



(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 1

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

Lucerne, 24th of April 2022

Table of Contents

List of Figures	iii
List of Tables	iv
List of Abbreviations and Acronyms	iv
1 Introduction	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
1.6 Literature Review.....	3
2 Methodology.....	4
2.1 Data Lake Architecture	4
2.1.1 AWS Lambda Function	4
2.1.2 AWS S3 Buckets	4
2.1.3 AWS RDS.....	4
2.1.4 Python.....	5
2.1.5 EventBridge	5
2.1.6 Amazon Key Management / Secrets Manager Services	5
2.2 Design choices, Trade-offs, and Assumptions	5
2.2.1 Google Trends	5
2.2.2 Edamam	7
2.2.3 Coop.....	7
2.3 Data Sources Definitions	8
2.3.1 Google Trends	8
2.3.2 Edamam	8
2.3.3 Coop.....	9
2.4 Data Imputation and Data Cleaning per Data Source	10
2.4.1 Google Trends	10
2.4.2 Edamam	12
2.4.3 Coop.....	12
2.5 Data analysis per data source.	13
2.5.1 Google Trends	13
2.5.2 Edamam	15
2.5.3 Coop.....	16
2.6 Database Schema	16
2.6.1 GoogleTrends	16

2.6.2	Edamam	17
2.6.3	Coop.....	17
2.7	Data Lake Consumers	18
3	<i>Discussions.....</i>	18
3.1	Google Trends.....	18
3.1.1	Quality of the data	18
3.1.2	Lists of Search Terms	18
3.1.3	Languages	19
3.1.4	Write Data from a Data Frame.....	19
3.1.5	Validation Tables	19
3.2	Edamam	19
3.3	Coop.....	19
4	<i>Conclusions.....</i>	20
5	<i>Future Work.....</i>	20
6	<i>Appendix</i>	21
7	<i>Annexes</i>	30
8	<i>Source.....</i>	32
8.1	Source Information	32
8.2	Source Images.....	33
8.3	Source Code	33
8.3.1	Google Trends	33
8.3.2	Coop.....	34

List of Figures

Figure 1 : Systems, Services & Tools	4
Figure 2 : Flow Diagram for Google Trends Data [own figure]	10
Figure 3 : Flow Diagram for Edamam Data [own figure]	12
Figure 4 : Flow Diagram for Coop Data [own figure]	13
Figure 5: Screenshot of the Table diet_canton_trend	14
Figure 6: Number of Null Values in the Table diet_canton_trend.....	14
Figure 7: Number of Null Values in the Table evolution_key_diet.....	15
Figure 8: Entity Relationship Diagram for Google Trends API	17
Figure 9: Entity Relationship Diagram for Edamam API.....	17
Figure 10: Entity Relationship Diagram for Coop Products.....	18

List of Tables

Table 1: Details of ‘recipes’ Table.....	15
Table 2: Number of Recipes for Each Search Query	16
Table 3 : Coop Product	16
Table 4: AWS Credit Conditions for Each Data Source.....	20
Table 5: Evaluation Grid.....	30
Table 6: Code Project Leads	31

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
RDS	Relational Database Service
SQL	Structured Query Language
URL	Uniform Resource Locator

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 will use a Google Trends API. Second, thanks to Edamam API, we will suggest them various recipes which are based on these meals, diet types, or ingredients preferences. Finally, we will also help them to plan their budget. Thus, we want to provide them with an estimation of the total recipe cost. 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 a Data Lake system where we load the data extracted from the three data sources mentioned previously. In the second part of the project, we will build a Data Warehouse, where we will clean, analyze, and prepare our data to be able to answer our research questions. Finally, we will visualize our data using Tableau Desktop and provide access 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 are 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 methods, which some of us are only learning this semester. Therefore, in order to understand the data that we collected, we need to find a solution to aggregate similar data into one unique data, for example, keto diet, ketogene diet and keto, which are the same concept. Only then, we will be able to provide a precise analysis of our extracted data to the restaurant owners.

5. Limitations within the AWS environment

Currently, the course benefits of only some of the Amazon Web Services. Therefore, the complexity of our project solution strongly depends on the services that are at our disposal.

1.6 Literature Review

History of food dates back to the beginning of humanity. The story starts with the conditions of accessibility and fighting for food [Recordati 2014/2015]. Over time, but still in a distant past, it evolves to simple diets like wild animals, fruits, and vegetables. With the civilization, industrialization, innovations and economic growth, the habit of eating meals with high nutritional value leaves its place to attractive, tasty but rather unhealthy food. Companies increase the level of processed foods and synthetic fertilizers to maximize their profit which seems to be a win-win case for both customers and firms since the costs drop. As people become more conscious about their eating habits and the impact on their health [Gynell et al. 2022], the current trend becomes to eat meals that have been processed as little as possible.

Consumer behavior can be defined as “the study of individuals, groups, or organizations and the processes they use to select, secure, use, and dispose of products, services, experiences, or ideas to satisfy needs and impacts that these processes have on the consumer and society” [Kuester 2012]. To survive in today’s competitive market, a restaurant must now be agile and adjust itself according to the customer eating behaviors.

As the world becomes more globalized, the restaurant’s customer base has changed. Clients come from all around the world. They often expect to have a large variety of meal options, but also, they expect specific types of meals and high-quality dishes [Türker & Süzer 2022]. As the customers are not only local people anymore, the restaurant owners have to adjust their menus and the selection of their ingredients to satisfy their new clientele.

With the globalization and the technological innovations, a big pool of consumer data is formed and become available to everyone's use. Nevertheless, to be able to get applicable answers to business questions, this raw and disparate data pool must be processed according to the need. Current data analysis methods and enhanced technologies allow the user to easily filter the required information from the organized version of this data ocean.

By definition, a data warehouse is a type of data management system which is designed to enable and support business intelligence activities, especially analytics, with the help of large amount of data that has been derived from multiple sources [Oracle 2022]. A properly implemented data warehouse supported with well-developed data lakes will ensure better business operations [Gardner 1998]. A success can only be mentioned when there is a strong harmony between the technology and the business purpose [Weir et al. 2003].

With the innovations in the business world, the size and complexity of the data warehouses are increasing, leading to further necessities in software, security, and hardware requirements [Coffey 2021]. To cover the high demand, cloud-based data warehousing gain importance considering its accessibility, elasticity,

improved performance, high data storage, enhanced integration, and improved disaster recovery [Rehan 2021, Levy 2021].

2 Methodology

2.1 Data Lake Architecture

The utmost benefit of a data lake is that you can store any kind of data in one place incurring a low cost and effortlessly pulling data as analytical needs arise. Due to this centralization of data, it helps a lot to combine different data sources together and helps to find new insights by combining data. The services offered by AWS provides scalability, flexibility and versatility in a relatively inexpensive way compared to a traditional data storage.

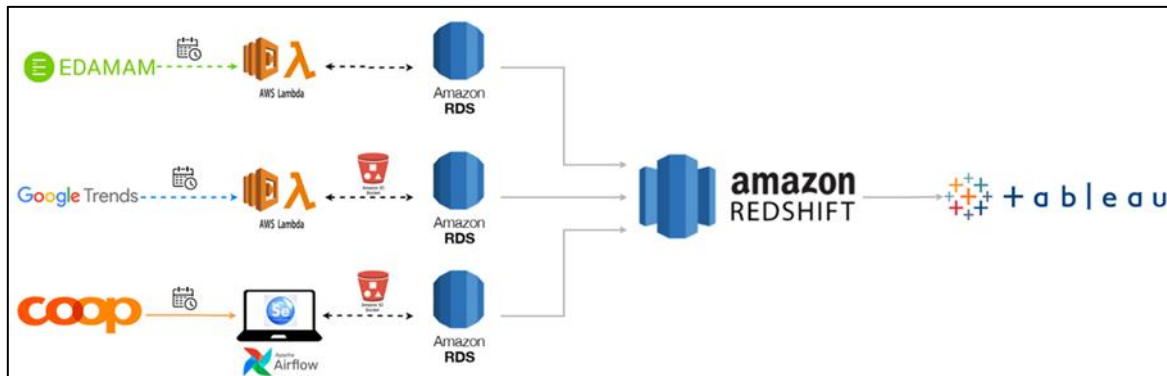


Figure 1 : Systems, Services & Tools

In this section, we will shortly describe all Systems & Services which we have used in this project. Most of the information can be found directly on the AWS documentation online.

2.1.1 AWS Lambda Function

The AWS Lambda function is useful to run code without any concerns about server provisioning and managing. And due to the very cost-efficient pricing (pay only for the compute time that you consume), we tried to use this tool in the full extent. It suits very well if you are implementing functions with short process time.

2.1.2 AWS S3 Buckets

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.

2.1.3 AWS RDS

We use the Amazon Relational Database Service (AWS RDS) as our main relational database. RDS allows to easily deploy scalable databases and at a low cost. Because of the numerous settings of the database that could be easily defined, especially in terms of storage, processing power and backup, we could quickly start using the database. We decided to use PostgreSQL as our database management system because it provides benefits which suits perfectly with the project's needs.

To be on the safer side, we decided to create for each data source separate database instances. This makes sure, that if something wrong happens with the database instance itself, it will only impact a part of all collected data.

2.1.4 Python

Python is our main programming language. The advantages are that there is a big community and large set of packages to choose from and implement in our solution. We mainly use Python for fetching data from the two APIs and for the web scrapping.

2.1.5 EventBridge

To trigger the whole ETL process within the AWS environment, we used the Amazon EventBridge service. It is a serverless event bus service which connects to our AWS services and targets such as AWS Lambda functions. With this orchestrator tool, we have more control on when and how each sub-process will be triggered. By defining the triggering time carefully, we prevented to confront with the limitation from the data source side.

