



(Source: Therail.media. Online)

Data-Driven Restaurants

How to Use Data to Create Attractive Menus?

School: Lucerne University of Applied Sciences and Arts (HSLU)

Program: Master of Science in Applied Information and Data Science

Chair of the Department: Andreas Brandenburg

Course: Data Warehouse and Data Lake Systems 2

Examiners: Luis Terán, José Mancera, Aigul Kaskina

Authors:

Ezgi Köker Gökgöl, 20-289-328, 8200 Schaffhausen, ezgi.koekergoekgoel@stud.hslu.ch

Vithushan Mahendran, 12-737-573, 4663 Aarburg, vithushan.mahendran@stud.hslu.ch

Alexandra Alinka Zimmermann, 16-209-173, 6003 Lucerne, alexandra.zimmermann@stud.hslu.ch

Project Source Code: https://github.com/vithu92/deep_divers

Tableau Dashboard: [LINK](#)

Lucerne, 8th of June 2022

Table of Contents

1	<i>Introduction</i>	1
1.1	Topic Introduction	1
1.2	Problem Definition	1
1.3	Goals.....	1
1.4	Business and Research Questions.....	2
1.5	Limitations.....	2
2	<i>State of the Art</i>	3
2.1	Data Sources	3
2.2	Google Trends	4
2.3	Edamam.....	4
2.4	Coop	4
3	<i>Methodology</i>	5
3.1	Data Quality	5
3.1.1	Google Trends	5
3.1.2	Edamam	6
3.1.3	Coop.....	7
3.2	Production Data Pipelines.....	8
3.2.1	Google Trends	8
3.2.2	Edamam	8
3.2.3	Coop.....	9
3.3	Data Warehouse architecture, System etc. in technical details.....	9
3.3.1	Database architecture	10
3.3.2	Tools used.....	10
3.3.3	New User Role.....	11
3.3.4	ER diagram	11
3.4	Visualization.....	11
3.4.1	Visualization Design Justification	11
3.4.2	Tableau Visualization (with public link)	12
3.5	Answer to research/business questions.....	13
4	<i>Discussion</i>	16
4.1	Google Trends	16
4.2	Edamam.....	16
4.3	Coop Website	17
4.4	Tableau Dashboard.....	18
5	<i>Conclusions</i>	18
6	<i>Future Work</i>	19

6.1	Google Trends	19
6.2	Edamam	19
6.3	Coop	19
6.4	Data warehouse.....	19
6.5	Visualization (Tableau)	19
7	<i>Final thoughts</i>	20
8	<i>Appendix</i>	21
Appendix A	Data Lake for Google Trends Data Source	21
Appendix B	Data Lake for Edamam Data Source.....	21
Appendix C	Data Lake of Coop	22
Appendix D	File Repository Coop	22
Appendix E	Code Snippet: upload data to RDS.....	23
Appendix F	Data in Meat Canton Trend Table.....	24
Appendix G	Python Function to Translate Queries to English	24
Appendix H	Python Function to Remove Stop Words.....	25
Appendix I	Python Function to Normalize Diet Names	26
Appendix J	First Rows of the Diet Search Trend Table in the Data Warehouse	26
Appendix K	Extract of the Python Code in the Function clean_product_search	27
Appendix L	First Rows of the Product Search Trend Table in the Data Warehouse.....	27
Appendix M	First Rows of the Recipe Search Trend Table in the Data Warehouse	27
Appendix N	List of Ingredients and their Quantities.....	28
Appendix O	Example Output for Search Word ‘apples’	28
Appendix P	Query that is Used to Get Prices of each Ingredients.....	28
Appendix Q	Function to transform the unit of a product	29
Appendix R	Example how we have transformed the units properly	29
Appendix S	Function to compute the price per unit	29
Appendix T	Python Code for the Data Pipeline for Google Trends Data.....	30
Appendix U	Setup with all Permission except "Super User" to all tables for user "deep_diver"..	31
Appendix V	Setup only SELECT permission to all tables for user "tableau_user"	31
Appendix W	Database Schema	32
Appendix X	Final Data Visualisation Draft	33
Appendix Y	Tableau Dashboard: Canton Filter	34
Appendix Z	Tableau Dashboard: Last 7 Days Filter.....	35
Appendix AA	Tableau Dashboard: Product Filter	36
Appendix BB	Tableau Dashboard: Diet Name Filter	37
Appendix CC	Price Calculation for ‘Banana-chilli pickle’ (max price)	37
Appendix DD	Price Calculation for ‘Cheesy Chicken Italiano’ (0 price).....	37

Appendix EE	Comparison of Tableau Dashboard Versions.....	38
9	<i>Annexes (Work Split)</i>	39
10	<i>Sources</i>	41
10.1	Source Information	41
10.2	GitHub Repository.....	42

List of Abbreviations and Acronyms

API	Application Programming Interface
AWS	Amazon Web Services
CSV	Comma-Separated Values
ETL	Extract, Transform and Load
HTML	Hypertext Markup Language
IAM	Identity and Access Management
JSON	JavaScript Object Notation
KMS	Key Management Service
NLP	Natural Language Processing
RDS	Relational Database Service
SQL	Structured Query Language
URL	Uniform Resource Locator

List of Figures

Figure 1: Example Data in Diet Canton Trend Table	6
Figure 2: Database architecture.....	10
Figure 3: Final Tableau Dashboard.....	13
Figure 4: Diet preference for Canton Lucerne (top) and Canton Zurich (bottom)	14
Figure 5: Evolution of Diet Trend historically.....	14
Figure 6: Screenshot of Trends from 29 th May 2022 for the Canton Zurich	15
Figure 7: First Five Recipes with Highest Prices.....	17
Figure 8: Example Output of the API for an Ingredient	17

List of Tables

Table 1: Data Pipeline for Coop Data Source.....	9
Table 2: AWS Credit Conditions.....	18
Table 3: Evaluation Grid.....	39
Table 4: Code Project Leads	40

1 Introduction

1.1 Topic Introduction

Eating at a restaurant is synonym of pleasure for most of us. One can enjoy taking a break from their daily routine, being pampered for a night-time, while having someone cook for them. One can appreciate the company of their friends or their beloved ones and possibly discover new trendy meals.

From a restaurant's point of view, the choice of the meals is crucial to attract and retain customers and to improve the resource management. Nowadays, people tend to have a look at the menu before going to a restaurant and sometimes expect to see specific meals. Therefore, it is today essential for the restaurants to keep the menus up to date with the latest trends and be aware of the customers' preferences in order to stay ahead of competition [Unilever Food Solutions 2022].

An increasing number of companies have nowadays data analysts which can reveal insights from the large amount of data generated daily. They tend to adopt more data-driven strategies. In many situations, using data helps to make better decisions. For a restaurant owner, data can be used to better plan their food supply, manage their budget, and also to better design their menu. Indeed, if a restaurant is able to collect data about the food preferences of its customers, for instance through surveys or experiments, they would gain a competitive advantage. Another option would be to analyze their clients' web search history, which surely reveals their interests and reduces human conformity biases. Such data is available to anyone on Google Trends. Anyone can know what people often search online, thus their interests and their food preferences. Such insights can be extremely valuable for restaurants designing their menus. They can easily adapt their menus to include ingredients or meals which are trendy at that moment.

1.2 Problem Definition

Designing the menu of a restaurant can be challenging. It should reflect the cook's skills thus, be somehow unique, but also match the customers' tastes so that it attracts most of them. Determining the latter can be extremely complex without the right tool or without a sample large enough. However, we think that, based on the web search history, people's food preferences can be implied. People usually first search online for what they want to eat or to cook. They inform themselves about the new food trends and new cuisines. They look up restaurants' menus, recipes, fancy meals, or anything that they wish they could eat at that moment. Providing such information to restaurant owners can help them in the design of their menu. Once the customers' taste is known, cooks can elaborate recipes based on other similar recipes and add their personal touch to the meals before serving them on the menu. Lastly, to determine the price of the meals or to decide when to offer them on the menu, the ingredients' costs should be monitored and optimized.

1.3 Goals

For this project, we target Pop-Up restaurants, where cooks have much freedom to try new food and meal concepts. They can indeed change their menus very easily.

For the business part, the goal of this project is to help restaurant owners to improve their business. First, we want to support them in the design of their menu and in the choice of their meals based on the population's food taste. To do so, we use a Google Trends API. Second, thanks to Edamam API, we can suggest them various recipes which are based on these meals, diet types, or ingredients preferences. Finally, we also help them to plan their budget. Thus, we provide them with an estimation of the total recipe cost. Therefore, we fetch the ingredients price from the Coop website.

For the technical part, the objectives of our project are first to extract data from different data sources, to be precise, from two APIs, namely Google Trends and Edamam, and to scrape data from the Coop website. Second, we build three data lake systems where we load the data extracted from the three data sources mentioned previously. In the second part of the project, we build a data warehouse, where we clean, analyze, and prepare our data to be able to answer our research questions. Finally, we visualize our data using Tableau Desktop. We provide access to the dashboard to our end-users in order to help them to make better decisions when designing their menus.

1.4 Business and Research Questions

In order to better understand the food sector and then be able to help restaurant owners in the design of their menus, we decided to answer the following research questions for our project:

Are there distinctive food preferences between different regions of Switzerland (e.g. among the Swiss cantons or the language speaking regions)?

We want to gain insights on the customer food preferences in the different cantons of Switzerland, and among its different speaking regions. Indeed, our first assumption is that food tastes are not the same in Switzerland and might depend on the canton or on the speaking areas. Some localities might prefer traditional meals whereas other might lean towards more international and trendy dishes. One example could be the meat preferences. Can we find out whether some cantons prefer typical meat, such as beef or pork, and other have a tendency for alternative protein sources such as tofu or seitan? Such information could be helpful for restaurant owner, that would then be able to adjust their menus according to the location and the local population's taste.

Given a specific food-related search term, are there any changes in the population's food preference over time?

We want to have a better understanding of the evolution of the food preferences in Switzerland, which would then improve our recommendations to restaurant owners. For instance, we can suppose that the search term "chocolate" increases in popularity around Easter but then decreases at the arrival of sunny days.

To answer these first two questions, we use Google Trends data.

For each of the meal category (vegan, vegetarian, etc.), which food items are the most often used in these recipes?

Once we know the meal category preferences of the local population, we want to analyze the recipes that belong to such category. For instance, we ask ourselves whether there are noticeable patterns in the vegan recipes. If so, a restaurant owner would then be able to create a more personalized recipe based on these reoccurring patterns. To answer this question, we use Edamam data.

Are there any distinctive patterns among the discounted items sold by Coop?

Restaurant owners usually must plan their budget and design their menu in the most economical way. If we can show that there are discount patterns amongst the products sold by Coop, restaurant owners can adjust their shopping lists accordingly and buy ingredients when they are the cheapest. It will help them to optimize their budget. To answer this question, we use the scraped data from the Coop website.

1.5 Limitations

This project has some limitations, which are developed below:

1. Short time frame

Since the project started in March, we are only able to collect data from that moment on. Google Trends API does not allow us to fetch detailed historical data over the last years. Indeed, either the data is incomplete, or it lacks details. For Coop website, we have the same challenge. We can only track the prices and the discounts from March on. Therefore, the patterns that we will uncover in the second part of the project will only reflect that short time frame. Additionally, as the dataset is rather small, we should be careful with our conclusions and their accuracy.

2. Number of calls to Edamam API

With the free access version of Edamam API, we are only allowed to fetch 10'000 recipes per month [Edamam 2022]. Therefore, the number of recipes that we can suggest to restaurant owners is large but limited. Additionally, we should be careful to make recommendations which are suitable to a menu in a restaurant.

3. Fetching data from Coop

To estimate the recipe costs, we use the prices on Coop website. However, larger restaurants usually do not buy their ingredients in supermarkets but rather in wholesalers, whose quantities are more suitable to restaurants. Nevertheless, we decided to work with Coop website as it was the option that we were the most familiar with.

4. Text processing

Some of the data that we obtain from Google Trends are textual data. Such data type is more challenging to analyze and rely on some Natural Language Processing (NLP) methods, which some of us are only learning this semester. Therefore, in order to understand the data that we collected, we found simple solutions clean and aggregate similar data into one unique data, for example, *keto diet*, *ketogene diet* and *keto*, which are the same concept. Only then, we were able to provide a precise analysis of our extracted data to the restaurant owners. Nevertheless, there are a few unexpected patterns that we could not handle due to our newly acquired but limited skills in NLP.

5. Limitations within the AWS environment

Currently, the course benefits of only some of the Amazon Web Services (AWS) for a budget of \$100. Therefore, the complexity of our project solution strongly depends on the services and on the budget that are at our disposal.

2 State of the Art

The course Data Warehouse and Data Lake Systems is divided into two parts. In the first one, we built three data lakes using three data sources, namely Google Trends, Edamam, and Coop. This project is the second part of the course, for which we built a data warehouse and created a dashboard on Tableau Desktop. Before detailing our final solution, it is worth summarizing what was done in the first part of the course.

2.1 Data Sources

We first describe the three data sources that we used for this project.

Google Trends

“Google Trends is a website by Google that analyzes the popularity of top search queries in Google Search across various regions and languages” [Wikipedia 2022]. The searching behavior on Google Web Search, Google News, Google Images, Google Shopping, and on YouTube can be analyzed using Google Trends [Hackernoon 2021]. Google Trends returns a measure normalized to the time and the location. It ranges from 0 to 100, which represents the relative popularity, in terms of search volume, of the given query over a specified period of time, for a given geography. 100 is the maximum and 0 represents a lack of data [Google Trends 2022]. Since there are no publicly available Google Trends API [Celis 2020], we decided to use Pytrends, which is an unofficial API for Google Trends and can be considered as a wrapper. The library “pytrends” needs to be installed to then be able to connect and query the Google Trends API.

Edamam

Edamam is a website related to better eating that aims to capture the World's food knowledge and distils to help users to make informed choices at the store and in the kitchen [Edamam 2022]. Using the website, it is possible to analyse the nutritional values of a self-defined meal or individual items, as well as to access various recipes related with a search query, while having the option of selecting different diet options (e.g., vegan, vegetarian) or allergens (e.g., gluten, soy). The website has three possible API's, Nutritional Analysis, Food Database and Recipe Search. For this study, we used Recipe Search API.

Coop Website

Coop is one of the largest grocery stores in Switzerland. Nevertheless, it does not only sell food, but also flowers, books, and other commodities. For this project we were only interested in their food product. To scrap from the Coop product pages, we used the Python packages named “selenium” and “webdriver manager”. So, by combining both packages mentioned previously, we can call the inbuilt local API of the browser software, which in our case is Google Chrome. With this API, we can load specific webpages and retrieve the whole html content as well.

In the next sections, we briefly describe what we did for each data source to build our data lakes.

2.2 Google Trends

First, the data was weekly extracted from the Google Trends API and loaded as six CSV files into an S3 Bucket on AWS. Each file corresponded to one trend category and captured the trends of the previous week. The first trend category, diet canton trend, was the weekly trend evolution of key diets in each Swiss canton. The second one, meat canton trend, was the same but for five types of meat. Then, the diet search trend category contained information about the diet names which were looked up by each speaking region of Switzerland (German, French, Italian) each week. The evolution key diet category tracked trend evolutions of ten key diets across all Switzerland. The fifth category, product search, allowed us to extract related online searches of 29 products (banana, bread, zucchini, etc.). We were indeed interested in getting insights about what people wanted to cook when they searched for each of these products. The last category, recipe search, fetched the weekly searched recipe names in each speaking region of Switzerland.

