“What’s on the Menu?” - Phase 2

Project Authored By: Scott Swanson

Course: CS513 at University of Illinois Urbana-Champaign

Date: Summer 2024

Dataset Courtesy of The New York Public Library

Workflow Description

Data was first loaded into OpenRefine. The data set was examined and the cells had their whitespace trimmed, and consecutive whitespace was collapsed down. Numeric attributes were converted to numeric data types, and date attributes were converted to date data types. Default values were then imputed for data that was deemed important enough to do this with. However, this step was later learned to be futile because the chosen default values were often "N/A", which was treated as a null value by `pandas`, which resulted in them becoming `NULL` (or blank in the CSV) (this was definitely a lesson-learned).

From there, data was standardized. For dates, this was simple in that the ISO-8601 standard was used. However, for others, this had to be done via clustering. Various clustering methods were used, including: Fingerprint, n-Gram fingerprint (n-Gram size was just left at 2), Metaphone3, and Cologne Phonetic. Each cluster was manually evaluated for authenticity and repeated n-times until a sufficient data set was obtained.

Some manual value editing was done, and then unused attributes were dropped from the data set. This workflow was repeated for each data set: `Dish`, `Menu`, and `MenuPage`. The exception to the process was `MenuItem`. Since it was such a large data set, all data cleaning was instead done via `pandas` in Python.

Following the OpenRefine work, the output was used as input into `pandas` for some more complex work to be done. The work that was done here depended on the data set, but as an example for `Menu`, blank `currency` and `currency_symbol` values were replaced with the most common occurrences of each piece of data, respectively. One bit of logic that was uniformly done in `pandas` across all the datasets, however, was removing duplicate rows.

Referential integrity was verified in a slightly unconventional way as a proof-of-concept. Traditionally, one would do this via logical programming language such as Prolog or Datalog (a subset of Prolog), or even load the data into a relational database. However, referential integrity could just as easily be proved with other tools, such as `pandas`, which is what was done.

There were three different foreign keys that needed to be verified: `dish_id` (references `id` in `Dish`) in `MenuItem`, `menu_page_id` (references `id` in `MenuPage`) in `MenuItem`, and `menu_id` (references `id` in `Menu`) in `MenuPage`. The idea is simple: check if all occurrences of these foreign keys exist in their referenced entity. If a given occurrence does not exist in the referenced entity, then simply remove the record from the source entity. After all occurrences have been removed, save the output.

This comprehensive workflow ensured the data was clean, standardized, and ready for analysis, ultimately producing meaningful insights and visualizations to inform stakeholders about historical price trends.

Workflow Comparison: What Changed From Planning?

In the original workflow specified in Phase 1, three core data cleaning stages were mentioned: key value imputation, data standardization, duplicate identification and removal. However, in reality, the imputation step was extrapolated to also include some non-key attributes

because of their importance to the goals and integrity at large. For example, `date` in the `Menu` was vital to U_1 which meant imputing blank values with a default value, in this case, “1900-01-01T00:00:00Z”.

There was also another stage that was added, which was attribute removal. There were some attributes that had either zero data in them, or if they did have any data, it was minimal and didn’t contribute anything of meaning to the overall goals of the dataset. Therefore, they were removed to decrease the size of the data and keep the focus on the data that mattered.

Technically, there was a substage added as well, which comes off of the referential integrity stage. In this substage, rows in violation of the integrity constraint were just removed. These changes were then saved to the output of the data cleaning process.

I had also originally planned on using Datalog and SQL, but removed those completely in favor of just doing the referential integrity and validation queries in `pandas`. This was partially because I found more problems with data as I was going through this which was annoying to deal with in a DBMS because I’d have to fix the data in the input and then import it again each time. Doing the validation queries in `pandas` lent itself better to a more agile approach to the process, so I used that instead.

Narrative for U_1: Popular Dish Pricing Trends in the US Over Time

Motivation

The primary motivation behind Use Case U1 was to track and analyze the price trends of popular dishes over time. This analysis aimed to provide insights into how the prices of these dishes have evolved, taking into account different periods and locations. By examining historical pricing data, we sought to identify trends such as seasonal variations, the impact of inflation, and changes in culinary preferences over time. The ultimate goal was to generate a clear, consumable visualization that could inform decision-making for stakeholders such as restaurateurs, culinary historians, and economic analysts.

