CS 513 Final Project - Farmer's Market

Authors

- Alex Antonison (ada4)
- Andrew Vojak (vojak1)
- Steven Bobo (sbobo3)

Introduction

The Data Harvester's final project involved cleaning a Farmer's Market dataset. The Farmer's Market data was downloaded from https://www.ams.usda.gov/local-food-directories/farmersmarkets. The Farmers Market Directory is a list that notes instances where two or more farm vendors are selling products to customers at a consistent location.

The primary goal of this project for Data Harvester's is to understand, clean, and enhance the data to help people be able to find Farmer's Markets that suit their needs. With this, Data Harvester's will make the process of cleaning and preparing the data automated to ensure that when future Farmer's Markets are added to the dataset, they can be automatically processed.

Initial Assessment

Dataset Structure and Quality Assessment

// You should describe the structure and content of the dataset and quality issues that are apparent from an initial inspection.

The Farmer's Market data is 8,687 records with 59 columns. Each record in the dataset represents a different farmer's market location.

- Farmers Market Name and Identifiers
  - Each farmer's market is uniquely identifiable with the `FMID` that is a number.
  - The `MarketName` column is a more user friendly method of identifying a Farmers Market, however there are instances where numerous Farmers Markets can have the same name but are not related, an example being "Livingston Farmers Market" which exists in 5 different states.
  - These columns are always filled out which is to be expected.
  - As these names are meant to be unique to identify a specific farmers market, we have decided to not normalize the MarketName column.
- Online Information
  - There are numerous fields to capture different types of digital media such as a `website`, `facebook`, `twitter`, `youtube`, as well as an `OtherMedia` column.
  - These columns have a lot of blanks. While the website column is 59% full and the Facebook column is 45% populated; twitter, youtube, and OtherMedia are mostly empty at below 15% populated.
  - For the Facebook column, it is a combination of links and names.
- Location Information
  - The location information across `street`, `city`, `County`, `State`, and `zip` all have high fill rates with the lowest being `zip` at 89%.
  - When evaluating `street`, most instances an address is provided but there are a handful of instances where common names such as `Main Street` are used.
  - The `x` and `y` columns represent longitude and latitude and are populated 99% of the time.
  - There is a `Location` column that indicates the type of location such as a "Private business parking lot" or "Local government building grounds." This is properly filled out and does not look like it will require any further transformations.
- Seasonal Availability
  - Although there are four locations for a farmers market to specify date and time, it is rare for a farmers market to specify more than one. It is worth noting that even the first section of noting what season a farmers market is available is only populated about 65% of the time. This would be an opportunity to work with farmers markets to better populate their availability to better inform the general public of when they are open.

- Accepted forms of Payment
  - In these columns, `Credit`, `WIC`, `WICcash`, `SFMNP`, and `SNAP`, it indicates what forms of payment a farmers market will accept. Whether it be a credit card or various government assisted programs.
  - As to be expected, all of these columns are 100% populated.
  - Based on an inspection using facets, these are all consistently `Y` or `N` and would not require further transformation.
- Type of Goods Sold
  - The following 30 columns are a series of flags that indicate what kind of goods are sold at the farmers market ranging from organic to kinds of meats.
  - Although the `Organic` column is 100% populated, all of the other columns are around 66% populated.
- Record Metadata
  - There is an `updateTime` column that appears to for the most part has a date timestamp but around 30% of the time only has a year for 2009, 2011, and 2013.

Use Cases

// You should also describe a (hypothetical or real) use case of the dataset and derive from it some data cleaning goals that can achieve the desired fitness for use.

The farmers market dataset can be used by people interested in finding farmers markets close to them to either purchase or sell items. To help with this, we have the following data cleaning goals:

- Cleanse up the online information columns to ensure it is easier for people find information about farmers markets close to them. Additionally, to ensure future consistency, put data validation checks in place to for properly entered hyperlinks to online content.
- Transform the Seasonal Availability columns to help make it easier to determine when farmers markets are open.
- Transform the updateTime to be a consistent date timestamp.
- Cleanse the location information to ensure that only valid information is entered in each field.

// Additionally

// Are there use cases for which the dataset is already clean enough?

The farmers market data as it currently is does provide useful information to people seeking farmers markets. However, there are numerous instances where the data entered is malformed and would require a person to interpret the entered information about a farmers market.

// Are there use cases for which the dataset will not be clean enough?

As there are parts of the dataset that predominantly blank, it would be difficult to perform a comprehensive analysis on aspects of farmers markets like goods sold, online presence, and seasonal availability.

Data Cleansing

Data Cleansing Steps

// Document the process and result of this phase, both in narrative form along with supplementary information (e.g., which columns were cleaned and what changes were made?).

