS GHG emission 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 (67%).
- 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. It provides GHG emission data based on life-cycle assessments for major food categories in Denmark.
- 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.

Note that the food groups were slightly adjusted from the Cool Food Calculators for better assignment of GHG emission factors on ingredients procured by UBC Food Services. 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.

2.3 ASSUMPTIONS

- To make the process of recipe evaluation consistent, accurate, and structured, several assumptions were made when evaluating their GHG emissions. 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 final weight of serving takes into account loss or addition of weight during 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.
- The GHG emission factor for water is zero, and we ignored the water use in the cooking process.
- We excluded the GHG emissions from sauces and dressings that have no dominant ingredients.

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 the 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. 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.

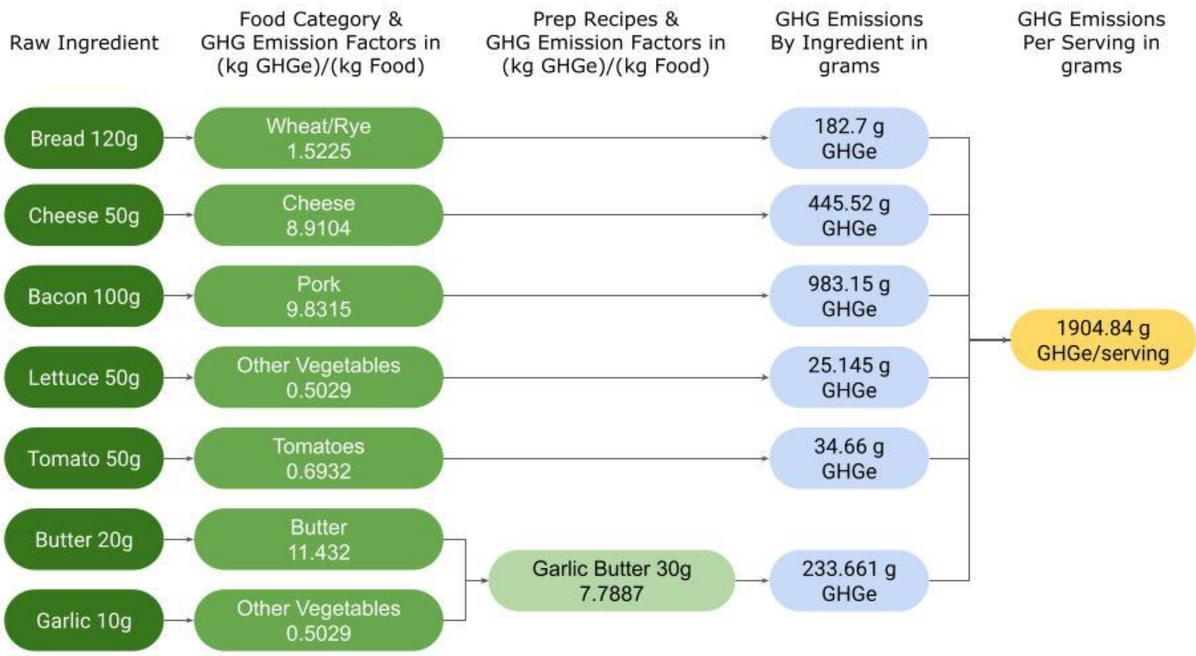


Figure 1: Flowchart for Calculating the GHG Emissions of a Bacon Sandwich

2.5 BASELINE AND LABEL CUT-OFFS

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* 2 for the design and meaning of the labels implemented during the summer pilot.

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. In addition, 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.

The methods for determining cut-offs for the three levels of the label 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 2: Traffic Light Labelling System

2.6 ADDITIONAL ATTRIBUTES

Besides GHG emissions, the evaluation framework also considers the incorporation of one additional attribute for the fall launch 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, 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 capability for qualitative attributes, their impact on climate change mitigation and adaptation, and evaluation survey results.

3. RESULTS

3.1 EXTERNAL FRAMEWORK

The evaluation of menu items is an automatic process that is conducted by an external 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 Optimum Control and does most of the calculating process. See Appendix A for the code that constructed the external framework. The process flowchart for the whole evaluation process is shown in *Figure 3*:

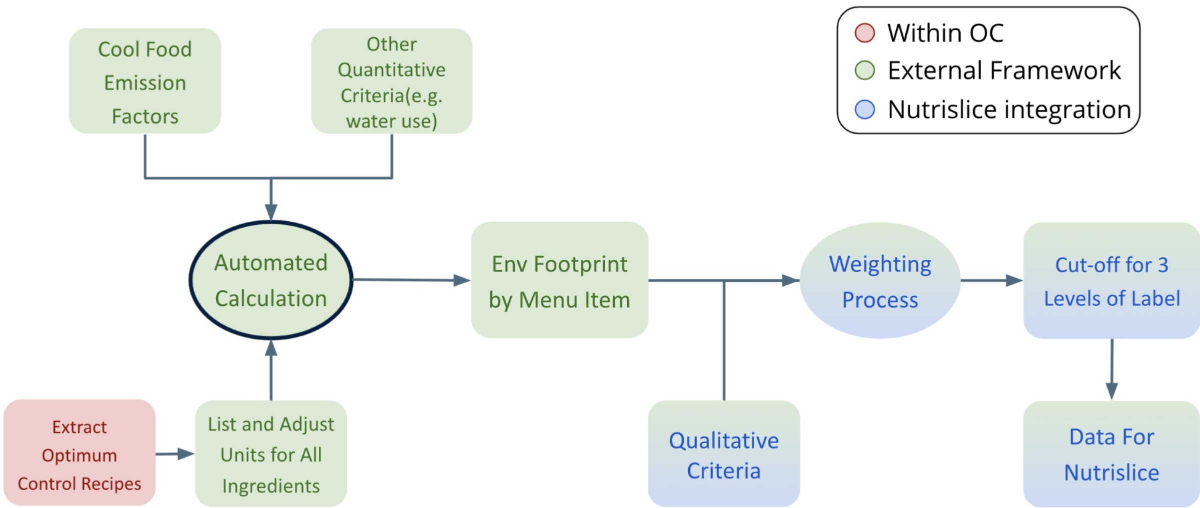


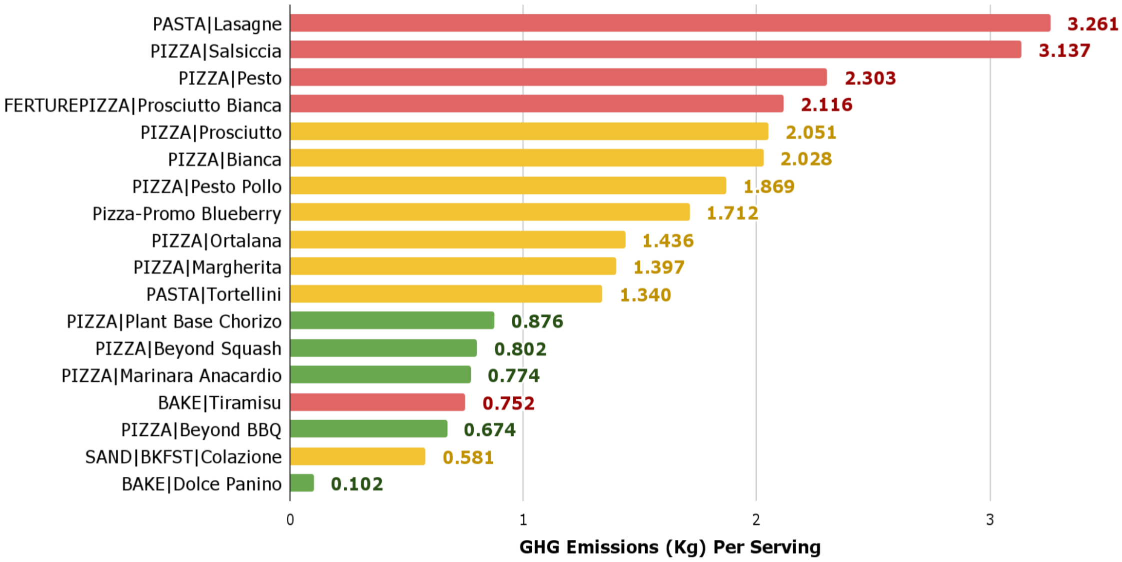
Figure 3: Evaluation Framework Flowchart

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, we are planning to assign the ingredients with additional quantitative criteria data (i.e., land use) for the fall launch. After these data gets processed in the automated calculation step/external 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 UBC Food Services.

3.2 SUMMER PILOT

The summer pilot for the operationalization of CFFS labels took place at the Mercante, one of the UBC Food Services retail venues that remained open during summer 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 external framework, is shown in *Figure* 4:



Note: the GHG emission results are based on 2021 data

Figure 4: GHG Emissions (Kg) Per Serving for Summer Pilot

The corresponding CFFS labels are available to students on the menu boards and also on the Nutrislice. See *Figure* 5 for the actual look of labels.

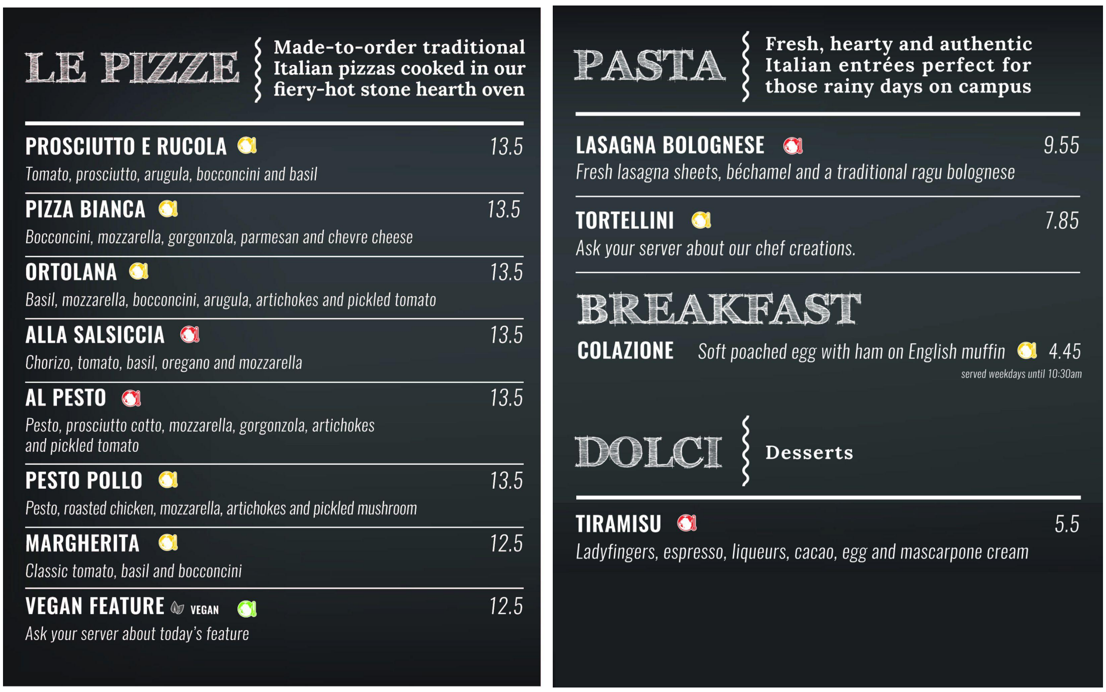
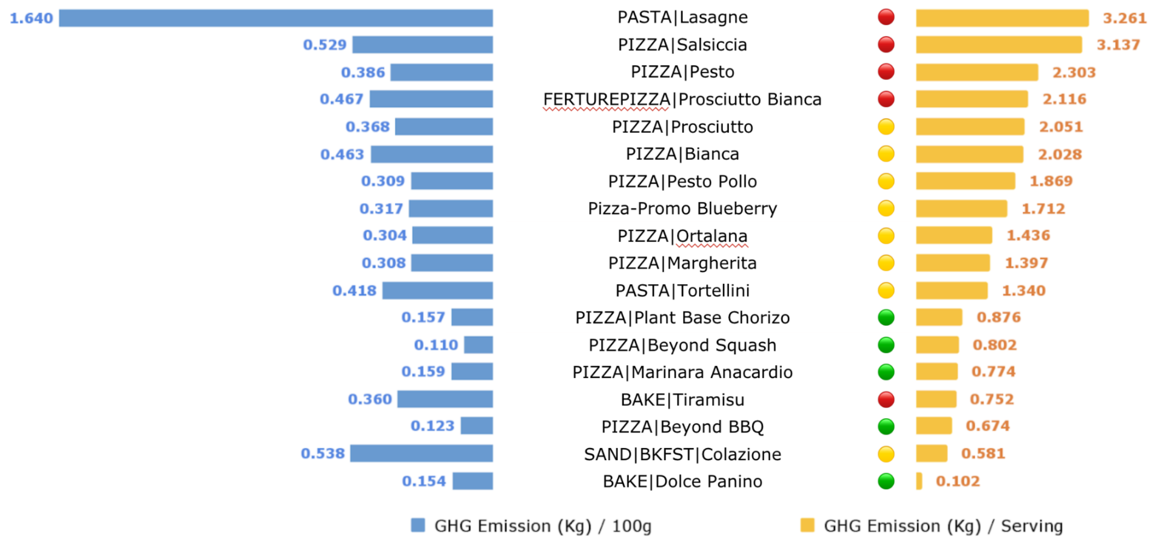


Figure 5: CFFS Label on Menu Board

The external 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, the total emissions are low due to the small serving size. 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, see *Figure* 6.