2.1.6 Amazon Key Management / Secrets Manager Services

Amazon Key Management and Secrets Manager Services gives us a centralized control over the credentials which we use in our projects. Thanks to the good integration of the AWS Services, it makes it very easy to retrieve credentials and keys anywhere in the AWS environment. The data will be saved encrypted and decrypted using the keys.

2.2 Design choices, Trade-offs, and Assumptions

Our project is based on various assumptions and trade-offs. In this section, we present them, the design choices, and the adjustments that we made for each data source.

2.2.1 Google Trends

Pytrends

Since there are no publicly available Google Trends API [Celis 2020], we decided to use Pytrends, the unofficial Google Trends API. We used some of its methods to fetch data available on Google Trends.

To use pytrends, we first had to create a TrendReq object and define the time zone. We set it to Central Europe (tz = 60), in order to fetch the web trends from Switzerland.

One of the pytrends' methods, namely `build_payload`, allows us to filter the categories to which the search queries correspond. For instance, the search query "apple" could be both a company and a fruit. For this project we used two categories (cat): 457, which corresponds to the "Special & Restricted Diets" and 122 to "Cooking & Recipes" [Patrick Trasborg 2017]. We also specified the geolocation to Switzerland (geo = 'CH') and filtered the results to only web search (gprop = "). Regarding the time frame, we tested different values. Ideally, we wanted to have historical web search data over the last few years on a weekly or daily basis. Unfortunately, the data that we could extract was not complete for Switzerland. Therefore, the time frame was set up to get the Google web trends of the last seven days starting at the end of March 2022. For one table, namely `keyword_trend_time`, we could fetch the weekly trends from April 2021 to nowadays.

Data Cleaning

The applied filters mentioned above allowed us to fetch filtered, thus cleaner data. We obtained meaningful data that matched one of the two selected food categories thus already removing the noise as much as possible. Consequently, we decided not to clean the data anymore for this part of the project. We were aware that we might still get noisy data, but we left the further data cleaning step for the next part of the course. We will probably use the AWS Glue DataBrew service to clean and normalize our data.

Search Terms and Output

`related_queries` returns a Python dictionary with all the top and rising queries that are related to the search term and their normalized measures. We selected only the top queries as it provided more information.

To fill the `meat_canton_trend` table, we input a list of 5 meat types, namely beef, chicken, pork, seitan, and tofu into our `pytrends` method and fetched their normalized values for the canton of Switzerland. We selected these meat types as they were for us the most frequent categories sold by a restaurant. We also selected some vegetarian alternatives. The objective was to compare the cantons to find out their meat preferences and to highlight any differences. Being able to reveal patterns in our data would help the restaurant owners to choose which meat to offer on their menu depending on the location of their restaurant.

To fill the `diet_canton_trend` and `evolution_key_diet` we built a list with 10 famous diets names [Wikipedia 2022], namely low carb, keto, fasting, detox, vegan, plant based, planetary, clean, low fat and mediterranean. Using the normalized values available on Google Trends, the goal was to fetch the diets preferences in the cantons of Switzerland (for `diet_canton_trend` table) and for Switzerland in general (`evolution_key_diet` table). This would help the restaurant owners to find out whether there is a demand for a given diet in the canton where his restaurant is located.

Based on the food categories that Coop sells, we created a list of food items that were meaningful to restaurant owners. The list can be found in the Appendix A. From this list, we fetched the related search queries to these terms and filled the `product_search` table with the result. The idea was to find out what Swiss people usually search about these words and inform the restaurant owners. For example, when people search for soup, they often looked up noodle, broccoli, pea, or lentil soup. Knowing that, a restaurant owner could then create a meal for their menu and better fit their customers' preferences.

Languages

For `diet_search_trend` and `recipe_search` tables we needed to fetch the related queries to diet and recipe search terms for the speaking regions of Switzerland. However, in Switzerland there are 4 official languages. Working with textual data, solving this was for us a real challenge. Therefore, we first decided to select only 3 of the 4 languages, namely German, French, and Italian and leave the Romansh out. Then we also assumed that people who searched for `diät` or `rezept` would come from the German (G) part of Switzerland, `diet` or `recette` from the French (F) part, and `dieta` or `ricetta` from the Italian (I) region. Based on that, we created another column `region` with the corresponding region letter, namely G, F and I.

Lambda Functions

We chose to work with Lambda Functions over Apache Airflow to automate the ETL process. The two main reasons were first that our ETL tasks were small and thus would always last less than 15 minutes. Second, Lambda Function is an AWS service, already implemented in our working environment, which facilitated the integrations of the other services that we used.

We decided to build three Lambda functions to perform the whole ETL process. One function would fetch the data from the Google Trend API. Another would write the data into the data tables. The last one would validate the written data. Each function ran at thirty minutes interval from each other. This allowed us to solve problems that could have occurred in one function and run it again without impacting the other functions.

CSV Files Time Stamp

Each week, data was fetched and saved into a CSV file stored in a S3 Bucket. We wanted to keep a record of the raw data before processing it. Additionally, we did not want to directly upload the fetched data into the tables and chose to first store them into a S3 Bucket to ensure that we could recover the data if something went wrong in the ETL process.

All the files are named according to a clear rule:

`{table name}_{timestamp when data was fetched}.csv`

2.2.2 Edamam

Data Fetching

To be able to make recipe suggestions to the restaurant owners, we used Edamam – Recipe Search API and obtained various recipes related with the search query (for example ‘chicken’). Fetching of the required data is done by ‘requests’ package and the JSON formatted response is prettified by using ‘json’ package.

Data Cleaning

While extracting the data, we saw that there were some responses which did not involve the search query at all. After the examination, we found that there is a ‘Try Also’ column at the right-hand side of each recipe page, which gives a list of similar recipes (Appendix B). If one of the recipes in this column has the search query in its name, then the API returns it as a response, although the original recipe is not related. For example, when the search query is ‘Tofu’, ‘Roasted Shrimp and Asparagus Scampi’ recipe came as one of the responses although it did not involve tofu as an ingredient. This is because the ‘Stir-fried Tofu and Green Beans’ recipe is in the first place of the ‘Try Also’ list on the Scampi recipe’s page. So, this problem is solved by adding a check whether the ingredient list of the current recipe involves the search query itself (**Fehler! Verweisquelle konnte nicht gefunden werden.**). If not, it is not added to the data frame.

Lambda Functions

We decided to work with Lambda Function on AWS for the ETL process of Edamam API. Similar to Google Trends, this choice was based on our ETL process being relatively simple and Lambda Functions being part of our working environment while Apache Airflow is not.

In addition to the limitation mentioned in chapter 1.5, Developer version of the Edamam – Recipe Search API allows 10 calls per minute. As it can be seen from Appendix A, the refined version of the search query list has 29 terms. So, it is not possible to obtain results for each term in one run of the code. In order to solve this issue, we created three lambda functions, divided the list to three separate sets of 10 calls and run the code accordingly. For all three functions, different id/keys of the API are used to be able to run them simultaneously. Since all three Lambda functions are similar to each other regarding the code content apart from the divided list of the search queries and API keys, only one Lambda function with complete list will be submitted in the code submission.

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. Since we can only obtain maximum 20 recipes with each call, all three Lambda Functions are set to be periodically run at 8-hour intervals with AWS EventBridge Service.

Moreover, RDS credentials on the code are encrypted in order to increase the security. First, a secret is defined for all RDS credentials by using AWS Secrets Manager service. Then, to allow Lambda function accessing this secret, a ‘read’ allowance policy of the secret is attached to the role of the Lambda function (robomaker_students) from AWS IAM service.

2.2.3 Coop

Beside finding food trends and matching with them suitable recipe, we would like to fetch more information about the ingredients such as the prices which then gives the user of our product more insight about the cost of each recipe.

As one of the largest retail and wholesale companies in Switzerland, Coop suits best as the source for getting products data. Unfortunately, Coop does not provide any public API for fetching product data. As a work-around, we decided to get these data via using web scrapping tools.

Analyzing the Coop website structure, we saw that the food products are categorized into multiple sub-groups, and we can directly filter them by specific URLs. We used this to scrap each sub-category

individually which helped the scrapping process and gave us more control to scale the data amount. See **Fehler! Verweisquelle konnte nicht gefunden werden.** for more insights.

There was no data cleaning necessary, because while doing the web scrapping, we have limited to a minimum of fields which needed to be scrapped such as the product name, price, and promotions. As the main goal of these data is to compute the cost for a recipe, there is no further data fetching or cleaning needed.

2.3 Data Sources Definitions

2.3.1 Google Trends

Description

“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].

Pytrends 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.

Pytrends Methods

Pytrends build_payload method is used to specify the search terms. Then, we used three pytrends methods, namely interest_over_time, related_queries, and interest_by_region to fetch the relative trends about these search queries.

interest_over_time returns a dataframe with the time frame as index and normalized measures for the specified time frame as column. It also creates a column “isPartial”, containing binary values, informs us whether the normalized values are estimates (1) or not (0). We excluded this column in our final data frame as there were a large majority of 0 and because the column was not relevant for us.

related_queries returns a Python dictionary with all the top and rising queries that are related to the search term and their normalized measures.

interest_by_region returns a data frame with the specified regions as index, the search query names as columns and their normalized measures as values. We defined the level “region” to get the measures for all the cantons of Switzerland. We also specified that the function should return Google Trends data for low volume regions (inc_low_vol = True) and excluded the ISO code of the Swiss cantons (inc_geo_code = False).

Limitations

It is possible to input a list of search queries into the build_payload function, however we cannot write more than 5 elements in the list. For the queries that require more than 5 elements, we had to merge the resulting data frames into one final data frame.