- Farmers Market Name and Identifiers

  - No work needs to be done regarding the `FMID` as each record is unique.
  - No work needs to be done regarding `MarketName` since even though there are instances where a farmers market has a duplicate name, the additional information provided in a record provides enough context to identify the correct farmers market.

- Online Information

  - For Youtube, I am creating the column `Facebook_Link` for valid Youtube links using the following GREL:

```
if(value.contains("facebook.com"), value, null)
```

- For Youtube, I am creating the column `Twitter_Link` for valid Youtube links using the following GREL:

```
if(value.contains("twitter.com"), value, null)
```

- For Youtube, I am creating the column `Youtube_Link` for valid Youtube links using the following GREL:

```
if(value.contains("youtube.com"), value, null)
```

- For OtherMedia, I am only selecting instances where a single valid website link was provided and created a column `OtherMedia_Link` using the following GREL:

```
if(
  length(
    value.find(/^(http:\/\/www\.|https:\/\/www\.|http:\/\/|https:\/\/)?[a-z0-
9]+([\-\.]{1}[a-z0-9]+)*\.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$/))
  == 1,
  value, "")
```

- Location Information
  - For street, we have created a column `street_valid` that detects instances where a street number and name are included and pulls out the first valid address found. This will rule out instances where entires such as "Downtown" or "Main Street" were provided.

```
value.match(/^\d+\s[A-z.]+\s[A-z]+/).get(0)
```

    - Attempted to use OpenRefine for this and the application continued to freeze on `"Working..."`. Instead, used Python to accomplish this. Ideally this would have been done within OpenRefine and included in the task list.
  - For `city`, we created a column `city_valid` that will check for only valid words using the following GREL command:

```
if(
    length(
      value.find(/[a-zA-Z ]+/))
    == 1,
    trim(value), "")
```

  - For `County`, we created a column `County_valid` that will check for only valid words using the following GREL command:

```
if(
    length(
      value.find(/[a-zA-Z ]+/))
    == 1,
    trim(value), "")
```

- For `zip`, we crated a `zip_valid` column that will check for only valid zip codes using the following GREL command:

```
if(
    length(
      value.find(/(\b\d{5}(?:-\d{4})?\b)/))
    == 1,
    trim(value), "")
```

- As we had to find valid addresses in python, we were unable to create the column `valid_full_address` in OpenRefine.
- We renamed `x` to `longitude` and `y` to `latitude` to make the column names more clear.
- We converted `longitude` and `latitude` to numbers using `Edit Cells > Common transforms > To number`

- Seasonal Availability
  - With each `SeasonDate` column, we are going to create `SeasonStartDate` and `SeasonEndDate` columns to allow for a person to more easily search on what farmers markets start and end.
  - We will use the following GREL commands to pull out the first and second date:

```
if(
    length(
      value.find(/[0,1]?\d{1}\/(([0-2]?\d{1})|([3][0,1]{1}))\/(([1]{1}[9]{1}[9]
{1}\d{1})|([2-9]{1}\d{3}))/))
    == 2,
    value.find(/[0,1]?\d{1}\/(([0-2]?\d{1})|([3][0,1]{1}))\/(([1]{1}[9]{1}[9]
{1}\d{1})|([2-9]{1}\d{3}))/).get(0), "")

if(
    length(
      value.find(/[0,1]?\d{1}\/(([0-2]?\d{1})|([3][0,1]{1}))\/(([1]{1}[9]{1}[9]
{1}\d{1})|([2-9]{1}\d{3}))/))
    == 2,
    value.find(/[0,1]?\d{1}\/(([0-2]?\d{1})|([3][0,1]{1}))\/(([1]{1}[9]{1}[9]
{1}\d{1})|([2-9]{1}\d{3}))/).get(1), "")
```

  - Once created, we then converted each date to an ISO date for ease of use with other applications. We used the `Edit Cells > Common transforms > To date` function for this.
- Accepted forms of Payment
  - Upon inspection, these columns are all appropriately filled out and do not require any further transformation.
- Type of Goods Sold
  - The only column that requires transformation is the `Organic` column to replace "-" with "" to be consistent in other instances with missing values. The new column is `Organic_valid`. The following GREL was used:

```
value.replace("-","")
```

- Record Metadata
    - We transformed the `updateTime` to a valid date using `Edit Cells > Common transforms > To date`.

Data Cleansing Quantified

// Can you quantify the results of your efforts? Also, provide provenance information from OpenRefine. Pay close attention to what OpenRefine includes and does not include in its operation history!

