CS513: Data Cleaning

Phase-II Report: Farmer's Market Data

Jahnvi Patel (jpate201)

File Locations

Phase-I Report: https://uofi.box.com/s/ahebkd60gqm2zhfmjib5fk82cpxd2a56
Dirty Data: https://uofi.box.com/s/sexixvibvaja4gqgpvjwmo8oa2c05wfd
Clean Data (OpenRefine): https://uofi.box.com/s/kdbnmezpz91bu8kohqjqcyavhti8ie7d
OpenRefine History: https://uofi.box.com/s/5l5evrvlpvpruscy2t2tem6uy3zu27fm
SQLite Queries: https://uofi.box.com/s/4ove9o9aj41ri3we7w2ztey3eq36unv8
SQLite DB Export: https://uofi.box.com/s/2ys1s8spal6kiezg0hoqnhwwcxyd8zwd
Jupyter Notebook: https://uofi.box.com/s/1vy7oi0ou120q7wzp5db526g85x35unt
Workflow Model: https://uofi.box.com/s/mbjj5v0g05t4ximljlhywveuu94nc09h
Proposed Schema Model: https://uofi.box.com/s/wdanzndsjeukjjxrcg97rvxj6phte2ai

Queries Export:

1. Organic Produce: https://uofi.box.com/s/tpaukt3omq1xuigkozpkc1c60377yxty
2. Vegetables, Meat, and Seafood: https://uofi.box.com/s/k0kz2f4tf2rould1d05tig0okki976zz
3. All Payments: https://uofi.box.com/s/ylzyaq0r5mg286n9r8jukc090y7fzvaq
4. Credit accepted: https://uofi.box.com/s/ejuxtoght1fzy7v9cly0w2kwhxmejnlg
5. WIC accepted: https://uofi.box.com/s/g18s4diykp1wao0yw4ryvqgh7egar2tm
6. WICCash accepted: https://uofi.box.com/s/6l1ypfwlv1jernyemqyyzl36lcszw66v
7. SFMNP accepted: https://uofi.box.com/s/pd03m0dkrnxdevxpk85kmr1tx9g2s3vr
8. SNAP accepted: https://uofi.box.com/s/rucxpnyo61xq6ujdagxzi0m1op8xw0sh

Full Download: https://uofi.box.com/s/fa5kzvbccr3bofd3vtcnfp4jf3vlsqrr

Please email me at jpate201@Illinois.edu if you have any issues with access.

Handling of Phase-I Report

Points received in Phase-1: 100/100

[ ] Improved/Extended Phase-1
[x] No change necessary

Introduction

One of constant issues in data analytics is finding and fixing dirty data. Failure to do so may result in inaccurate data analyses and erroneous conclusions. The aim of the project is to utilize and apply the data cleaning techniques learned in CS 513: Data Cleaning to clean a dirty dataset.

In this report, we go through various steps of analysis to create viable data schema and workflows to clean the data so it can be used to query chosen use cases from Phase-1.

Selected Dataset

For this project, we will utilize the ***Farmer's Market*** Dataset. This dataset contains a number of farmers' markets across the United States, along with data such as their name, location, open times, payment methods accepted, and products sold.

Use Cases

Before we embark on cleaning the data, we must identify a purpose for which the data needs to be clean. In this section, we will identify three use cases: target use case (U1), “zero data cleaning” (U0) use case, and “never (good) enough” (U2) use case.

- **Target Use Case (U1):**
    - “Which Farmers’ Markets sell vegetables, meats, *and* seafood?”
        - This question would help a customer find the appropriate farmer’s markets that sell vegetables, meats, and seafood to help plan their grocery trip.
    - “Which Farmers’ Markets sell organic produce?”
        - This use case would help the customer find appropriate farmers’ markets that sell organic produce
    - “Could I map the density of Farmer’s Market which accept Credit/WIC/WICcash/SFMNAP/SNAP across the U.S.?
        - With almost complete data found in location and products portion of the dataset, it would be interesting to see the question mapped out across the Farmer’s Market in the U.S.

- **“Zero Data Cleaning” Use Case (U0):**
    - “Which Farmer’s Market accepts Credit, Cash, WIC, WICash, SFMNP, and/or SNAP?”
        - As the above attribute values are all filled, no data cleaning is required to answer this use case.
    - “Can I create a map of farmer’s markets by city, state, and county to get an understanding of the density of the farmer’s market in the U.S?”
        - As *Street*, *City*, *County*, and *State* information is available to a large extent, a user would be able to get a good idea of this question without any data cleaning. Also, by using the x and y coordinate fields, this question would be easily and accurately answered.
    - “Can I determine the popularity of X product in Y State/City/Zip?”
        - As the product attribute values are abundantly available in the Farmer’s Market Dataset, this question could be answered using the given information.

- **“Never Good Enough” Use Case (U2):**
    - “Which farmer’s market can I visit on X date?”
        - The Farmer’s Market Dataset is very limited and poorly displays the dates and times that the markets are open and available for a visit. *Season1Date* attribute is mostly filled with date ranges and *Season1Time* is filled with weekdays and times the said market is open. Though, *Season2Date*, *Season2Time*, *Season3Date*, *Season3Time*, *Season4Date*, and *Season4Time* are either missing data or display inaccurate or redundant data. Moreover, the seasons do not accurately represent the four seasons. For example, in the United States, Summer lasts from June 21st to September 22nd. If we presume *Season1Date* to be the summer season, most date ranges do not adhere to this range. With 8666 total Farmer’s Market, it would be impossible to sort out the data.
    - “Could I scrape the social media URLs for information about the Farmer’s Market?”
        - As most of the URLs for the social media sites are missing, non-URL strings, or no longer exist, this question would not yield useful results.

Describe the (original) Dataset

The Farmer's Market Dataset contains 59 columns and 8666 rows.

The attribute name *FMID* is a primary key (PK) that uniquely identifies each record. Followed by the attribute name *MarketName*, which identifies each farmer's market. The next five field names are dedicated to social media URLs, which are filled with inconsistent and missing data.

This is followed by locational data such as *street*, *city*, *County*, *State*, and *Zip*. These attributes are all nearly filled, though will require various forms of data cleaning.

Next, we have eight seasonal data attributes composed of *Season1Date, Season1Time, Season2Date, Season2Time, Season3Date, Season3Time, Season4Date, and Season4Time*; apart from Season1 date and times most of the other attributes have little to no information. Since these values have no functionality for U1, it makes sense to remove these attributes.