2.3.2 Edamam

Description

Edamam is a website related to better eating that aims to capture the World's food knowledge and distills it to help users make informed choices at the store and in the kitchen [Edamam 2022]. Using the website, it is possible to analyze 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 site has three possible API's, Nutritional Analysis, Food Database and Recipe Search. For this study, we use Recipe Search API.

Input Parameters

Input parameters to make a call to the API are:

- **type** (mandatory): type of recipes to search for: public
- **q** (mandatory): query text: chicken (for example)
- **app_id** (mandatory): the application id: obtained from the site with the subscription
- **app_key** (mandatory): the application key: obtained from the site with the subscription
- **random** (optional): select whether you want this query to respond with a random selection of 20 recipes based on the criteria filled: true

Header

Headers are block of information that you can get for each API call, about the request/response body, authorization, caching, or cookies of the API. They are useful to help tracking down the cause of any potential issues with the call, which may be related with Content-Type, Transfer-Encoding or Date [Apipheny 2022]. An example header of a call to Edamam can be seen in Appendix E.

Output

With each call to the API, we obtain 20 or less recipes related to the search query. By setting the optional parameter “random” to “true”, variability in the response is granted. If we didn’t set it so, the API returns same recipes in each call. As a response, we obtain image, label, URL, Edamam URL, diet labels, health labels and ingredient list for each of the recipe. Then, we allocated each output to different variables to be able to transfer them into a tabular form (**Fehler! Verweisquelle konnte nicht gefunden werden.**).

Limitations

Edamam – Recipe Search API, Developer version allows up to 10’000 calls per month and 10 calls/min.

2.3.3 Coop

Description

To scrap from the Coop product pages, we used the python packages named “selenium” and “webdriver manager”.

So, by combining both packages mention above, 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.

Settings for web scrapping

We have set some basic settings to prevent any error while fetching data. Some of these settings are specifically set for the Google Chrome Browser.

One Setting included to not visually show the browser while scrapping. With that, the whole process can be done in background and prevents any intervention by any users.

For more insights about all the settings, see **Fehler! Verweisquelle konnte nicht gefunden werden.**

Extraction of data

In our favor, we recognized that within the meta container of the HTML body, there is a JSON body with all the products which has been loaded from the Coop backend (**Fehler! Verweisquelle konnte nicht gefunden werden.**). So instead of using the actual HTML body, we used the raw data which the Coop frontend received from the backend. With that, we reduced the workload and reduced the risks of faulty scrapping.

2.4 Data Imputation and Data Cleaning per Data Source

2.4.1 Google Trends

We fetched data every Sunday evening, at 17h30 (15h30 UTC) to get the trends over the last 7 days.

We used three Lambda functions, namely `Fetch_Load_S3`, `Write_Data_Table` and `tables_data_validation`. `Fetch_Load_S3` fetched the data from Google Trends, created data frames and exported them into 6 CSV files. The latter ones were directly uploaded into the S3 bucket “datalakebucketdiet”. The files were named the same as their corresponding table. Each file name contained a timestamp which helped to track the creation date of the CSV files. `Write_Data_Table` extracted the most recent CSV file of each 6 categories and wrote their content into the corresponding data tables. `Tables_data_validation` created one validation table for each table to validate the number of new data that were written into the tables.

The flow diagram is shown in the Figure 2 below:

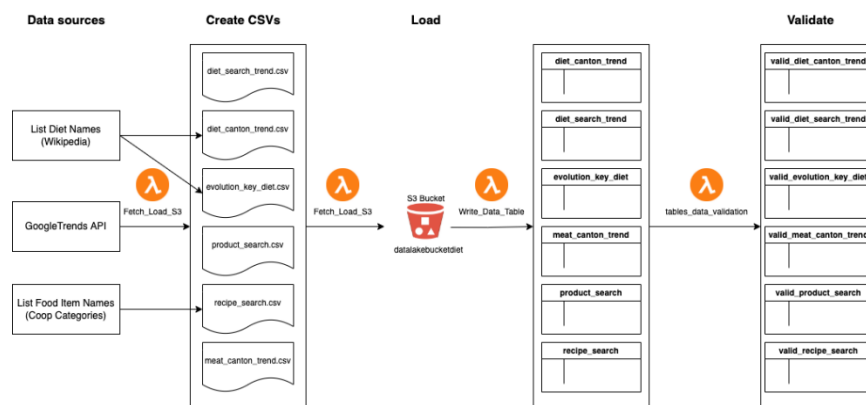


Figure 2 : Flow Diagram for Google Trends Data [own figure]

To create the Lambda function, we added a custom layer, created on Cloud9, containing the required Python libraries. The Lambda function `Fetch_Load_S3` used the layer `Layer_Fetch_Load_S3` and `Write_Data_Table` and `tables_data_validation` the layer `Write_Data_Table_Layer`. A screenshot of the creation of one Lambda Function can be found in the Appendix H.

All the Lambda functions were compatible with Python 3.7 and had a timeout of 14 minutes 59 seconds. `Fetch_Load_S3` and `Write_Data_Table` also connect to the S3 Bucket using the current AWS credentials, retrieved by means of the Session method of the “boto3” library [Stackoverflow 2016] and accessed the encrypted environment variable containing the name of the S3 Bucket. This solution allowed us to avoid displaying any credentials in the Python script. To create an encrypted environment variable [BiteSize Academy 2020], we first created a symmetric key on Amazon KMS, named `diet_key` and we added the execution role name of the Lambda function to the key usage permissions list to allow access to the key. Then, we created an environment variable for the S3 Bucket name and encrypted it with the created key. Therefore, the security was improved as the value of the encrypted environment variable can only be decrypted if the key is known.

For the `Write_Data_Table` and `tables_data_validation` Lambda Functions, we additionally created other encrypted environment variables for the database endpoint and database name, the username, and the password.

Fetch_Load_S3

The Lambda function contained various Python functions. Some automatically fetched the data and created data frames with an additional column containing the date when the data was extracted, while `load_df_into_s3` directly transformed each data frame into CSV files and uploaded them into the S3 Bucket. We explain below the creation of the data frames.

In order to discover the diets preferences for each speaking region of Switzerland, namely German, French and Italian, we saved the top related searched queries to *diet* in the three languages and their normalized measures into the data frame *diet_search_trend*. The column *region* will later allow us to group or sort by speaking area. A screenshot of a sample of the table's data can be found in the Appendix I.

Based on a list of famous diet names [Wikipedia 2022], we created a list with some of the most well-known diets. Since Google Trends API did not allow us to make queries with more than 5 keywords, we had to split the list into two list of 5 diet names, build two data frames separately, to finally merge them into one data frame. We also had to replace the space in the searched terms with underscores as the search terms would be used as column names in the PostgreSQL tables. We used these lists to build the data frames *diet_canton_trend* and *evolution_key_diet*.

Then, based on the food categories that Coop sells, we created lists of meaningful food items that would interest restaurant owners. We then fetched the related queries to these categories and input this data into the data frame *product_search*. A screenshot of the categories available at Coop and the corresponding list is available in the Appendix A. Again, since the Google Trends API does not allow us to make queries with more than 5 keywords, we decided to loop over each food category, and append the values into the final data frame. A screenshot of the code can be found in the Appendix J.

To create the data frame *recipe_search*, we fetched the weekly top related searched queries to *recipe*, for the 3 languages, and their normalized measures.

Finally, we created a list of 5 different meat types, namely beef, chicken, pork, seitan, tofu and weekly fetched the normalized measures, indicating the evolution of the preferences, for these meat groups for the canton of Switzerland. Each week, we saved the results into the data frame *meat_canton_trends*.

Using the scheduler service EventBridge, the data was fetched and uploaded into the S3 Bucket at 17h30 (15h30 UTC).

[Write_Data_Table](#)

The Lambda function was created to write the data from the most recent file of each CSV file category into the corresponding data table.

We first created a function, *connection_db*, which connected to the PostgreSQL database. To do so, it accessed the encrypted environment variables. A screenshot of the function can be found in the Appendix K. In order to directly write the data from a CSV file into the corresponding data table, we chose not to use the panda function *pandas.to_sql* as it did not enable enough flexibility to insert the data into the data tables. As we wanted to create 6 tables and repeat the same steps each time, we built a Python function, namely *insert_data*, which could upload any CSV files into the corresponding data table, without having to specify the column names [GeeksforGeeks 2022]. First, we created 6 empty data tables, named after the different CSV category file names. Then, we filled the tables each Sunday with the data from the most recent CSV files [SqlServerCentral 2021].

Using the scheduler service EventBridge, the data written into the tables every Sunday at 18h00 (16h00 UTC).

[tables_data_validation](#)

The Lambda function was created to verify and validate that the expected data was correctly fetched and written into the corresponding PostgreSQL data tables. Therefore, 6 validation tables were created to validate the content of their corresponding table. Each week, we verified that the *diet_canton_trend* and *meat_canton_trend* tables counted 26 cantons, *diet_search_trend* and *recipe_search* tables had 3 languages regions, *evolution_key_diet* table appended one row and that *product_search* table appended data for around 29 products. It was possible that certain weeks there were no related queries for some selected products.

Using the scheduler service EventBridge, data validations were performed every Sunday at 18h30pm (16h30 UTC).

2.4.2 Edamam

Every 8 hours, recipe data are fetched from Edamam - Recipe Search API using three Lambda functions (as explained in Chapter 2.2.2). Extracted data are transferred to pandas data frame, indexed with the search query and loaded to the recipes table created on the database. As an initial step of the loading, we created a PostgreSQL database with the name 'dledamam2' on AWS RDS service. Flow diagram of the Edamam data being transferred from the source to the database can be seen in Figure 3, below.