Source_Column	**Cleansed_Column**	**Source_Column**	**Cleansed_Column**	**% Difference**	**Tool**	**Note**
Organic	Organic_valid	8687	3644	-58.05%	OpenRefine	Replaced rows with just - with a blank to give a better idea of how complete the column is
street	street_valid	8397	4244	-49.46%	Python	Used regex to only include valid addresses
OtherMedia	OtherMedia_Link	718	460	-35.93%	OpenRefine	Removed invalid website URLs
Youtube	Youtube_Link	171	123	-28.07%	OpenRefine	Removed non-youtube URLS
Twitter	Twitter_Link	1008	761	-24.50%	OpenRefine	Removed non-twitter URLS
Facebook	Facebook_Link	3929	3399	-13.49%	OpenRefine	Removed non-facebook URLs
city	city_valid	8647	8451	-2.27%	OpenRefine	Used regex to only include rows that had words
County	County_valid	8172	8007	-2.02%	OpenRefine	Used regex to only include rows that had words

Source_Column	Cleansed_Column	Source_Column	Cleansed_Column	% Difference	Tool	Note
zip	zip_valid	7742	7724	-0.23%	OpenRefine	Only included numbers with 5 digits
x	longitude	8658	8658	0.00%	OpenRefine	Converted to number
y	latitude	8658	8658	0.00%	OpenRefine	Converted to number
updateTime	n/a	8687	8687	0.00%	OpenRefine	Changed to a valid date
FMID	n/a	8687	8687	n/a	n/a	No change
MarketName	n/a	8687	8687	n/a	n/a	No change
Website	n/a	5212	5212	n/a	n/a	No change
State	n/a	8687	8687	n/a	n/a	No Changes necessary
n/a	full_address	n/a	3844	n/a	Python	Constructed a full address for a Farmers Market
Season1Date	n/a	5479	n/a	n/a	n/a	No Change
n/a	Season1StartDate	n/a	409	n/a	OpenRefine	Extracted the first valid date from the Season1Date
n/a	Season1EndDate	n/a	409	n/a	OpenRefine	Extracted the second valid date from the Season1Date
Season1Time	n/a	5637	n/a	n/a	n/a	No Change
Season2Date	n/a	449	n/a	n/a	n/a	No Change
n/a	Season2StartDate	n/a	409	n/a	OpenRefine	Extracted the first valid date from the Season2Date
n/a	Season2EndDate	n/a	409	n/a	OpenRefine	Extracted the second valid date from the Season2Date

Source_Column	Cleansed_Column	Source_Column	Cleansed_Column	% Difference	Tool	Note
Season2Time	n/a	437	n/a	n/a	n/a	No Change
Season3Date	n/a	81	n/a	n/a	n/a	No Change
n/a	Season3StartDate	n/a	73	n/a	OpenRefine	Extracted the first valid date from the Season3Date
n/a	Season3EndDate	n/a	73	n/a	OpenRefine	Extracted the second valid date from the Season3Date
Season3Time	n/a	77	n/a	n/a	n/a	No Change
Season4Date	n/a	6	n/a	n/a	n/a	No Change
n/a	Season4StartDate	n/a	5	n/a	OpenRefine	Extracted the first valid date from the Season4Date
n/a	Season4EndDate	n/a	5	n/a	OpenRefine	Extracted the second valid date from the Season4Date
Season4Time	n/a	6	n/a	n/a	n/a	No Change

// If important information is missing in the latter, provide that information in narrative form.