Validation Queries

Top 10 Most Popular Dishes

```
import pandas as pd

def popular_dishes(dish_df):
      return dish_df.nlargest(10, 'times_appeared')
```

Formatted Output:

Popular Dishes

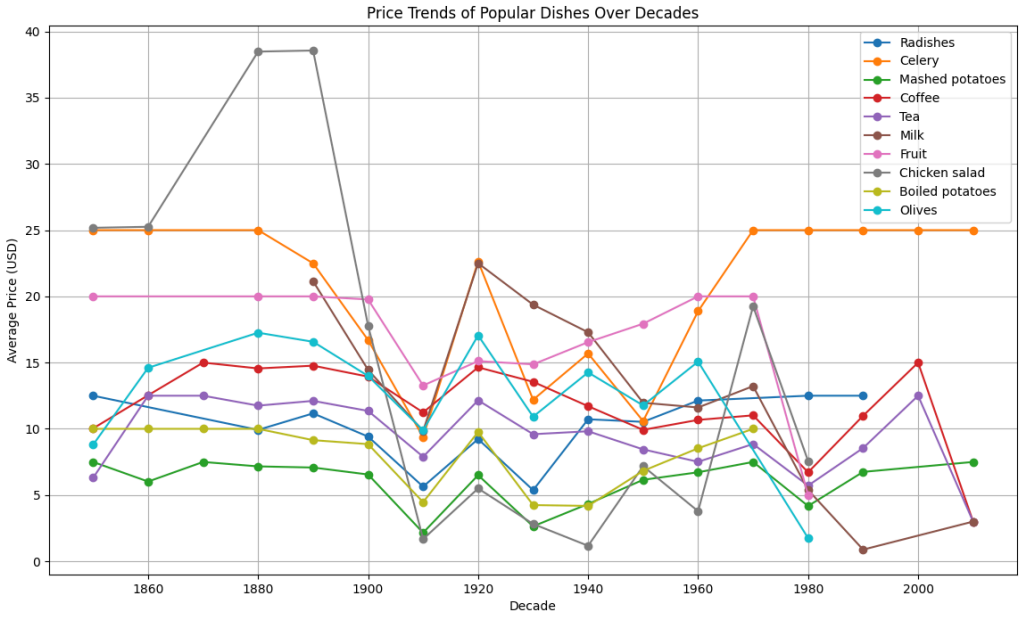
id	name	menus_appeared	times_appeared	first_appeared	last_appeared	lowest_price	highest_price
96	Coffee	7740	8484	1	2928	0.0	30.0
97	Tea	4159	4769	1858	2012	0.0	25.0
15	Celery	4246	4690	1	2928	0.0	50.0
1177	Olives	4319	4553	1858	1980	0.0	35.0
7	Radishes	3262	3346	1854	2928	0.0	25.0
83	Mashed potatoes	2583	2670	1852	2012	0.0	15.0
98	Milk	1894	2290	1890	2012	0.0	45.0
219	Boiled potatoes	2074	2129	1858	1974	0.0	20.0
112	Fruit	1919	2006	1854	2928	0.0	40.0
217	Chicken salad	1809	1879	1858	1987	0.0	100.0