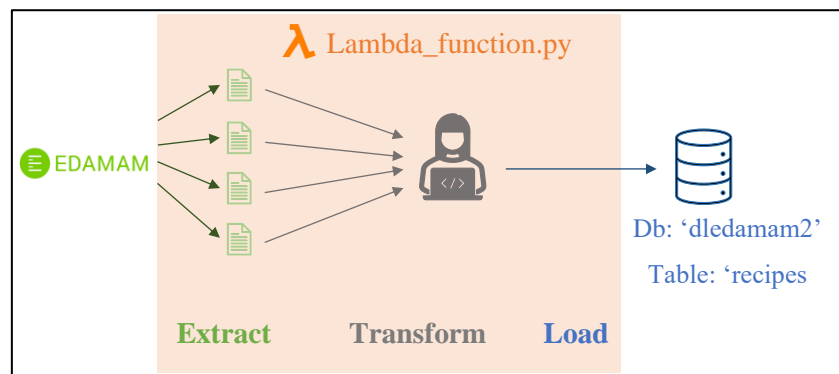


Figure 3 : Flow Diagram for Edamam Data [own figure]

Each recipe is transferred to the database after a two-stage try/except check. First, as we obtain (at most) 20 random results for each call, there is a possibility of getting duplicate responses from different calls. In the case of a duplicate recipe, python gives 'psycopg2.errors.UniqueViolation' error while loading the recipe to the database as the label of the recipe is the primary key of the table (Chapter 2.6.2). Thus, with a try/except statement, duplicate recipe is avoided, and the code continues with the next recipe (Appendix L). Second, there is also a possibility to obtain an empty response for a call. Although it is an empty response, search query index is added to the data frame and during the loading phase 'TypeError' occurs. This situation is also resolved by a try/except statement for the corresponding error type (Appendix L).

2.4.3 Coop

Data Transformation

The result of the scrapping has been saved as a JSON body. This JSON body has been parsed into a panda DataFrame.

As a DataFrame, we have selected the columns which we use for our project (product_name, price, promotion, source_url, etc.). With the source_url we were able to see the specific food category a food item is included. In the following example of a specific URL, you see that within the URL, you have also the categorization of it. With the help of string splitting, we were able to separately get these categories and save them in a separate column.

<https://www.coop.ch/en/food/fruit-vegetables/salad-vegetables/avocadoes/naturaplan-organic-avocado-1-piece/p/3465933?lang=en>

As you see in the URL above, each food tier can be retrieved directly from the product URL. This will help us to filter down product items and also to select the correct food item for a specific recipe. See **Fehler! Verweisquelle konnte nicht gefunden werden.** for further information.

Export and Upload Data

After the data transformation, we exported the pandas DataFrame as CSV. This means, that for each food category URL, we have created a csv file. As our list of food category contains 47 URL, each time our web scrapper runs, it will produce 47 csv files. Because we are scrapping multiple times to have an insights of price changes, we create for each run a separate export directory with the timestamp (e.g., 2022_04_03) and save all csv file inside this directory [**Fehler! Verweisquelle konnte nicht gefunden werden.**].

As soon as these CSV files has been created, a second process starts which then uploads the files to AWS S3 bucket. To have an overview of all scrapping sessions, we have implemented a function which creates a directory for each date and put all csv file inside this folder. See Appendix O for further information.

And at the end, a third script will load the latest CSV files into RDS. With that, in our database we are always working with the latest data. To prevent from duplicate products in the RDS, we have implemented a function which checks for each new row of data if this data already exists in the RDS. With that, we can make sure, that only the changes will be updated and not all data every time. See **Fehler! Verweisquelle konnte nicht gefunden werden.** for further insights.

Coop Scraper ETL Overview

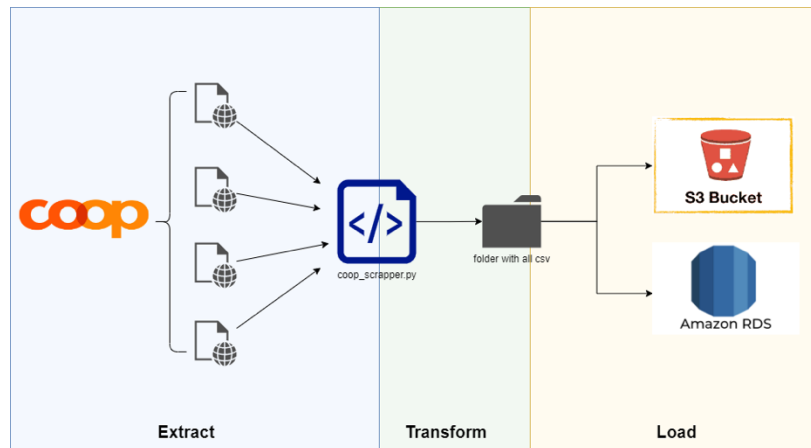


Figure 4 : Flow Diagram for Coop Data [own figure]

This ETL process will be triggered every Monday and Thursday because these days are known for new promotions and price changes of Coop products.

As our orchestrator tool we are using Apache Airflow, which contains 3 DAGs:

- 1) Extract and transform data (coop_scrapper.py)
- 2) Load files into AWS S3 (upload_s3_bucket.py)
- 3) Update data in RDS (upload_rds_postgres.py)

2.5 Data analysis per data source.

In this chapter we describe the extracted data which we fetched from each data source. The description corresponds to the situation of the 19th of April 2022. As we will continue fetching data from our sources, the number of data points will continue growing.

2.5.1 Google Trends

The database contains six tables. For each table, we created a validation table, which verifies that the data was correctly extracted and written in the suitable table. A screenshot of the validation tables can be found in the Appendix Q.

diet_canton_trend

The table has 104 rows and 12 columns.

```
# Content of the table
pd.read_sql_query('select * from "diet_canton_trend"',con=conn)
```

#	canton	low_carb	keto	fasting	detox	vegan	date_last7d	plant_based	planetary	clean	low_fat	mediterranean
0	Aargau	27	19	0	5	49	2022-03-29	0	33	67	0	0
1	Appenzell Auenroden	0	0	0	0	0	2022-03-29	0	0	0	0	0
2	Appenzell Auenroden	36	27	0	0	37	2022-03-29	0	0	0	0	0
3	Basel-Landschaft	29	19	0	4	48	2022-03-29	0	0	0	0	0
4	Basel-Stadt	22	8	0	4	66	2022-03-29	0	0	0	0	0
...
99	Thurgau	25	13	0	8	54	2022-04-17	0	0	0	0	0
100	Ticino	23	38	0	12	27	2022-04-17	0	0	0	0	0
101	Vaud	21	29	0	8	42	2022-04-17	0	0	0	0	0
102	Vaud	22	17	2	3	56	2022-04-17	0	0	0	0	0
103	Zurich	24	19	1	4	52	2022-04-17	20	0	40	40	0

Figure 5: Screenshot of the Table diet_canton_trend

As we can see in the Figure 5, the first column contains the names of the cantons of Switzerland. The data type is string. Each canton appears 4 times, as we fetched data over the last 4 weeks. The other columns, namely low_carb, keto, fasting, detox, vegan, plant_based, planetary, clean, low_fat and mediterranean correspond to popular diet names. Their values vary from 0 to 100, thus are integer type. As seen in the Figure 6, some of the diets, namely planetary, low_fat and mediterranean, contain almost only null values, meaning that there were not enough data to estimate the search trend for the given query. The column date_last7d contains data which type is date. It records the date when the data was fetched. It starts on the 29th of March 2022 and ends on the 17th of April 2022.

```
# number of 0 in each column
df = pd.read_sql_query('select * from "diet_canton_trend"',con=conn)
(df == 0).sum(axis = 0)
```

canton	0
low_carb	24
keto	18
fasting	95
detox	38
vegan	9
date_last7d	0
plant_based	97
planetary	102
clean	90
low_fat	100
mediterranean	101

Figure 6: Number of Null Values in the Table diet_canton_trend

The corresponding validation table, namely valid_diet_canton_trend, verifies that data was fetched each week for the 26 cantons.

diet_search_trend

The table contains 58 rows and 4 columns. A screenshot of the table can be found in the Appendix R. The first column contains the queries that each speaking region of Switzerland did on the given week. The type is string. The second column contains the normalized values of the search query. The type is therefore integer. Its values range from 2 to 100. The column region informs about the language region where such query was made. Its type is string. The last column records the date when the data was fetched, which corresponds to a date type. It goes from the 29th of March 2022 to the 17th of April 2022.

The corresponding validation table, namely valid_diet_search_trend, verifies that data was fetched each week for the 3 speaking regions.

evolution_key_diet

The table contains 54 rows and 11 columns. A screenshot of the table can be found in the Appendix S. Each row corresponds to one week, starting on the 4th of April 2021 to the 10th of April 2022. These dates are saved in the first column. The column type is therefore date. The 10 other columns correspond to the popular diet names that we mentioned above, and their type is integer. Their values range from 0 to 100. The number of null values is lower than for the diet_canton_trend table as we can see on the Figure 7.

The corresponding validation table, namely valid_evolution_key_diet, verifies that there is only one record for each week.

meat_canton_trend

The table contains 104 rows and 7 columns. A screenshot of the table can be found in the Appendix T. The first column contains the name of the cantons of Switzerland. Again, each canton appears 4 times, as we

fetches data over the last 4 weeks. The next 5 columns correspond to 5 meat types. Each column contains the normalized measures for a given canton in a given week. Therefore, their type is integer. The last column records the date when the data was fetched, which corresponds to a date type. It goes from the 29th of March 2022 to the 17th of April 2022.