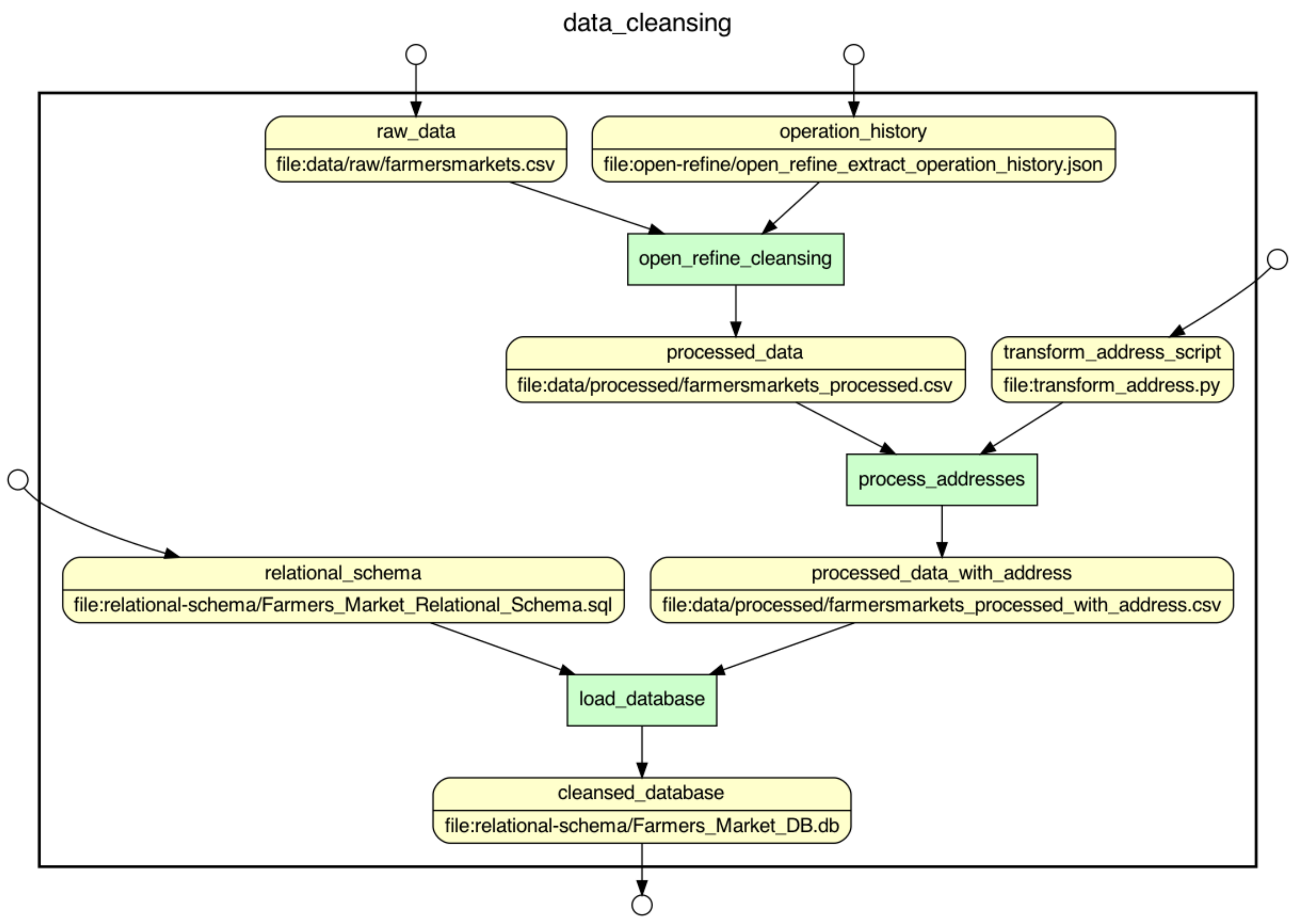
Workflow Model

// Identifying the key inputs, outputs of your workflow along with the dependencies

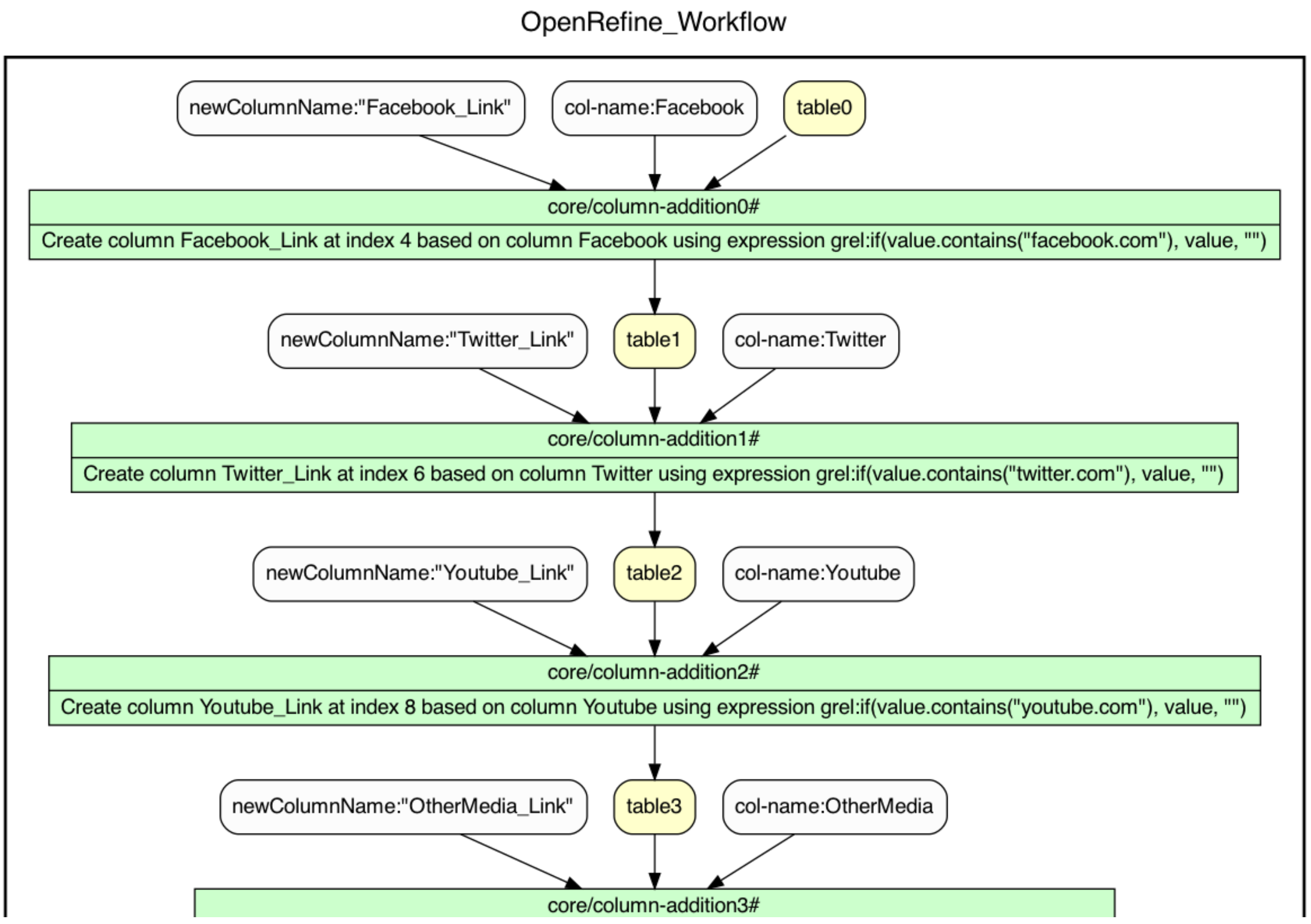
The primary input to the workflow is the raw data (`data/raw/farmersmarkets.csv`). A user can manually cleanse the data step-by-step using OpenRefine, however for the sake of reproducability and provenance we consider the OpenRefine operation history a key input, as it will allow others to derive the same processed data set. Next, we have a Python script to transform addresses (`transform_address.py`) which produces a cleansed and processed data set (`data/processed/farmersmarkets_processed_with_address.csv`) containing valid addresses. Finally, using the relation schema defined by `Farmers_Market_Relational_Schema.sql`, we can load the processed data into a database (`Farmers_Market_DB.db`).