Note: the GHG emission results are based on 2021 data

Figure 6: GHG Emissions (Kg) Per Serving vs. Per 100g for Summer Pilot

The label cut-offs for the summer pilot are shown in *Figure* 7. 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.

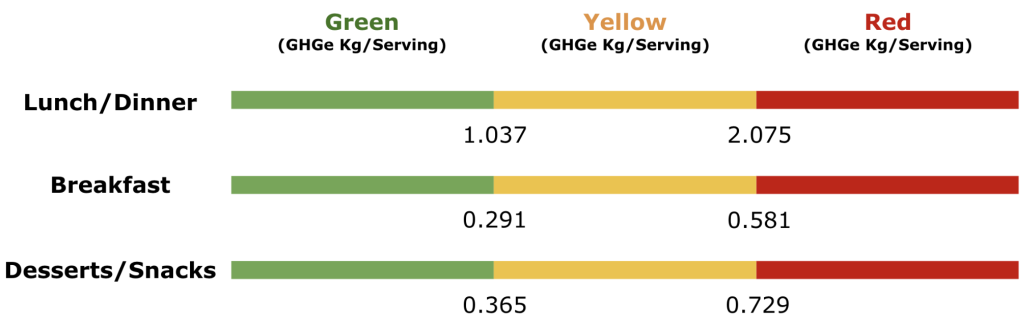


Figure 7: Label Cut-offs for Summer Pilot by Food Groups

3.3 FALL LAUNCH

The fall launch for the operationalization of CFFS labels will take place at the Open Kitchen, one of the three UBC Food Services residence dining halls that open during the 2021-2022 academic year. Besides GHG Emissions, the CFFS label is going to incorporate one additional attribute into the evaluation to produce a more comprehensive evaluation of the climate impact of menu items.

4. DISCUSSION

From the above analysis, we can see that food that contains ruminant meat and dairy products (i.e. beef, lamb, cheese, etc.) tends to have high GHG emissions, both per serving and per 100g measuring method. This could suggest that one way to lower the GHG emissions from the food system is by reducing the amount of 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 CO_2eq, which is equivalent to the emissions from a 11.96-kilometer drive in an average passenger vehicle (average of 206g CO_2 emissions per km driven, Canada Energy Regulator, 2019).

There are also some limitations in the evaluation framework. First, there are several processed products and packaged foods that are directly purchased from external suppliers, such as sauces, dressings, and snacks, etc. Therefore, the evaluation can only take the best estimation of their GHG emission factors by manually calculating the ingredients contained in these products using the available GHG factors.

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

Lastly, 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 the Optimum Control is incomplete for some items, such as the unit information and conversion data, which need to be adjusted and inserted manually.

5. RECOMMENDATIONS

5.1 RECOMMENDATIONS FOR ACTION AND IMPLEMENTATION

To improve the evaluation framework and make it more resilient and suited for expanded operations, the steps below could be used for development:

- 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.

5.2 RECOMMENDATIONS FOR FUTURE RESEARCH

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

6. CONCLUSION

In conclusion, the CFFS label evaluation framework is a resilient approach to conduct the evaluation process in an efficient and structured way that meets the needs for the future expansion of the CFFS label. However, there are a few limitations in the current framework due to the missing information from the data sources 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., Department of Economics, Institutionen för nationalekonomi med statistik, Handelshögskolan, Göteborgs universitet, Gothenburg University, & School of Business, Economics, and Law. (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

Poore, J., and T. Nemecek. 2018. "Reducing Food's Environmental Impacts through Producers and Consumers." *Science 360* (6392):987–92. doi:10.1126/science.aaq0216.

Searchinger, T., R. Waite, C. Hanson, J. Ranganathan, P. Dumas, and E. Matthews. 2019. "World Resources Report: Creating a Sustainable Food Future—A Menu of Solutions to Feed Nearly 10 Billion People by 2050 (Final Report)." Washington, DC: World Resources Institute. http://www.sustainablefoodfuture.org.

Waite, R., D. Vennard, and G. Pozzi. 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. Washington, DC: World Resources Institute. Available online at: www.coolfoodpledge.org.

APPENDICES

APPENDIX A [CODE FOR EXTERNAL FRAMEWORK]

Climate-Friendly Food Systems (CFFS) Labelling Project

The University of British Columbia

Created by Silvia Huang

Part I: Data Preprocessing

Set up and Import Libraries

In [1]:
```
#install libraries if needed
#!pip3 install pdpipe
#!pip install watermark
```

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 directory into the project folder
os.chdir("/Users/silvia/cffs-label")
```

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

Load Data Files

Set Data File Path

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

Out[5]:
```
['/Users/silvia/cffs-label/data/raw/OK 21-22 Sep-Dec/OK Al Forno_Custom Kitchen_Dim Sum_Global.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22 Sep-Dec/OK Square Meal.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22 Sep-Dec/OK Sandwich Kitchen_Sides_Soup.oc',
 '/Users/silvia/cffs-label/data/raw/OK 21-22 Sep-Dec/OK Grill Kitchen Break_Grill Kitchen Day_Grill Kitchen Features.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-7631	5 SPICE POWDER	1.000	ea	1.000	lb	SPICES
1	I-4971	ARTICHOKE 1/4 SALAD CUT TFC	6.000	LG CAN	2.500	Kg	PRODUCE
2	I-4473	AVOCADO (20CT) MX	20.000	CT	1.000	HEAD	PRODUCE
3	I-4973	AVOCADO PULP CHUNKY	12.000	bag	454.000	g	PRODUCE
4	I-4496	BAK CHOY BABY BC	30.000	lb	1.000	lb	PRODUCE
...	...	...	...	...	...	...	...
483	I-29081	YEAST B12 NUTRITIONAL BULK	10.000	Kg	1000.000	g	FOOD - GROCERY
484	I-2171	YOGURT STRAWB POUCH	4.000	POUCH	2.000	Kg	DAIRY
485	I-2281	YOGURT VANILLA STIRRED 650G	1.000	TUB	1.000	650G	DAIRY
486	I-1489	ZEST SUGARED LEMON	5.000	lb	1.000	lb	BAKING-RAW INGREDIENTS
487	I-4967	ZUCCHINI MED FCY MX	25.000	lb	1.000	lb	PRODUCE

488 rows × 7 columns

In [8]:

```
Items.shape
```

Out[8]:

```
(488, 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]:

```
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	I-3388	1.000	L	1.00000000	0.3058	P-10496