USD-Only Popular Dish Price Changes Over Time (By Decade)

```
import pandas as pd
import numpy as np

def convert_to_usd(price, currency):
	if currency == "CENTS":
	return np.ceil(price / 100)
	return price

def price_changes_over_time(dish_df, menu_df, menuitem_df, menupage_df,
popular_dishes_df):
	# Merge dataframes to get complete information.
	merged_df = menuitem_df.merge(dish_df, left_on='dish_id', right_on='id',
suffixes=('', '_dish'))
	merged_df = merged_df.merge(menupage_df, left_on='menu_page_id',
right_on='id', suffixes=('', '_page'))
	merged_df = merged_df.merge(menu_df, left_on='menu_id', right_on='id',
suffixes=('', '_menu'))

	# Filter for popular dishes in USD.
	filtered_df = merged_df[(merged_df['dish_id'].isin(popular_dishes_df['id']))
&
	                        (merged_df['currency'].isin(['DOLLARS',
'CENTS']))].copy()

	# Convert prices to USD.
	filtered_df['price'] = filtered_df.apply(lambda row:
convert_to_usd(row['price'], row['currency']), axis=1)
	filtered_df['high_price'] = filtered_df.apply(lambda row:
convert_to_usd(row['high_price'], row['currency']), axis=1)

	filtered_df['date'] = pd.to_datetime(filtered_df['date'], errors='coerce',
format='%Y-%m-%dT%H:%M:%SZ')

	# Extract decade from the date.
	filtered_df['decade'] = (filtered_df['date'].dt.year // 10) * 10

	# Track price changes over the decades.
	filtered_df = filtered_df[['dish_id', 'name', 'price', 'high_price',
'decade', 'location', 'sponsor']]

	price_summary = filtered_df.groupby(['dish_id', 'name', 'decade']).agg(
	avg_price=('price', 'mean'),
	min_price=('price', 'min'),
	max_price=('price', 'max')
	).reset_index()
```

```
    return price_summary.sort_values(by=['dish_id', 'decade'])
```

Formatted Output:



Tools Used

- **OpenRefine**: Utilized for initial data cleaning, examination, and standardization. Its powerful clustering algorithms and manual editing capabilities were critical for preparing the datasets.
- **`pandas`**: Leveraged for advanced data processing, including referential integrity verification, complex data adjustments, and final data cleaning for large datasets.
- **`matplotlib`**: Used for visualizing the final analysis, creating an informative line graph that showcased the price trends of popular dishes over time.

Data Changes

OpenRefine

- `Dish`
    - Trim whitespace in the columns `description`, `lowest_price`, `highest_price`, and `name`.
    - Condensed multiple spaces into one space in the columns `description`, `lowest_price`, `highest_price`, and `name`.

    - Converted `lowest_price`, `highest_price`, `menus_appeared`, `times_appeared`, `first_appeared`, `last_appeared`, and `id` into numerics.
    - Removed the `description` column.
- `Menu`
    - Trimmed whitespace in the columns `id`, `name`, `sponsor`, `event`, `venue`, `place`, `physical_description`, `occasion`, `notes`, `call_number`, `keywords`, `language`, `date`, `location`, `location_type`, `currency`, `currency_symbol`, and `status`.
    - Condensed multiple spaces into one space in the columns `id`, `name`, `sponsor`, `event`, `venue`, `place`, `physical_description`, `occasion`, `notes`, `call_number`, `keywords`, `language`, `date`, `location`, `location_type`, `currency`, `currency_symbol`, and `status`.
    - Transformed the values in columns `event`, `currency`, `status`, `place`, and `sponsor` to uppercase for consistency.
    - Applied a series of date transformations using `value.toDate()` and custom scripts to ensure date values are in the ISO-8601 format.
    - Standardized entries in the `event` column to ensure consistency (e.g., converting different representations of "DINNER" to a single standard "DINNER").
    - Standardized entries in the `place` column, normalizing various representations of locations to a consistent format.
    - Standardized entries in the `currency_symbol` column, ensuring uniform representation of symbols like `s`, `f`, and `kr`.
    - Replaced null or empty values in certain columns (e.g., `name`, `event`, `venue`, `place`, `occasion`, `notes`) with standard placeholders like "Unknown" or "N/A".
    - Removed the columns `keywords`, `language`, and `location_type`.
- `MenuPage`
    - Trimmed whitespace in all the columns.
    - Condensed multiple spaces into one space in all the columns.
    - Converted `id`, `menu_id`, `page_number`, `full_height`, and `full_width` into numerics.
    - Standardized invalid data ranges in `full_height`, `full_width`, and `page_number`.
        - For `full_height`, values were set to `0` if they were outside the range `0-13000`.
        - For `full_width`, values were set to `0` if they were outside the range `500-9200`.
        - For `page_number`, values were set to `0` if they were outside the range `1-75`.