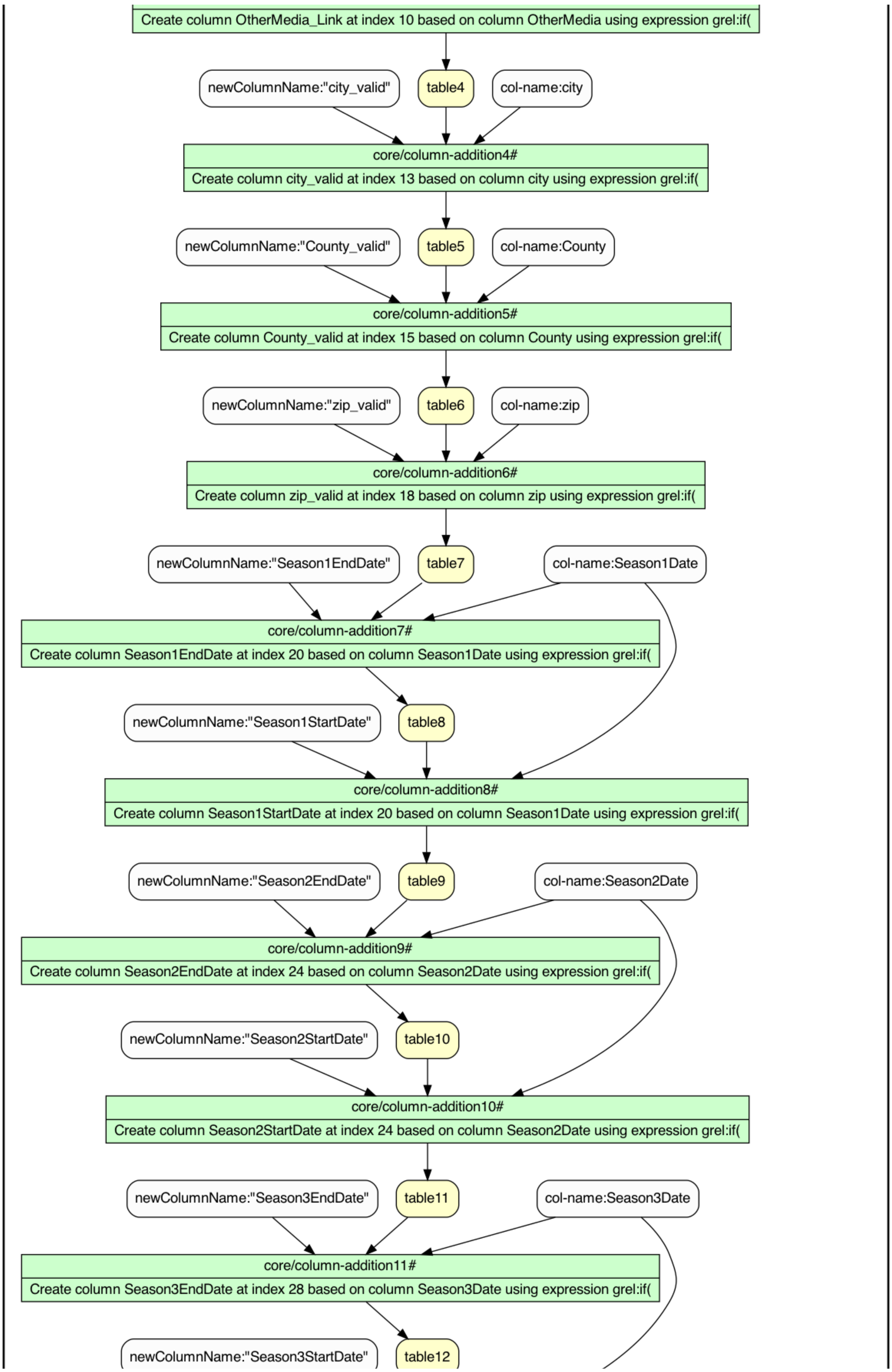
Between each stage of the workflow, the key dependencies are the `.csv` file produced from OpenRefine as well as the derived `.csv` file with valid addresses.

