Packaging Licencing Report Pipeline

Introduction

Packaging Licensing Fee Calculator has the objective of automating a pipeline dynamically integrated with Google Sheets that pushes to the DWH calculated costs related to utilized packaging. Key Problem: HelloFresh is required to report and pay fees related to the disposal and recycling of packaging materials. This includes packaging such as plastics, paper, and metal. Tracking these materials and calculating the corresponding fees used to be a time-consuming, manual process. As solution, this tool aggregates data from multiple sources to automatically calculate licensing fees based on the weight of packaging materials delivered each month.

Scope and Sources

The **scope** of this pipeline is limited to the DACH market and covers consumption of all packaging material that passes through the distribution centers. This includes delivered goods (paying customers) and non delivered, too (non paying customers).

> Example: a can of cooked beans is delivered in a box (paper), is made of metal (tin) and has a paper label on it (paper).

Data **Sources**: There are three main sources for this file:

- Consumption data from paying customers' deliveries (paying)
- And donated/disposed inventory movements (non paying)
- Weight conversion master data for each consumed article

Weight Conversion:

The purpose of this dataset is the conversion of materials (SKUs) to their weights. E.g., assuming I am shipping my can of cooked beans, I will need to know that the can is made of, say, 10 gr of metal, plus a 0,5 gr paper label. With this conversion I know that I could calculate the weight of delivered metal and paper for the can of cooked beans.

```
# show the head
display(packaging_weight_pd.head())
```

	sku	has_packaging	category	unit_weight_grams	packaging_type
0	alias-49384	yes	paper	0.6	label
1	alias-49384	yes	plastic	26.0	thermoform
2	alias-79751	yes	paper	0.6	label
3	alias-79751	yes	paper	0.6	label
4	alias-79751	yes	plastic	8.0	flowpack

> Google API: the master data conversion table in question did not exist as an **asset** at HelloFresh. This pipeline therefore integrates a **Google Sheet** file containing all information required and allowing stakeholders to interactively update the source. The data preparation then happens in the Python script, and includes handling categorical data, duplicate entries and other logical behaviours. More at the section "Challenges".

Customer Deliveries:

The `delivered_skus` dataset captures paying customers' delivered boxes. The entire content of the box, from the shipping label outside, to the marketing material envelope contained, or the cooling ice packs that keep the food cool throughout the delivery - all materials are listed and quantified. This represents the **Bill of Materials** of each single box.

```
display(delivered_skus_df.head())
```

	dc	hf_month	hf_week	delivery_date	source	destination	sku	sku_category	randomized_delivery_quantity
0	FI	2024-08	2024-W32	2024-08-09	pouch_data	paying_customer	alias-53602	C_09	4986
1	FI	2024-07	2024-W27	2024-07-04	pouch_data	paying_customer	alias-38086	C_09	428
2	PI	2024-07	2024-W27	2024-07-04	pouch_data	paying_customer	alias-35483	C_09	2846
3	PI	2024-07	2024-W27	2024-07-04	pouch_data	paying_customer	alias-60481	C_09	1074
4	FI	2024-08	2024-W32	2024-08-09	pouch_data	paying_customer	alias-10570	C_09	649

Because each single customer box is captured and because each contains hundreds of items, the amount of data is important. Note that for anonymization purposes much of the published dataset *in this current public repository* has been deleted, masked or randomized. However, the original SQL query is still available for reference here.

This table contains several operational categories of materials, like “food”, “marketing” or “cooling” items. These are organised in `sources`, including `'pouch_data' 'mk_bag_data' 'meal_selection' 'marketing_skus' 'ice_data' 'box_data'` . Each source (e.g. pouch_data) corresponds to a **json** array in the original table. And these arrays are parsed via SQL.

Donation and Disposal Deliveries:

Much like the `delivered_skus`, the `disposed_skus` dataset captures non-paying deliveries (donations and disposals). Next to delivered materials, there is more material that the company is consuming over the same period of time. And this corresponds to the material that is either disposed of or donated (HelloFresh always tries to donate 100% of the excess materials). Causes of excess might vary and include expiration of goods, overproduction or other inventory adjustments.

Pipeline Flow

Data Flow Overview

Calculation of all delivered materials -> Cross with packaging weights -> Determine weight of packaging per category

- Calculation of Delivered Materials: Consolidates data across customer, donation, and disposal deliveries.
- Cross-referencing with Packaging Weights: Matches SKUs with corresponding weights.
- Determination of Packaging Weight: Multiplies material quantities with material weights.