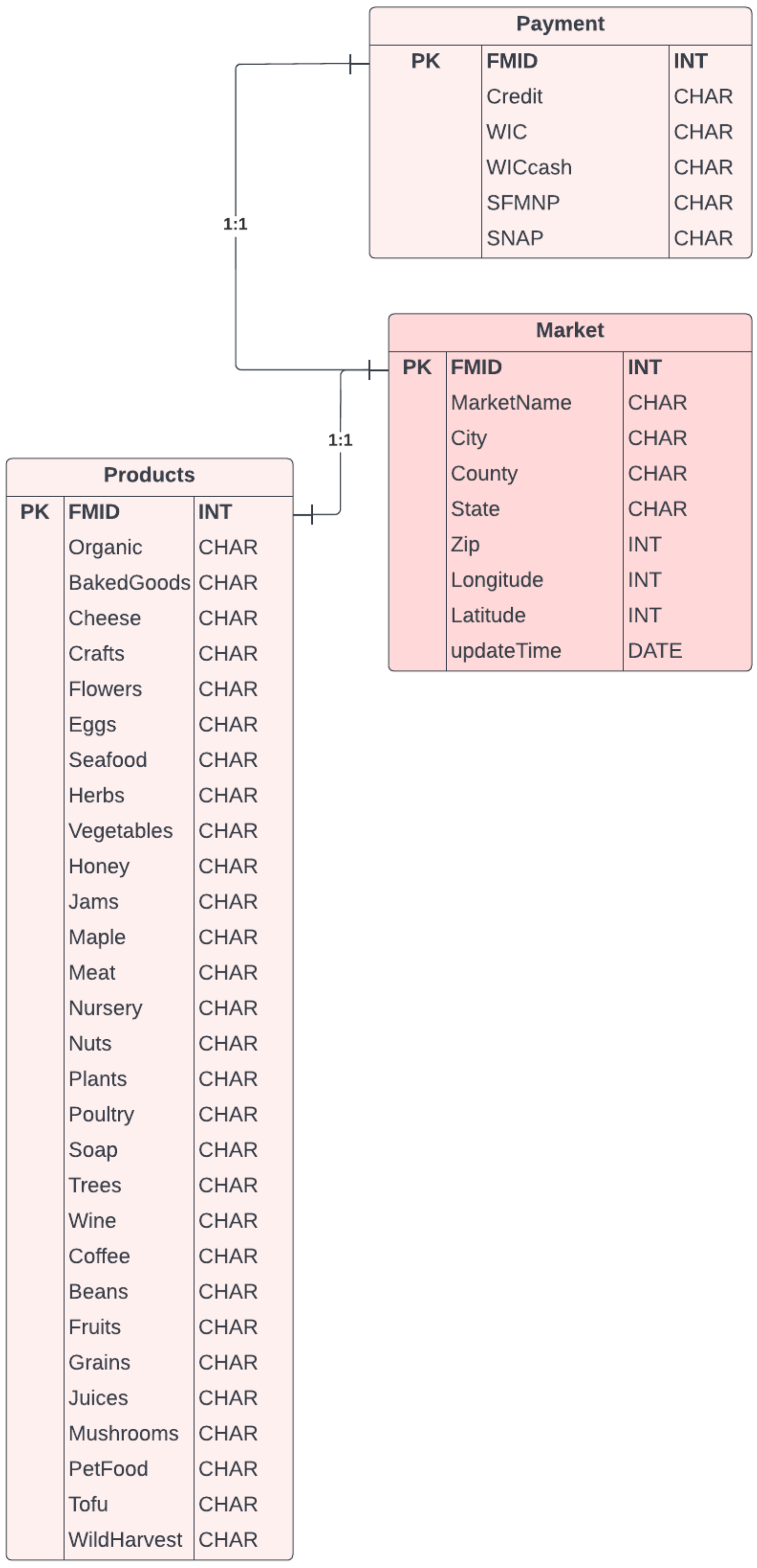
This is followed by *x* (Longitude) and *y* (Latitude) coordinates, which have near-complete data, and ideally, the attributes would be moved next to location information. Next, we have the *Location* field, which contains miscellaneous location-based information about it few of the farmer's markets.

Next, we information about payment methods accepted by the farmer's markets. *Credit, WIC* (Women, Infants, Children Program), *WICcash* (Women, Infants, Children Cash Program), *SFMNP* (Seniors Farmers' Market Nutrition Program), and *SNAP* (Supplemental Nutrition Assistance Program). The data is noted as Y/N.

Next attributes make up a large part of the database. They detail the products sold at the farmer's markets: *Organic, Bakedgoods, Cheese, Crafts, Flowers, Eggs, Seafood, Herbs, Vegetables, Honey, Jams, Maple, Meat, Nursery, Nuts, Plants, Poultry, Prepared, Soap, Trees, Wine, Coffee, Beans, Fruits, Grains, Juices, Mushrooms, PetFood, Tofu,* and *WildHarvested.* These attributes are missing a large number of data values, but a good understanding can be gathered from the values present.

At last, we have the *updateTime* attribute, which can presumably be described when the data record was last updated for each of the Farmer's Markets.

As social media data does not provide any important relevance to U1, we have removed the attributes from the dataset



Payment

PK	FMID	INT
	Credit	CHAR
	WIC	CHAR
	WICcash	CHAR
	SFMNP	CHAR
	SNAP	CHAR

Market

PK	FMID	INT
	MarketName	CHAR
	City	CHAR
	County	CHAR
	State	CHAR
	Zip	INT
	Longitude	INT
	Latitude	INT
	updateTime	DATE

Products

PK	FMID	INT
	Organic	CHAR
	BakedGoods	CHAR
	Cheese	CHAR
	Crafts	CHAR
	Flowers	CHAR
	Eggs	CHAR
	Seafood	CHAR
	Herbs	CHAR
	Vegetables	CHAR
	Honey	CHAR
	Jams	CHAR
	Maple	CHAR
	Meat	CHAR
	Nursery	CHAR
	Nuts	CHAR
	Plants	CHAR
	Poultry	CHAR
	Soap	CHAR
	Trees	CHAR
	Wine	CHAR
	Coffee	CHAR
	Beans	CHAR
	Fruits	CHAR
	Grains	CHAR
	Juices	CHAR
	Mushrooms	CHAR
	PetFood	CHAR
	Tofu	CHAR
	WildHarvest	CHAR

Figure 1 Proposed Database Schema for Farmers' Market Data

Farmer’s Market Database Dictionary		
Original Attribute Name	**Modified Attribute Name**	**Description**
FMID	FMID	Primary Key that uniquely identifies each record
MarketName	MarketName	Name of the Farmer’s Market
street	Street	Street address of the Farmer’s Market
city	City	City of the Farmer’s Market
County	County	County of the Farmer’s Market
State	State	State of the Farmer’s Market
zip	Zip	Zip code of the Farmer’s Market
x	Longitude	Longitude
y	Latitude	Latitude
Credit, WIC, WICcash, SFMNP, SNAP	Credit, WIC, WICcash, SFMNP, SNAP	Payment methods accepted
Organic, Bakedgoods, Cheese, Crafts, Flowers, Eggs, Seafood, Herbs, Vegetables, Honey, Jams, Maple, Meat, Nursery, Nuts, Plants, Poultry, Prepared, Soap, Trees, Wine, Coffee, Beans, Fruits, Grains, Juices, Mushrooms, PetFood, Tofu, and WildHarvested	Organic, BakedGoods, Cheese, Crafts, Flowers, Eggs, Seafood, Herbs, Vegetables, Honey, Jams, Maple, Meat, Nursery, Nuts, Plants, Poultry, Prepared, Soap, Trees, Wine, Coffee, Beans, Fruits, Grains, Juices, Mushrooms, PetFood, Tofu, and WildHarvested	Products sold at the Farmer’s Market
updateTime	UpdateTime	The last known date/time the record was updated