The corresponding validation table, namely `valid_meat_canton_trend`, verifies that data was fetched each week for the 26 cantons.

date	0
low_carb	0
keto	1
fasting	44
detox	9
vegan	0
plant_based	49
planetary	50
clean	51
low_fat	34
mediterranean	52

[product_search](#)

The table contains 1712 rows and 4 columns. A screenshot of the table's content can be found in the Appendix U. The first column query contains the query that was made. It has a string data type. The second column value contains the normalized values of the search term, and its data type is integer. They range from 8 to 100. The third column `data_last7d` records the date when the data was fetched, which corresponds to a date type. It goes from the 29th of March 2022 to the 17th of April 2022. The last column `product`, whose data type is string, contains 29 different product names, such as banana, pesto, beef. Melon contains only 3 related search queries whereas pesto, chicken, mousse and cake each recorded 100 related queries over the last 4 weeks.

The corresponding validation table, namely `valid_product_search`, counts the number of distinct products that had at least one related query on that given week. Ideally, there should be 29 products each week.

Figure 7: Number of Null Values in the Table
`evolution_key_diet`

[recipe_search](#)

The table contains 300 rows and 4 columns. A screenshot of the table can be found in the Appendix V. The first column records the query that was made on the specific week in the given speaking region. The type is string. The next column contains the related normalized value of the search query. Therefore, the type is integer. It ranges from 11 to 100. The third column `region` informs about the speaking region where the query was made. Its type is string. The last column contains the date when the data was fetched. The type of this column is date. It goes from the 29th of March 2022 to the 17th of April 2022.

The corresponding validation table, namely `valid_recipe_search`, verifies that data was fetched each week for the 3 speaking regions.

2.5.2 Edamam

Recipes table contain 8 columns with the information that can be seen in Table 1. As of 22nd April 2022, there are 19'255 recipes (rows) in total and recipe counts corresponding to each search query can be seen in Table 2.

Table 1: Details of 'recipes' Table

Column Name	Description	Non-value Count	Data Type
Label	Name of the recipe	0	String
Search_Query	Query text	0	String
Image	URL of the image of the recipe	0	String
Url	URL of the original recipe	0	String
Edamam_url	URL of the recipe on Edamam	0	String
Diets_Labels	List of the diet names that the recipe belongs to (e.g. 'high-fiber', 'low-fat')	6846	String
Health_Labels	List of the health labels that the recipe belongs to (e.g. 'vegan', 'vegetarian')	16	String
Ingredients	List of the ingredients	0	String

Table 2: Number of Recipes for Each Search Query

Search Query	Number of Recipes	Search Query	Number of Recipes	Search Query	Number of Recipes
Apple	1121	Aubergine	545	Chicken	951
Banana	1072	Courgette	745	Pork	925
Mango	1021	Chilli	1009	Seitan	223
Melon	759	Garlic	1043	Tofu	880
Strawberry	315	Quark	164	Pasta	700
Avocado	1097	Flan	31	Cake	306
Broccoli	1113	Mousse	56	Soup	259
Endive	743	Pudding	391	Bolognese	21
Tomato	1126	Fondant	122	Pesto	450
Potato	1139	Beef	928		

2.5.3 Coop

For each scrapping session there has been fetched approximately 8'241 products. For each product we have the following attributes:

Table 3 : Coop Product

Column name	Description	Non-value count	Data type
Title	Name of the product	0	String
Href	URL of the product	0	String
Quantity	Quantity of product	16	String
Rating	Rating given by customer	0	Float64
Price	Actual price	1	Float64
oldPrice	Price before promotion	7963	Float64
savingText	Promotion text (e.g., "2 for 1")	7731	String
Category_1	Tier 1 Category	0	String
Category_2	Tier 2 Category	0	String
Category_3	Tier 3 Category	0	String

According to this data above from April, the 21st 2022, we can see that there were totally 278 promotions on product on coop. Also, you can see that the splitting the URL into the 3 Food Categories works well without any errors.

So, in total we have 3 number type and 7 string type in our coop dataset.

2.6 Database Schema

In this section, we describe the schema of the three databases, one for each data source, that we created with the extracted data.

2.6.1 GoogleTrends

The Entity Relationship Diagram is presented below. As each table was created to contain specific information, there are no foreign keys.

Nevertheless, each row in each table can be uniquely identified with:

1. the combination of query and date_last7d for the tables diet_search_trend, product_search and recipe_search.
2. the combination of canton and date_last7d for the tables diet_canton_trend and meat_canton_trend

3. the date for the table evolution_key_diet

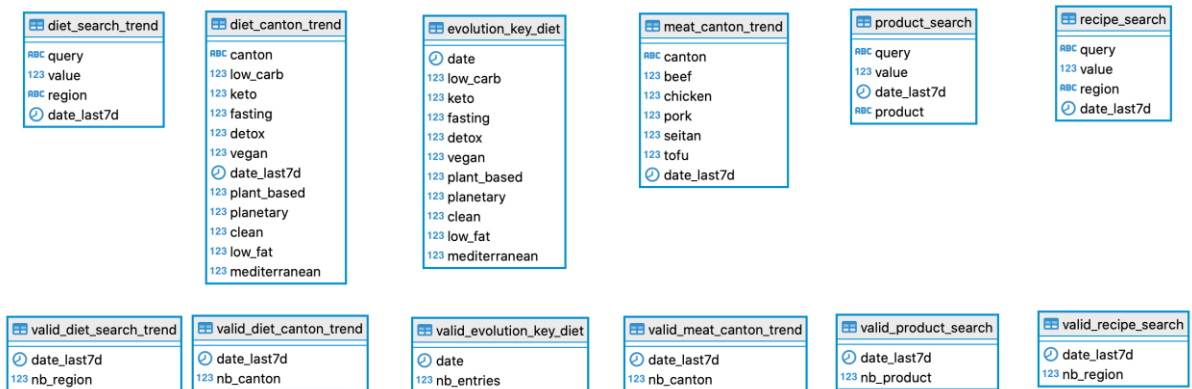


Figure 8: Entity Relationship Diagram for Google Trends API

2.6.2 Edamam

The Entity Relationship Diagram of ‘dledamam2’ database is shown Figure 9, below. From the image, it can be seen that the ‘label’ column, which contains the name of the recipe, is the primary key of the table. That means, it is unique and cannot be null.

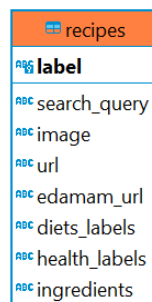


Figure 9: Entity Relationship Diagram for Edamam API

In the following steps of the project, we are aiming to combine ‘recipes’ table with other tables we extracted from Google Trends, on mainly search query column as well as specific diet types and health labels. Moreover, combination with the Coop data will be done over ingredients list.

2.6.3 Coop

The ER Diagram of the database “coop” shown in Figure 10 below, indicates that this database is very simple and just one layered. The challenge is to join product items with each recipe. This join will happen in the next part of the project, as for now fetching data and storing them separate is easier.

Product
123 index
ABC title
ABC href
ABC quantity
123 rating
123 price
123 oldPrice
ABC savingText
ABC category_1
ABC category_2
ABC category_3

Figure 10: Entity Relationship Diagram for Coop Products

2.7 Data Lake Consumers

The consumer of our data lake will be in the first instance the restaurant owner. By means of a visualized form such as a Tableau dashboard, the restaurant owner will see the latest food trends in Switzerland and get suggestions for recipes with an approximated cost.

A restaurant owner is someone who keeps the restaurant business running, dealing with suppliers and staff and not time resources to think about the marketing and the needs of consumers. His motivation is to keep the business successful and get positive feedback from consumers. It is sometimes very difficult to find helpful feedbacks and often there is not much time for deepened analyses about the needs of the consumers.

Especially for restaurant owners which are thinking about opening Pop-up Stores in different cities or would like to adapt their menu dishes accordingly to the needs of the restaurant customer, this tool will help a lot.

3 Discussions

In this section we will discuss the implementation of our project solution, the advantages and disadvantages of our design choices and the trade-offs that we made.

3.1 Google Trends

3.1.1 Quality of the data

As mentioned earlier, we filtered the search results on two categories, namely 122 and 457. This helped us to minimize the noise in our data and fetch only information related to food and diets. However, we still ended up with some noisy datapoints, such as *mango bar* or *mango lausanne* which is probably due to how the related_queries algorithm works. It might not know that these queries should be labelled to a business category rather than a food one. Therefore, further data cleaning will be required in the next part of our project.

3.1.2 Lists of Search Terms

Pytrends' methods allow us to input a list of five search terms maximum per call. Consequently, for each diet_search_trend and product_search query, we had to split the keywords into lists of five keywords and append the results into a single data frame, which was then saved into the corresponding CSV file. This process was obviously not optimal and required to loop over the lists, store the results into variables, and merge data frames into a unique data frame. As there were not many related queries our solution was still suitable. Nevertheless, it was rather slow. Indeed, for the product_search query, we had to split the list of food products into seven lists and create a list of these lists (Appendix J). To avoid any problems related to the number of queries we made to the Google Trends API, we also decided to add a sleeping time of 10 seconds between the queries, which slowed down the fetching process. Therefore, the "Extract" step should be improved in the future.

3.1.3 Languages

Our solution to differentiate the speaking regions of Switzerland and collect data was helpful. Nevertheless, even though `pytrends` allows to set the preferred language (hl), it would have been better to have an option to input only one word, namely *diet* or *recipe*, and use the `pytrends` method to apply our created function with corresponding translated word in German, French and Italian. We would not have had to create a list of the same search term translated into the three languages. The preferred language argument yet does not allow us to do so.

3.1.4 Write Data from a Data Frame

The function that allowed us to directly write the data from a data frame into a data table without having to specify the column names was optimal in terms of the portability of the code. However, we had to replace all spaces in the search terms by underscores before creating the CSV file. Indeed, it was not possible to have column names containing spaces as they would also become the column names in the PostgreSQL tables. This additional step was the only downside of our solution.