	IngredientId	Qty	Uom	Conversion	InvFactor	Recipe
1	I-4660	2.270	Kg	2.20462000	0.6942	P-10496
2	I-4598	1.000	CT	1.00000000	0.0013	P-12954
3	I-4679	1.000	BUNCH	1.00000000	0.0063	P-18318
4	I-4792	10.000	Kg	2.20462000	1.2048	P-18746
...	...	...	...	...	...	...
3222	R-61625	1.000	ea	1.00000000	1.0000	R-65038
3223	R-33558	1.000	ea	1.00000000	1.0000	R-65039
3224	R-33558	1.000	ea	1.00000000	1.0000	R-65040
3225	R-41877	1.000	ea	1.00000000	1.0000	R-65040
3226	R-64997	1.000	ea	1.00000000	1.0000	R-65042

3227 rows × 6 columns

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

Out[13]:
```
(3227, 6)
```

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

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

In [15]:
```
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-56398	BATCH|Guacamole	2.750	Kg	PREP
1	P-24750	CHOPPED|Cilantro	0.500	Kg	
2	P-41574	COOKED|Black Beans	30.000	Kg	PREP
3	P-26068	COOKED|Caramelized Onion	1.200	Kg	PREP
4	P-28258	COOKED|Chow Mein	48.081	Kg	PREP
...	...	...	...	...	...
488	P-16305	YIELD|Smokie (1pc)	1.000	ea	
489	P-50781	YIELD|Thai Basil	200.000	g	
490	P-50676	YIELD|Thyme	300.000	g	

	PrepId	Description	PakQty	PakUOM	InventoryGroup
491	P-46833	YIELD|Yam Fries	800.000	g	
492	P-57145	YIELD|Yellow Pepper	8.300	Kg	

493 rows × 5 columns

In [18]:
```
Preps.shape
```
Out[18]:
```
(493, 5)
```

In [19]:
```
Preps.dtypes
```
Out[19]:
```
PrepId            object
Description       object
PakQty            object
PakUOM            object
InventoryGroup    object
dtype: object
```

In [20]:
```
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-30154	ADD|Crackers	OK - CUSTOM KITCHEN
1	R-56337	ALF|Flatbread|Mediterranean	OK - AL FORNO
2	R-61779	ALF|Flatbread|Mushroom Pesto	OK - AL FORNO
3	R-50590	ALF|Flatbread|OK	OK - AL FORNO
4	R-50494	ALF|Flatbread|Proscuitto	OK - AL FORNO
...	...	...	...
316	R-64095	THANKSGIVING ONION GRAVY	OK - SQUARE MEAL
317	R-30673	THANKSGIVING PUMPKIN PIE	OK - SQUARE MEAL
318	R-35341	VEG|Bowl|Polenta	OK - SIDES
319	R-56637	VEG|French Toast|Eggnog	OK - SIDES
320	R-56451	VEG|FrenchToast|Corn Flake	OK - GRILL KITCHEN BREAKFAST

321 rows × 3 columns

In [23]:
```
Products.shape
```
Out[23]:
```
(321, 3)
```

In [24]:
```
Products.dtypes
```
Out[24]:
```
ProdId         object
Description    object
SalesGroup     object
dtype: object
```

In [25]:
```
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
...	...	...	...	...	...	...
265	I-25492	0.00495050	1.0000	ea	202.0000	g
266	I-27407	0.01333333	1.0000	ea	75.0000	g
267	I-43559	0.02000000	1.0000	CT	50.0000	g
268	I-47525	0.00408163	1.0000	cup	245.0000	g
269	I-63034	0.01098901	1.0000	CT	91.0000	g

270 rows × 6 columns

In [28]:
```
Conversions.shape
```

Out[28]:
```
(270, 6)
```

In [29]:
```
Conversions.dtypes
```

Out[29]:
```
ConversionId      object
Multiplier        object
ConvertFromQty    object
ConvertFromUom    object
ConvertToQty      object
ConvertToUom      object
dtype: object
```

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

Data Summary

In [31]:
```
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	488	7
Preps	493	5
Ingredients	3227	6
Products	321	3
Conversions	270	6

Climate-Friendly Food Systems (CFFS) Labelling Project

The University of British Columbia

Created by Silvia Huang

Part II: Data Cleaning

Set up and Import Libraries

In [1]:
```
#install libraries if needed
#!pip3 install pdpipe
```

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 directory into the project folder
os.chdir("/Users/silvia/cffs-label")
```

In [4]:
```
#enable reading data in the scrolling window
#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-7631	5 SPICE POWDER	1.0	ea	1.0	lb	SPICES
1	I-4971	ARTICHOKE 1/4 SALAD CUT TFC	6.0	LG CAN	2.5	Kg	PRODUCE
2	I-4473	AVOCADO (20CT) MX	20.0	CT	1.0	HEAD	PRODUCE
3	I-4973	AVOCADO PULP CHUNKY	12.0	bag	454.0	g	PRODUCE
4	I-4496	BAK CHOY BABY BC	30.0	lb	1.0	lb	PRODUCE

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

Out[7]:
```
(488, 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	I-3388	1.00	L	1.00000	0.3058	P-10496
1	I-4660	2.27	Kg	2.20462	0.6942	P-10496
2	I-4598	1.00	CT	1.00000	0.0013	P-12954
3	I-4679	1.00	BUNCH	1.00000	0.0063	P-18318
4	I-4792	10.00	Kg	2.20462	1.2048	P-18746

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

Out[10]: (3227, 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-56398	BATCH|Guacamole	2.750	Kg	PREP
1	P-24750	CHOPPED|Cilantro	0.500	Kg	NaN
2	P-41574	COOKED|Black Beans	30.000	Kg	PREP
3	P-26068	COOKED|Caramelized Onion	1.200	Kg	PREP
4	P-28258	COOKED|Chow Mein	48.081	Kg	PREP

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

Out[13]: (493, 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-30154	ADD|Crackers	OK - CUSTOM KITCHEN
1	R-56337	ALF|Flatbread|Mediterranean	OK - AL FORNO
2	R-61779	ALF|Flatbread|Mushroom Pesto	OK - AL FORNO
3	R-50590	ALF|Flatbread|OK	OK - AL FORNO
4	R-50494	ALF|Flatbread|Proscuitto	OK - AL FORNO

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

Out[16]: (321, 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]: (270, 6)

Update Conversion List

In [20]:
```
Update_Conv = pd.read_csv(os.path.join(os.getcwd(), "data", "cleaning", "update", "Conv_UpdateConv.csv"))
Update_Conv.head()
```

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

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
...	...	...	...	...	...	...
479	P-7523	0.035242	16.0	ea	454.00	g
480	P-7637	0.016909	1.0	srvg	59.14	ml
481	P-9779	0.068267	768.0	slice	11250.00	g
482	I-29665	0.025000	1.0	each	40.00	g
483	I-32263	0.033333	1.0	ea	30.00	g

484 rows × 6 columns

In [24]:
```
path = os.path.join(os.getcwd(), "data", "preprocessed", "Conversions_List.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