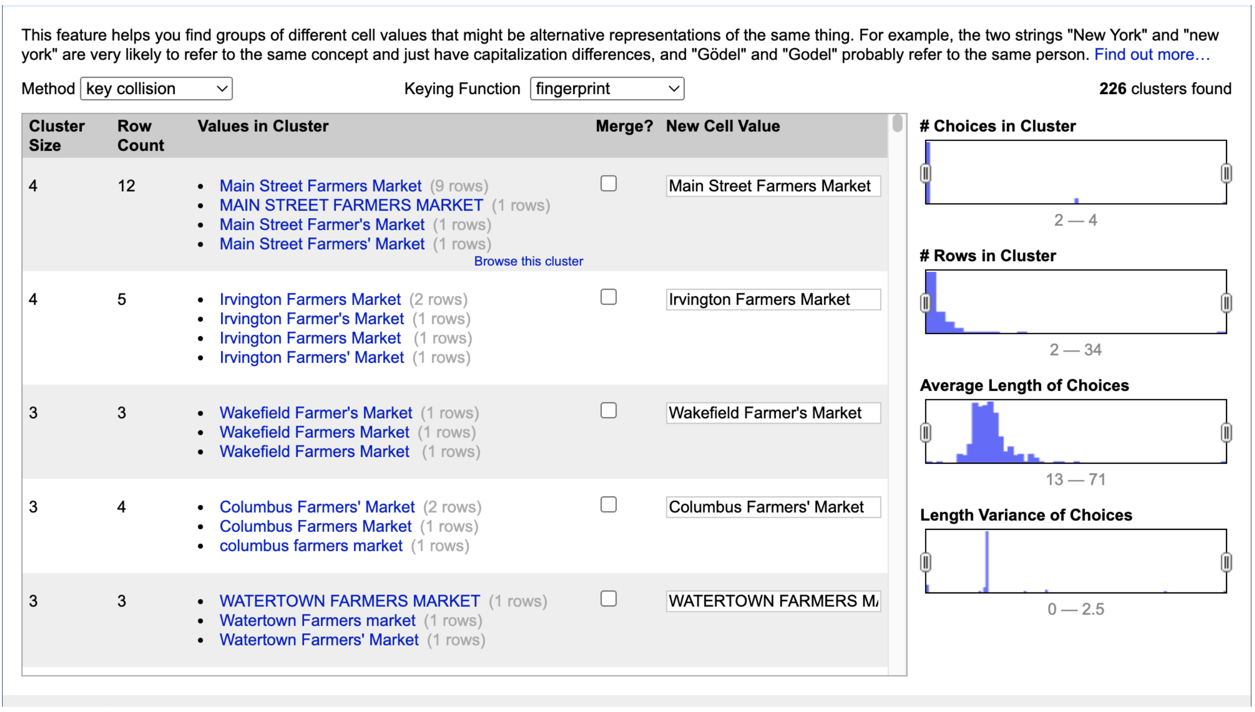
Figure 2 Database Dictionary for Farmer’s Market Database

Data Quality Problems

At an initial glance, we can notice several data quality issues present in the Farmer's Market Dataset.

The goal is to achieve data quality sufficient to fulfill our target use case, U1. To achieve this, we must focus on reducing errors and improving the consistency of the farmers' market names, location data, accepted payment information, and product data. Though in general proper data cleaning exercises will improve the accuracy, consistency, and reliability of *all* attributes.

Using the text facet in OpenRefine, we can observe that farmers' markets, street addresses have inconsistencies in title cases and spellings. The same is consistent with other attributes such as city and county. We must canonicalize the data here using clustering in OpenRefine.



This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. Find out more…

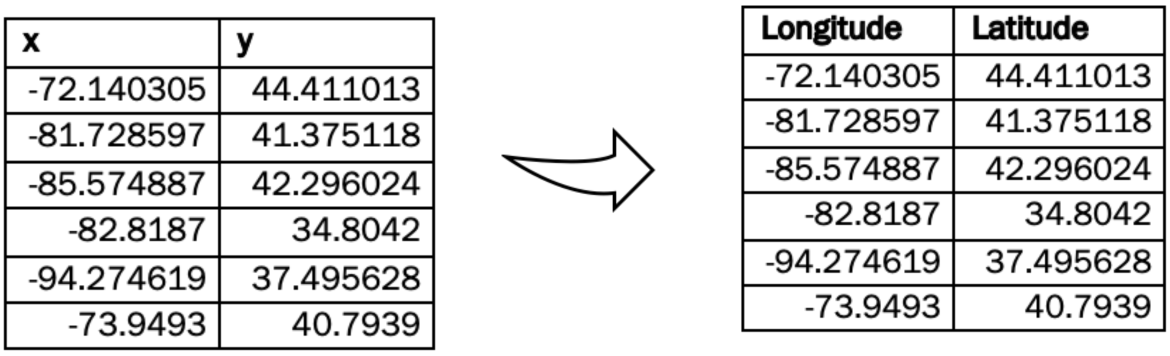
Method: key collision   Keying Function: fingerprint   **226** clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
4	12	• Main Street Farmers Market (9 rows) • MAIN STREET FARMERS MARKET (1 rows) • Main Street Farmer's Market (1 rows) • Main Street Farmers' Market (1 rows) Browse this cluster	☐	Main Street Farmers Market
4	5	• Irvington Farmers Market (2 rows) • Irvington Farmer's Market (1 rows) • Irvington Farmers Market (1 rows) • Irvington Farmers' Market (1 rows)	☐	Irvington Farmers Market
3	3	• Wakefield Farmer's Market (1 rows) • Wakefield Farmers Market (1 rows) • Wakefield Farmers Market (1 rows)	☐	Wakefield Farmer's Market
3	4	• Columbus Farmers' Market (2 rows) • Columbus Farmers Market (1 rows) • columbus farmers market (1 rows)	☐	Columbus Farmers' Market
3	3	• WATERTOWN FARMERS MARKET (1 rows) • Watertown Farmers market (1 rows) • Watertown Farmers' Market (1 rows)	☐	WATERTOWN FARMERS M/

Figure 3 *MarketName* cluster in OpenRefine

Social media attributes (*Website, Facebook, Twitter, Youtube, OtherMedia*) do not provide any important relevancy to the use case U1, hence it is logical to remove the attributes from the dataset.

It is also vital to rename some of the attributes to improve consistency. For instance, x and y attributes would become Longitude and Latitude, respectively. This includes attribute names that need a proper title case and spell check for persistency with the rest of the dataset



x	y
-72.140305	44.411013
-81.728597	41.375118
-85.574887	42.296024
-82.8187	34.8042
-94.274619	37.495628
-73.9493	40.7939

Longitude	Latitude
-72.140305	44.411013
-81.728597	41.375118
-85.574887	42.296024
-82.8187	34.8042
-94.274619	37.495628
-73.9493	40.7939

Figure 4 x and y transformation to Longitude and Latitude

It is important to observe the social media attributes (*Website, Facebook, Twitter, Youtube, and OtherMedia*) as they have a vast number of missing and inaccurate information, and it is not contributing to our goal U1 use case. It must also be noted that the Location attribute does not provide any significant use to our U1 goal, hence it can be removed from the dataset as well.

As SeasonXDate and SeasonXTime do not follow the date ranges of the four seasons, it is not prudent to allocate them eight individual attributes. It will be simpler for comprehension to pull this data into a separate table and combine the date attributes into a single attribute, Season Date, which contains the date ranges that the farmer's market is open; along with a single attribute time, which combines the times.

It can also be observed that the attributes are not correct data types. It will be vital to add this to the plan to correct it. The attributes that contain date and time information will need to be made ISO-compliant standard date form YYYY-MM-DD.