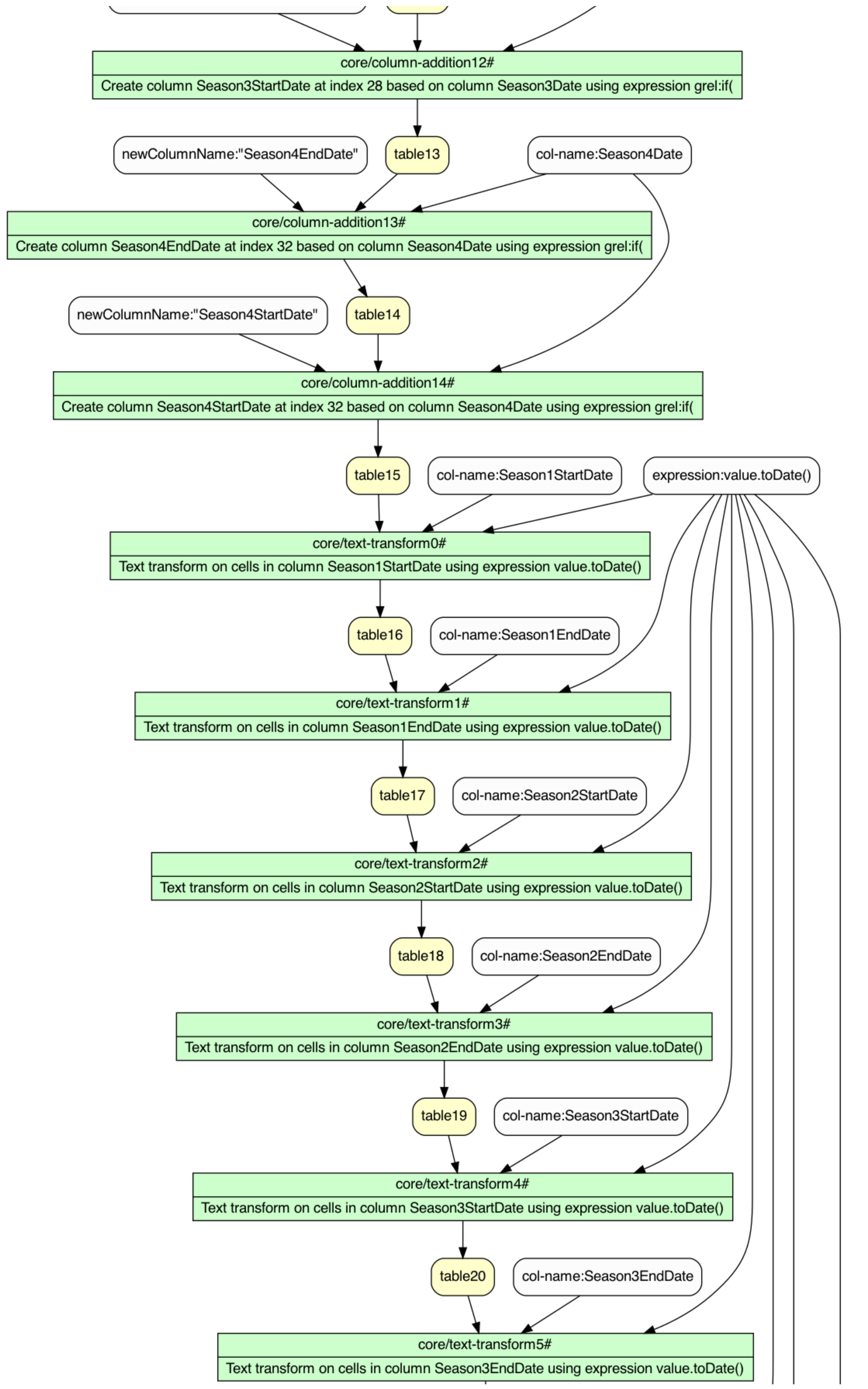
The overall cleansing workflow can be visualized using YesWorkflow:


data_cleansing
raw_data
file:data/raw/farmersmarkets.csv
operation_history
file:open-refine/open_refine_extract_operation_history.json
open_refine_cleansing
processed_data
file:data/processed/farmersmarkets_processed.csv
transform_address_script
file:transform_address.py
process_addresses
relational_schema
file:relational-schema/Farmers_Market_Relational_Schema.sql
processed_data_with_address
file:data/processed/farmersmarkets_processed_with_address.csv
load_database
cleansed_database
file:relational-schema/Farmers_Market_DB.db

The OpenRefine operation history can be exported as a `.json` file, and visualized using or2ywtool. This tool generates intermediate YesWorkflow data that graphviz turns into an image:


OpenRefine_Workflow
newColumnName:"Facebook_Link"
col-name:Facebook
table0
core/column-addition0#
Create column Facebook_Link at index 4 based on column Facebook using expression grel:if(value.contains("facebook.com"), value, "")
newColumnName:"Twitter_Link"
table1
col-name:Twitter
core/column-addition1#
Create column Twitter_Link at index 6 based on column Twitter using expression grel:if(value.contains("twitter.com"), value, "")
newColumnName:"Youtube_Link"
table2
col-name:Youtube
core/column-addition2#
Create column Youtube_Link at index 8 based on column Youtube using expression grel:if(value.contains("youtube.com"), value, "")
newColumnName:"OtherMedia_Link"
table3
col-name:OtherMedia
core/column-addition3#


Create column OtherMedia_Link at index 10 based on column OtherMedia using expression grel:if(
newColumnName:"city_valid"
table4
col-name:city
core/column-addition4#
Create column city_valid at index 13 based on column city using expression grel:if(
newColumnName:"County_valid"
table5
col-name:County
core/column-addition5#
Create column County_valid at index 15 based on column County using expression grel:if(
newColumnName:"zip_valid"
table6
col-name:zip
core/column-addition6#
Create column zip_valid at index 18 based on column zip using expression grel:if(
newColumnName:"Season1EndDate"
table7
col-name:Season1Date
core/column-addition7#
Create column Season1EndDate at index 20 based on column Season1Date using expression grel:if(
newColumnName:"Season1StartDate"
table8
core/column-addition8#
Create column Season1StartDate at index 20 based on column Season1Date using expression grel:if(
newColumnName:"Season2EndDate"
table9
col-name:Season2Date
core/column-addition9#
Create column Season2EndDate at index 24 based on column Season2Date using expression grel:if(
newColumnName:"Season2StartDate"
table10
core/column-addition10#
Create column Season2StartDate at index 24 based on column Season2Date using expression grel:if(
newColumnName:"Season3EndDate"
table11
col-name:Season3Date
core/column-addition11#
Create column Season3EndDate at index 28 based on column Season3Date using expression grel:if(
newColumnName:"Season3StartDate"
table12


core/column-addition12#
Create column Season3StartDate at index 28 based on column Season3Date using expression grel:if(
newColumnName:"Season4EndDate"
table13
col-name:Season4Date
core/column-addition13#
Create column Season4EndDate at index 32 based on column Season4Date using expression grel:if(
newColumnName:"Season4StartDate"
table14
core/column-addition14#
Create column Season4StartDate at index 32 based on column Season4Date using expression grel:if(
table15
col-name:Season1StartDate
expression:value.toDate()
core/text-transform0#
Text transform on cells in column Season1StartDate using expression value.toDate()
table16
col-name:Season1EndDate
core/text-transform1#
Text transform on cells in column Season1EndDate using expression value.toDate()
table17
col-name:Season2StartDate
core/text-transform2#
Text transform on cells in column Season2StartDate using expression value.toDate()
table18
col-name:Season2EndDate
core/text-transform3#
Text transform on cells in column Season2EndDate using expression value.toDate()
table19
col-name:Season3StartDate
core/text-transform4#
Text transform on cells in column Season3StartDate using expression value.toDate()
table20
col-name:Season3EndDate
core/text-transform5#
Text transform on cells in column Season3EndDate using expression value.toDate()