3.1.5 Validation Tables

This method should be improved since it fully recreated the validation tables each Sunday. Over time and with more data this solution is not optimal.

3.2 Edamam

The data we extracted from Edamam API was already clean and by applying filters to avoid irrelevant, duplicate and empty recipes we enhanced the data quality on our database. In the first part of the project, we only included search queries related with the ingredients. As an improvement, it is possible to include calling the recipes directly with a specific kind of diet name or health label from the API.

3.3 Coop

As for now, we have implemented a way to use Apache Airflow as our orchestrator tool to trigger the ETL process for Coop data. This is done locally on our personal computer. This should be improved by trying to get this on the AWS environment. With that, the data fetching is consistent and there is no need to be physically attended to run the ETL process. As an inspiration from the presentation of the other teams, we saw that there is a way to use the AWS service EC2 to run a Ubuntu or an Windows instance and run Apache Airflow directly over this system.

4 Conclusions

Considering business point of view, with this study, we aim to create a recommendation tool based on actual data, which will offer trendy menu options for Pop-Up restaurant owners to enable them enhancing their business. Having huge amount of data available does not mean that it is useful by itself. In order to be able to make profitable deductions, data from various sources have to be combined, cleaned, filtered and represented in a user-friendly way. With our final product, a restaurant owner is going to be able to see the trendy ingredients and diets, changes in these trends over time, various new menu options related with these and overall cost of them. Thus, they will have all the information needed to go through a menu improvement. In addition, they can judge whether it is economically worth the change in the menu also by considering their other resources such as cooks and teams.

Considering technical point of view, until this point, we have completed the most important part of the project which is constructing the data lakes. From now on, we will mostly continue by creating our data warehouse and filtering data to answer our research questions.

As of 23rd April 2022, the credit conditions on AWS accounts can be seen for each data source in **Fehler! Verweisquelle konnte nicht gefunden werden.**

Table 4: AWS Credit Conditions for Each Data Source

Source	AWS Credit Condition
Google Trends	Used \$27.2 of \$100
Edamam	Used \$12.7 of \$100
Coop	Used \$31.9 of \$100

5 Future Work

As for now, if a certain food item is trendy, the results from the recipe trends data source varies a lot. An example would be if you say the trendy food item is mango, then the related recipe queries also contain very similar ingredients like “Mango Bar”, which is not a recipe. So, we need to find a way to make sure that the recipe which we are fetching contains the trendy food item as it is and no other procced or subproducts of it.

Another issue is that Coop provides many different products for certain food items. An example, there are many products with different labels like “Bio” or “Fairtrade” for Tomatoes. These products have different prices which makes it difficult to match ingredients with the correct product. A way to solve that would be to define that the cost which will be associate with the recipe should be only an approximation. Therefore, we would take the average price of all tomato products.

Finally, we do not yet search for specific diet type (e.g., vegan, vegetarian) in Edamam API. This would add some more diverse offering in recipes.

6 Appendix

Appendix A: Example of the Food Categories Available on the Coop website and Our Corresponding List

Food	Wine	Household & Pet	Cosmetics & Health	Lounge	Inspiration & Gifts	Promotions
Fruit & Vegetables		Dairy Products & Eggs	Meat & Fish	Bread & Baked Goods	Inventories	Sweets & Snacks
Special Offers						
Organic Fruit & Vegetable			Salads	Root Vegetables		More Vegetables
Fruit			Leaf Lettuce	Potatoes		Courgettes, Pumpkin & Aubergines
Apples & Pears			Pre-Packaged Fresh Salads	Carrots & Parsnips		Sweetcorn, Peas & Beans
Grapes & Berries				Celeriac & Kohlrabi		Leek & Fennel
More Garden Fruit			Salad Vegetables	Beetroots		Leaf Vegetables & Artichokes
Bananas			Tomatoes	Cassava		Mushrooms
Kiwi & Melons			Radishes			Asparagus
Pineapple & Mangoes			Cucumbers	Cabbage		Prepared Vegetables
Citrus Fruit			Peppers	Broccoli & Cauliflower		
More Exotic Fruit			Avocados	Brussel Sprouts & Cabbage		Herbs & Spices
Nuts, dates and figs			Fruit & Vegetable Baskets	Sauerkraut & Red Cabbage		Onions & Garlic
				Plant based alternatives to meat		Fresh Herbs
						Sprouts & Watercress
						Chilli, Lemongrass & Ginger

```
# 4) product_search
# Based on the list of categories in Coop
fruit = ['apple', 'banana', 'mango', 'melon', 'strawberry']
salads_veggie = ['avocado', 'broccoli', 'endive', 'tomato', 'potato']
more_veg = ['aubergine', 'courgette']
herbs_sprices = ['chilli', 'garlic']
creme_dessert = ['quark', 'flan', 'mousse', 'pudding', 'fondant']
meat = ['beef', 'chicken', 'pork', 'seitan', 'tofu']
pasta_cake_soup_sauce = ['pasta', 'cake', 'soup', 'bolognese', 'pesto']


# List of these lists
all_food = [fruit, salads_veggie, more_veg, herbs_sprices, creme_dessert, meat, pasta_cake_soup_sauce]
```

Appendix B: ‘Try Also’ column at the right-hand Side Bar of a Recipe

EDAMAM

tofu

REFINE SEARCH BY Calories, Diet, Ingredients



Zucchini-Tofu Croquettes

See full recipe on: [Big Girls Small Kitchen](#)

Save

EMAIL PIN IT SHARE TWEET GOOGLE+

7 Ingredients


1 large zucchini

Nutrition

77 4% 8

CALORIES / SERVING DAILY VALUE SERVINGS

Try also



Crispy Tofu and Rice Noodles in Red Curry Coconut Sauce

200 CALORIES 9 INGREDIENTS

tastykitchen.com

Appendix C: Allocation of the JSON Response to Separate Variables

```
for entry in recipes:
    entry_items = {}
    # Check whether the search query is actually within the ingredient list
    check = [True for el in entry['recipe']['ingredientLines'] if search_string in el]
    if check:
        entry_items['Image'] = entry['recipe']['image']
        entry_items['Label'] = entry['recipe']['label']
        entry_items['Url'] = entry['recipe']['url']
        entry_items['Edamam_url'] = entry['recipe']['shareAs']
        entry_items['Diets_Labels'] = entry['recipe']['dietLabels']
        entry_items['Health_Labels'] = entry['recipe']['healthLabels']
        entry_items['Ingredients'] = entry['recipe']['ingredientLines']

        all_recipe[entry_items['Label']] = entry_items

return all_recipe
```

Appendix D: List of URLs for Each Coop Food Category

```
{
  "url":
  [
    "https://www.coop.ch/en/food/fruit-vegetables/fruit/c/m_0002?page=1&pageSize=100&q=%3Aname-asc&sort=name-asc",
    "https://www.coop.ch/en/food/fruit-vegetables/root-vegetables/c/m_0016?q=%3AmostBought&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/fruit-vegetables/cabbage/c/m_0022?q=%3AmostBought&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/fruit-vegetables/fresh-salad-vegetables/c/m_0028?q=%3Aname-asc&sort=name-asc&pageSize=100",
    "https://www.coop.ch/en/food/fruit-vegetables/salads/c/m_0050?q=%3AmostBought&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/fruit-vegetables/more-vegetables/c/m_0034?q=%3AmostBought&sort=topRated&pageSize=100&page=1",
    "https://www.coop.ch/en/food/fruit-vegetables/fresh-herbs-spices/c/m_0044?q=%3AmostBought&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/dairy-products-eggs/cheese-counter/c/m_1909?q=%3AtopRated&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/dairy-products-eggs/milk/c/m_0056?q=%3AtopRated&sort=name-asc&pageSize=300&page=1",
    "https://www.coop.ch/en/food/dairy-products-eggs/cream/c/m_0061?q=%3AtopRated&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/dairy-products-eggs/butter-margarine/c/m_0084?q=%3AtopRated&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/dairy-products-eggs/yogurt/c/m_0066?q=%3AtopRated&sort=name-asc&pageSize=400&page=1",
    "https://www.coop.ch/en/food/dairy-products-eggs/quark/c/m_0071?q=%3AtopRated&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/dairy-products-eggs/desserts-cream/c/m_0074?q=%3AtopRated&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/dairy-products-eggs/eggs/c/m_0083?q=%3AtopRated&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/meat-fish/meat-from-the-butcher-shop/c/m_2333?q=%3AtopRated&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/meat-fish/fish-from-the-seafood-market/c/m_1893?q=%3AtopRated&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/meat-fish/packaged-fresh-meat/c/m_0088?q=%3AtopRated&sort=name-asc&pageSize=300&page=1",
    "https://www.coop.ch/en/food/meat-fish/cold-cuts-sausage-products/c/m_0097?q=%3AtopRated&sort=name-asc&pageSize=100&page=1",
    "https://www.coop.ch/en/food/meat-fish/packaged-fish/c/m_0107?q=%3AtopRated&sort=name-asc&pageSize=200&page=1",
    "https://www.coop.ch/en/food/meat-fish/ready-to-cook-meat-fish/c/m_0599?q=%3AtopRated&sort=name-asc&pageSize=200&page=1",
    "https://www.coop.ch/en/food/bread-baked-goods/bread-from-the-bakery/c/m_0119?q=%3AtopRated&sort=name-asc&pageSize=100&page=1",
  ]
}
```