Some zip codes appear to be only 4 characters. We must apply integrity constraints to confirm that all zip codes have 5 characters.

Initial Plan

The initial plan for the data was as follows:

1. OpenRefine:
   a. Remove attributes not relevant to U1
   b. Remove trailing, leading, and consecutive whitespaces
   c. Assign appropriate data types to the attributes
   d. Rename attribute names for consistency
   e. Convert Y to 1 and N to 0 in the data
   f. Use text facets to clean up the data
   g. Transform updateTime to ISO-compliant YYYY-MM-DD
   h. Apply integrity constraints to remove the zip codes with less than 5 characters
2. Python:
   a. Confirm all FMIDs are unique
   b. Combine season date and season times into SeasonDate and SeasonColumn, respectively
3. SQL:
   a. Export the data and use SQL to separate the dataset into its natural groupings
   b. Confirm the data types conform to the ones specified in the database schema
4. Jupyter:
   a. Export the data and use python libraries to visualize the data

Workflow Model

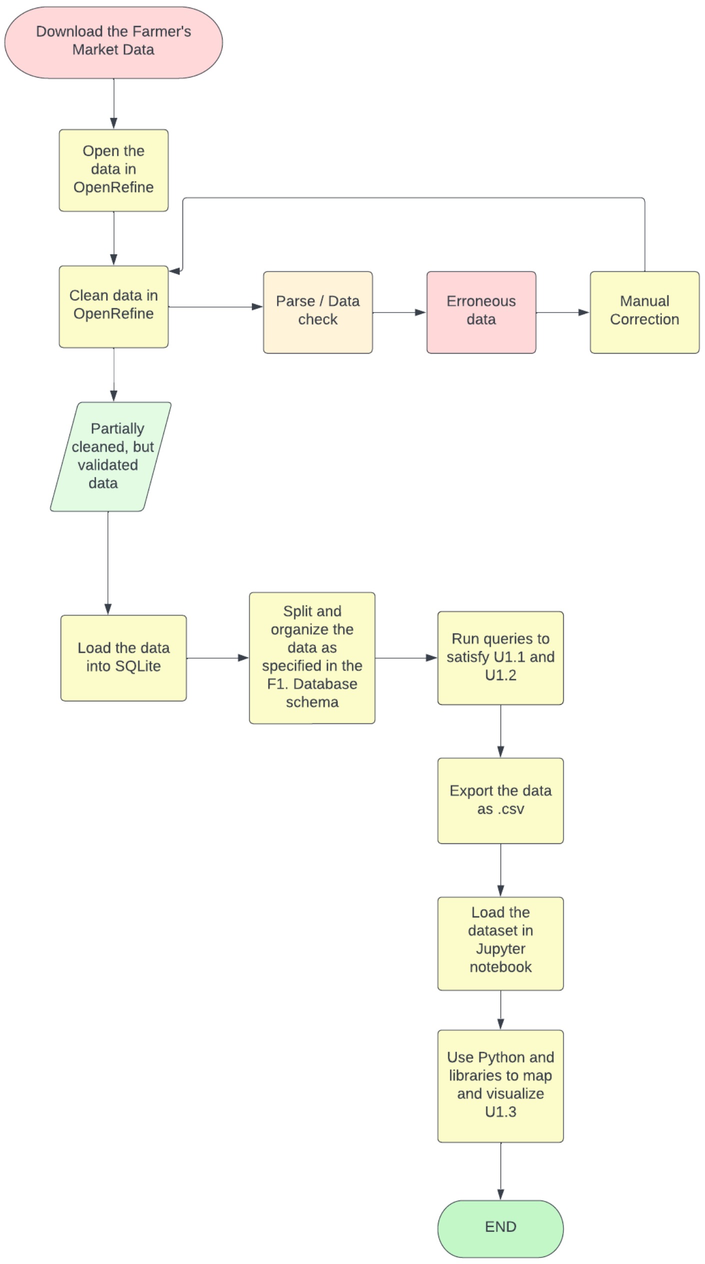

Download the Farmer's Market Data
Open the data in OpenRefine
Clean data in OpenRefine
Parse / Data check
Erroneous data
Manual Correction
Partially cleaned, but validated data
Load the data into SQLite
Split and organize the data as specified in the F1. Database schema
Run queries to satisfy U1.1 and U1.2
Export the data as .csv
Load the dataset in Jupyter notebook
Use Python and libraries to map and visualize U1.3
END

Figure 5 Data Cleaning Workflow

Data Cleaning Steps Performed, Data Improvision, and Workflow Narrative

For this project, we set out to fulfill the U1 goals we set for when the dataset was initially explored in Phase-I. The workflow model initiates the cleaning process once we throughly analyzed the dataset. The next important goal is to develop a strategy to clean our dataset to reach our U1 goal with the tools at our disposal.

As we have had some practice with tools such as OpenRefine, SQL, Jupyter, and Python, earlier in the class, we can use them to clean, organize, and visualize the Farmer's Market database.

During the cleaning process, we redesign and reformat the data, identify important attributes, remove the unnecessary information, remove redundancies, improve consistencies, and remedy missing information. After which, we can use the cleaned data to run queries and visualizations to reach U1 goals.

For the cleaning process, we use the tool, OpenRefine. The goal here is to take the raw Farmers' Market data from its ad-hoc format into a format that can be converted into a relational database. OpenRefine allows us to read and transform the data into a comma-separated-values file, which can be imported into SQLite for our next step.

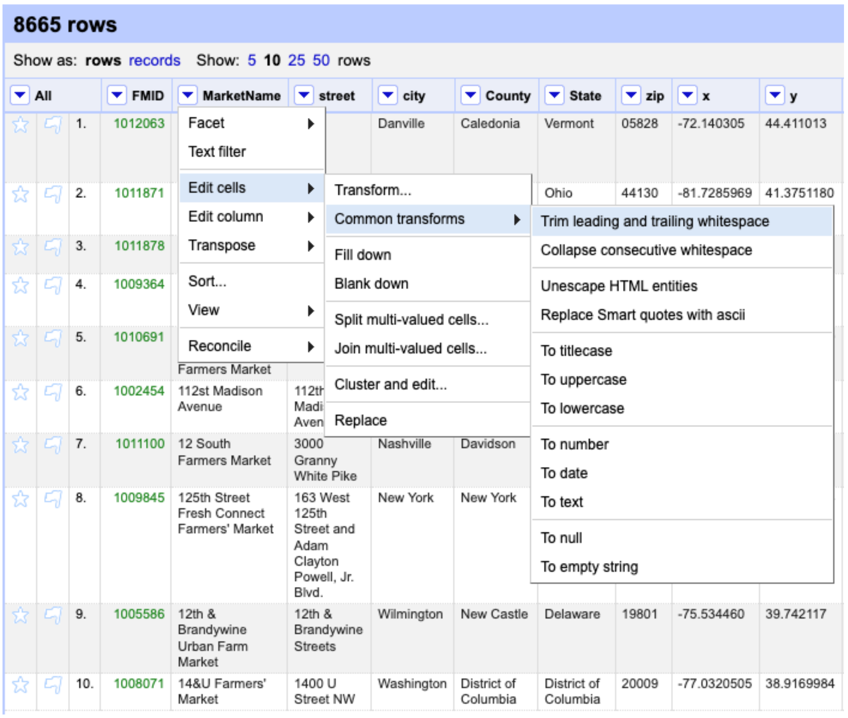
OpenRefine

To begin, we download the raw Farmers-Market data from sample datasets and import into OpenRefine.