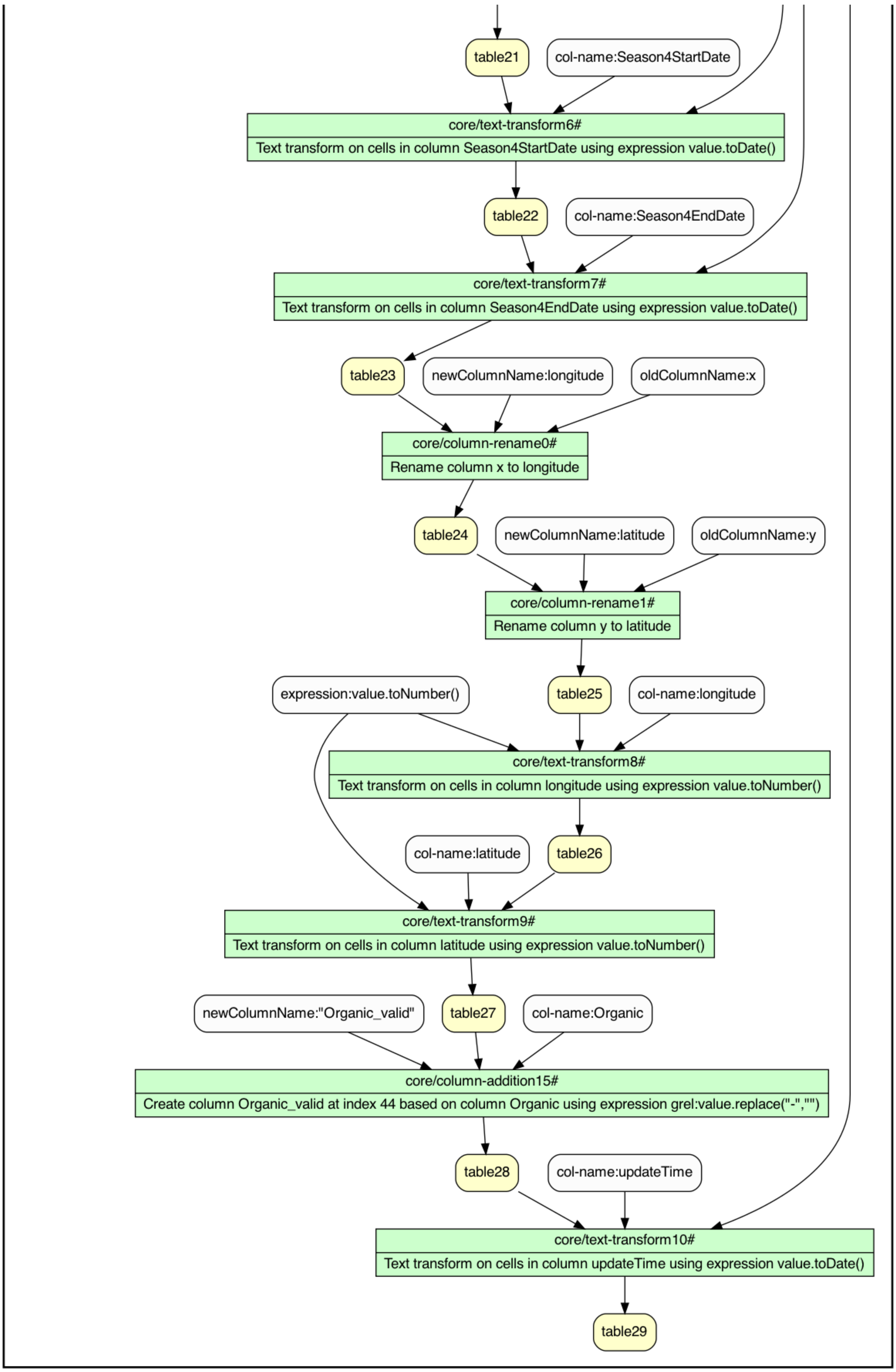


table21
col-name:Season4StartDate
core/text-transform6#
Text transform on cells in column Season4StartDate using expression value.toDate()
table22
col-name:Season4EndDate
core/text-transform7#
Text transform on cells in column Season4EndDate using expression value.toDate()
table23
newColumnName:longitude
oldColumnName:x
core/column-rename0#
Rename column x to longitude
table24
newColumnName:latitude
oldColumnName:y
core/column-rename1#
Rename column y to latitude
expression:value.toNumber()
table25
col-name:longitude
core/text-transform8#
Text transform on cells in column longitude using expression value.toNumber()
col-name:latitude
table26
core/text-transform9#
Text transform on cells in column latitude using expression value.toNumber()
newColumnName:"Organic_valid"
table27
col-name:Organic
core/column-addition15#
Create column Organic_valid at index 44 based on column Organic using expression grel:value.replace("-","")
table28
col-name:updateTime
core/text-transform10#
Text transform on cells in column updateTime using expression value.toDate()
table29