Then, each week, once the data was fetched and saved as CSV files into the S3 bucket, the data from the most recent CSV file of each of the six categories was written into its corresponding PostgreSQL table in the data lake. Finally, a validation table was created for each of the six tables in the data lake. These tables were only useful to us as they validated that the correct number of data points were written into the tables. An overview of the data lake is in the Appendix A.

It is worth mentioning that we are still extracting data every Sunday and loading them to the data lake.

2.3 Edamam

With the aim of making suggestions to the restaurant owners, various recipes related with 29 search queries (e.g. ‘tofu’) are fetched from the Edamam – Recipe Search API. The recipes that are obtained from the API consists of information regarding the diet labels (e.g. low-carb, high fibre), health labels (e.g. dairy-free, gluten-free), meal type (e.g. lunch, dinner), dish type (e.g. main course, desert), ingredients list as well as the quantity of each ingredient in grams. As there is a 10 calls/min limitation of the API and it is possible to only obtain 20 (or less) recipes with each call, data are being fetched with three Lambda functions every four hours and they are directly transferred to the ‘recipes’ table created with PostgreSQL on our data lake. The entity diagram of the ‘recipes’ table and an example entry can be seen in Appendix B. It should be noted that the information on the ‘price’ column is not extracted from the API, rather calculated afterwards. We will give the details of the calculation in Chapter 3.1.2.

2.4 Coop

The ambition was to implement the web scrapping process on AWS EC2 to make the whole project based on cloud application. But due to the large cost, we decided to do the web scrapping locally on our own devices and upload it directly to S3 & RDS (see Appendix C). For this purpose, there are three separate scripts which will be run individually, as you can see in the Appendix D.

The web scrapping happens every Monday and Thursday – this was set like this because Coop releases new promotions on these days.

First, we have implemented a function, which checks existing rows on the RDS Database and looks if the rows have been changed – if so, then updates these rows. Due to a small mistake in the code, it does not properly recognise rows in which changes has to be made and create new rows instead. That was also the reason, why the data table had a size of 6.3 GB previously.

After some research, we found out, that in Pandas there is already a function implemented, which has a parameter named “if_exists”, that will check automatically for changes in the rows and only changing these. Because of the optimised and low-code format, we decided to use this function instead of ours (see Appendix E).

3 Methodology

To build a meaningful dashboard and perform powerful analyses, it is not possible to directly work with the raw data that is stored in the three data lakes. Indeed, raw data tends to be imperfect, or ambiguous, and thus it needs to be cleaned. Once this step is done, we can build our data warehouse, which contains the processed data from each data source. Finally, we can build our final dashboard, which aims to help the end-users to gain valuable insights on the data, visualize trends, find interesting patterns, and get meaningful recipe recommendations.

3.1 Data Quality

In this section we present the data cleaning and aggregation that we did for the data in each data lake. First, we explored the raw data to measure its quality and identify what needed to be cleaned. Then we processed the raw data in Lambda functions for Google Trends and Edamam on AWS to obtain data of higher quality.

3.1.1 Google Trends

Cleaning the data from Google Trends was rather complex. Luckily the data from three categories, namely *evolution key diet*, *meat canton trend* and *diet canton trend*, were already clean enough to be used in a dashboard. Indeed, their raw structure was very good and always consistent over the weeks. An example of the data from the meat canton trend table can be found in the Appendix F. The three tables contained weekly information about trends either at a canton level, for the meat canton trend and the diet canton trend categories, or at a Swiss level, for the evolution key diet category. We could thus directly upload these raw data into the data warehouse as separate tables.

Next, to be able to clean and aggregate data in the remaining tables more effectively, we created a function to translate all queries to English using the library `deep_translator` and appended a new column (`query_en`) with the translated queries. A code snippet can be found in Appendix G. Then, we also removed the stop words from all translated queries and appended another column (`query_en_no_stopw`) which contained translated queries without stop words to the data frames. Based on our use case and to simplify the aggregation, we additionally had to adjust the list of stop words which needed to be omitted. For example, we wanted to exclude the words “recipe”, “plan”, all the adjectives such as “best”, “healthy”, or “easy” and some proper names, “betty bossi” and “Jamie Oliver”. We also had to remove stop words in German, French and Italian since for a few queries the translated function did not identify the language (for example Swiss German) and thus it did not translate them properly. The code can be found in Appendix H.

Thereafter, the data from *diet search trend* table required to be further cleaned and aggregated. The goal was to display which diet names were searched each week in each speaking region of Switzerland (G, F, I). The first challenge was that people did not always used the same key words to search for the same information. Therefore, Google considered that these searches were distinct. Consequently, their *values* were normalized separately, which was the second challenge. The Figure 1 illustrates these issues (see columns *query* or *query_en*). In row 1, 2 and 4, we can observe that people in German speaking region (region G) looked up for the keto diet, either using the complete or the abbreviated form. Therefore, we used the library `Regex` to identify similar diet names in the column `query_en_no_stopw` and created a new column (*diet_category*) where we replaced values with one unique name for each diet. As a result, the column contained only normalized values. Finally, since we only wanted to have the same diet name only once per week and per region, we aggregated the data frame per week, region and diet name and selected the maximum value in the value column. These steps were done in a function, namely `clean_diet_search_trend`. The first lines of code can be found in the Appendix I. We could then create a

table using the columns `diet_category`, `region`, `date_last_7d`, and `value`, stored in the data warehouse. The first rows of this table can be found in the Appendix J.

Figure 1: Example Data in Diet Canton Trend Table

query	value	region	date_last7d	query_en	query_en_no_stopw	diet_category
keto diät	100	G	2022-03-29	keto diets	keto	keto
ketogene diät	60	G	2022-03-29	ketogenic diet	ketogenic	keto
low carb diät	22	G	2022-03-29	low carb diät	low carb	low_carb
keto diät rezepte	13	G	2022-03-29	keto diet recipes	keto	keto

(Source: Own Figure)

The data in the *product_search* table required a lot of cleaning. First, after having translated all the queries and removed the stop words, we manually translated some of the most frequent Swiss German words to English, such as “schoggi” or “wähe”. Then, we removed the queries which were not related to recipe ingredients. For example, the queries about how to grow zucchini, or the calories content of one of the 29 products, or the recommended quantity of pasta per person were removed from the table. We also had to omit the queries which were related to technology product, for instance “apple watch”, to bars, restaurants, and shops, for example the clothes shop “mango” in Lausanne, “beef steakhouse bern”, or to vehicle, such as the scotter “chilli 5000”. Finally, we tried to normalize the data as much as possible by taking care of the word order and of the singular and plural forms of words. For example, “banana pancake” was not considered the same as “pancake banana”. We thus created an additional column *query_clean* which contained the normalized and cleaned query. Finally, since we only wanted to have the same product search query only once per week and per product category (there are 29 product categories), we aggregated the data frame per week, product and query and selected the maximum value in the value column. These steps were done in the Python function *clean_product_search*. Some lines of code can be found in the Appendix K. We could then create a table containing the columns *query_clean*, *product*, *date_last7d*, and *value*, stored was in the data warehouse. The first rows of this table can be found in the Appendix L.

The last table, namely *recipe_search*, did not require much further data processing after having translated the queries to English and removed the stop words. We created a Python function, *clean_recipe_search*, to remove recipe queries which did not contain meaningful information for restaurants, for example “cooking with kids” or “Easter cake”. It also aggregated the queries per week, region, and recipe name in order to have the same recipe name only once per week and per region. The first rows of this table can be found in the Appendix M.

3.1.2 Edamam

Initial cleaning of the recipe data is adjusting the format of the ingredients and their quantities in the data fetching stage. In the response of the API, there are separate blocks for each of the ingredients in the recipe which consist of information about the name, quantity, measure, and food category of the corresponding ingredient. During fetching, each of the ingredient is coupled with its quantity as a list and the lists for all ingredients are gathered as one list of lists. Example part of the response for an ingredient and resulting list of lists for one recipe is shown in the Appendix N. The reason for doing this adjustment is to ease the price calculations which will be explained in the following stages of this chapter.

To be able to give some insight to the restaurant owner, we calculated prices of recipes using their ingredient lists and prices that we get from the ‘Product’ table corresponding to each ingredient. The steps can be summarized as follows:

- 1) Get the ingredients and their quantities from the ‘recipes’ table.
- 2) For each ingredient, pre-process the name:
 - Delete all ‘in ...’:

Some ingredient names involve 'in ...' in it (e.g. 'Albacore Tuna in Water'). If we search the 'Product' table with this name to find its' price at Coop, there will be no match. Thus, the part starts with 'in' and followed by another word (until the end of the name) is dropped.

- Delete all 'all...':
Likewise, some names involve 'all..' in it (e.g. 'all-purpose flour') which do not have a match as it is. So, the part starts with 'all' and followed by other characters (until the end of the word) is dropped.
 - Convert the name to lower case:
As there is no standard format for creating the ingredients list in Edamam, the names will appear as the creator of the recipe defines it. That means the same thing may be defined in various ways (e.g 'Cherry tomatoes', 'Cherry Tomatoes', 'cherry tomatoes'). To overcome this problem, names are transformed into lower case.
 - Special case of beetroot:
As the vegetable version of beetroot is defined as 'beets' in coop list, searching as 'beetroot' gives inaccurate results such as 'Kambly ApériSticks Beetroot' which has a very high price compared to the vegetable version (vegetable = 4.7 CHF/kilo, kambly = 18.4 CHF/kilo). Then, the total price becomes extremely incorrect. Therefore, all the 'beetroot's are replaced with 'beets'.
 - Apply natural language processing to the pre-processed names:
Extract nouns and proper nouns from the names. For example, for 'Albacore Tuna', search word become just 'tuna' so there would be a match to it.
- 3) Search the pre-processed names of the ingredients from the 'Product' table to get their prices.
 - For each search, there exist multiple results which are related with the name of the ingredient. To select the correct price, we examined the options and we realized that in most cases the shortest title and the lowest price corresponds to the basic ingredient. So, we sorted the results according to these two criteria and got the first option. An example output for search word 'apples' can be found in the Appendix O.
 - For some items like kiwi, lemon or avocado, the quantity in the 'Product' table is 'PCE' (piece) rather than grams. So, we created a list of most important items that has the quantity of pieces and we refine the query that is used to get prices by additionally defining quantity of the item (Appendix P).
 - 4) As the amounts and prices negligibly small for oil, salt, pepper and sugar they are excluded from the calculations to increase the performance.
 - 5) Total price is calculated by using the quantity of the ingredient within the recipe and the price for 100 grams which is obtained from the 'Product' table.

3.1.3 Coop

The reason for fetching product information from Coop was to enrich our recipe suggestions with the cost so that the stakeholder could have better insights of the profit and optimize its budget. As the product comes in different packaging and different units (fruits in gram or kilo price, liquid product in litre or millilitre) we have to unify them so that we can compute the estimated price per recipe more realistically. Instead of creating for each unit transformation a separate function, we implemented a generic function which we can use to transform any unit to any other unit.

As you can see in the Appendix Q, the dataframe is the parameter, the unit which you want to change, "new_unit" as the targeted unit and the factor to multiply with.

In the Appendix R, you can see an example of how we changed all the different unit into the targeted unit. For instance, all liquid unit has been changed to millilitre. With these units transformations, we were able to precisely compute the price per recipe.

The next challenge was, that in many recipes the number of a certain product item was not given in grams (e.g., 1 lemon instead of 40g of lemon). To cope this issue, we decided to compute the price per 100 units.

In the Appendix S, you can see we have implemented a generic function which calculate the price per unit for the whole data frame with the given unit using regex. The reason why we must additionally check if there is an “x” in the row[‘quantity’] is that in several products the price was given for multipack (e.g., 6x100g strawberry yoghurt).

3.2 Production Data Pipelines

Once the data quality was enhanced, we could write the clean data into the data warehouse. In the section below, we describe the data pipelines of each data source.

3.2.1 Google Trends

The raw data from Google Trends was stored as CSV files in the S3 bucket in the data lake. To clean the data as described in Chapter 3.1.1, we created an additional Lambda function, *dw_clean_insert_data*, which accessed the most recent CSV files in the S3 bucket of each category (evolution key diet, meat canton trend, diet canton trend, diet search trend, product search, recipe search). Using the function BytesIO, we could directly read the data from the CSV files as Pandas data frames, without having to download them locally. Next, the Lambda function performed the steps mentioned in Chapter 3.1.1 and created six data tables, one for each category. Finally, the clean and aggregated data was written into their respective data table in the data warehouse, namely *evolution_key_diet*, *meat_canton_trend*, *diet_canton_trend*, *diet_search_trend*, *product_search_trend*, and *recipe_search_trend*. The corresponding Python code can be found in the Appendix T. As already mentioned, the data from the first three categories (evolution key diet, meat canton trend, diet canton trend) are directly written into their data tables in the data warehouse. To hide important information in the Python script, we encrypted the S3 bucket name, host, data warehouse name, username and the password in the Lambda function.

This data pipeline is automated thanks to EventBridge and runs every Sunday at 18h30 (16h30 UTC), namely one hour after the data was extracted from Google Trends. This one-hour buffer time allows us to solve any issues if the data extraction faces a problem.

3.2.2 Edamam

We fetch the recipe data from the Edamam by using the Recipe Search API. Cleaning and modification of the data are mostly carried out by the Lambda function that is created for this data source. After the initial modifications, the data is directly loaded to our ‘recipes’ table on our data lake. As the data source is not dynamic, extracted (and formatted) information is directly transferred to the PostgreSQL database, which is created on AWS RDS service, through Lambda function rather than using S3 Buckets.

The only exception to the modifications carried out on Lambda function is the price calculation that is explained on Chapter 3.1.2. As there are more than 40,000 recipes, it is taking quite long (approximately one day) to calculate prices for all recipes. Due to this fact, Lambda function is not suitable to carry out these computations as Lambda functions are more appropriate to simple and short time runs. Thus, Jupyter notebook was chosen for the price calculation process (Price_Calculation.ipynb). After calculations are done, already existing ‘recipe’ table is modified by adding a ‘price’ column. That means no new table is created in the data warehouse that we are building but rather we modify the existing data lake.

3.2.3 Coop

We fetch our data from coop by using web scrapping. Therefore, we are running three different scripts separately, on our local computer.

Table 1: Data Pipeline for Coop Data Source

Extract	Coop_scrapper.py This script extracts products from the Coop webstore using selenium library. The selected data will be saved in separate CSV for each product category and one master CSV containing all the products.
Transform	Coop_scrapper.py As soon as the data is downloaded, the functions “unifying_unit” and “price_per_unit” will be performed to transform the extracted data into the form which can be used later for further computation in the data warehouse.
Load	Upload_rds-postgres.py & upload_s3_bucket.py These scrips upload the transformed data to RDS and S3 bucket

(Source: Own Figure)

As these scripts will be run locally on our own computer, you can find all these script in our GitHub repository.