We can uncheck "Trim leading & trailing whitespaces from strings", as done in OpenRefine homework. This will give us a better idea of how dirty the data initially is. We also rename the project as "FarmersMarket" for clarity.

Next, let's remove the attributes that are not needed to analyze U1 goals. These attributes have inaccurate, unreliable, and/or unfinished data and this information does not need to be cleaned to answer our U1 questions. The following attributes are removed: *Website, Facebook, Twitter, YouTube, OtherMedia, Location, Season1Date, Season1Time, Season2Date, Season2Time, Season3Date, Season3Time, Season4Date,* and *Season4Time.*

To begin cleaning the remainder to data, trim leading and trailing whitespaces for all attributes as well as collapse consecutive whitespaces for all attributes manually.

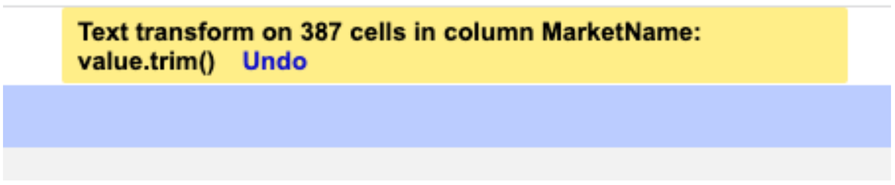


8665 rows

Show as: **rows** records Show: 5 **10** 25 50 rows

All	FMID	MarketName	street	city	County	State	zip	x	y
1.	1012063			Danville	Caledonia	Vermont	05828	-72.140305	44.411013
2.	1011871					Ohio	44130	-81.7285969	41.3751180
3.	1011878								
4.	1009364								
5.	1010691	Farmers Market							
6.	1002454	112st Madison Avenue	112th Madi Aven						
7.	1011100	12 South Farmers Market	3000 Granny White Pike	Nashville	Davidson				
8.	1009845	125th Street Fresh Connect Farmers' Market	163 West 125th Street and Adam Clayton Powell, Jr. Blvd.	New York	New York				
9.	1005586	12th & Brandywine Urban Farm Market	12th & Brandywine Streets	Wilmington	New Castle	Delaware	19801	-75.534460	39.742117
10.	1008071	14&U Farmers' Market	1400 U Street NW	Washington	District of Columbia	District of Columbia	20009	-77.0320505	38.9169984

Figure 6 Trim all leading and trailing whitespaces in *MarketName*



Text transform on 387 cells in column MarketName: value.trim() Undo

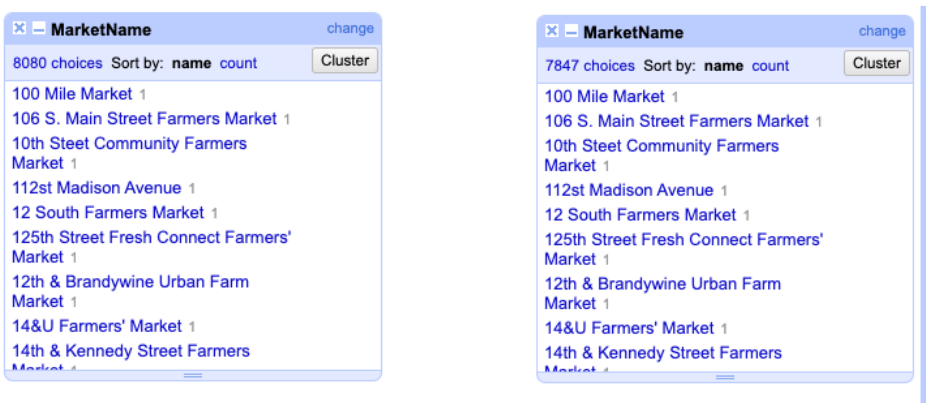
Figure 7 Display of bulk trimming whitespaces in MarketName

Next, we remedy incorrect data types and assign appropriate data types to the data values. The data that will remain as is for attributes: *MarketName, street, city, County, State, and* all the Product values. The values that will be converted into a numeric format is: *FMID, zip, x, y*. We can update the *updateTime* to be a of data type Date.

After the correcting data types, let's rename attributes for consistency: *x* to *Longitude*, *y* to *Latitude*, *street* to *Street*, *city* to *City*, zip to Zip, *Bakedgoods* to *BakedGoods*. We can also make the data values for the following attributes title case: *Street*, *City*, *County*, and *State*.

To further clean this data, we use facets. Facets is a useful tool in OpenRefine. Faceting allows us to look for patterns and trends, as well provides us a big-picture look at the data at hand. We can filter down to specific subset of the dataset and explore it further to analyze or clean.

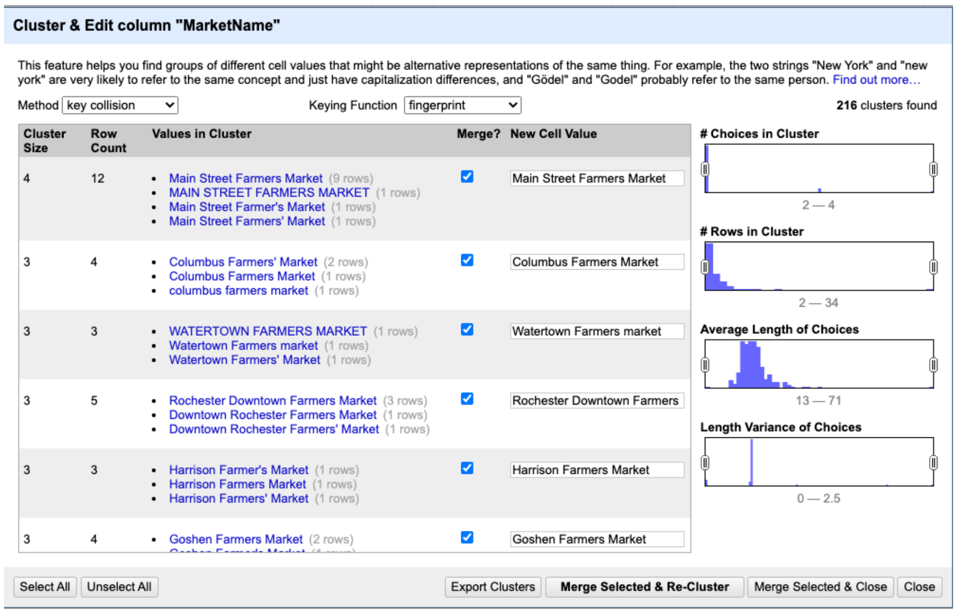
Using the text facet feature, we clean the data values of the following attributes: *MarketName*, *Street*, City, and County. A text facet is relatively straightforward; it compares the contents of all the data values in the relevant column. It makes no educated guesses regarding typos or close matches allows us to assign a new cell value that can merge the closely resembling data into a cohesive cell value.