	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]:

```
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
0	I-8856	6.000	ea	1.0	1.0000	R-64671
1	I-64492	1.000	LOAF	1.0	0.0625	P-26234
2	I-1254	0.500	CT	1.0	0.5000	P-28369
3	I-1273	1.000	LOAF	1.0	0.1351	P-58370
4	I-62225	4.000	CT	1.0	0.4444	P-64456
5	I-62736	1.000	ea	1.0	1.0000	R-28249

	IngredientId	Qty	Uom	Conversion	InvFactor	Recipe
8	I-2281	0.076	650G	1.0	1.0000	R-30524
9	I-2669	0.125	PIE	1.0	1.0000	R-30673
13	I-1223	1.000	CT	1.0	1.0000	R-54456
30	I-1252	1.000	CT	1.0	1.0000	R-64997

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-56398	BATCH|Guacamole	2.750	Kg	PREP	2750.000000	g
1	P-24750	CHOPPED|Cilantro	0.500	Kg	NaN	500.000000	g
2	P-41574	COOKED|Black Beans	30.000	Kg	PREP	30000.000000	g
3	P-26068	COOKED|Caramelized Onion	1.200	Kg	PREP	1200.000000	g
4	P-28258	COOKED|Chow Mein	48.081	Kg	PREP	48081.000000	g
...	...	...	...	...	...	...	...
488	P-16305	YIELD|Smokie (1pc)	1.000	ea	NaN	112.000005	g
489	P-50781	YIELD|Thai Basil	200.000	g	NaN	200.000000	g
490	P-50676	YIELD|Thyme	300.000	g	NaN	300.000000	g
491	P-46833	YIELD|Yam Fries	800.000	g	NaN	800.000000	g
492	P-57145	YIELD|Yellow Pepper	8.300	Kg	NaN	8300.000000	g

493 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-33556	COOKED| Fried Fish	1.0	each	NaN	1.0	each
1	P-64456	POP-UP|Coconut|Flan|LM	9.0	PTN	NaN	9.0	PTN
2	P-64513	POP|Salmon|En|Papillote|LM	1.0	PTN	NaN	1.0	PTN
3	P-44585	PREP|Lime|WEDGE	8.0	piece	NaN	8.0	piece

	PrepId	Description	PakQty	PakUOM	InventoryGroup	StdQty	StdUom
4	P-64944	Sesame|Tuna	1.0	PTN	NaN	1.0	PTN

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
0	P-33556	COOKED| Fried Fish	1.0	each	NaN	1.0	each
1	P-64456	POP-UP|Coconut|Flan|LM	9.0	PTN	NaN	9.0	PTN
2	P-64513	POP|Salmon|En|Papillote|LM	1.0	PTN	NaN	1.0	PTN
3	P-44585	PREP|Lime|WEDGE	8.0	piece	NaN	8.0	piece
4	P-64944	Sesame|Tuna	1.0	PTN	NaN	1.0	PTN

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]:
```
(1937, 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	5	7
Items_Nonstd	10	6

Climate-Friendly Food Systems (CFFS) Labelling Project

The University of British Columbia

Created by Silvia Huang

Part III: Update Information and Mapping

Set up and Import Libraries

In [1]:
```
#install libraries if needed
#!pip3 install pdpipe
```

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 directory into the project folder
os.chdir("/Users/silvia/cffs-label")
```

In [4]:
```
#enable reading data in the scrolling window
#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-56398	BATCH|Guacamole	2.750	Kg	PREP	2750.0	g
1	P-24750	CHOPPED|Cilantro	0.500	Kg	NaN	500.0	g
2	P-41574	COOKED|Black Beans	30.000	Kg	PREP	30000.0	g
3	P-26068	COOKED|Caramelized Onion	1.200	Kg	PREP	1200.0	g
4	P-28258	COOKED|Chow Mein	48.081	Kg	PREP	48081.0	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
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

	Category ID	Food Category	g N lost/kg product
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_7.csv"))
New_Items_Added
```

Out[11]:

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup
0	I-13422	59	BURGER 4OZ NATURAL HALAL	1	cs	42	CT	MEAT
1	I-63034	59	BURGER VEG MALIBU GARDENBURGER	48	CT	1	CT	FOOD - GROCERY
2	I-64468	4	CHICK BREAST CRUNCH BREADED FZ	4	Kg	1	Kg	POULTRY
3	I-1254	24	CIABATTA BUN 5"X 5" PLAIN	12	CT	1	CT	BREAD
4	I-62225	11	EGG LRG 15 DOZEN LOOSE	15	DOZ	12	CT	DAIRY
5	I-62736	24	HAMBURGER BUN WW VEGAN 100gr	1	ea	1	ea	BREAD
6	I-64492	24	LOAF GARLIC BREAD	1	LOAF	1	LOAF	BREAD
7	I-3356	9	MILK CONDENSED SWEET	24	SM CAN	300	ml	FOOD - GROCERY
8	I-1223	24	PANINI SUB ITALIAN - 7"	12	CT	1	CT	BREAD
9	I-53707	54	PAPRIKA BULK	5	lb	1	lb	SPICES
10	I-5115	12	SAL SOX FLT S/ON PBO PRV OW	1	lb	1	lb	SEAFOOD
11	I-29357	55	Soup 1 (10L mon-fri)	10	L	1	L	PRODUCTION FOOD
12	I-29356	55	Soup 2 (10L mon-fri)	10	L	1	L	PRODUCTION FOOD
13	I-1273	24	SOURDOUGH BREAD COUNTRY	1	LOAF	1	LOAF	BREAD

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup
14	I-63866	37	TOMATO POLPA MUTTI	10	Kg	1	Kg	FOOD - GROCERY
15	I-2281	10	YOGURT VANILLA STIRRED 650G	1	TUB	1	650G	DAIRY

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)
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-56398	BATCH|Guacamole	2.750	Kg	PREP	2750.0	g
1	P-24750	CHOPPED|Cilantro	0.500	Kg	NaN	500.0	g
2	P-41574	COOKED|Black Beans	30.000	Kg	PREP	30000.0	g
3	P-26068	COOKED|Caramelized Onion	1.200	Kg	PREP	1200.0	g
4	P-28258	COOKED|Chow Mein	48.081	Kg	PREP	48081.0	g

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

Out[16]: (493, 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]:
```
(1937, 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
...	...	...	...	...	...	...	...	...	...
1932	I-29357	55	Soup 1 (10L mon-fri)	10.00	L	1.0	L	PRODUCTION FOOD	0.0000
1933	I-29356	55	Soup 2 (10L mon-fri)	10.00	L	1.0	L	PRODUCTION FOOD	0.0000
1934	I-1273	24	SOURDOUGH BREAD COUNTRY	1.00	LOAF	1.0	LOAF	BREAD	1.5225
1935	I-63866	37	TOMATO POLPA MUTTI	10.00	Kg	1.0	Kg	FOOD - GROCERY	0.6932
1936	I-2281	10	YOGURT VANILLA STIRRED 650G	1.00	TUB	1.0	650G	DAIRY	2.9782