3.3 Data Warehouse architecture, System etc. in technical details

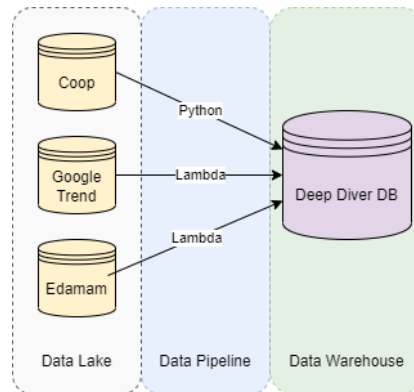
In the previous module "Data Warehouse and Data Lake Systems 1" our project goal was to create data lake for each data source. These was done by creating different RDS PostgreSQL instances, each saving one data source. The next step will be to clean up the raw data to use in our visualization tool.

First, we wanted to implement an AWS Redshift instance and connect all three data lake into it. Nevertheless, we did not use AWS Redshift since it was much more expensive than an AWS RDS. It also required much time to be setup and it is more suitable when there are more data lakes to manage. So, in our case, working with three RDS databases, one additional RDS Database storing the cleaned data would be sufficient.

3.3.1 Database architecture

The following figure describe our database architecture. Once the data was cleaned, after having been through the data pipelines, it was uploaded into a separate RDS Database (PostgreSQL), our data warehouse.

Figure 2: Database architecture



(Source: Own Figure)

3.3.2 Tools used

The tools we used to implement the data lake & data warehouse are listed below:

Amazon RDS (PostgreSQL)

We used the AWS RDS service to create database instances for each data source and to create the data warehouse storing the clean combined data. As database management system, we selected PostgreSQL.

DBeaver

To simply access all database instances from all team member accounts, we used the DBeaver software. We were able to quickly gain insights into the data. It also gave us the options to add new roles and assign them to specific tables.

PyCharm Professional

Using this tool, we were able to implement all the scripts for the web scrapping and for its cleaning process. This tool provides many features such as debug-modus, formatter, code-suggestions, etc. Also we would use the in-build git-client to push the changes directly into the repository.

AWS Lambda function

This AWS service helped us to fetch, transform and load from the data sources Edamam and Google Trends automatically, using scheduler methods. We used Python as our programming language.

S3 bucket

When retrieving data, storing them is very crucial. Especially when the data is continuously generated as it is with Google Trends. The main benefits of AWS S3 is that it is available all the time, low cost, accessible anytime and simple to manage. Therefore, we use S3 Buckets as target storage for our extracted data but also as source for importing data into the database.

Jupyter Notebook

To implement Python code and directly see the output, Jupyter notebook is better. Several scripts were implemented with Jupyter Notebook as you can see in the Git repository.

3.3.3 New User Role

To have access to all data lakes and data warehouse anytime, we set up a user role named "deep_diver" in each RDS Database instances. Thus, we are able to access all data tables even when the database is created by other AWS accounts. To prevent any damage with this user, we assign all permission except "super-user". We could then ensure that the data cannot be deleted with the user role "deep_diver" (see Appendix U).

Additionally, we created one additional user role with very limited permission for the visualisation tool Tableau. With that we can precisely set the permission for accessing data from the frontend side (see Appendix V).

3.3.4 ER diagram

Combining the fetched data in a way to use them for our visualization need to be very precisely made. Based on the way of joining of data tables, the information we create can be differ. In our case, we would like to join the data tables in that way, so that we can fulfill our project goals and answer the research questions. You can find the database schema in the Appendix W.

Additionally, we created additional tables which represents combined information of certain tables. These will allow us to precisely control the quality of data which is used in our dashboard visualisation.

The following tables were additionally created with a Lambda function:

- Top_product_trend
According to the product trend from the table "product_search_trend", we looked up for recipes matching the product trend.
- Top_recipe_trend
According to the recipe trend from the table "recipe_search_trend", we looked up the most matching recipes from our "recipe" table

3.4 Visualization

In this section, we first describe some visualization concepts which justify the design choices of our dashboard. Then we present and describe our final dashboard.

We decided to use Tableau Desktop, which is a software provided by Tableau. It allows to visualize and analyze data, which can might be stored on-prem or in the cloud [Tableau 2022]. Depending on the data type, the software recommends basic graphs to quickly start visualizing data. One can also build powerful graphs to fit specific needs and make new calculated fields to create additional variables from the data.

3.4.1 Visualization Design Justification

Data visualization has many advantages. First, it helps to understand the data quicker. It is particularly useful when the dataset is large. By plotting data, one can faster discover patterns and gain insights. It is indeed simpler to compare trends between cities using a line or a bar plot than with raw numbers. Second, information is usually retained more easily when it is visually displayed than when it is described with words.

The Visual Information Seeking Mantra (VISM) principles [Shneiderman, 1996], namely “overview first, zoom and filter, then details on demand”, highlight the basic features that data visualizations should have. The visualization should first provide an overview over all the data, to gain a global understanding.

Then, depending on the user's interests, they should be able to get further details on specific items in the data visualization and filter out what is less meaningful to them. Finally, whenever they need, information about specific details should be quickly available. These basic features are available in our dashboard thanks to our use of filters.

Regarding the specific design of the data visualization, the types of graphs and colors are important.

First, the choice of graphs depends on the data that we want to present. In our dashboard, we used the bar chart, table graph, and the line plot. The bar chart is easy to read and is often used to plot numeric values and categorical variables [Yi n.d.]. It is usually recommended to use bar graphs when we want to compare subgroups in the data. On the other hand, tables organize data into rows and columns. They are basic and are usually chosen to simply display individual values. Line graphs are usually used to show the changes over time [Freeman 2022]. It helps to see trends and possible seasonality effects in the data.

Additionally, colors have an impact on how the information is perceived and interpreted. They are indeed closely related to emotions. It is therefore essential to choose colors wisely. Each color has a meaning. For our dashboard, we used three colors: green shades, brown shades, and white. This choice was based on the message that we wanted to transmit and the emotions that we wanted to create. Green is often associated with health, balance, and harmony [Color Meanings, n.d.]. Brown represents the reliability, honesty, and the soil. Lastly, white is usually synonym of balance, clarity, and simplicity.

Finally, it was essential that we avoided cognitive overload and thus displayed the right quantity of information in the dashboard. The first objective of a data visualization is indeed to simplify a large amount of data into clear graphs. Therefore, we focused on creating a dashboard that was quickly comprehensible and which captured the attention of the user. To do so, we simplified the graphs as much as possible, by removing, for example, the grid lines for the bar plots, the table, and the lists. Additionally, we removed the headers and the Y-axis of the bar plots and directly displayed their values on top of the bars. The graphs were thus faster to understand. Furthermore, for the table and one bar plot, we chose to display only the top three and top five observations respectively. The same logic applied to the list of recipes. We selected to show ten recipes in each list. The restaurant owners would thus receive a large but limited list of recommendations from which they could choose.

3.4.2 Tableau Visualization (with public link)

For this project, we wanted to provide our end-users, the pop-up restaurant owners, with a Tableau dashboard which would allow them to take better and data-driven decisions. Therefore, it was essential that we created a dashboard which was powerful but also easy to understand. Additionally, it should enable us to answer our research questions.

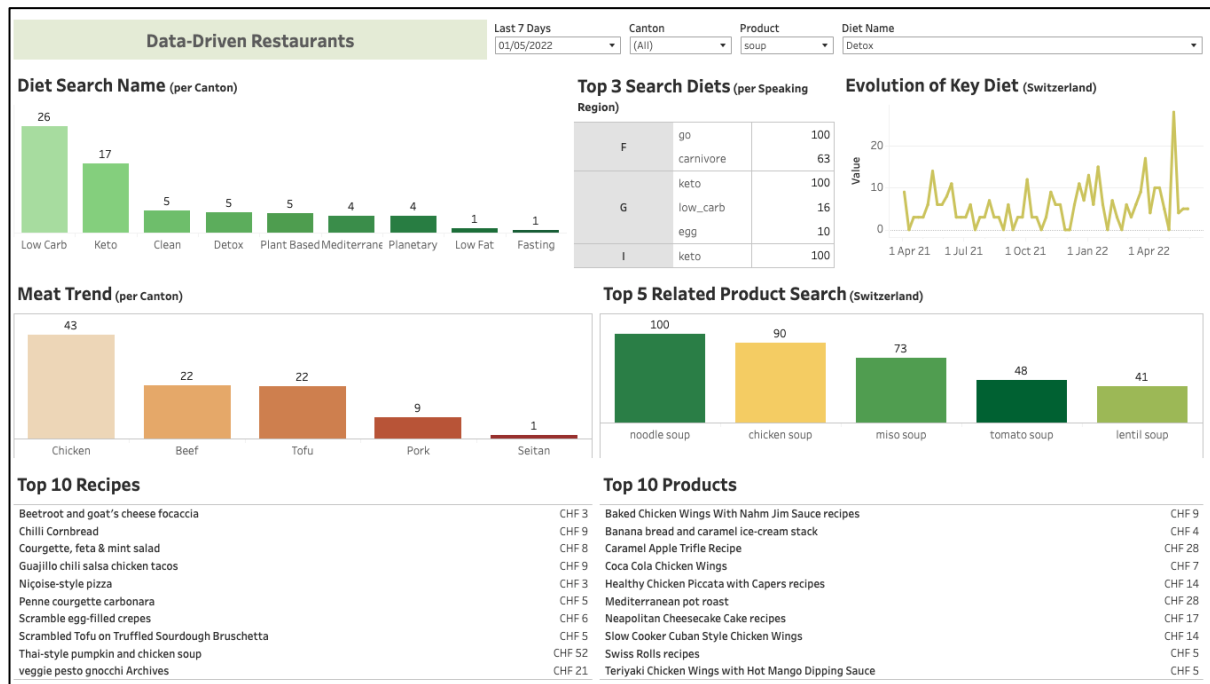
Before starting to build our dashboard, we brainstormed to decide what we wanted to show. Our data was rich in information so there were plenty of ways to display it. We tried different solutions and we ended up with the final draft, which can be found in the Appendix X. We used it as template to build our Tableau Dashboard.

Our solution is available following this link:

https://public.tableau.com/shared/TS3B94HSP?:display_count=n&:origin=viz_share_link

It is also presented below. It is organized as follow: on top, there are the different filters which can be adjusted to the end-user's needs (VISM: zoom, filter, details on-demand). In the middle part, there are five plots which display the observed trends corresponding to the situation created by the filters. At the bottom, the user finds a recommendation of the best ten recipes, based on the recipe trends data. They can also benefit from other recommendations of recipes based on the product trend data. The added value of these two lists is that they also provide the cost of each recipe.

Figure 3: Final Tableau Dashboard



(Source: Own Figure)

First, by setting the *Canton* filter to *All*, the restaurant owner has an overview of the diet and meat trends (plots: *Diet Search Name*, *Meat Trend*) for the chosen week (filter: *Last 7 Days*). If they are interested in a specific canton, they can select it in the filter *Canton*, see Appendix Y.

Then, they can also change the date (filter: *Last 7 Days*), to visualize the trends for another week. The filter will change the four following graphs: *Diet Search Name*, *Meat Trend*, *Top 5 Related Product Search*, and *Top 3 Search Diets*. An example can be found in Appendix Z.

Moreover, to have further information about the related queries of another product, the restaurant owner can change the *Product* filter. An example can be found in the Appendix AA.

The filter *Diet Name* changes the graph *Evolution of Key Diet*. An example can be found in Appendix BB. The end-user can then see the trend evolution of a specific diet and gain insights on its popularity in Switzerland. They can hover the mouse over the line plot to see the specific date and the corresponding value. It might thus help them to decide whether they should offer meals of a certain diet and when it is the best moment to add these meals on their menu.

Finally, thanks to the two lists in the bottom part (*Top 10 Recipes*, *Top 10 Products*), the restaurant owners can benefit from a suggestion of 20 recipes, based the trendy online searches for recipes and products. The end-users can also have an estimation of the recipe price, which will help them to optimize their budget and already estimate the price for which they will offer the meals on their menu.

3.5 Answer to research/business questions

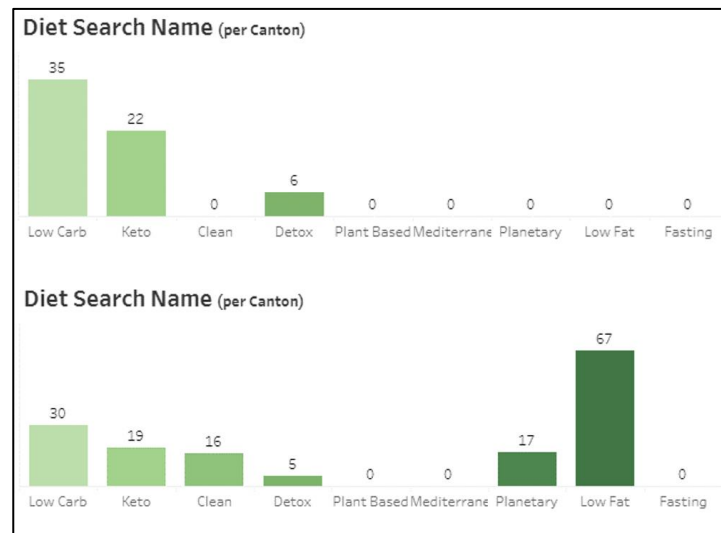
In this chapter, we answer our research questions.

1. Are there distinctive food preferences between different regions of Switzerland (e.g. among the Swiss cantons or the language speaking regions)?

With the information fetched from Google Trends, we were not only able to get the distinctive food preferences in Switzerland, but we were also able to distinct them between different regions.

One part of the dashboard shows the bar chart of the diet preference trend based on the canton. As you can see in the Figure 4, you can see the different diet trend in canton Lucerne and Zurich. Clearly the most trending diet in Lucerne is the "Low Carb" diet whereas in Zurich the most searched diet is the "Low Fat" diet.

Figure 4: Diet preference for Canton Lucerne (top) and Canton Zurich (bottom)

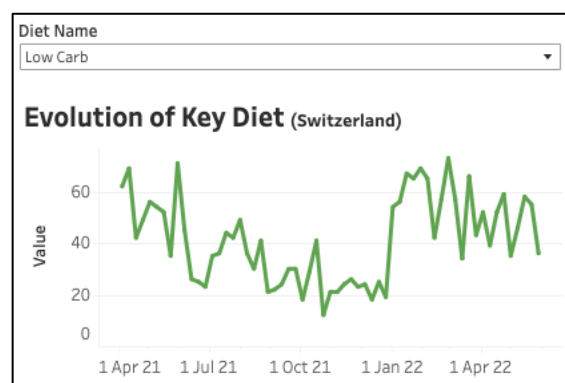


(Source: Own Figure)

2. *Given a specific food-related search term, are there any changes in the population's food preference over time?*

The goal here was to visualize the historical changes of certain trends in Switzerland. To show that, we took the diet trends for each data-fetch runtime and plot it as a line plot.