MarketName change
8080 choices Sort by: name count Cluster
- 100 Mile Market 1
- 106 S. Main Street Farmers Market 1
- 10th Steet Community Farmers Market 1
- 112st Madison Avenue 1
- 12 South Farmers Market 1
- 125th Street Fresh Connect Farmers' Market 1
- 12th & Brandywine Urban Farm Market 1
- 14&U Farmers' Market 1
- 14th & Kennedy Street Farmers

MarketName change
7847 choices Sort by: name count Cluster
- 100 Mile Market 1
- 106 S. Main Street Farmers Market 1
- 10th Steet Community Farmers Market 1
- 112st Madison Avenue 1
- 12 South Farmers Market 1
- 125th Street Fresh Connect Farmers' Market 1
- 12th & Brandywine Urban Farm Market 1
- 14&U Farmers' Market 1
- 14th & Kennedy Street Farmers

Figure 8 *MarketName* before (L) and after (R) text facet and clustering modification. Note the number of choices.



Cluster & Edit column "MarketName"

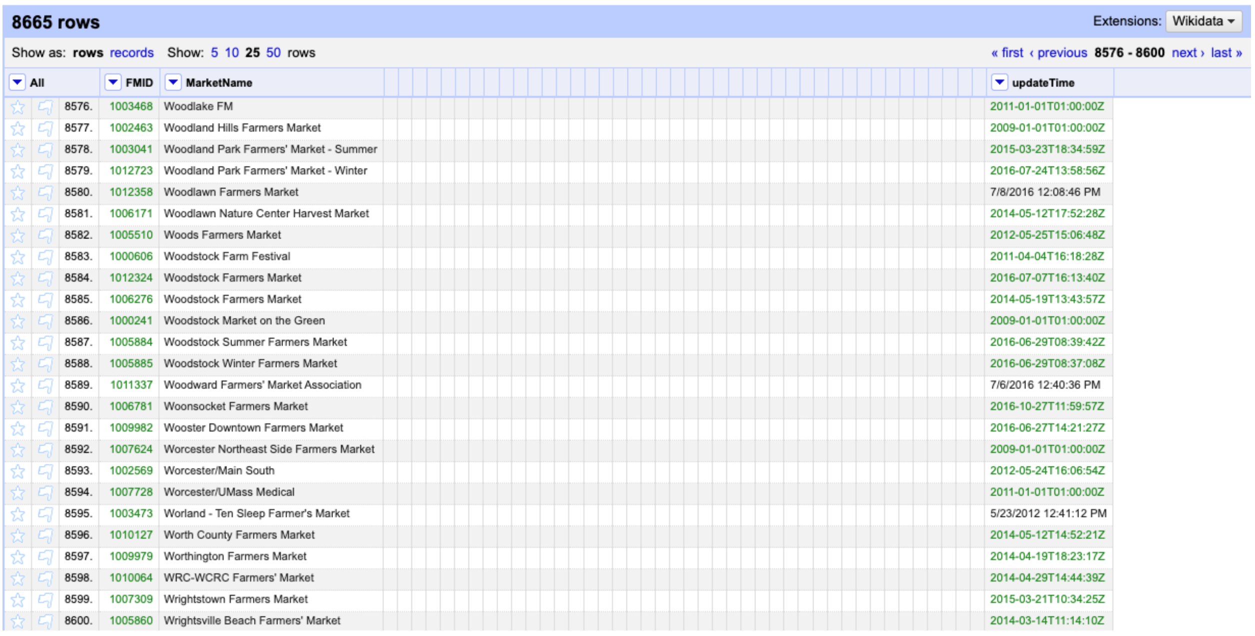
This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. Find out more...

Method: key collision   Keying Function: fingerprint   216 clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
4	12	Main Street Farmers Market (9 rows); MAIN STREET FARMERS MARKET (1 rows); Main Street Farmer's Market (1 rows); Main Street Farmers' Market (1 rows)	✓	Main Street Farmers Market
3	4	Columbus Farmers' Market (2 rows); Columbus Farmers Market (1 rows); columbus farmers market (1 rows)	✓	Columbus Farmers Market
3	3	WATERTOWN FARMERS MARKET (1 rows); Watertown Farmers market (1 rows); Watertown Farmers' Market (1 rows)	✓	Watertown Farmers market
3	5	Rochester Downtown Farmers Market (3 rows); Downtown Rochester Farmers Market (1 rows); Downtown Rochester Farmers' Market (1 rows)	✓	Rochester Downtown Farmers
3	3	Harrison Farmer's Market (1 rows); Harrison Farmers Market (1 rows); Harrison Farmers' Market (1 rows)	✓	Harrison Farmers Market
3	4	Goshen Farmers Market (2 rows)	✓	Goshen Farmers Market

Figure 9 *MarketName* while clustering ambiguous values

Next, let’s convert the *updateTime* attribute to ISO-complaint format date, YYYY-MM-DD. Although, the date data type converts the *updateTime* column into YYYY-MM-DDTHH:MM:SSZ format, it also contains the time information that we do not need.



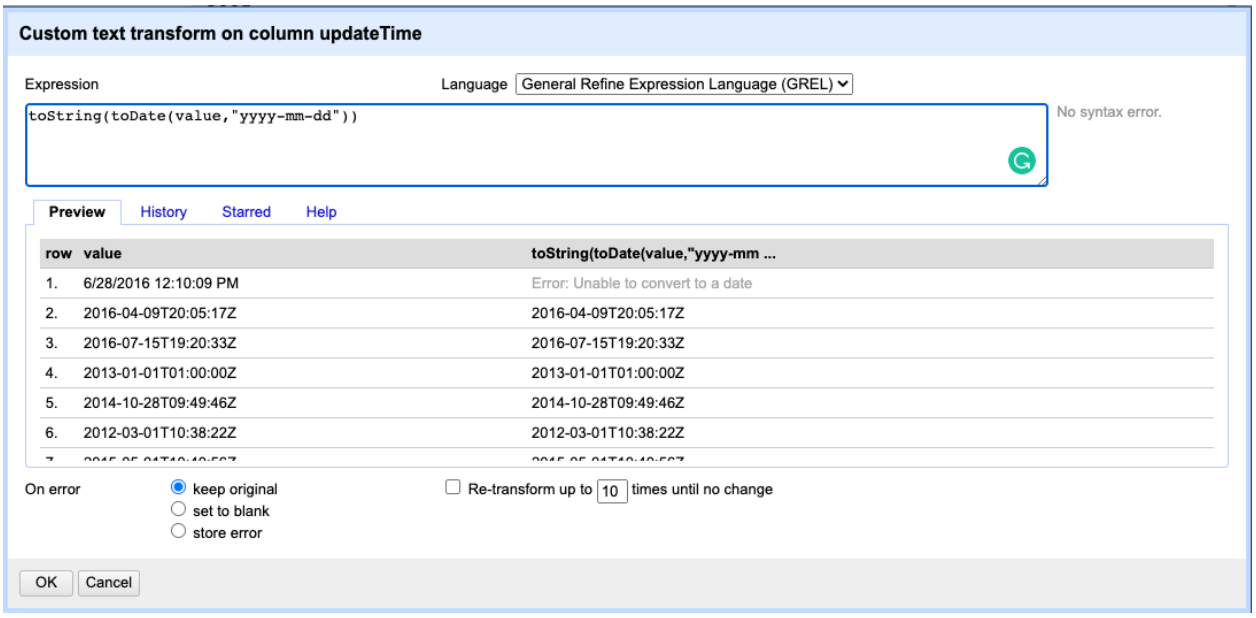
8665 rows   Extensions: Wikidata

Show as: rows records   Show: 5 10 25 50 rows   « first ‹ previous 8576 - 8600 next › last »