The original pipeline runs daily and overwrites the entire scope of the dataset

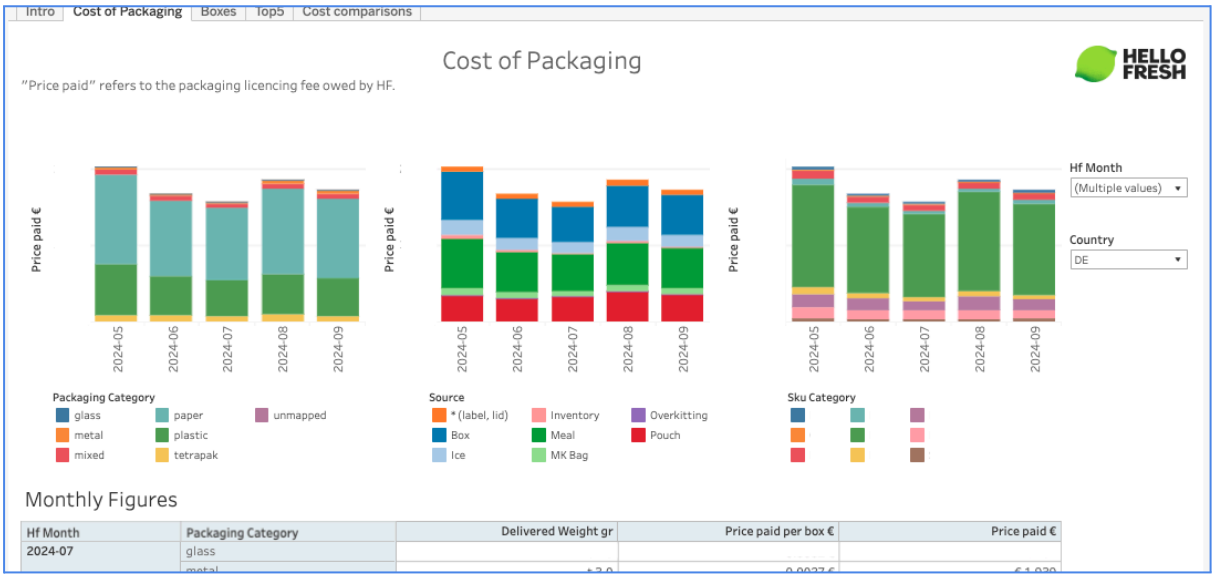
Results

Raw Data Output

In the original pipeline developed for HelloFresh we would push the final dataframe to the Data Warehouse, allowing stakeholders to query the data, analyze it and pull it into Tableau. As this step of the pipeline isn't possible for the purpose of showcasing publicly the project, we only focus on the script process and skip to the visualization.

Visualization

A Tableau dashboard is created to show



The main purpose of the dashboard is monitoring and analyzing costs. For privacy and anonymity purposes all sensitive legends and units have been canceled out with white boxes.

Challenges and Current Limitations

Delivered Materials

By knowing the HelloFresh operation, during the discovery phase we noticed that the original source `delivered_skus` is lacking critical information regarding materials, specifically: it is missing rows pertaining to some of the materials being delivered to the customers. This is a shortcoming of the source, which we wanted to address *within the Python script* until a more stable solution would be offered by the Tech teams responsible for publishing a more comprehensive source.

The following items are missing from the source: **Labels** and **Box Lids**. We therefore:

- Add labels (also called meal-kit labels, or **MK Labels**) using the logic: for every row of 'mk_bag_data' add 1 meal-kit label. This is an accurate logic that captures the delivered MK Labels without having them available in the original source.
- Add **Box Lids** using the logic: each box has 1 lid. Notice that differently sized boxes have the same lid shape, here.

> Python: in order to identify and tell these quantities apart from the actual published source, we introduced an explicative source: `source='later_manipulation'`

Packaging Weights

Because the Google file responsible for this source is dynamically connected to the Python pipeline and because it is managed by the stakeholders, it can present a few challenges like typos, duplicate entries, empty values or even incoherent logic (e.g. an item has **False** value for the `has_packaging` column, yet has a positive integer in `unit_weight_grams`).

- **Typos**: it is really hard to enforce validation rules on a spreadsheet. Especially for categorical variables. When it came to handling hundreds of cases like ‘Paper’, ‘Papre’, ‘Paper??’ and the like we opted for a downstread validation

```
valid_categories = ['paper', 'plastic', 'mixed',
'tetrapak', 'metal', 'glass']
```

- **Duplicate Entries**: the source contains multiple entries for certain SKUs. This might be because older entries where re-written, e.g. In order to maintain clarity, we opted for keeping the first entry in the dataset as a rule.

```
drop_duplicates(subset=['sku', 'has_packaging',
'category_clean'], keep='first'
```

- And in the cases where values where missing or incoherent, we used the apply method to enforce a logical dataset. For instance:

```
apply(lambda row: 0 if not row['has_packaging'] else
row['unit_weight_grams'],axis=1)
```

Results

The estimated impact of this project is a saving of 10 man-hours per month. However, the most significant change brought about by this project lies in a **thorough analysis of the data sources** and the subsequent adoption of a more accurate approach. In short, it resulted in cost savings of approximately **€120,000 per month**.

These savings stem from replacing manual reporting, which relied on highly inaccurate inventory transactions, with more reliable downstream tables. The manual process also lacked any form of test cases, leading to uncontrolled repetitive errors. For example, the inventory teams would mark packaging materials as full pallets and later adjust the value. If negative adjustments were not accounted for in the reporting process, it would falsely indicate that the "consumption" of packaging material was much higher than it actually was. For a large-scale operation like HelloFresh, such inaccuracies could lead to €120,000 in excess packaging fees every month.

Resources

Tableau Dashboard

Github repository

Jupyter Notebook