Figure 5: Evolution of Diet Trend historically



(Source: Own Figure)

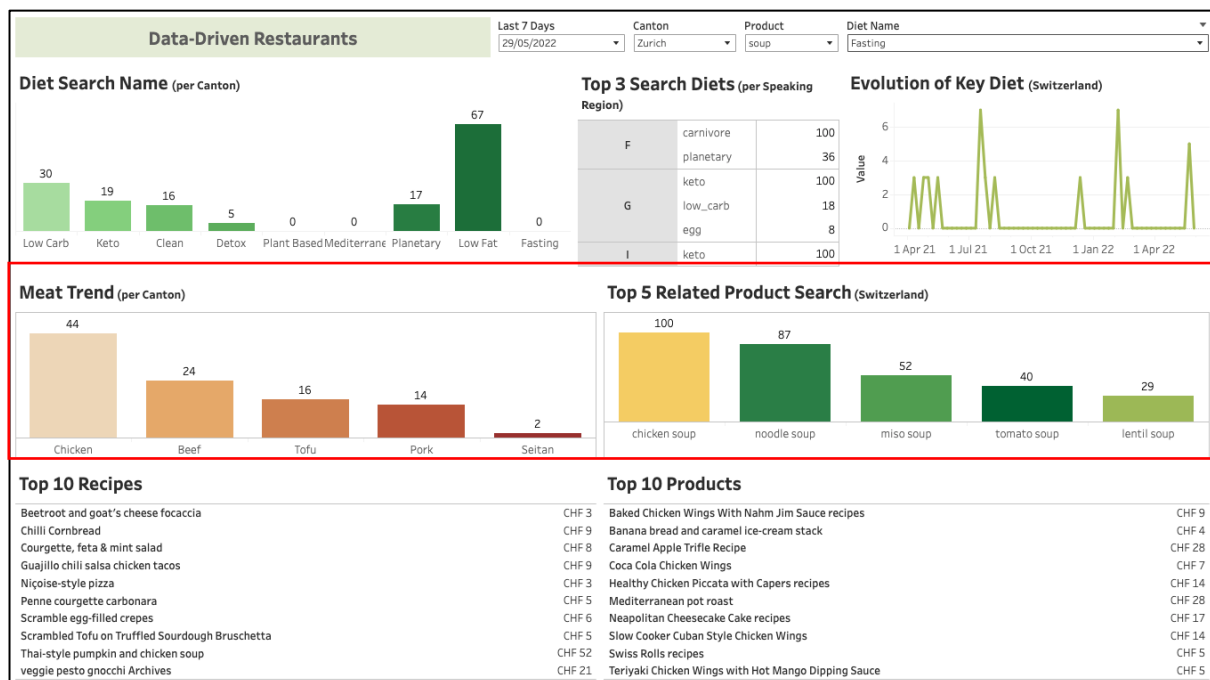
With a dropdown menu, one can select a certain diet trend see how the trends has been changing over time.

In the Figure 5, you can spot that the "Low carb" diet is a diet trend, which has seasonality during springtime. Which would indicate that most people are trying to lose their "Christmas weight-gain" by looking for low carb diets. This can be a very good insight that stakeholder can use to make restaurants more attractive to the customer during certain period.

3. *For each of the meal category (vegan, vegetarian, etc.), which food items are the most often used in these recipes?*

One other insights which can be crucial to meet the needs of restaurant customer are the food preferences. This will also help the restaurant owner to know the demand of each product.

Figure 6: Screenshot of Trends from 29th May 2022 for the Canton Zurich



(Source: Own Figure)

As you can see in Figure 6, majority of people living in Zurich seem to like chicken, followed by beef. With these insights, we can help to precisely select recipes which meets the needs and make the restaurant more attractive. So based to this Dashboard, recipes containing chicken and are made for a "low fat" diet will be the best suitable recipes.

4. *Are there any distinctive patterns among the discounted items sold by Coop?*

At the end of our project, we decided to remove this question from our research topics and give more attention to the price computations. During the run of the data warehouse part of our project, we realized that calculating and suggesting the prices as accurately as possible will benefit the restaurant owners to a greater extent compared to observing the discount pattern of each ingredient.

4 Discussion

In this section, we will discuss the advantages and disadvantages of our solution. We first go through the solution implemented for each data lake and finally we evaluate our dashboard.

4.1 Google Trends

Processing the data from Google Trends was rather challenging but we are satisfied with the final outcome. First, we had to translate all the queries to English using the Python library *Google Translator*. Translating helped us to have less variation in the data. Nevertheless, Swiss German language was not always detected by the program. It is indeed a challenging language to translate due to its numerous dialects. Swiss-German is a well-known challenge in NLP.

Furthermore, although we only have basic skills in NLP, we were able to clean the raw data as much as possible. We used the Python library *nltk* to identify and remove stop words from the data. We also adjusted the list of stop words to our use case which allowed us to have even cleaner data. Next, we could identify and remove noisy data from our dataset, such as “apple watch” or “mango shop”, but there are still noisy observations which might not have been detected. Indeed, our solution was to use the Lambda function to manually remove the data which was not related to food. There might be more effective solutions. One additional challenge was the word order. Google Trends does not treat “chocolate cake” as being the same as “cake chocolate”. Same problem when queries contained typing mistakes, such as “tatar” instead of “tartar”. We were able to clean, reorder, and normalize the observations that we detected but our method can be improved.

Moreover, once the data was processed, we decided to aggregate it to have the same value appearing only once per aggregation basis. We chose to use the maximum since the values were normalized by Google Trends to express a form of ranking. It is worth noting that Google Trends does not provide documentation about how these values are normalized, nor on which basis. It might have helped us to better aggregate our data. As a consequence, we are not very confident about how to interpret these numbers. Nevertheless, we can already affirm that a searched query with the value 100 was more looked up than a query having a value of 20. Last, there are still many observations which have a value of 0, which does not look good in our visualization. Google Trends explains that such value is allocated when there is not enough data to normalize it.

Finally, unlike the first part of the project for which we built the data lakes, we decided to create a unique Lambda function to extract the data from the S3 bucket, clean it and write the processed data into the data warehouse. It allowed us to do everything at once and simplify the work. We ensured that the Lambda function could process the data in less than 15 minutes. This is indeed the maximum time that a Lambda function can run. Nevertheless, there are two disadvantages of this solution. First, it is risky to do everything at once. If there is one problem in the function, it stops, and all the data processing needs to be done again. Second, we chose not to save the cleaned data into the S3 bucket. Therefore, we have no weekly backup. In future work, we will adjust the lambda function. This improvement will be discussed in chapter 6.1.

4.2 Edamam

The calculated prices showed that there are some discrepancies over the values. There are two main types of the observed cases. First, there are some recipes with extremely high price values (Figure 7). When we made a check we found that the problem is at the response obtained from the API. As you can see in Figure 8 below, actual ingredient is the banana from chillies but mistakenly they defined the name of the ingredient as ‘chillies’ and the quantity as 3 kilograms. When the search is made with the keyword ‘chillies’ the price from the ‘Product’ table comes as ‘Fine Food Ground Bird's Eye Chillies’ which has 16 CHF/100 gr. Thus, the end price for 3 kilos is critically over calculated. We have checked some part of the resting overpriced recipes and same reason is valid for each. Details for the price calculations for the most expensive recipe (Banana-chilli pickle) can be found in Appendix CC.

Figure 7: First Five Recipes with Highest Prices

label	Prices
Banana-chilli pickle	547.88
Radish Canapés	450.45
Kung Pao Beef recipes	436.16
Sesame Orangeyaki Stirfry with Tofu recipes	382.11
THAI STYLE GREEN CURRY PASTE	324.44

(Source: Own Figure)

Figure 8: Example Output of the API for an Ingredient

```
"text": "3 kilogram banana chillies",  
"quantity": 3.0,  
"measure": "kilogram",  
"food": "chillies",  
"weight": 3000.0,  
"foodCategory": "vegetables",
```

(Source: Own Figure)

Second, there are 62 recipes with 0 price value. When we check them, we found that it is resulting from the pre-processing of the ingredient names. For example, in the recipe of ‘Cheesy Chicken Italiano’ there is one ingredient named as ‘Rotini Pasta’ and after the pre-processing, it is still ‘rotini pasta’ which will return nothing as a price. Details for the price calculations for a zero-price recipe (Cheesy Chicken Italiano) can be found in Appendix DD.

All these problems and the ones that we try to resolve by pre-processing (Chapter 3.1.2) are the ones that we can detect. To detect each fault, every recipe must be checked one-by-one which will not make sense. Although it needs improvement, in general the price calculation process is advantageous to use as it gives a general idea about the cost of the recipe to the restaurant owner which was the main aim at the beginning of the project. Moreover, it is always possible to define further settings to improve the accuracy of the calculations.

4.3 Coop Website

When we did the web scrapping, we are not sure what kind of product we were going to use for our project. Therefore, we scrapped all the products of coop. This led to many data points which were not used in our project. Plenty of data storage were used but never used for the actual work. This could have been improved by selecting only subcategories of the coop products. Additionally, some of the products are the same but in different packaging. These could also be reduced by only selecting one packaging size per product.

So, for instance if we have *tomatoes* as a specific food item which has been listed in a recipe, we found more than one entry for tomatoes in the coop database. To come up with a generic solution, we just compute the average price of all tomato’s product. This leads to an over- or underestimated cost for the recipe. But nevertheless, it was better to use this methodology than ignoring and only selecting randomly one of the specific food items.

Even though we scrapped the English version of the coop web shop, many products contain the German naming. This leads to, that these products will not be considered in the recipe cost computation as these will only work with product items in English language. We could have checked each product by using python language for translation from German to English.

4.4 Tableau Dashboard

We are satisfied of our final dashboard. It is easy to understand, the colors are harmonious, and the amount of information is balanced. It is worth mentioning that we decided to adjust the dashboard after the final presentation, on the 19th of Mai, see Appendix EE. We removed the list of recipes on the right and added the line plot. Indeed, we found confusing to have too many lists of recipes. We also thought that it was necessary to plot the trend evolution of key diets in Switzerland. The result is finally airier.

The dashboard has also a clear structure and is logical. The title and the filters are on the top, the graphs in the middle and the recommendations at the bottom.

Last, there are a few points that could be improved. For example, we think that it is not clear which graph is connected to which filters. Second, it is not possible to select many dates at once, thus the meat trends cannot be compared over time. The improvements will be discussed in chapter 6.5.

5 Conclusions

Nowadays, we are moving away from collecting and hording Big Data towards processing it in real-time. Thus, having massive amounts of data will not have matter, if profitable deductions cannot be made from it. To gain useful insights from the data, various sources must be processed, combined and represented in a user-friendly way. However, even these are not enough by themselves as the processes must be fast, data must be up-to-date, and deductions must be accurate.

Any business that is supported by a database with the aforementioned qualifications will find itself ahead of its competitors. Ideally, such databases can be a part of any sector, however, certain industries will benefit more by their utilizations. Since the culinary industry is a timeless business sector, it would perfectly benefit from an efficient database management. In light of these intuitions, we aimed to create a recommendation tool based on actual data, which will offer trendy menu options for Pop-Up restaurant owners to enhance their business. With our prototype, a restaurant owner can see the trendy ingredients and diets, changes in these trends over time, various new menu options related with these items and the overall costs. Consequently, they will have all the information needed to go through a menu improvement. While the tool has certain limitations, we believe that we have achieved the initial goal we set out to do. Further developments can focus on improving the issues related with data collections, language processing, and price calculations.

As of 08th June 2022, the credit conditions on AWS accounts can be seen for each data source in Table 2.

Table 2: AWS Credit Conditions

Source	AWS Credit Condition
Google Trends	Used \$52.1 of \$100
Edamam	Used \$38 of \$100
Coop	Used \$97.80 of \$100
Total	Used \$187.9 of \$300

(Source: Own Figure)

6 Future Work

In this section, we discuss the areas to improve and provide suggestions for future work.

6.1 Google Trends

As mentioned in chapter 4.1, we created only one Lambda function to extract raw data from the S3 bucket, clean it, and upload it into the data warehouse. Doing everything at once is risky because if a problem occurs in the function, the whole data processing is blocked and needs to be done again. Therefore, in the future, we will split the Lambda function into two functions, one to clean the data and one to write the values into the data tables stored in the data warehouse. Additionally, we will save the clean data into a new folder in the S3 bucket to have a weekly backup of our data. As a result, if any problems occur, we will be able to react and recover our work quicker.

Furthermore, regarding the data cleaning step, we will improve the methods to process raw data. In the future, we will be more familiar with powerful NLP techniques which will allow us to better normalize textual data.

Finally, we will request additional information to better understand how Google Trends normalizes its values. We will thus be able to provide more suitable aggregated values in our dashboard, which will be more comprehensible.

6.2 Edamam

Similar to Google Trends, there are two planned future steps. First one is splitting up the Lambda functions to be able to both track the source more easily if a problem occurs on the run of the code and shorten the resolution process by only repeating the steps from where the code stops. Second one is to use S3 buckets for the backups. As Edamam is not a dynamic source and no trend will be analysed over recipes, we did not use S3 buckets. But for a future work, it will be advantageous to integrate this service as S3 buckets will also serve as a general backup for a serious scenario such as a corruption in the database.

Moreover, there is one more major improvement for the recipes which is the improvement of the price calculations. More methods will be integrated to enhance the text processing and price matching for each ingredient.

6.3 Coop

A better filtering or classification of the product will improve the performance and reduce the data volume. As we have been scrapping the data frequently, we might have also consider showing the price changes per product. This might also helped the stakeholder to know the time to purchase the products.

6.4 Data warehouse

Of course, adding further data tables which enhance the information such as the product's price changes, other food-related trends or involving other social platform such as TikTok (food-channels) will result in better analyses of the needs of restaurant customers. Furthermore, initializing a dynamically changeable data warehouse such as AWS Redshift, will help to change data sources without too much additional workload.

6.5 Visualization (Tableau)

In chapter 4.4, we discussed about some improvements to be done on our Tableau dashboard.

First, we would like to improve the speed of our dashboard. It takes sometimes too long to be loaded.

Next, we suggested that we should improve the filters. Indeed, it is not immediately clear which filters are connected to which graph(s). We could for example briefly (a fraction of a second) highlight the graphs when they are impacted by a filter and change. Another option would be to use graphs as filters, which we decided not to implement for this part of the project. For instance, it would be possible to click on one diet

category, and see its trend evolution in the line plot. Additionally, if the diet name was in the top 3 diets of the given week (i.e. values in the *table plot*), it could be highlighted.

Furthermore, we will implement a solution which will allow our end-users to select multiple dates and display the trends on the *Diet Search Name* and *Meat Trend* bar plots. In the line plot, they will be able to select up to three diet names to compare their trends. These changes could help our end-user to visualize trends for specific diet or meat categories.

Moreover, the list of recommendations, at the bottom of the dashboard, could be improved. For example, the restaurant owners could select and filter for a specific product, and corresponding recipes would be directly suggested. For now, the recommendations include all trendy recipe and product trends queries and randomly select 20 recipes. Furthermore, we would like to add more details about the suggested recipes. When the end-users hover over a recommended recipe, they would see in the tooltip the ingredients of the recipe. Finally, they could click on the recipes, which would direct them to the Coop website. There, they will be able to adjust the order and save the products in the basket before buying them.