All	FMID	MarketName	updateTime
8576.	1003468	Woodlake FM	2011-01-01T01:00:00Z
8577.	1002463	Woodland Hills Farmers Market	2009-01-01T01:00:00Z
8578.	1003041	Woodland Park Farmers' Market - Summer	2015-03-23T18:34:59Z
8579.	1012723	Woodland Park Farmers' Market - Winter	2016-07-24T13:58:56Z
8580.	1012358	Woodlawn Farmers Market	7/8/2016 12:08:46 PM
8581.	1006171	Woodlawn Nature Center Harvest Market	2014-05-12T17:52:28Z
8582.	1005510	Woods Farmers Market	2012-05-25T15:06:48Z
8583.	1000606	Woodstock Farm Festival	2011-04-04T16:18:28Z
8584.	1012324	Woodstock Farmers Market	2016-07-07T16:13:40Z
8585.	1006276	Woodstock Farmers Market	2014-05-19T13:43:57Z
8586.	1000241	Woodstock Market on the Green	2009-01-01T01:00:00Z
8587.	1005884	Woodstock Summer Farmers Market	2016-06-29T08:39:42Z
8588.	1005885	Woodstock Winter Farmers Market	2016-06-29T08:37:08Z
8589.	1011337	Woodward Farmers' Market Association	7/6/2016 12:40:36 PM
8590.	1006781	Woonsocket Farmers Market	2016-10-27T11:59:57Z
8591.	1009982	Wooster Downtown Farmers Market	2016-06-27T14:21:27Z
8592.	1007624	Worcester Northeast Side Farmers Market	2009-01-01T01:00:00Z
8593.	1002569	Worcester/Main South	2012-05-24T16:06:54Z
8594.	1007728	Worcester/UMass Medical	2011-01-01T01:00:00Z
8595.	1003473	Worland - Ten Sleep Farmer's Market	5/23/2012 12:41:12 PM
8596.	1010127	Worth County Farmers Market	2014-05-12T14:52:21Z
8597.	1009979	Worthington Farmers Market	2014-04-19T18:23:17Z
8598.	1010064	WRC-WCRC Farmers' Market	2014-04-29T14:44:39Z
8599.	1007309	Wrightstown Farmers Market	2015-03-21T10:34:25Z
8600.	1005860	Wrightsville Beach Farmers' Market	2014-03-14T11:14:10Z

Figure 10 *updateTime* after conversion to Date format

We can clean this column by applying the following GREL command toString(toDate(value, “YYYY-MM-DD”)) converting the string to date in the YYYY-MM-DD format.



Custom text transform on column updateTime

Expression   Language: General Refine Expression Language (GREL)

```
toString(toDate(value,"yyyy-mm-dd"))
```

No syntax error.

Preview | History | Starred | Help

row	value	toString(toDate(value,"yyyy-mm ...
1.	6/28/2016 12:10:09 PM	Error: Unable to convert to a date
2.	2016-04-09T20:05:17Z	2016-04-09T20:05:17Z
3.	2016-07-15T19:20:33Z	2016-07-15T19:20:33Z
4.	2013-01-01T01:00:00Z	2013-01-01T01:00:00Z
5.	2014-10-28T09:49:46Z	2014-10-28T09:49:46Z
6.	2012-03-01T10:38:22Z	2012-03-01T10:38:22Z

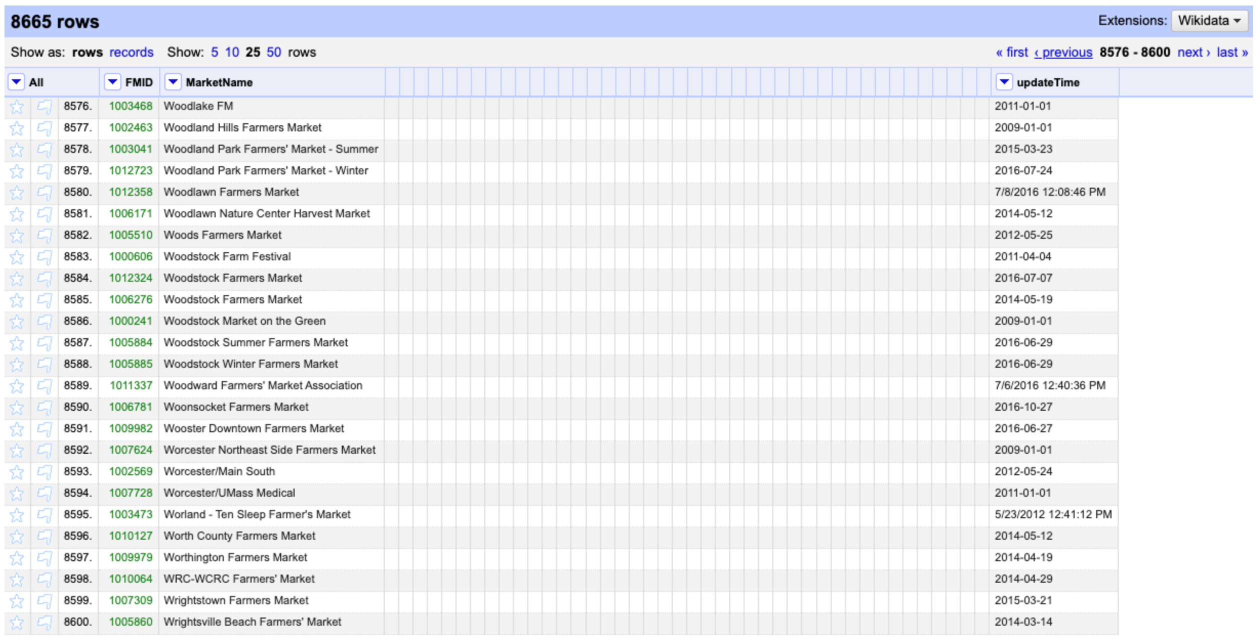
On error: (● keep original) ○ set to blank ○ store error

☐ Re-transform up to 10 times until no change

OK Cancel

Figure 11 Applying GREL to *updateTime*

Here we run into an issue where any time between 12 PM to 1 PM was giving an "invalid date" error and we could not apply the GREL command. Note, as shown in Figure 12, that not all dates were able to take on the YYYY-MM-DD format.