1937 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

	ItemId	CategoryID	Description	CaseQty	CaseUOM	PakQty	PakUOM	InventoryGroup	Active Total Supply Chain Emissions (kg CO2 / kg food)	g N lost/kg product
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT	41.3463	329.50
...	...	...	...	...	...	...	...	...	...	...
1932	I-29357	55	Soup 1 (10L mon-fri)	10.00	L	1.0	L	PRODUCTION FOOD	0.0000	0.00
1933	I-29356	55	Soup 2 (10L mon-fri)	10.00	L	1.0	L	PRODUCTION FOOD	0.0000	0.00
1934	I-1273	24	SOURDOUGH BREAD COUNTRY	1.00	LOAF	1.0	LOAF	BREAD	1.5225	14.80
1935	I-63866	37	TOMATO POLPA MUTTI	10.00	Kg	1.0	Kg	FOOD - GROCERY	0.6932	7.90
1936	I-2281	10	YOGURT VANILLA STIRRED 650G	1.00	TUB	1.0	650G	DAIRY	2.9782	26.07

1937 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	1677.200	61309.000
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.200	61309.000
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.200	61309.000
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT	41.3463	329.50	1677.200	61309.000
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT	41.3463	329.50	1677.200	61309.000
...	...	...	...	...	...	...	...	...	...	...	...	...
1932	I-29357	55	Soup 1 (10L mon-fri)	10.00	L	1.0	L	PRODUCTION FOOD	0.0000	0.00	1.000	1.000
1933	I-29356	55	Soup 2 (10L mon-fri)	10.00	L	1.0	L	PRODUCTION FOOD	0.0000	0.00	1.000	1.000
1934	I-1273	24	SOURDOUGH BREAD COUNTRY	1.00	LOAF	1.0	LOAF	BREAD	1.5225	14.80	419.200	12821.700
1935	I-63866	37	TOMATO POLPA MUTTI	10.00	Kg	1.0	Kg	FOOD - GROCERY	0.6932	7.90	77.000	4480.700
1936	I-2281	10	YOGURT VANILLA STIRRED 650G	1.00	TUB	1.0	650G	DAIRY	2.9782	26.07	262.409	13002.612

1937 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]:
```
(1937, 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	1677.200	61309.000
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.200	61309.000
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.200	61309.000
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT	41.3463	329.50	1677.200	61309.000
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT	41.3463	329.50	1677.200	61309.000
...	...	...	...	...	...	...	...	...	...	...	...	...
1932	I-29357	55	Soup 1 (10L mon-fri)	10.00	L	1.0	L	PRODUCTION FOOD	0.0000	0.00	1.000	1.000
1933	I-29356	55	Soup 2 (10L mon-fri)	10.00	L	1.0	L	PRODUCTION FOOD	0.0000	0.00	1.000	1.000
1934	I-1273	24	SOURDOUGH BREAD COUNTRY	1.00	LOAF	1.0	LOAF	BREAD	1.5225	14.80	419.200	12821.700
1935	I-63866	37	TOMATO POLPA MUTTI	10.00	Kg	1.0	Kg	FOOD - GROCERY	0.6932	7.90	77.000	4480.700
1936	I-2281	10	YOGURT VANILLA STIRRED 650G	1.00	TUB	1.0	650G	DAIRY	2.9782	26.07	262.409	13002.612

1937 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

Part IV: Data Analysis

Set up and Import Libraries

In [1]:
```
#install libraries if needed
#!pip3 install pdpipe
```

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 directory into the project folder
os.chdir("/Users/silvia/cffs-label")
```

In [4]:
```
#enable reading data in the scrolling window
#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-7631	5 SPICE POWDER	1.0	ea	1.0	lb	SPICES
1	I-4971	ARTICHOKE 1/4 SALAD CUT TFC	6.0	LG CAN	2.5	Kg	PRODUCE
2	I-4473	AVOCADO (20CT) MX	20.0	CT	1.0	HEAD	PRODUCE
3	I-4973	AVOCADO PULP CHUNKY	12.0	bag	454.0	g	PRODUCE
4	I-4496	BAK CHOY BABY BC	30.0	lb	1.0	lb	PRODUCE

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	I-3388	1.00	L	1.00000	0.3058	P-10496
1	I-4660	2.27	Kg	2.20462	0.6942	P-10496
2	I-4598	1.00	CT	1.00000	0.0013	P-12954
3	I-4679	1.00	BUNCH	1.00000	0.0063	P-18318
4	I-4792	10.00	Kg	2.20462	1.2048	P-18746

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]:
```
(493, 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-30154	ADD|Crackers	OK - CUSTOM KITCHEN
1	R-56337	ALF|Flatbread|Mediterranean	OK - AL FORNO
2	R-61779	ALF|Flatbread|Mushroom Pesto	OK - AL FORNO
3	R-50590	ALF|Flatbread|OK	OK - AL FORNO
4	R-50494	ALF|Flatbread|Proscuitto	OK - AL FORNO