7 Final thoughts

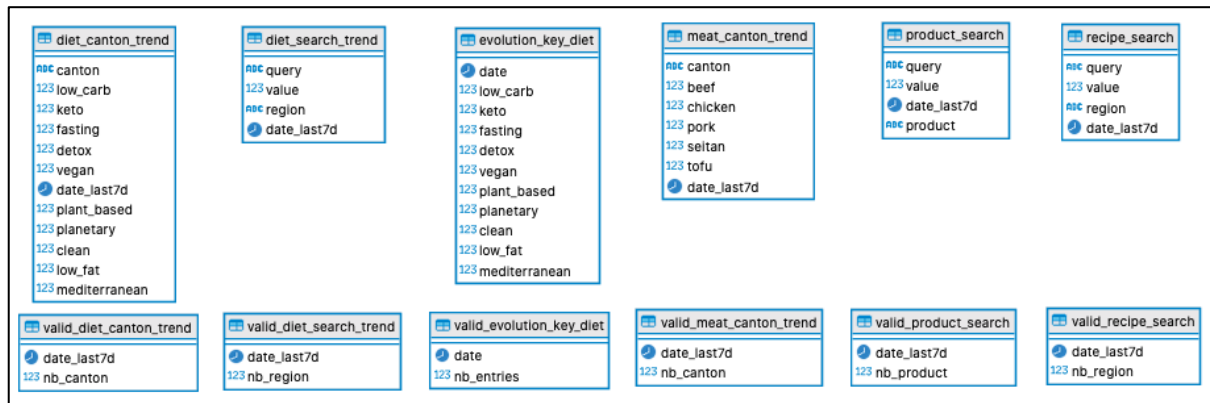
Data is the new gold – and this is what we have learned in this semester. Never have we thought that we could change traditional industries such as restaurants into a data-driven businesses. By using data, we are filling the gap of customer needs by analysing true human behaviour on the Internet. It also shows how much people nowadays rely on the Internet and by doing so how much information they share.

We enjoyed working on this project. We were impressed to see its evolution, from extracting raw data, building data lakes, cleaning data, building a data warehouse, until the creation of a Tableau dashboard. The team was very motivated and enthusiastic to build the end-product. We learned how to fetch data using an API and a wrapper and by scraping it. We could explore various services on AWS, such as the Cloud9, S3 bucket, Lambda functions and EventBridge. We deepened our skills on building and maintaining PostgreSQL databases and on creating a powerful data visualization.

This data-driven restaurant project has a lot of potential. We could contact Coop or other retailers and the pop-up restaurant community and offer them a partnership. Who knows where it will bring us...

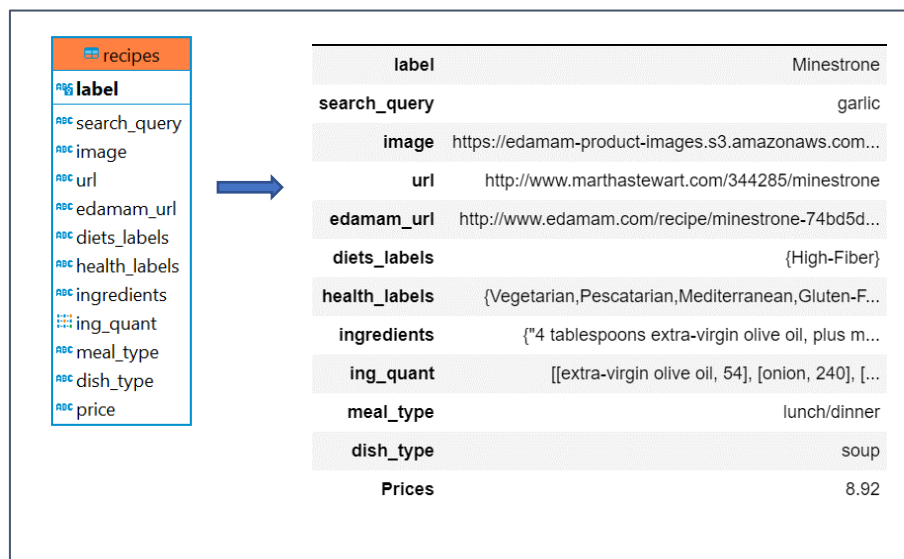
8 Appendix

Appendix A Data Lake for Google Trends Data Source



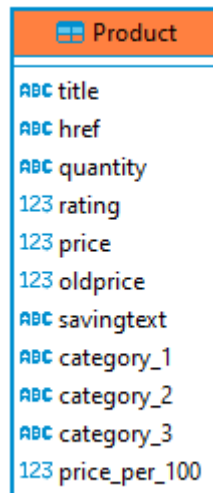
(Source: Own Figure)

Appendix B Data Lake for Edamam Data Source



(Source: Own Figure)

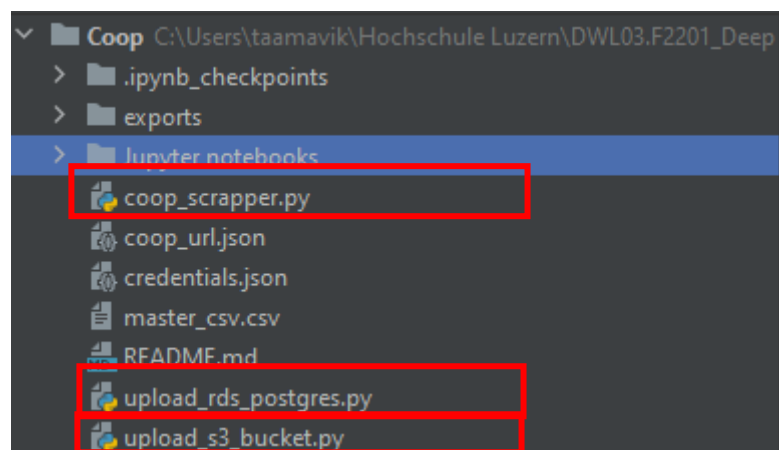
Appendix C Data Lake of Coop



Product	
ABC	title
ABC	href
ABC	quantity
123	rating
123	price
123	oldprice
ABC	savingtext
ABC	category_1
ABC	category_2
ABC	category_3
123	price_per_100

(Source: Own Figure)

Appendix D File Repository Coop



(Source: Own Figure)

Appendix E Code Snippet: upload data to RDS

```
def upload_dataframe_to_table(table_name: str, df: pd.DataFrame, username: str = "deep_diver",
                             password: str = "Amazon2022"):
    engine = create_engine(
        f'postgresql://{username}:{password}@deepdiver.ca7k4yfxv4gc.us-east-1.rds.amazonaws.com:5432/deep_diver_db')
    with engine.connect() as conn:

        # df_before = df.shape
        table_exists = False
        # Check if table already exists
        ins = inspect(engine)
        for _t in ins.get_table_names():
            if table_name in _t:
                table_exists = True
        if table_exists:
            sql_query = f"alter table public.\"Product\" alter column \"savingText\" type text;"
            engine.execute(sql_query)
            df.to_sql(name=f'{table_name}', con=engine, if_exists='replace', index=False)
        # # read table
        # sql_query = f"SELECT * FROM public.\"{table_name}\""
        # sql_df = pd.read_sql(sql=sql_query, con=conn)
        # df = pd.concat((df, sql_df), ignore_index=True).drop_duplicates(keep=False)
        # df.drop(['index'], axis=1, inplace=True)
        # df_after = df.shape
        # removed_rows = df_before[0]-df_after[0]
        # print(f"Total {removed_rows} rows has been removed because they exists already in DB")
        else:
            df.to_sql(name=f'{table_name}', con=engine, if_exists='replace', index=False)

    logging.info(f"Connetion Status: {bool(conn)}")
```

(Source: upload_rds_postgres.py)

Appendix F Data in Meat Canton Trend Table

canton	beef	chicken	pork	seitan	tofu	date_last7d
Aargau	21	42	15	2	20	2022-05-29
Zurich	24	44	14	2	16	2022-05-29
Vaud	23	40	13	3	21	2022-05-29
Valais	20	42	14	6	18	2022-05-29
Ticino	35	27	13	5	20	2022-05-29
Thurgau	27	39	14	3	17	2022-05-29
St. Gallen	18	39	19	3	21	2022-05-29
Nidwalden	23	62	0	0	15	2022-05-29
Lucerne	19	43	18	2	18	2022-05-29
Grisons	33	29	14	2	22	2022-05-29
Geneva	19	48	14	3	16	2022-05-29
Canton of Zug	18	63	11	0	8	2022-05-29
Canton of Uri	0	0	0	0	0	2022-05-29
Canton of Solothurn	25	37	18	3	17	2022-05-29
Canton of Schwyz	24	42	19	0	15	2022-05-29
Canton of Schaffhausen	24	48	12	0	16	2022-05-29
Canton of Obwalden	0	0	0	0	0	2022-05-29
Canton of Neuchâtel	16	46	12	0	26	2022-05-29
Canton of Jura	28	48	0	0	24	2022-05-29
Canton of Glarus	33	67	0	0	0	2022-05-29
Canton of Fribourg	20	30	18	8	24	2022-05-29
Canton of Bern	24	35	16	3	22	2022-05-29
Basel-Stadt	21	51	13	2	13	2022-05-29
Basel-Landschaft	18	42	17	2	21	2022-05-29
Appenzell Outer Rhodes	11	48	0	0	41	2022-05-29
Appenzell Innerrhoden	0	0	0	0	0	2022-05-29

(Source: Own Figure)

Appendix G Python Function to Translate Queries to English

```
# Translate queries to English
def translate_en(df: DataFrame):
    """
    To translate all queries to English.
    It detects automatically the language (source='auto') and translate it to English.
    :return: The dataframe with an additional column (query_en) which contains the translated query.
    """
    # create a new column for the translation
    df['query_en'] = ''

    for index, row in df.iterrows():
        to_translate = row['query']
        translated = GoogleTranslator(source='auto', target='en').translate(to_translate)
        # create a column with translation to english
        df['query_en'].iloc[index] = translated
    return df
```

(Source: dw_clean_insert_data.py)

Appendix H Python Function to Remove Stop Words

```
# Remove stopwords
def remove_stopw(df: DataFrame):
    """
    To remove all stop words from the translated queries.
    The list of stop words is adjusted to the use case. It also omits adjectives and some proper names.
    :return: The dataframe with an additional column (query_en_no_stopw)
    which contains the translated query without stop words.
    """

    # specify language for stopwords
    all_stopwords = stopwords.words('english')

    # adjust the list of stopwords according to our topic
    all_stopwords += ['diet', 'diets', 'diät', 'dieta',
                     'plan', 'plans',
                     'recipes', 'recipe', 'recettes', 'recette', 'rezepte', 'rezept', 'ricette', 'ricetta',
                     'healthy', 'easy', 'vegan', 'quick', 'best', 'petite', 'petit', 'round', 'blue', 'homemade',
                     'special', "together", "weisses",
                     'without', 'mit',
                     'betty', 'bossi', 'betty bossi', 'thermomix', 'fooby',
                     'make', "restaurant", 'menu', "life", "sous vide", "original",
                     'oven', 'guide', 'four', 'ofen', 'forno', 'poêle',
                     'im', 'in', 'au', "good", "easter", "ideas", "idea",
                     'prepare', 'dishes', 'dish', 'freeze', "cook", "cottura",
                     'picking', 'pick', 'near', "veggies", "veggies",
                     'fresh', "savory",
                     'jamie', "oliver"
                    ]

    # add stop from other languages
    all_stopwords += stopwords.words('french')
    all_stopwords += stopwords.words('italian')
    all_stopwords += stopwords.words('german')

    # clean queries, they don't contain any stop words
    df['query_en_no_stopw'] = df['query_en'].apply(lambda x: ' '.join([word for word in x.split()
                                                                    if word not in (all_stopwords)]))

    # remove empty spaces and put to lower case
    df['query_en_no_stopw'] = df['query_en_no_stopw'].str.strip().str.lower()

    # replace hyphen (chocolate-mousse)
    df['query_en_no_stopw'] = df['query_en_no_stopw'].str.replace('-', ' ')

    # replace empty values by NaN
    df['query_en_no_stopw'].replace('', np.nan, inplace=True)

    # drop the rows where there are no values
    df.dropna(subset=['query_en_no_stopw'], inplace=True)
    df.reset_index(inplace=True, drop=True)

    return df
```

(Source: dw_clean_insert_data.py)

Appendix I Python Function to Normalize Diet Names

```
# Diet_trends
def clean_diet_search_trend(df_diet: DataFrame):
    """
    To normalise the diet names and have a unique diet name for each diet name.
    :return: The dataframe with an additional column (diet_category)
    which contains the normalised and unique diet names (normalised from the column query_en_no_stopw).
    """

    df_diet['diet_category'] = ""

    for index, row in df_diet.iterrows():
        # replace empty strings by NaN (if the query_en value contained only stopwords)
        if re.match(r"^\s?$", row['query_en_no_stopw']):
            df_diet['diet_category'].iloc[index] = np.nan

        # look for ketogene diets
        elif re.match(r".*[kc](eto)(ge)?", row['query_en_no_stopw']):
            df_diet['diet_category'].iloc[index] = "keto"

        # look for low carb diets
        elif re.match(r".*(carb)", row['query_en_no_stopw']):
            df_diet['diet_category'].iloc[index] = "low_carb"

        # planetary diet
        elif re.match(r".*(planet)", row['query_en_no_stopw']):
            df_diet['diet_category'].iloc[index] = "planetary"
```

(Source: dw_clean_insert_data.py)

Appendix J First Rows of the Diet Search Trend Table in the Data Warehouse

diet_category	region	date_last7d	value
carnivore	F	2022-03-29	100
egg	G	2022-03-29	7
go	F	2022-03-29	91
keto	G	2022-03-29	100
keto	I	2022-03-29	100
low_carb	G	2022-03-29	22
mediterranean	G	2022-03-29	5
planetary	F	2022-03-29	68
shake	G	2022-03-29	7

(Source: Own Figure)

Appendix K Extract of the Python Code in the Function clean_product_search

```
# remove row containing "grow" -> not related to cooking
if re.match(r".*(grow).*",row['query_clean']):
    df_product['query_clean'].iloc[index] = np.nan

# remove when people search for information
## about calories of meal / product
if re.match(r".*kcal.*|.calorie.*|.kalorie.*",row['query_clean']):
    df_product['query_clean'].iloc[index] = re.sub(r'.*kcal.*|.calorie.*|.kalorie.*', '', row['query_clean'])
## remove everytime they search for the cooking temperature = search for information
if re.match(r".*temper.*",row['query_clean']):
    df_product['query_clean'].iloc[index] = np.nan
## remove everytime they search for a product sold in Migros, Coop
if re.match(r".*migros.*",row['query_clean']):
    df_product['query_clean'].iloc[index] = np.nan
if re.match(r".*coop.*",row['query_clean']):
    df_product['query_clean'].iloc[index] = np.nan
## remove everytime they search for quantities per person
if re.match(r".*person.*",row['query_clean']):
    df_product['query_clean'].iloc[index] = np.nan
# remove when people search to buy something
if re.match(r".*(buy).*",row['query_clean']):
    df_product['query_clean'].iloc[index] = np.nan
```

(Source: dw_clean_insert_data.py)

Appendix L First Rows of the Product Search Trend Table in the Data Warehouse

query_clean	product	date_last7d	value
zucchini soup	courgette	2022-03-29	8
zucchini pasta	pasta	2022-03-29	24
zucchini pancake	courgette	2022-03-29	11
zucchini gratin	courgette	2022-03-29	23
yorkshire pudding	pudding	2022-03-29	38
wild garlic pesto	garlic	2022-03-29	31