Appendix E: Header from a Call to Edamam API

ly Cookies Headers (12) Test Results		200 OK 975 ms 394.83 K
KEY	VALUE	
Server	openresty	
Date	Fri, 15 Apr 2022 21:31:28 GMT	
Content-Type	application/json	
Transfer-Encoding	chunked	
Connection	keep-alive	
Cache-Control	private	
Expires	Thu, 01 Jan 1970 00:00:00 GMT	
X-Served-By	ip-10-0-1-153.ec2.internal/10.0.1.153	
vary	accept-encoding	
Content-Encoding	gzip	
Strict-Transport-Security	max-age=15552001	
X-Request-ID	84da5502172fea14dfbd635bdd62c83f	

Appendix F: Code Snippet of the Web Scraper Settings

```
def web_scrapper(url: str = None) -> BeautifulSoup:
    # Chrome Settings
    chrome_option = Options()
    chrome_option.add_argument("--headless") # background task; don't open a window
    chrome_option.headless = True
    chrome_option.add_argument('--disable-gpu')
    chrome_option.add_argument('--no-sandbox') # I copied this, so IDK?
    chrome_option.add_argument('--disable-dev-shm-usage') # this too
    chrome_option.add_argument('--disable-extensions')
    # Inititalize Chrome Service and the webdriver
    chrome_service = Service(ChromeDriverManager(print_first_line=False, log_level=logging.NOTSET).install())
    chrome_service.SuppressInitialDiagnosticInformation = True
    driver = webdriver.Chrome(service=chrome_service, options=chrome_option)
    # Scrap html body
    driver.get(url)
    html = driver.page_source
    # Convert to a beautiful soup object
    return BeautifulSoup(html, "lxml")
```

Appendix G: Raw Data Inside HTML

The screenshot shows a web browser displaying a page titled "Fruit & Vegetables" with a "Fruit" section. The page lists various fruits and vegetables, including Apples & Pears, Grapes & Berries, More Garden Fruit, Bananas, Kiwi & Melons, Pineapple & Mangoes, Citrus Fruit, and More Exotic Fruit. The page also features a "Filter" button and a "Sort" button. The DevTools console is open, showing the raw HTML data for the page, including the product details for "Abate Fetel pears ca. 1kg" and "Äpfel Envy I 1kg". The HTML data is displayed in a structured format, showing the product name, price, and other details.

Appendix H: Creation of a Lambda Layer for Google Trends

Layer configuration

Name

Layer_Fetch_Load_S3

Description - optional

To fetch and load data into S3 as CSVs

☐ Upload a .zip file
 ☒ Upload a file from Amazon S3

Amazon S3 link URL

Paste an S3 link URL to your function code .zip.

https://datalakebucketdiet.s3.amazonaws.com/fetch_load_layer.zip

Compatible architectures - optional [Info](#)

Choose the compatible instruction set architectures for your layer.

☐ x86_64
 ☐ arm64

Compatible runtimes - optional [Info](#)

Choose up to 15 runtimes.

Runtimes

Python 3.7 X

Appendix I: Data Sample in the diet_search_trend Table

query	value	region	date_last7d
keto diät	100	G	2022-03-29
ketogene diät	60	G	2022-03-29
low carb diät	22	G	2022-03-29
keto diät rezepte	13	G	2022-03-29
keto rezepte	13	G	2022-03-29
was ist keto diät	10	G	2022-03-29
eier diät	7	G	2022-03-29
ketogene rezepte	7	G	2022-03-29
diät shake	7	G	2022-03-29
atkins diät	6	G	2022-03-29
mediterrane diät	5	G	2022-03-29
keto diät plan	4	G	2022-03-29
carnivore diet	100	F	2022-03-29
dukan diet	91	F	2022-03-29
planetary health diet	68	F	2022-03-29
dieta keto	100	I	2022-03-29
dieta cetogenica	49	I	2022-03-29

Appendix J: Looping over All Product Categories for the product_search Table

```
# 4) product_search
# Based on the list of categories in Coop
fruit = ['apple', 'banana', 'mango', 'melon', 'strawberry']
salads_veggie = ['avocado', 'broccoli', 'endive', 'tomato', 'potato']
more_veg = ['aubergine', 'courgette']
herbs_sprices = ['chilli', 'garlic']
creme_dessert = ['quark', 'flan', 'mousse', 'pudding', 'fondant']
meat = ['beef', 'chicken', 'pork', 'seitan', 'tofu']
pasta_cake_soup_sauce = ['pasta', 'cake', 'soup', 'bolognese', 'pesto']

# List of these lists
all_food = [fruit, salads_veggie, more_veg, herbs_sprices, creme_dessert, meat, pasta_cake_soup_sauce]

# Create dataframe
df_all_products = pd.DataFrame(columns = ['query', 'value', 'date_last7d', 'product'])

for sublist in all_food:
    pytrend.build_payload(kw_list= sublist, cat = 122, geo = 'CH', timeframe= 'now 7-d')

    print('Begin:', sublist)

    for product in sublist:
        time.sleep(10) # to avoid being blacklisted by GoogleTrends
        try:
            print(product)
            df_prod = pytrend.related_queries()[product]['top']
            df_prod['product'] = [product for n in range(len(df_prod))] # food item name in column "product"
            df_all_products = df_all_products.append(df_prod, ignore_index = True) #
        except:
            print(f'No queries for {product} this week!')
            pass

df_all_products['date_last7d'] = [date.today() for n in range(len(df_all_products))]
df_all_products
```

Appendix K: Function to Create a Connection to PostgreSQL Database

```
# Connection to the PostgreSQL database
def connection_db():
    try:
        conn = psycopg2.connect(
            f"host={DECRYPTED_HOST} dbname={DECRYPTED_DB} user={DECRYPTED_USR} password={DECRYPTED_PWD}")
        return conn
    except psycopg2.Error as err:
        print("Error: Could not make connection to the Postgres database")
        print(err)
```

Appendix L: try/except Solutions in the Loading Phase of Edamam Data

```
# Enter the values
sql_in = """
INSERT INTO recipes (Label, Search_Query, Image, Url, Edamam_url,
                    Diets_Labels, Health_Labels, Ingredients)
VALUES(%s,%s,%s,%s,%s,%s,%s,%s)
"""

for i in range(len(x1)):
    try:
        values = (x1.index[i], x1.iloc[i][0], x1.iloc[i][1], x1.iloc[i][2], x1.iloc[i][3],
                  x1.iloc[i][4], x1.iloc[i][5], x1.iloc[i][6])
        cur.execute(sql_in, values)
    except psycopg2.errors.UniqueViolation:
        # In case of duplicate recipes
        pass

except TypeError:
    # In case of empty response
    pass
```

Appendix M: Code Snippet for Coop Data Transformation

```
def extract_data(url: str = None) -> pd.DataFrame:
    soup: BeautifulSoup = web_scrapper(url=url)
    json_body = json.loads(soup.find_all("meta", attrs={"data-pagecontent-json": True})[0]["data-pagecontent-json"])
    for index in range(len(json_body['anchors'])):
        if json_body['anchors'][index]['name'] == "productTile-new":
            product_data = json_body['anchors'][index]['json']['elements']

    if not product_data:
        raise ("No product data found")
    df = pd.read_json(StringIO(json.dumps(product_data)))
    if 'saving' not in df.columns:
        df['saving'] = np.nan
    if 'savingText' not in df.columns:
        df['savingText'] = np.nan

    selected_columns = ["title", "href", "quantity", "ratingValue", "price", "saving", "savingText"]

    df = df[selected_columns]

    # renaming
    if 'saving' in df.columns:
        df.rename(columns={'saving': 'oldPrice'}, inplace=True)
    if 'ratingValue' in df.columns:
        df.rename(columns={'ratingValue': 'rating'}, inplace=True)

    df['category_1'] = df['href'].apply(lambda row: row.split("/")[3])
    df['category_2'] = df['href'].apply(lambda row: row.split("/")[4])
    df['category_3'] = df['href'].apply(lambda row: row.split("/")[5])

    return df
```

Appendix N: Your caption name

Appendix O: Specific Function to Create Folder on S3 Accordingly

```
def create_folder_on_s3(bucket_object, bucket_name: str) -> str:
    today = datetime.now()
    month = today.strftime("%m")
    day = today.strftime("%d")
    folder_name = f"{today.year}_{month}_{day}"
    try:
        bucket_object.put_object(Bucket=bucket_name, Key=(folder_name + '/'))
        return folder_name
    except Exception as e:
        logging.error("Something went wrong while creating folder on S3")
        logging.error(e)
```

Appendix P: Optimized Way to Update Database

```
def upload_dataframe_to_table(table_name: str, df: pd.DataFrame, username: str = "master",
                             password: str = "master1234"):
    engine = create_engine(
        f'postgresql://{username}:{password}@coop.ca7k4yfxv4gc.us-east-1.rds.amazonaws.com:5432/coop_product')
    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)
            # 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")
            logging.info(f"Connetion Status: {bool(conn)}")

            df.to_sql(name=f'{table_name}', con=engine, if_exists='append')
```

Appendix Q: Screenshot of Validation tables' Content

valid_diet_canton_trend	valid_diet_search_trend	valid_evolution_key_diet
date_last7d nb_canton	date_last7d nb_region	date nb_entries
2022-04-17 26	2022-04-17 3	2022-04-10 1
2022-04-10 26	2022-04-10 3	2022-04-03 1
2022-04-03 26	2022-04-03 3	2022-03-27 1
2022-03-29 26	2022-03-29 3	2022-03-20 1
		2022-03-13 1
		2022-03-06 1

valid_recipe_search	valid_product_search	valid_meat_canton_trend
date_last7d nb_region	date_last7d nb_product	date_last7d nb_canton
2022-04-17 3	2022-04-17 29	2022-04-17 26
2022-04-10 3	2022-04-10 28	2022-04-10 26
2022-04-03 3	2022-04-03 29	2022-04-03 26
2022-03-29 3	2022-03-29 29	2022-03-29 26