In [13]:
```
Conversions = pd.read_csv(os.path.join(os.getcwd(), "data", "preprocessed", "Conversions_List.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
...	...	...	...	...	...	...
479	P-7523	0.035242	16.0	ea	454.00	g
480	P-7637	0.016909	1.0	srvg	59.14	ml
481	P-9779	0.068267	768.0	slice	11250.00	g
482	I-29665	0.025000	1.0	each	40.00	g
483	I-32263	0.033333	1.0	ea	30.00	g

484 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	1677.200	61309.000
1	I-10869	1	BEEF STIRFRY COV FR	5.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.200	61309.000
2	I-7064	1	BEEF OUTSIDE FLAT AAA	1.00	Kg	1.0	Kg	MEAT	41.3463	329.50	1677.200	61309.000
3	I-37005	1	BEEF MEATBALLS	4.54	Kg	1000.0	g	MEAT	41.3463	329.50	1677.200	61309.000
4	I-37002	1	BEEF INSIDE ROUND SHAVED	9.00	Kg	1000.0	g	MEAT	41.3463	329.50	1677.200	61309.000
...	...	...	...	...	...	...	...	...	...	...	...	...
1932	I-29357	55	Soup 1 (10L mon-fri)	10.00	L	1.0	L	PRODUCTION FOOD	0.0000	0.00	1.000	1.000
1933	I-29356	55	Soup 2 (10L mon-fri)	10.00	L	1.0	L	PRODUCTION FOOD	0.0000	0.00	1.000	1.000
1934	I-1273	24	SOURDOUGH BREAD COUNTRY	1.00	LOAF	1.0	LOAF	BREAD	1.5225	14.80	419.200	12821.700
1935	I-63866	37	TOMATO POLPA MUTTI	10.00	Kg	1.0	Kg	FOOD - GROCERY	0.6932	7.90	77.000	4480.700
1936	I-2281	10	YOGURT VANILLA STIRRED 650G	1.00	TUB	1.0	650G	DAIRY	2.9782	26.07	262.409	13002.612

1937 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

	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 [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
...	...	...	...	...	...	...
649	P-50641		1.0	un	650.00	g
650	P-49636		12.0	ea	1440.00	g
651	P-14833		1.0	PTN	500.00	g
652	P-47365		2.0	each	100.00	g
653	NaN	NaN	NaN	NaN	NaN	NaN

654 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)/StdUo
0	P-56398	BATCH|Guacamole	2.750	Kg	PREP	2750.0	g	1486.216896	0.540443	11.000185	0.0040
1	P-24750	CHOPPED|Cilantro	0.500	Kg	NaN	500.0	g	362.904983	0.725810	5.700834	0.0114
2	P-41574	COOKED|Black Beans	30.000	Kg	PREP	30000.0	g	12625.079663	0.420836	97.526164	0.0032
3	P-26068	COOKED|Caramelized Onion	1.200	Kg	PREP	1200.0	g	2629.549652	2.191291	22.086666	0.0184
4	P-28258	COOKED|Chow Mein	48.081	Kg	PREP	48081.0	g	54907.550000	1.141980	518.100000	0.0107
...	...	...	...	...	...	...	...	...	...	...	
488	P-16305	YIELD|Smokie (1pc)	1.000	ea	NaN	112.0	g	1101.128000	9.831500	14.873600	0.1328
489	P-50781	YIELD|Thai Basil	200.000	g	NaN	200.0	g	228.111417	1.140557	3.583377	0.0179
490	P-50676	YIELD|Thyme	300.000	g	NaN	300.0	g	228.316600	0.761055	3.586600	0.0119
491	P-46833	YIELD|Yam Fries	800.000	g	NaN	800.0	g	397.440000	0.496800	5.000000	0.0062
492	P-57145	YIELD|Yellow Pepper	8.300	Kg	NaN	8300.0	g	5029.000000	0.605904	79.000000	0.0095

493 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
0	P-33556	COOKED| Fried Fish	1.0	each	NaN	1.0	each
1	P-64456	POP-UP|Coconut|Flan|LM	9.0	PTN	NaN	9.0	PTN
2	P-64513	POP|Salmon|En|Papillote|LM	1.0	PTN	NaN	1.0	PTN
3	P-44585	PREP|Lime|WEDGE	8.0	piece	NaN	8.0	piece
4	P-64944	Sesame|Tuna	1.0	PTN	NaN	1.0	PTN

In [44]:

```
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]:

```
for index, row in Products.iterrows():
    filter_products(index, row)
```

```
P-64456 61 R-64457
P-64944 79 R-64945
P-33556 81 R-41868
P-33556 102 R-33558
P-64513 144 R-64487
```

In [46]:

```
Products['GHG Emission (g) / 100g'] = round(100*Products['GHG Emission (g)']/Products['Weight (g)'], 3)
Products['N lost (g) / 100g'] = round(100*Products['N lost (g)']/Products['Weight (g)'], 3)
Products['Freshwater Withdrawals (ml) / 100g'] = round(100*Products['Freshwater Withdrawals (ml)']/Products['Weight (g)']
Products['Stress-Weighted Water Use (ml) / 100g'] = round(100*Products['Stress-Weighted Water Use (ml)']/Products['Weight
```

In [47]:

```
Products
```

Out[47]:

	ProdId	Description	SalesGroup	Weight (g)	GHG Emission (g)	N lost (g)	Freshwater Withdrawals (ml)	Stress-Weighted Water Use (ml)	GHG Emission (g) / 100g	N lost (g) / 100g	Freshwa Withdraw (ml) / 1C
0	R-30154	ADD|Crackers	OK - CUSTOM KITCHEN	6.000000	9.135000	0.088800	2515.199950	7.693020e+04	152.250	1.480	41920.(
1	R-56337	ALF|Flatbread|Mediterranean	OK - AL FORNO	145.000000	313.698698	3.172877	56392.170652	2.459800e+06	216.344	2.188	38891.
2	R-61779	ALF|Flatbread|Mushroom Pesto	OK - AL FORNO	205.000000	597.890333	5.311554	84931.503114	3.685929e+06	291.654	2.591	41430.(
3	R-50590	ALF|Flatbread|OK	OK - AL FORNO	165.000000	366.387741	3.575417	58538.457445	2.584866e+06	222.053	2.167	35477.8
4	R-50494	ALF|Flatbread|Proscuitto	OK - AL FORNO	210.000000	781.674378	8.992882	137562.460815	5.211512e+06	372.226	4.282	65505.9
...	...	...	...	...	...	...	...	...	...	...	
316	R-64095	THANKSGIVING ONION GRAVY	OK - SQUARE MEAL	90.000000	39.004197	0.640408	1600.018883	5.590614e+04	43.338	0.712	1777.7
317	R-30673	THANKSGIVING PUMPKIN PIE	OK - SQUARE MEAL	0.125000	0.190312	0.001850	52.400000	1.602713e+03	152.250	1.480	41920.(
318	R-35341	VEG|Bowl|Polenta	OK - SIDES	325.000000	833.017684	7.191962	67809.930649	4.046229e+06	256.313	2.213	20864.5
319	R-56637	VEG|French Toast|Eggnog	OK - SIDES	329.500001	758.878624	9.393660	110045.912281	3.348081e+06	230.312	2.851	33397.8
320	R-56451	VEG|FrenchToast|Corn Flake	OK - GRILL KITCHEN BREAKFAST	389.500001	927.297378	9.185953	139427.538821	3.749196e+06	238.074	2.358	35796.5

316 rows × 12 columns

In [48]:

```
Products.shape
```

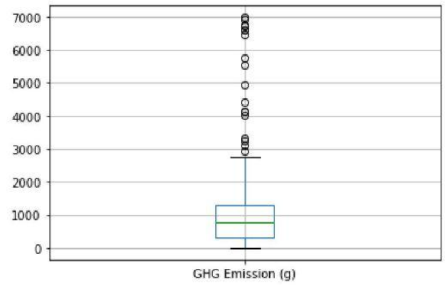
Out[48]:

```
(316, 12)
```

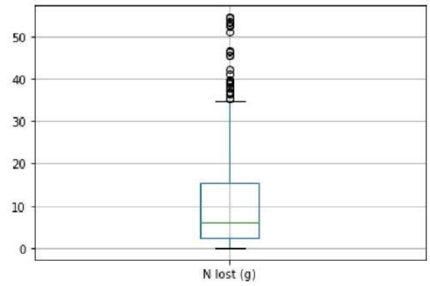
In [49]:

```
path = os.path.join(os.getcwd(), "data", "final", "Recipes Footprints.csv")
Products.to_csv(path, index = False, header = True)
```

