Data Engineering − Intermediate Report 4

Daan Brugmans (s1080742)
Radboud University
Nijmegen, Netherlands
daan.brugmans@ru.nl

ABSTRACT

This document is the final report for the Radboud University's Data Engineering course. Every section was written consecutively as a weekly assignment. The report describes the design and construction of a data pipeline. The code for the implementation of the pipeline described here can be found at the following URL: https://github.com/daanbrugmans/ru-data-engineering-23-24

1 DATA & PROJECT PROPOSAL

For the Data Engineering project, I would like to build a pipeline that serves data to data scientists working on a beer recommendation system. The pipeline should collect from multiple sources that serve data(sets) of beer and their attributes. Examples of attributes that should be served to the data scientists are name, brewery, alcohol by volume (ABV), international bitterness unit (IBU), category/style, textual description, flavor profile, and personal ratings. The pipeline should serve two sets of data to the data scientists: a big general dataset containing beer data from varying publicly available sources that can (and should) be updated, and a small dataset of an end user of the beer recommendation system that includes personal ratings of beers the end user has had before. These datasets will be served tabular. The data scientists can then use the ratings of the end user and the data of both datasets to build a system that can recommend beers from the big dataset to the end user's non-specified preferences, which can be extracted from the end user's data using machine learning. A realistic use case for such a recommendation system could exist for social media platforms revolving around beer and online retailers of beer that can use the system to increase revenue and beer sales.

Due to the scope of the Data Engineering project, I will implement only the parts of the pipeline that are needed to serve the small dataset of an end user. Although this requires that I construct a complete and working pipeline, I will move the focus away from data cleaning and joining, for example.

For the small dataset of an end user, I will provide my own data of beer ratings that I have collected on Untappd. Untappd [1] is a social medium where users rate beers they try and share their ratings with friends by registering their rating on the medium ("check-in"). I participate in Untappd and have collected a dataset of these check-ins with ratings. I have access to an export of my check-in data via a GDPR request. This export contains information about a beer's name, category/style, brewery, check-in location, purchasing location, flavor profile experienced by the end user, rating, and timestamp. The data will be provided in CSV format and contain almost 400 check-ins, slightly over 300 of which being

unique, so while I do not expect for missing data to be a major issue, duplicates will be more prominent.

2 DATA QUALITY

Before wrangling the data, I will talk about the data quality of the various data sources. Quantitative measures of data quality were measured using code. A class called the `DataQualityAssessor` assesses the data's quality using a few metrics. A code snippet of this class can be found in figure 1. This class is used in a Jupyter notebook that allows easy visualizations of data distributions and qualities. This notebook can be found at the following URL: https://github.com/daanbrugmans/ru-data-engineering-23-24/blob/main/code/analysis.ipynb

2.1 Untappd

The Untappd data contains all of the features I need. Most of these features do not have missing data: beer name, brewery name, beer type, ABV, IBU, and personal rating have 0% data missing.

The flavor profiles experienced by the user have 1% missing data, but since that's so little, I expect to be able to handle that issue easily. The missing flavor profile data may be caused due to how a user selects flavors: although a user can select any combination of flavors from a set list, most users will often only select from the top 5 most chosen flavors for a beer, since Untappd suggests those automatically. Flavors that are not in the top 5 require more user interaction, and many users will refrain from manually searching for specific flavors. This may mean that some beers with missing flavor profile data do have certain flavors as experienced by the user, but none of which being in the top 5 most chosen flavors, and the user did not manually search for the less common flavors.

More problematic is the feature for textual descriptions: 77% of the check-ins do not have a textual comment, and of the remaining 23%, not all comments are semantically relevant to the beer itself. I expect that this feature is not one I can make good use of.

Based on box-and-whiskers-plot and barplot visualizations, there seem to be no outliers present in the data. Two features are units of measure, ABV and IBU, and they are formatted correctly: ABV as a percentage, and IBU as a float. The check-in timestamp is formatted to the standard ISO format, and floats use periods as the decimal mark. The dataset is in an unnormalized form.

Although the Untappd data provides a unique ID for each record, for our purposes, that is not the primary key of our data: that would be beer name + brewery name. We will assume that a brewery does not make multiple different beers that share the same name. Given this primary key, the Untappd dataset consists of 18% duplicates. This can be fixed during data wrangling: duplicates are different check-ins of the same beer. Features like beer name and brewery will be consistent across duplicates. For some features, we must apply some transformation. For example, personal ratings can be

```python
class DataQualityAssessor:
    def __init__(self, dataset: pd.DataFrame) -> None:
        self.dataset = dataset

    def get_null_rates(self) -> pd.DataFrame:
        df_is_null = self.dataset.copy().isna()
        null_rates = {}

        for column_name in df_is_null.columns:
            row_count = df_is_null[column_name].size
            null_row_count = df_is_null[df_is_null[column_name] == True][column_name].size

            null_rates[column_name] = round(null_row_count / row_count, 2)

        null_rates_df = pd.DataFrame([null_rates]).T
        null_rates_df.columns = ["NULL Rate"]

        return null_rates_df

    def get_duplicate_rate(self, primary_key: list[str]) -> float:
        row_count = len(self.dataset)
        row_count_without_duplicates = len(self.dataset.drop_duplicates(subset=primary_key))

        return round(1 -  row_count_without_duplicates / row_count, 2)

    def draw_boxplots(self) -> None:
        sns.boxplot(data=pd.melt(self.dataset.select_dtypes(include="number")), x="value", y="variable")
        plt.xscale("log")

    def draw_categorical_feature(self, feature_name: str) -> None:
        sns.countplot(data=self.dataset, y=feature_name, order=self.dataset[feature_name].value_counts().index)

    def draw_continuous_feature(self, feature_name: str) -> None:
        sns.histplot(data=self.dataset, x=feature_name)
```

Figure 1: A code snippet for the Data Quality Assessor.

averaged, and flavor profiles of all check-ins can be merged into one list.

3 DATA WRANGLING

Preferably, our pipeline should serve two different sets of data: the small personal dataset with a user's ratings, and the big general dataset to query beers from. We will store these datasets as parquet files. These datasets should share the same features, with the exception of the personal rating for the personal dataset. This list of features should be as follows:

- Beer Name
- Brewery Name
- Beer Category (General)
- Beer Type (Specific)
- Alcohol By Volume (ABV)
- International Bitterness Unit (IBU)
- Textual Description
- Flavor Profile

We want to represent these features in a tabular manner. Most of these features already fit this structure. However, flavor profiles pose a problem, as they are currently stored as a list of varying size for every record in the Untappd dataset. My proposed solution is to wrangle the flavors into binary features: every flavor found in any

flavor profile becomes a feature. If a beer's flavor profile contains that flavor, the feature's value is 1, otherwise it is 0. The main problem with this approach is that these flavor features are only stored in the Untappd data. If we want to use the features for the general dataset, we will have to extract that knowledge somehow. For example, it could be collected from the Untappd website, or the data scientists could use NLP techniques on a beer's name and description to fill in the features. Regardless, making full use of the flavor features will require additional work.

The Untappd dataset only has one column about beer category, but it actually contains both beer category and beer type: beer type is the full value, while beer category is the substring prior to the "-" (if it is present).

The Untappd data source is currently represented in a tabular form and is stored in a CSV file. I will wrangle it into the desired representation using Python and Pandas, applying further operations on dataframes. Since all data is stored in a single CSV of manageable size, I can perform this process in bulk.

REFERENCES

[1] Untappd. 2024. Untappd.   https://untappd.com/