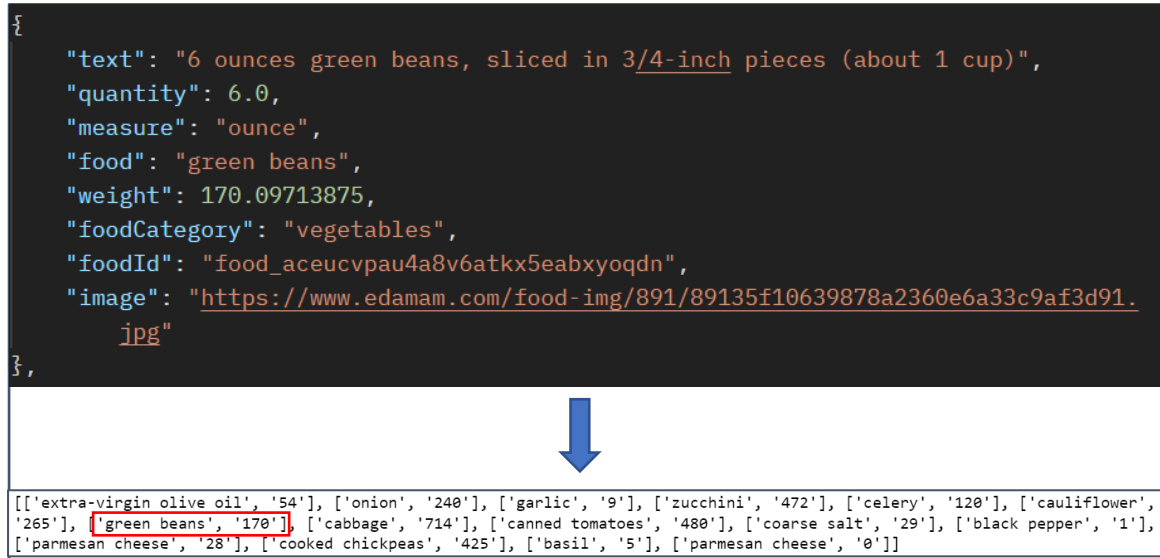
(Source: Own Figure)

Appendix M First Rows of the Recipe Search Trend Table in the Data Warehouse

query_clean	region	date_last7d	value
brownies	G	2022-03-29	13
bolognese	G	2022-03-29	12
waffles	G	2022-03-29	11
pizza	F	2022-03-29	15
crepes	I	2022-03-29	83
piadina	I	2022-03-29	26
pesto	I	2022-03-29	47

(Source: Own Figure)

Appendix N List of Ingredients and their Quantities



(Source: Own Figure)

Appendix O Example Output for Search Word 'apples'

		name	amount	price	pp 100 g
0		Prix Garantie apples	2500 g	3.95	0.16
1		Prix Garantie Apples Jonagold ca. 1kg	1000 g	2.90	0.29
2		Golden Apples 4-6 pieces ca. 1kg	1000 g	3.20	0.32
3		Braeburn Apples 4-6 Pieces ca. 1kg	1000 g	3.30	0.33
4		JaMaDu Kids Apples 8 Pieces	1000g	3.50	0.35
5		Boskoop Category I Apples ca. 1kg	1000 g	3.95	0.40
6		Crimson Snow I Apples ca. 1kg	1000 g	3.95	0.40
7		Gala Victor Apples 4-6 Pieces 1kg	1000 g	4.95	0.50
8		Pink Lady Apples 4-6 Pieces ca. 1kg	1000 g	4.95	0.50
9		JaMaDu Naturaplan Organic Kids Apples ca. 650g	650g	3.75	0.58

Appendix A: Example Output for Search Word 'apples'

(Source: Own Figure)

Appendix P Query that is Used to Get Prices of each Ingredients

```
# Get the first entry for the search item
sql_prod = """
SELECT *
FROM
(SELECT title, quantity, price, price_per_100
FROM "Product"
WHERE (LOWER(title) LIKE %s AND quantity LIKE %s)
ORDER BY LENGTH(title) ASC
FETCH FIRST 20 ROW ONLY
) AS temp
ORDER BY price_per_100 ASC
FETCH FIRST 1 ROW ONLY;
"""
```

(Source: Own Figure)

Appendix Q Function to transform the unit of a product

```
def unifying_unit(df: pd.DataFrame, unit: str, new_unit: str, calculator=int):
    for index, row in df.iterrows():
        if isinstance(row['quantity'], str):
            if re.search(r'\d+' + unit, row['quantity']):
                if "x" not in row['quantity']:
                    old_value = row['quantity']
                    new_value = int(float(old_value.split(unit)[0]) * calculator)
                    df.at[index, 'quantity'] = f"{new_value} {new_unit}"
    return df
```

(Source: coop_scrapper.py)

Appendix R Example how we have transformed the units properly

```
df_master = unifying_unit(df=df_master, unit="kg", new_unit="g", calculator=1000)
df_master = unifying_unit(df=df_master, unit="l", new_unit="ml", calculator=1000)
df_master = unifying_unit(df=df_master, unit="dl", new_unit="ml", calculator=100)
df_master = unifying_unit(df=df_master, unit="cl", new_unit="ml", calculator=10)
```

(Source: coop_scrapper.py)

Appendix S Function to compute the price per unit

```
def price_per_unit(df: pd.DataFrame, unit: str, calculator: int=100):
    for index, row in df.iterrows():
        if isinstance(row['quantity'], str):
            if re.search(r'\d+\s*' + unit, row['quantity']):
                if "x" not in row['quantity']:
                    value_no_unit = int(float(row['quantity'].split(unit)[0]))
                    price_float = float(row['price'])
                    if value_no_unit != 0:
                        new_value = (price_float / value_no_unit) * calculator
                        df.at[index, 'price per 100'] = round(new_value, 2)
    return df
```

(Source: coop_scrapper.py)

Appendix T Python Code for the Data Pipeline for Google Trends Data

```
# Category files in S3 bucket in the data lake: loop over them and load data into the data tables
search_categories = ["diet_search_trend", "diet_canton_trend", "evolution_key_diet",
                    "meat_canton_trend", "product_search", "recipe_search"]

# Goal: For each category, access all CSV files and get only the most recent one
# then clean and aggregate the data to write it into its proper data table in the data warehouse
for search_category in search_categories:
    print(f"{search_category} starting")
    csv_in_category = [] # create a list of all CSVs of this search category

    # 1) Get all CSV files of this given search category
    # Source: https://www.sqlservercentral.com/articles/reading-a-specific-file-from-an-s3-bucket-using-python
    for file_name_in_category in my_bucket.objects.filter(Prefix=search_category): # select the files in bucket
        file_name = file_name_in_category.key # get the csv file names of this search category
        if file_name.find(".csv") != -1: # append only CSV files format into the list
            csv_in_category.append(file_name_in_category.key)

    csv_in_category.sort() # Sort files by date
    print(f"CSV in the category: {csv_in_category}") # help to spot errors when running the code

    last_fetch_category = csv_in_category[-1] # select only the most recent CSV file of the search category
    print(f"last fetch: {last_fetch_category}") # verification check

    # 2) Select only the last_fetch_category of the category and write its content into data table
    obj = s3.Object(ENCRYPTED_BUCKET, last_fetch_category)
    data = obj.get()['Body'].read()

    # 3) Clean and Write data into tables
    # diet_canton_trend, evolution_key_diet, meat_canton_trend -> no further cleaning needed
    if search_category in ['diet_canton_trend', 'evolution_key_diet', 'meat_canton_trend']:
        # directly insert data into data tables
        insert_data(pd.read_csv(io.BytesIO(data), header=0, delimiter=",", low_memory=False), f'{search_category}')

    # additional normalisation depending on the table:
    else:
        data = pd.read_csv(io.BytesIO(data), header=0, delimiter=",", low_memory=False)
        # translate to english
        data = translate_en(data)
        # remove stopwords
        data = remove_stopw(data)

        # diet_search_trend
        if search_category == 'diet_search_trend':
            data = clean_diet_search_trend(data) # runs its proper function to clean and aggregate the data
            # insert data into data tables
            insert_data(data, 'diet_search_trend') # runs the function to write data into its table in the data warehouse

        # product_search
        elif search_category == 'product_search':
            data = clean_product_search(data) # runs its proper function to clean and aggregate the data
            # insert data into data tables
            insert_data(data, 'product_search_trend') # runs the function to write data into its table in the data warehouse

        # recipe_search
        elif search_category == 'recipe_search':
            data = clean_recipe_search(data) # runs its proper function to clean and aggregate the data
            # insert data into data tables
            insert_data(data, 'recipe_search_trend') # runs the function to write data into its table in the data warehouse
```

(Source: dw_clean_insert_data.py)

Appendix U Setup with all Permission except "Super User" to all tables for user "deep_diver"

Name: Object ID:

☐ Super User
 ☒ Inherit
 ☒ Create Role
 ☒ Create Database
 ☒ Can Login
 ☐ Replication
 ☐ Bypass RLS

Enter a part of object name here

Roles

Settings

Permissions

Source

public

Tables

Product

diet_canton_trend

diet_search_trend

evolution_key_diet

meat_canton_trend

product_search_trend

recipe_search_trend

recipes

recipes_trend

top_product_trend

top_recipe_trend

Views

Materialized Views

Functions

Sequences

Permission	With GRANT	With Hierarchy
<input checked="" type="checkbox"/> SELECT	X	X
<input checked="" type="checkbox"/> INSERT	X	
<input checked="" type="checkbox"/> UPDATE	X	
<input checked="" type="checkbox"/> DELETE	X	
<input checked="" type="checkbox"/> TRUNCATE	X	
<input checked="" type="checkbox"/> REFERENCES	X	
<input checked="" type="checkbox"/> TRIGGER	X	

Grant All Revoke All

public."Product", public.diet_canton_trend, public.diet_search_trend, public.evolution_key_diet, public.meat_canton_trend, public.product_search_trend, public.recipe_search_trend, public.recipes, public.recipes_trend, public.top_product_trend, public.top_recipe_trend

(Source: Own Figure)

Appendix V Setup only SELECT permission to all tables for user "tableau_user"

Name: Object ID:

☐ Super User
 ☐ Inherit
 ☐ Create Role
 ☐ Create Database
 ☒ Can Login
 ☐ Replication
 ☐ Bypass RLS

Enter a part of object name here

Roles

Settings

Permissions

Source

public

Tables

Product

diet_canton_trend

diet_search_trend

evolution_key_diet

meat_canton_trend

product_search_trend

recipe_search_trend

recipes

recipes_trend

top_product_trend

top_recipe_trend

Views

Materialized Views

Functions

Sequences

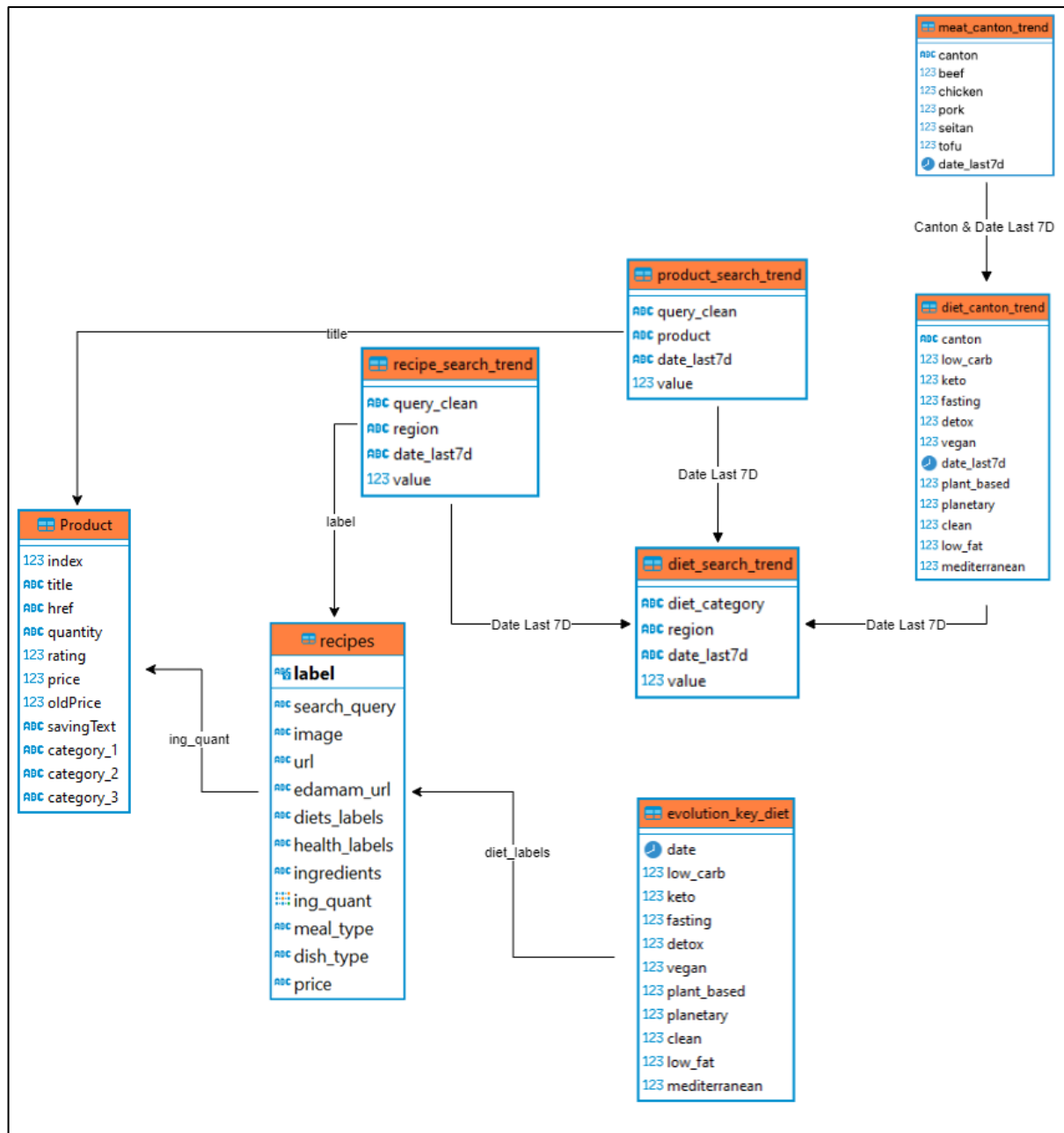
Permission	With GRANT	With Hierarchy
<input checked="" type="checkbox"/> SELECT		X
<input type="checkbox"/> INSERT		
<input type="checkbox"/> UPDATE		
<input type="checkbox"/> DELETE		
<input type="checkbox"/> TRUNCATE		
<input type="checkbox"/> REFERENCES		
<input type="checkbox"/> TRIGGER		

Grant All Revoke All

public."Product", public.diet_canton_trend, public.diet_search_trend, public.evolution_key_diet, public.meat_canton_trend, public.product_search_trend, public.recipe_search_trend, public.recipes, public.recipes_trend, public.top_product_trend, public.top_recipe_trend