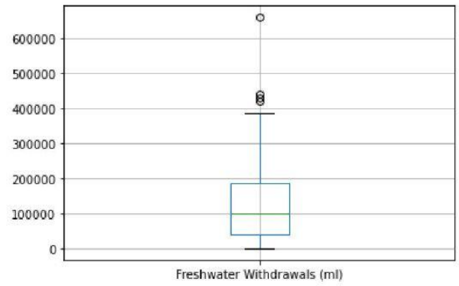
In [50]:
```
Products.boxplot(column=['GHG Emission (g)'], return_type='axes')
```
Out[50]: <AxesSubplot:>



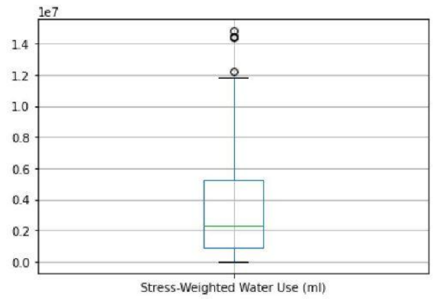
In [51]:
```
Products.boxplot(column=['N lost (g)'], return_type='axes')
```
Out[51]: <AxesSubplot:>



In [52]:
```
Products.boxplot(column=['Freshwater Withdrawals (ml)'], return_type='axes')
```
Out[52]: <AxesSubplot:>

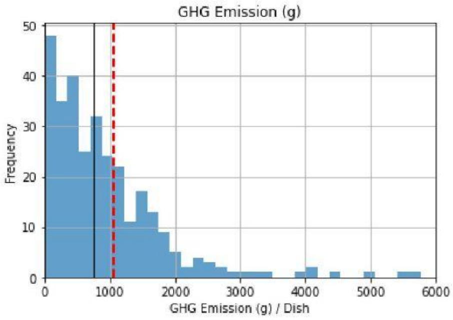


In [53]:
```
Products.boxplot(column=['Stress-Weighted Water Use (ml)'], return_type='axes')
```
Out[53]: <AxesSubplot:>

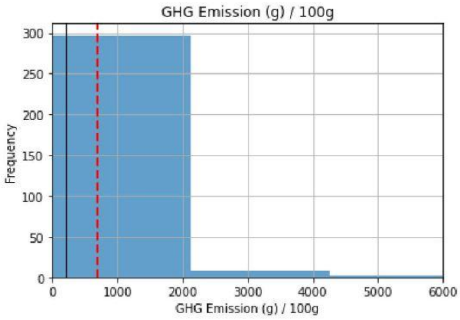


In [54]:
```
Products.hist(column=['GHG Emission (g)'], bins= 40, alpha = 0.7)
plt.axvline(Products['GHG Emission (g)'].mean(), color='r', linestyle='dashed', linewidth=2, label = 'mean' )
plt.axvline(Products['GHG Emission (g)'].median(), color='k', linewidth=1, label = 'median')
plt.xlabel('GHG Emission (g) / Dish')
plt.ylabel('Frequency')
```

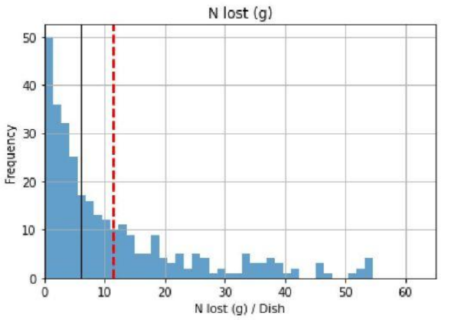
```
plt.xlim([0, 6000])
plt.savefig('GHGe_dish.png')
```



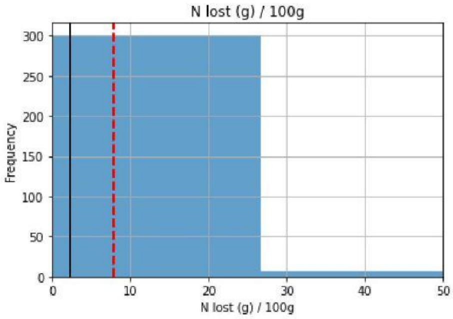
In [55]:
```
Products.hist(column=['GHG Emission (g) / 100g'], bins= 40, alpha = 0.7)
plt.axvline(Products['GHG Emission (g) / 100g'].mean(), color='r', linestyle='dashed', linewidth=2, label = 'mean' )
plt.axvline(Products['GHG Emission (g) / 100g'].median(), color='k', linewidth=1, label = 'median')
plt.xlabel('GHG Emission (g) / 100g')
plt.ylabel('Frequency')
plt.xlim([0, 6000])
plt.savefig('GHGe_100g.png')
```



In [56]:
```
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([0, 65])
plt.savefig('N lost_dish.png')
```



In [57]:
```
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([0, 50])
plt.savefig('N lost_100g.png')
```


N lost (g) / 100g
Frequency
300
250
200
150
100
50
0
0
10
20
30
40
50
N lost (g) / 100g

APPENDIX B [GHG EMISSION FACTORS LIST]

Category ID	Food Category	Active Total Supply Chain Emissions (kg CO2 / kg food)
1	beef & buffalo meat	41.3463
2	lamb/mutton & goat meat	41.6211
3	pork (pig meat)	9.8315
4	poultry (chicken, turkey)	4.3996
5	butter	11.4316
6	cheese	8.9104
7	ice cream	4.0163
8	cream	6.9824
9	milk (cow's milk)	2.2325
10	yogurt	2.9782
11	eggs	3.6615
12	fish (finfish)	4.9798
13	crustaceans (shrimp/prawns)	21.1274
14	mollusks	2.4351
15	animal fats	6.9693
16	other legumes	1.6042
17	beans and pulses (dried)	1.6776
18	peas	0.6995
19	peanuts/groundnuts	1.692
20	soybeans/tofu	1.7542
21	other grains/cereals	1.4785
22	corn (maize)	0.9734
23	oats (oatmeal)	2.3017
24	wheat/rye (bread, pasta, baked goods)	1.5225
25	rice	2.5345
26	tree nuts and seeds	4.2854
27	almond milk	0.7021
28	oat milk	0.9943
29	rice milk	0.6972
30	soy milk	0.489
31	other fruits	0.4306
32	apples	0.3581
33	bananas	0.7115

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