Appendix R: Screenshot of diet_search_trend Table's Content

```
pd.read_sql_query('select * from "diet_search_trend"',con=conn)
```

⚡	query ⚡	value ⚡	region ⚡	date_last7d ⚡
0	keto diät	100	G	2022-03-29
1	ketogene diät	60	G	2022-03-29
2	low carb diät	22	G	2022-03-29
3	keto diät rezepte	13	G	2022-03-29
4	keto rezepte	13	G	2022-03-29
5	was ist keto diät	10	G	2022-03-29
6	eier diät	7	G	2022-03-29
7	ketogene rezepte	7	G	2022-03-29
8	diät shake	7	G	2022-03-29
9	atkins diät	6	G	2022-03-29
10	mediterrane diät	5	G	2022-03-29
11	keto diät plan	4	G	2022-03-29
12	carnivore diet	100	F	2022-03-29
13	dukan diet	91	F	2022-03-29
14	planetary health diet	68	F	2022-03-29
15	dieta keto	100	I	2022-03-29
16	dieta cetogenica	49	I	2022-03-29
17	keto diät	100	G	2022-04-03
18	ketogene diät	49	G	2022-04-03

Appendix S: Screenshot evolution_key_diet Table's Content

```
pd.read_sql_query('select * from "evolution_key_diet"',con=conn)
```

⚡	date ⚡	low_carb ⚡	keto ⚡	fasting ⚡	detox ⚡	vegan ⚡	plant_based ⚡	planetary ⚡	clean ⚡	low_fat ⚡	mediterranean ⚡
0	2022-03-27	43	17	0	17	39	0	0	0	47	0
1	2021-04-04	62	47	0	9	100	0	0	0	0	0
2	2021-04-11	69	33	3	0	74	0	0	0	30	0
3	2021-04-18	42	31	0	3	65	31	0	0	0	0
4	2021-04-25	49	36	3	3	74	30	0	0	0	0
5	2021-05-02	56	19	3	3	61	0	0	0	0	0
6	2021-05-09	54	40	0	6	71	0	0	0	31	0
7	2021-05-16	52	35	3	14	74	0	0	30	0	0
8	2021-05-23	35	35	0	6	46	0	0	0	31	0
9	2021-05-30	71	34	0	6	54	0	0	0	31	0
10	2021-06-06	45	25	0	8	36	0	0	0	0	30
11	2021-06-13	26	17	0	11	49	0	0	0	0	0
12	2021-06-20	25	17	0	3	42	0	0	0	30	0
13	2021-06-27	23	34	0	3	68	0	0	0	31	0
14	2021-07-04	35	18	0	3	56	32	32	0	32	0
15	2021-07-11	36	18	0	6	66	0	0	33	33	0
16	2021-07-18	44	14	7	0	48	0	0	0	0	0
17	2021-07-25	42	26	3	3	49	0	35	0	0	0
18	2021-08-01	49	23	0	3	55	0	0	0	0	35

Appendix T: Screenshot meat_canton_trend Table's Content

```
pd.read_sql_query('select * from "meat_canton_trend"',con=conn)
```

	canton	beef	chicken	pork	seitan	tofu	date_last7d
0	Aargau	21	44	11	4	20	2022-03-29
1	Appenzell Innerrhoden	0	0	0	0	0	2022-03-29
2	Appenzell Outer Rhodes	14	39	0	0	47	2022-03-29
3	Basel-Landschaft	18	52	8	1	21	2022-03-29
4	Basel-Stadt	20	54	9	3	14	2022-03-29
...
99	Thurgau	20	35	19	2	24	2022-04-17
100	Ticino	32	26	14	7	21	2022-04-17
101	Valais	27	35	15	2	21	2022-04-17
102	Vaud	25	40	13	3	19	2022-04-17
103	Zurich	25	45	12	2	16	2022-04-17

Appendix U: Screenshot product_search Table's Content

```
pd.read_sql_query('select * from "product_search"',con=conn)
```

	query	value	date_last7d	product
0	banana bread	100	2022-03-29	banana
1	banana cake	18	2022-03-29	banana
2	banana bread recipe	16	2022-03-29	banana
3	banana bread recette	14	2022-03-29	banana
4	banana muffins	13	2022-03-29	banana
...
1707	basilikum pesto rezept	9	2022-04-17	pesto
1708	pesto rosso rezept	9	2022-04-17	pesto
1709	bärlauch pesto rezept	9	2022-04-17	pesto
1710	pesto rouge	8	2022-04-17	pesto
1711	avocado pesto	8	2022-04-17	pesto

Appendix V: Screenshot recipe_search Table's Content

```
pd.read_sql_query('select * from "recipe_search"',con=conn)
```

	query	value	region	date_last7d
0	betty bossi rezept	100	G	2022-03-29
1	lasagne rezept	51	G	2022-03-29
2	brot rezept	48	G	2022-03-29
3	wildeisen rezept	47	G	2022-03-29
4	pancakes rezept	37	G	2022-03-29
...
295	crema pasticceria ricetta	14	I	2022-04-17
296	gnocchi ricetta	12	I	2022-04-17
297	pancake ricetta	11	I	2022-04-17
298	pasta frolla ricetta	11	I	2022-04-17
299	pasta frolla	11	I	2022-04-17

7 Annexes

Table 5: 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			
Literature review (Academic / Business)	5	x		
Methodology	50			
Data lake architecture, System etc. in technical details	8		x	
Design choices / trade-offs /assumptions	7	x	x	x
Data sources definitions (technical)	7	x	x	x
Data imputation, data cleaning per data source.	7	x	x	x
Data analysis per data source.	7	x	x	x
Database schema, model, table descriptions and/or files.	7	x	x	x
Data Lake consumers (personas)	7		x	
Discussion	15			
Proper discussion of the solution (advantages, disadvantages, trade-offs in terms or technologies used etc.)	15	x	x	x
Conclusions	7			
Proper discussions of the project outcomes.	7	x		

Future Work	5			
Areas to improve in the data lake, what would you do different etc.	5		x	

Table 6: Code Project Leads

Responsibilities	Ezgi Köker Gökgöl	Vithushan Mahendran	Alexandra Alinka Zimmermann
Google Trends API (Ingestion)			x
Edamam API (Ingestion)	x		
Coop Webscraping (Ingestion)		x	

8 Source

8.1 Source Information

[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>

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>

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/>

[Coffey 2021] Patrick Coffey: *The impact of cloud-based data warehousing*. available: <https://towardsdatascience.com/the-impact-of-cloud-based-data-warehousing-eff1cadcf6f2>, accessed 12th March 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.

[Gardner 1998] Stephen R. Gardner: Building the data warehouse. Commun. ACM 41, v.9 (Sept. 1998), pp. 52–60. Doi:10.1145/285070.285080.

[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.

[Gynell et al. 2022] Gynell I., Kemps E. & Prichard I.: The effectiveness of implicit interventions in food menus to promote healthier eating behaviours: A systematic review. *Appetite* (2022). V. 173, pp. 105997 . Doi: 10.1016/j.appet.2022.105997.

[Hackernoon 2021] ScrapperAPI: *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.

[Kuester 2012] Kuester, Sabine: *MKT 301: Strategic Marketing & Marketing in Specific Industry Contexts*. Lecture Notes, University of Mannheim, pp. 110.

[Levy 2021] Jeremy Levy: *3 Reasons a Cloud Data Warehouse Is Critical To Customer Analysis*. available: <https://insidebigdata.com/2021/04/17/3-reasons-a-cloud-data-warehouse-is-critical-to-customer-analysis/>, accessed 12th March 2022.

[Oracle 2022] Oracle: *What Is a Data Warehouse?* available: <https://www.oracle.com/database/what-is-a-data-warehouse/>, accessed 12th March 2022.

[Recordati 2014/2015] Recordati, Gaia Bruna Patrizia: *The food industry: history, evolution and current trends*. Bachelor's Degree Thesis in Management, LUISS Guido Carli, relatore Paolo Boccardelli, pp. 106.

[Rehan 2021] Afnan Rehan: *6 Benefits of Adopting a Cloud Data Warehouse for Your Organization*. available: <https://www.astera.com/type/blog/cloud-data-warehouse-benefits>, accessed 12th March 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.

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

[Türker & Süzer 2022] Nuray Türker, and Özkan Süzer: Tourists' food and beverage consumption trends in the context of culinary movements: The case of Safranbolu. *International journal of gastronomy and food science* (2022). V. 27, pp. 100463. Doi: 10.1016/j.ijgfs.2021.100463.

[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.

[Weir et al. 2003] Weir, R., Peng, T., & Kerridge, J.: Best practice for implementing a data warehouse: a review for strategic alignment. In *Design and Management of Data Warehouses 2003: Proceedings of the 5th Intl. Workshop DMDW'2003*, Berlin, Germany, September 8, 2003.

[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.

8.2 Source Images

[Shareicone.net] Shareicon.net: Amazon, Delivery, Bucket, s3, Content, objects, with, storage icon [Icon]. Available <https://www.shareicon.net/amazon-delivery-bucket-s3-content-objects-with-storage-92172>, accessed 9th April 2022.

[Awesome Logo] Awesome Logo: Discussion of Using Vandium Io To Create AWS Lambda. Available <https://awe-logo.blogspot.com/2015/02/aws-lambda-logo-png.html>, accessed 9th April 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.

8.3 Source Code

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

8.3.1 Google Trends

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, "Is there a way to get access_key and secret_key from boto3? [duplicate],"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>

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>

[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.

[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.

8.3.2 Coop

Bajju Muthukandan, “7. WebDriver API”. Available: <https://selenium-python.readthedocs.io/api.html>, accessed 29.03.2022