Python with `pandas`

- `Dish`
  - Missing `lowest_price` values were filled in using the mean (rounded to two decimal places) of the entire column.
  - Missing `highest_price` values were filled in using the mean (rounded to two decimal places) of the entire column.
  - Duplicate rows were removed from the data set.
- `Menu`
  - Missing `currency` values were filled in using the most common occurrence in the column.
  - Missing `currency_symbol` values were filled in using the most common occurrence in the column.
  - Rows that didn't contain `sponsor`, `event`, `venue`, and `place` were removed from the data set.
  - Duplicate rows were removed from the data set.
- `MenuPage`
  - Duplicate rows were removed from the data set.
- `MenuItem`
  - Trimmed whitespace in all the columns.
  - Condensed multiple spaces into one space in all the columns.
  - Columns `created_at` and `updated_at` were converted to the ISO-8601 datetime format.
  - Missing `price` and `high_price` values were imputed based on several conditions:
    - If `price` exists and `high_price` is missing, `high_price` was set to the corresponding `highest_price` from `Dish`.
    - If `high_price` exists and `price` is missing, `price` was set to the value of `high_price`.
    - If both `price` and `high_price` were missing, `price` was set to the median of `lowest_price` and `highest_price` from `Dish`, and `high_price` was set to `highest_price`.
  - If `price` was higher than `high_price`, `high_price` was set to the value of `price`.
  - Rows with empty or `NaN` values in the `dish_id` column were removed.
  - Duplicate rows were removed from the data set.

Problems and Lessons Learned

- Choosing default values like “N/A” just gets parsed by `pandas` as `NaN`. See here for more details on `read_csv()` and look at the section under `na_values`.
- I didn’t realize, going into this, how manual the process was, despite the amount of tools I had access to.
- I didn’t quite understand the value of referential integrity until this project. The idea made sense, but it took being “in the weeds” to really get how important that topic is.

Next Steps

Implementing a formal, comprehensive, data analysis and visualization platform would be what I would try to build using this project next.This platform would automate the data cleaning, processing, and visualization steps, ensuring scalability, reliability, and user-friendliness. Below is an outline of how this formal solution could be structured and what it would do.

Design

1. **Data Ingestion**
    a. **Data Sources**: The platform would support importing data from various sources, such as CSV files, databases, and APIs.
    b. **Automated Data Loading**: A robust data ingestion pipeline would automatically load and parse incoming datasets, checking for file integrity and schema consistency.
2. **Data Cleaning and Standardization**
    a. **Cleaning Module/Service**: Implement a data cleaning module that uses algorithms similar to OpenRefine for whitespace trimming, collapsing consecutive whitespaces, and converting data types.
    b. **Standardization Module/Service**: Standardize date formats to ISO-8601 and use clustering methods to unify similar text entries. Implement logic to handle missing values intelligently, avoiding issues like those encountered with "N/A".
3. **Data Processing and Integration**
    a. **Integration with `pandas`**: Use `pandas` for advanced data manipulation, ensuring the platform can handle large datasets efficiently.
        i. Maybe for this, we could also make the `currency` and `currency_symbol` work more intelligently by using the `location` to identify what the `currency` choice is in that geographic area.
    b. **Referential Integrity Verification Module/Service**: Implement checks to ensure referential integrity using pandas, with detailed logs and reports on any inconsistencies found.
4. **Analysis and Computation**

    a. **Popular Dishes Identification**: Automate the identification of popular dishes based on the `times_appeared` attribute. Maybe this version lets you filter between time periods, location, etc.
    b. **Trend Analysis**: Perform statistical analyses to identify trends, seasonal variations, and other patterns in the data.
5. **Visualization and Reporting**
    a. **Interactive Visualization**: Use libraries like `matplotlib` and `plotly` to create interactive, dynamic visualizations. Each dish would be represented by a different colored line on a time-series graph.
    b. **Dashboard Integration**: Build a web-based dashboard using frameworks like `dash` to display the visualizations and allow users to interact with the data.
    c. **Export Options**: Provide options to export the visualizations and reports in various formats, such as PNG, PDF, and Excel.