(Source: Own Figure)

Appendix W Database Schema

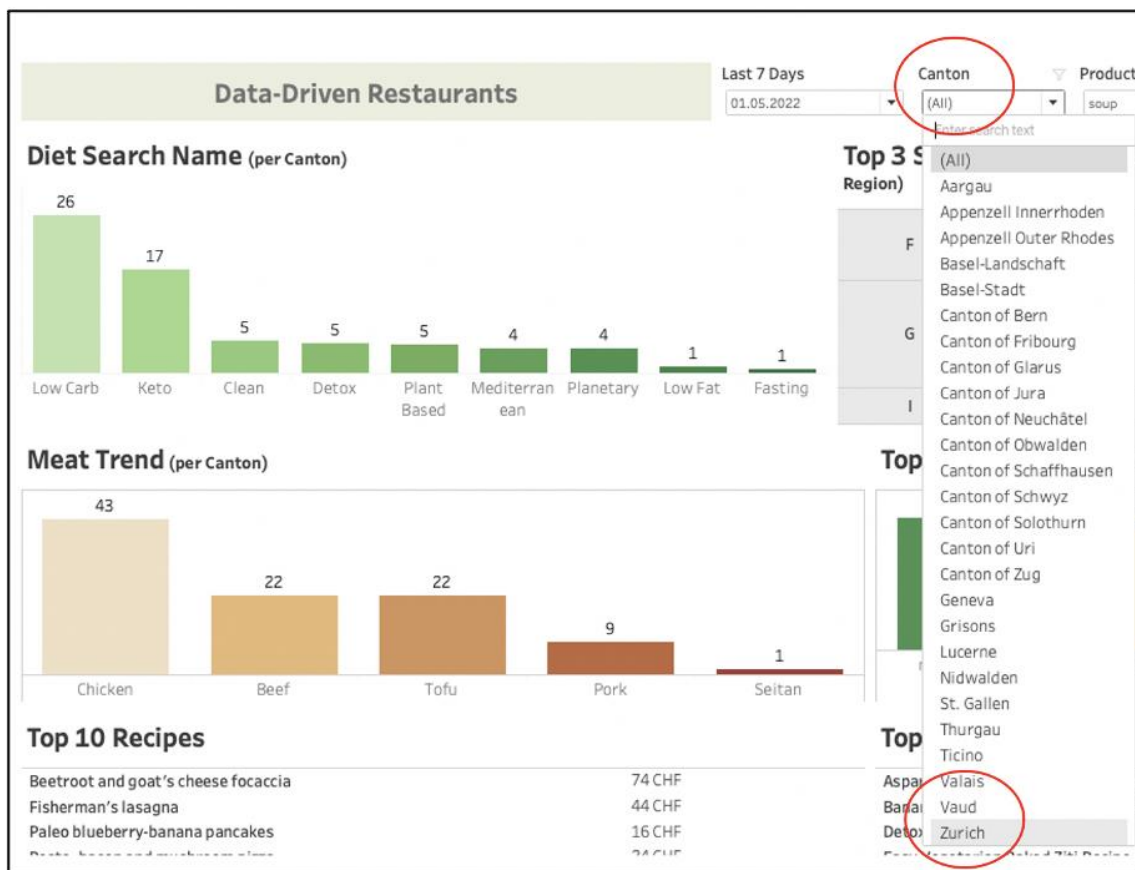


(Source: Own Figure)

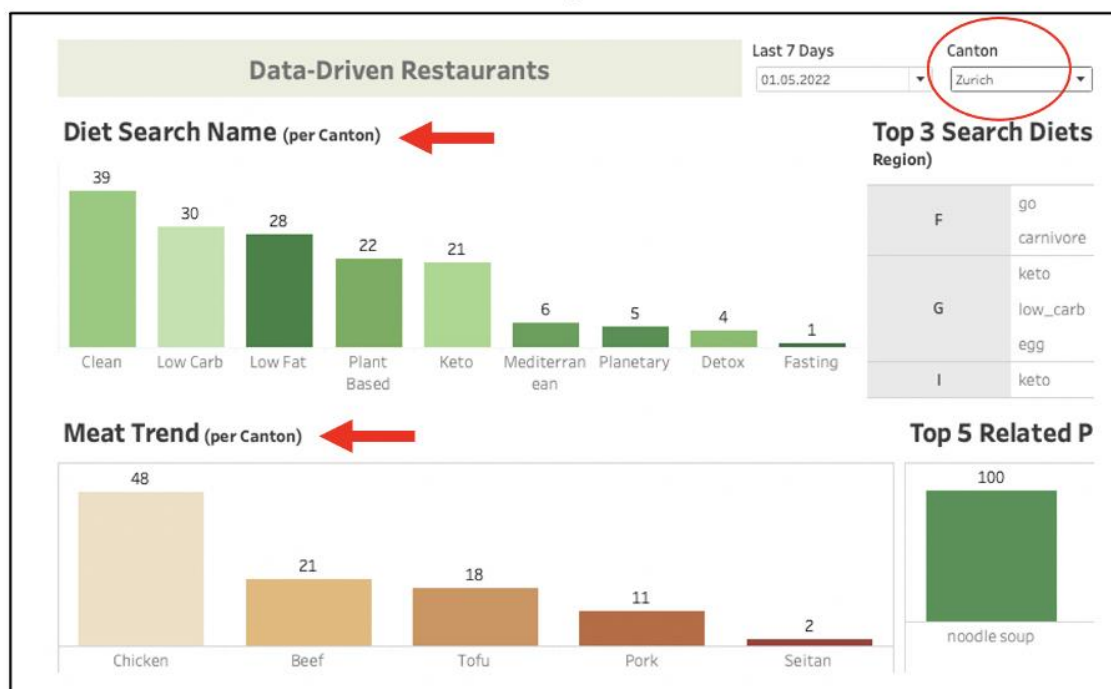


(Source: Own Figure)

Appendix Y Tableau Dashboard: Canton Filter

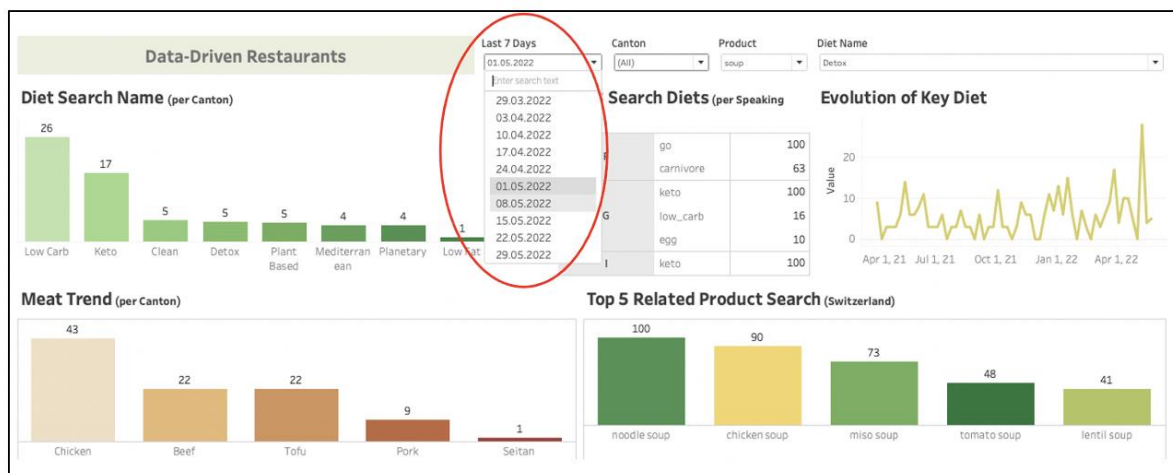


Changes :

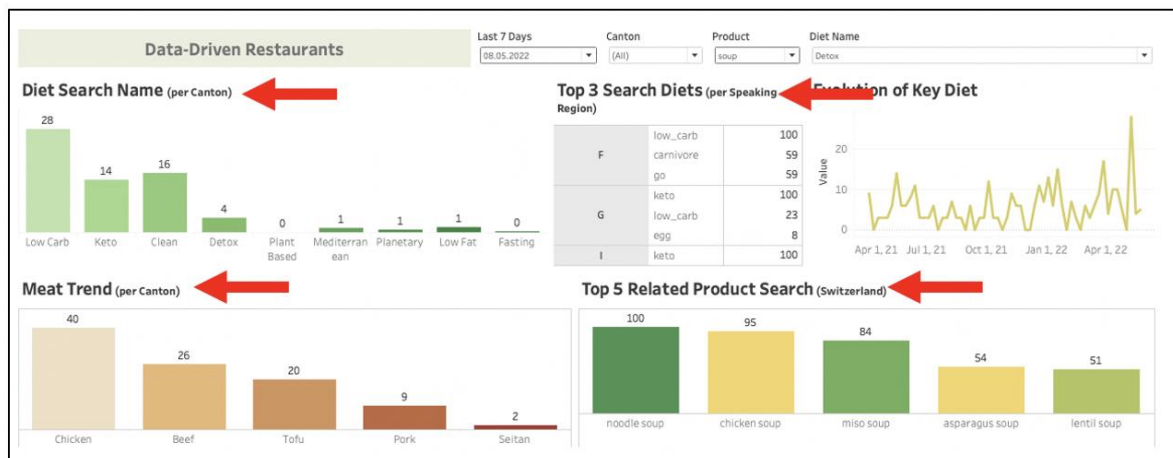


(Source: Own Figure)

Appendix Z Tableau Dashboard: Last 7 Days Filter

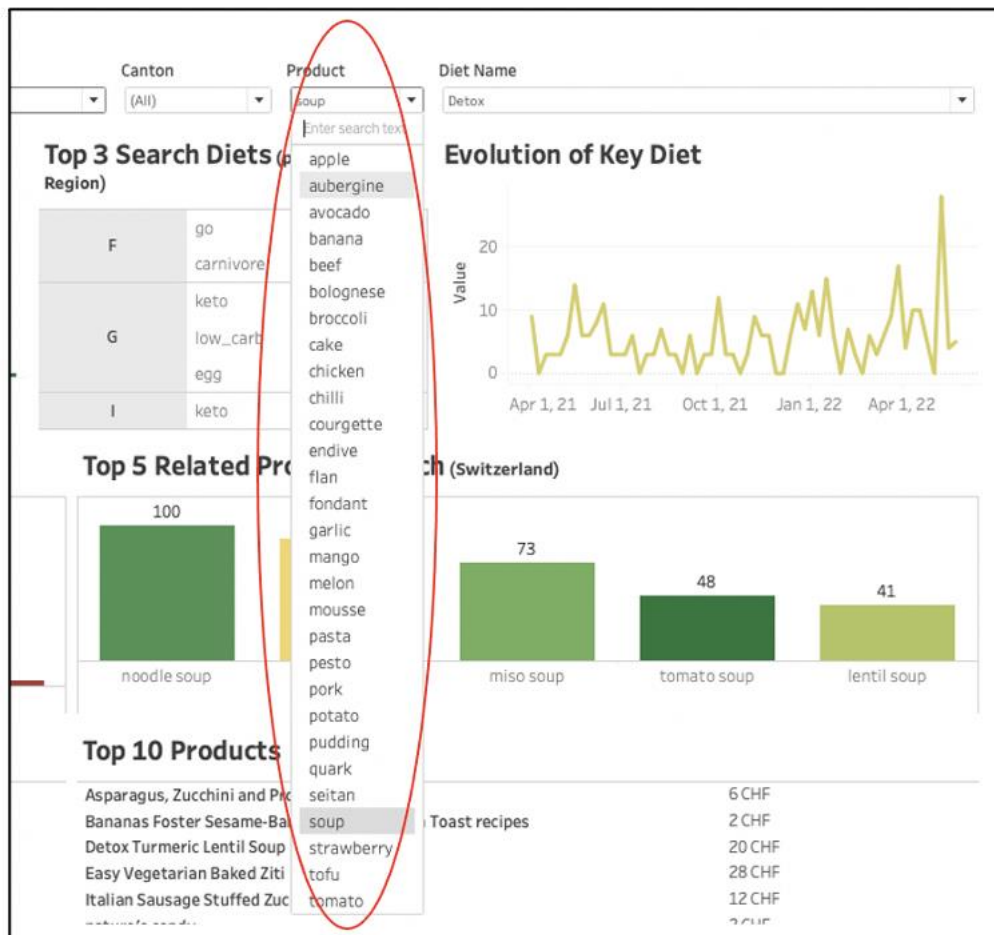


Changes:

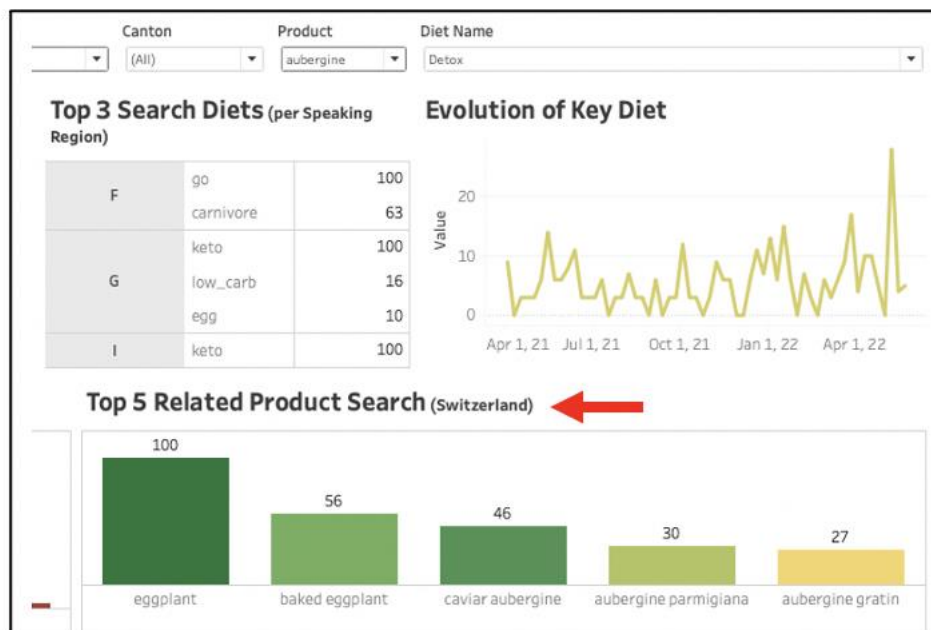


(Source: Own Figure)

Appendix AA Tableau Dashboard: Product Filter

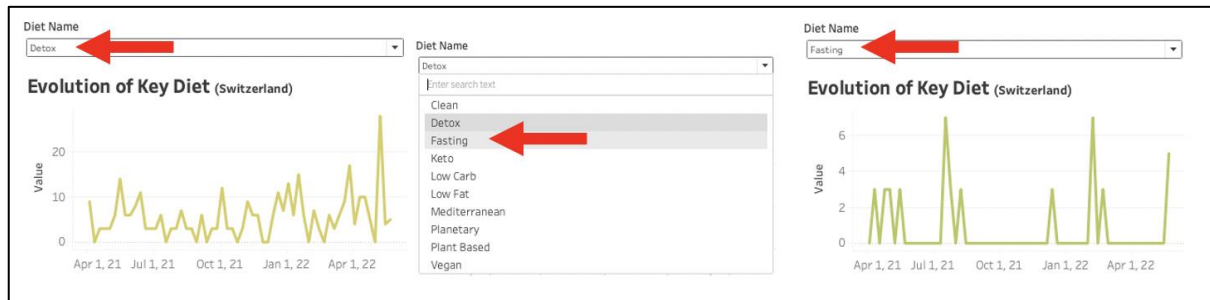


Changes:



(Source: Own Figure)

Appendix BB Tableau Dashboard: Diet Name Filter



Appendix B: Tableau Dashboard: Diet Name Filter

(Source: Own Figure)

Appendix CC Price Calculation for 'Banana-chilli pickle' (max price)

	ing	quant	ing_c	ing_s	c_title	c_quant	c_price	c_pp1	price	Total_price	
0	chillies	3000	chillies	chillies	Fine Food Ground Bird's Eye Chillies	40g	6.45	16.12	0	483.60	
1	chillies	360	chillies	chillies	Fine Food Ground Bird's Eye Chillies	40g	6.45	16.12	0	58.03	
2	salt	219	salt	salt		salt	0	0.00	0.00	0	0.00
3	dill seeds	13	dill seeds	dill seeds		dill seeds	0	0.00	0.00	0	0.00
4	aniseed	80	aniseed	aniseed	Whole Aniseed	25g	0.95	3.80	0	3.04	
5	yellow mustard seeds	67	yellow mustard seeds	mustard seeds	Yellow Mustard Seeds	50g	1.60	3.20	0	2.14	
6	mustard seeds	13	mustard seeds	mustard seeds	Yellow Mustard Seeds	50g	1.60	3.20	0	0.42	
7	fennel seeds	12	fennel seeds	fennel seeds	Naturaplan Organic Fennel Seeds	26g	1.40	5.38	0	0.65	
8	peanut oil	216	peanut oil	peanut oil		peanut oil	0	0.00	0.00	0	0.00

(Source: Own Figure)

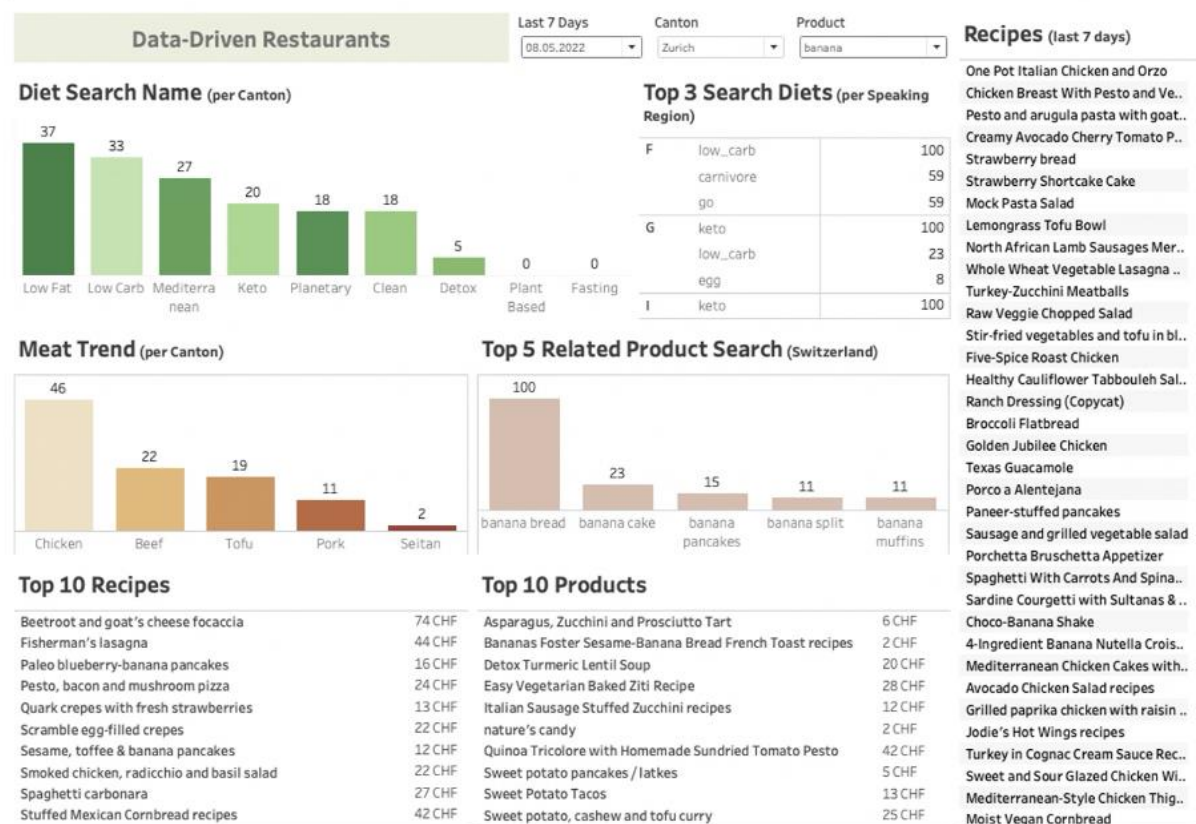
Appendix DD Price Calculation for 'Cheesy Chicken Italiano' (0 price)

	ing	quant	ing_c	ing_s	c_title	c_quant	c_price	c_pp1	price	Total_price
0	rotini pasta	315	rotini pasta	rotini pasta	rotini pasta	0	0	0	0	0
1	boneless skinless chicken breasts	454	boneless skinless chicken breasts	skinless chicken breasts	skinless chicken breasts	0	0	0	0	0
2	Pasta Sauce	396	pasta sauce	pasta sauce	pasta sauce	0	0	0	0	0
3	VELVEETA	170	velveeta	velveeta	velveeta	0	0	0	0	0

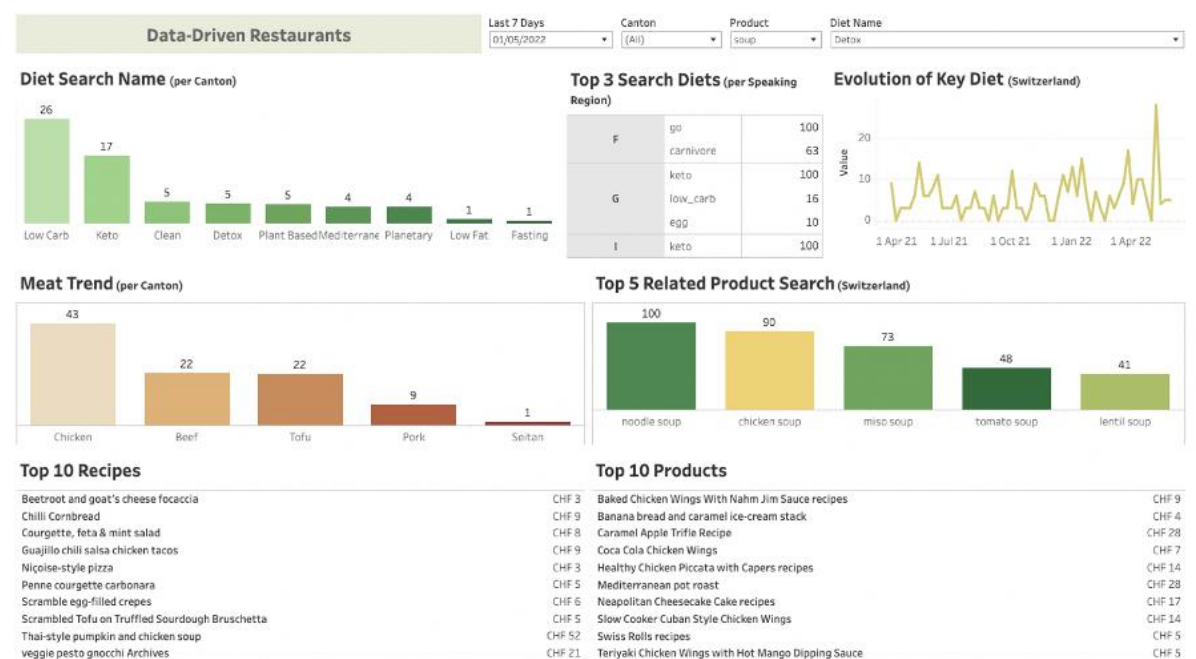
(Source: Own Figure)

Appendix EE Comparison of Tableau Dashboard Versions

Version Presented on the 19th of Mai 2022



Final Version



(Source: Own Figure)

9 Annexes (Work Split)

Table 3: Evaluation Grid

Evaluation Criteria	Maximum Possible Points	Ezgi Köker Gököl	Vithushan Mahendran	Alexandra Alinka Zimmermann
Introduction	18			
Table of contents	3	x	x	x
Topic introduction	3			x
Problem definition	3			x
Goals	3			x
Business / Research questions	3			x
Limitations	3			x
State of the art	5			
Previous state of your project (from last part)	5	x	x	x
Methodology	42			
Data Quality	7	x	x	x
Production data pipelines	7	x	x	x
Data warehouse architecture, system etc. in technical details	7		x	
Visualization design justification	7			x
Tableau visualization (with public link)	7			x
Answer to research/business questions	7	x	x	
Discussion	15			
Proper discussion of the solution (advantages, disadvantages, and data results insights etc.)	15	x	x	x
Conclusions	7			
Proper discussions of the project outcomes.	7	x		
Future Work	5			
Areas to improve in the data warehouse, what would you do different etc.	5	x	x	x

Table 4: Code Project Leads

Responsibilities	Ezgi Köker Gökgöl	Vithushan Mahendran	Alexandra Alinka Zimmermann
Google Trends API			x
Edamam API	x		
Coop Webscraping		x	
Tableau Dashboard (Tableau Desktop and Tableau Public)			x

10 Sources

10.1 Source Information

Abhishek Agarrwal. "Tableau Tutorial 122 - How to use Index function for ranking data" (November 26, 2018). Accessed Accessed 16th of May 2022 [Online Video]. Available: <https://www.youtube.com/watch?v=uRxU-wYcRU>

[Apipheny 2022] Apipheny: *What are API headers?*. Available <https://apipheny.io/api-headers/#:~:text=API%20headers%20are%20like%20an,track%20down%20any%20potential%20issues>, accessed 19th April 2022.

Anonym, "Is there a way to get access_key and secret_key from boto3?,"stackoverflow, blog, December 21, 2016 [Online]. Available: <https://stackoverflow.com/questions/41270571/is-there-a-way-to-get-access-key-and-secret-key-from-boto3>

Anonym, "Python remove stop words from pandas dataframe,"stackoverflow, blog, April 08, 2015 [Online]. Available: <https://stackoverflow.com/questions/29523254/python-remove-stop-words-from-pandas-dataframe>

Anonym, "Save Dataframe to csv directly to s3 Python,"stackoverflow, blog, July 01, 2016 [Online]. Available: <https://stackoverflow.com/questions/38154040/save-dataframe-to-csv-directly-to-s3-python>

Anonym, "Using NLTK corpora with AWS Lambda functions in Python,"stackoverflow, blog, February 22, 2017 [Online]. Available: <https://stackoverflow.com/questions/42382662/using-nltk-corpora-with-aws-lambda-functions-in-python>

BiteSize Academy. "How to use environment variables with a Lambda function? (and how to encrypt them with KMS)" (May 8, 2020). Accessed 16th of April 2022 [Online Video]. Available: <https://www.youtube.com/watch?v=J9QKS0NrH7I&t=277s>

B. Shneiderman, "The eyes have it: a task by data type taxonomy for information visualizations," Proceedings 1996 IEEE Symposium on Visual Languages, 1996, pp. 336-343, doi: 10.1109/VL.1996.545307.

L. Celis, "Google Trends API," The Ad Tech Blog, blog, March 07, 2022 [Online]. Available: <https://blog.leocelis.com/2020/07/03/google-trends-api/>

[Color Meanings, n.d.] Color Meaning: *Color Meanings – The Power and Symbolism of Colors*. Available: <https://www.color-meanings.com/#brown>, accessed 4th June 2022.

[Edamam 2022] Edamam: *Company*. Available <https://www.edamam.com/about/company>, accessed 18th April 2022.

[Edamam 2022] Edamam: *Recipe Search API*. Available <https://developer.edamam.com/edamam-recipe-api>, accessed 18th April 2022.

[Freeman 2022] Edraw: *Line Graph - When to Use It?*. Available: <https://www.edrawsoft.com/chart/when-to-use-line-graph.html>, accessed accessed 05th June 2022.

[GeeksforGeeks 2022] Geeks for Geeks: *How to write Pandas DataFrame to PostgreSQL table?*. Available <https://www.geeksforgeeks.org/how-to-write-pandas-dataframe-to-postgresql-table/>, accessed 26th March 2022.

[Google Trends 2022] Google Trends Support: *FAQ about Google Trends data*. Available <https://support.google.com/trends/answer/4365533?hl=en>, accessed 20th April 2022.

[Hackernoon 2021] ScraperAPI: *Build a Powerful Google Trends Scraper Using PyTrends: A Step-By-Step Guide*. Available <https://hackernoon.com/build-a-powerful-google-trends-scraper-using-pytrends-a-step-by-step-guide>, accessed 10th April 2022.

[SqlServerCentral 2021] Sql Server Central: *Reading a Specific File from an S3 bucket Using Python*. Available <https://www.sqlservercentral.com/articles/reading-a-specific-file-from-an-s3-bucket-using-python>, accessed 26th March 2022.

[Tableau 2018] Tableau: *Combining 3 Line Graphs*. Available: <https://community.tableau.com/s/question/0D54T00000C6K5MSAV/combining-3-line-graphs>, accessed 16th May 2022.

[Tableau 2022] Tableau: *Tableau Desktop*. Available: <https://www.tableau.com/products/desktop>, accessed 05th June 2022.

[The Rail Media 2020] Tobias Foster: 11 Big Data Strategies to Boost Your Restaurant's Sales [Image]. available: <https://www.therail.media/stories/2020/2/19/11-big-data-strategies-to-boost-your-restaurants-salesa>, accessed 11th March 2022.

Patrick Trasborg (2017) google-trends-api [Source Code] <https://github.com/pat310/google-trends-api/wiki/Google-Trends-Categories>

[Unilever Food Solutions 2022] Unilever Food Solutions: *How to update your menus: new trends and designs*. Available: <https://www.unileverfoodsolutionsarabia.com/en/chef-inspiration/trend-watch/how-to-update-your-menus.html>, accessed 11th March 2022.

[Wikipedia 2022] Wikipedia: *Google Trends*. Available https://en.wikipedia.org/wiki/Google_Trends, accessed 10th April 2022.

[Wikipedia 2022] Wikipedia: *List of diets*. Available https://en.wikipedia.org/wiki/List_of_diets, accessed 26th March 2022.

[WordStream n.d.] Word Stream: *Google Trends: What Is Google Trends?*. Available: <https://www.wordstream.com/google-trends>, accessed 11th March 2022.

[Yi n.d.] Chartio: *A Complete Guide to Bar Charts*. Available: <https://chartio.com/learn/charts/bar-chart-complete-guide/>, accessed 11th March 2022.

10.2 GitHub Repository

Project Source Code: https://github.com/vithu92/deep_divers