8665 rows

Extensions: Wikidata

Show as: rows records Show: 5 10 25 50 rows

« first ‹ previous 8576 - 8600 next › last »

All	FMID	MarketName	updateTime
8576.	1003468	Woodlake FM	2011-01-01
8577.	1002463	Woodland Hills Farmers Market	2009-01-01
8578.	1003041	Woodland Park Farmers' Market - Summer	2015-03-23
8579.	1012723	Woodland Park Farmers' Market - Winter	2016-07-24
8580.	1012358	Woodlawn Farmers Market	7/8/2016 12:08:46 PM
8581.	1006171	Woodlawn Nature Center Harvest Market	2014-05-12
8582.	1005510	Woods Farmers Market	2012-05-25
8583.	1000606	Woodstock Farm Festival	2011-04-04
8584.	1012324	Woodstock Farmers Market	2016-07-07
8585.	1006276	Woodstock Farmers Market	2014-05-19
8586.	1000241	Woodstock Market on the Green	2009-01-01
8587.	1005884	Woodstock Summer Farmers Market	2016-06-29
8588.	1005885	Woodstock Winter Farmers Market	2016-06-29
8589.	1011337	Woodward Farmers' Market Association	7/6/2016 12:40:36 PM
8590.	1006781	Woonsocket Farmers Market	2016-10-27
8591.	1009982	Wooster Downtown Farmers Market	2016-06-27
8592.	1007624	Worcester Northeast Side Farmers Market	2009-01-01
8593.	1002569	Worcester/Main South	2012-05-24
8594.	1007728	Worcester/UMass Medical	2011-01-01
8595.	1003473	Worland - Ten Sleep Farmer's Market	5/23/2012 12:41:12 PM
8596.	1010127	Worth County Farmers Market	2014-05-12
8597.	1009979	Worthington Farmers Market	2014-04-19
8598.	1010064	WRC-WCRC Farmers' Market	2014-04-29
8599.	1007309	Wrightstown Farmers Market	2015-03-21
8600.	1005860	Wrightsville Beach Farmers' Market	2014-03-14

Figure 12 *updateTime* after applying the GREL command

To resolve this, we can remove the keyword "PM" from all updateTime data values, using the command: replace(value, "PM", "") which replaces all the "PM" instances in *updateTime* with blanks.

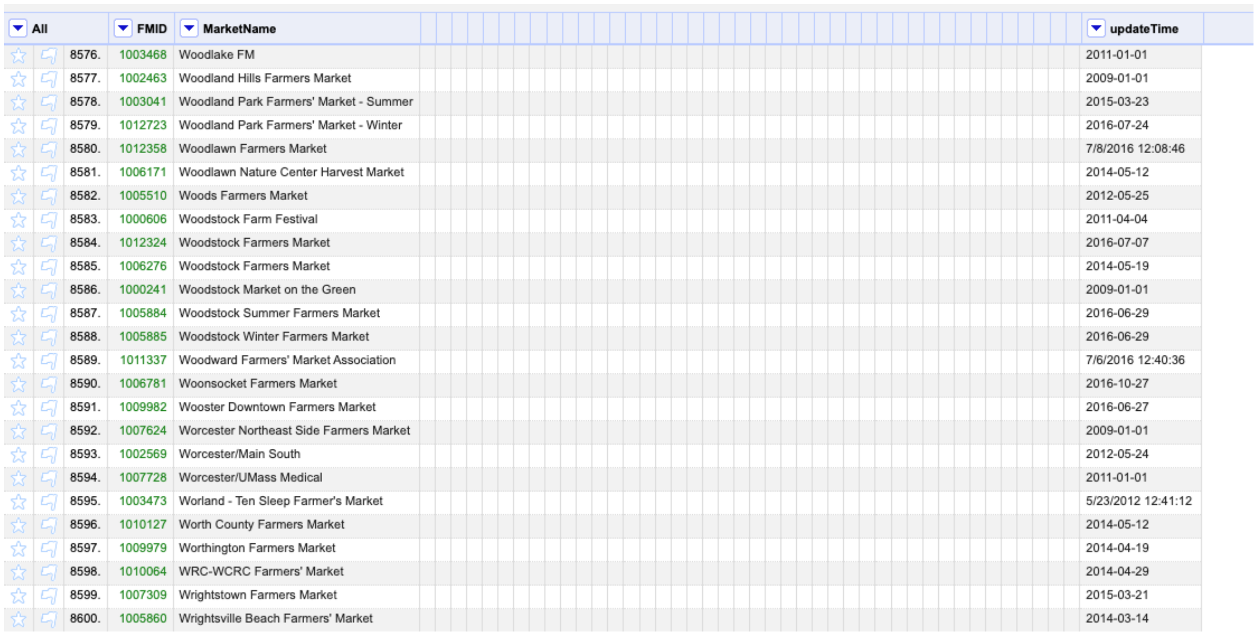
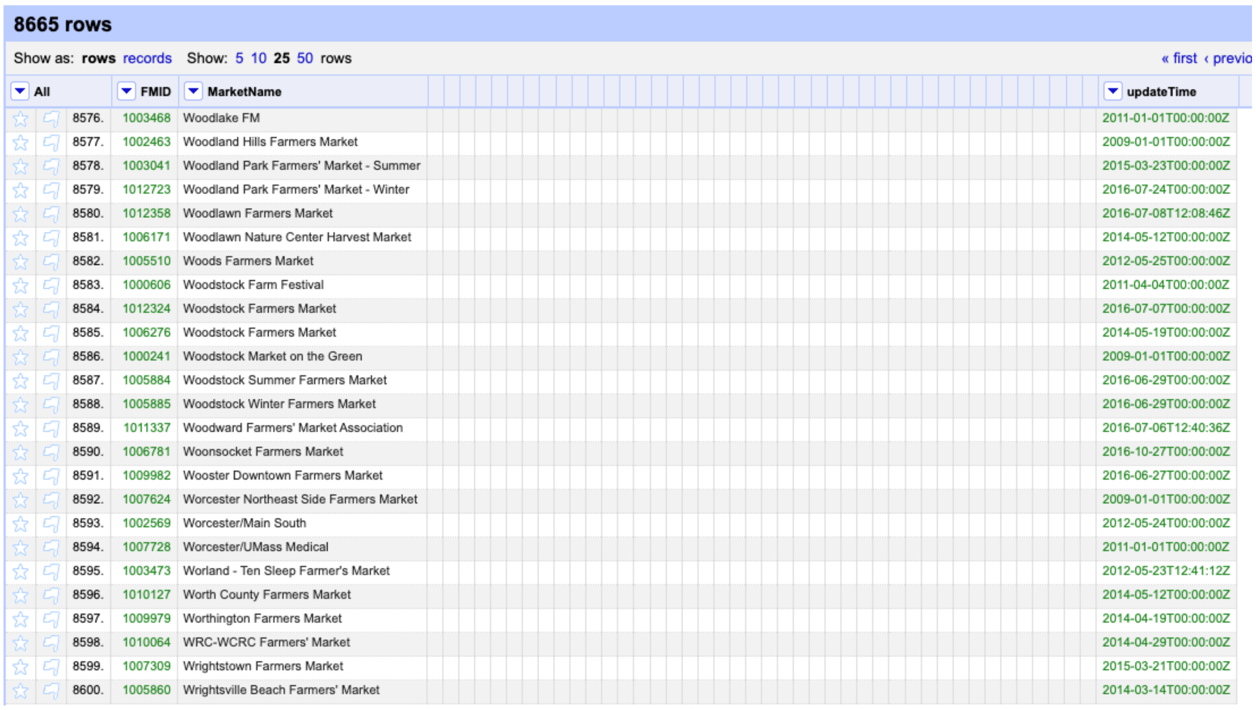

All	FMID	MarketName	updateTime
8576.	1003468	Woodlake FM	2011-01-01
8577.	1002463	Woodland Hills Farmers Market	2009-01-01
8578.	1003041	Woodland Park Farmers' Market - Summer	2015-03-23
8579.	1012723	Woodland Park Farmers' Market - Winter	2016-07-24
8580.	1012358	Woodlawn Farmers Market	7/8/2016 12:08:46
8581.	1006171	Woodlawn Nature Center Harvest Market	2014-05-12
8582.	1005510	Woods Farmers Market	2012-05-25
8583.	1000606	Woodstock Farm Festival	2011-04-04
8584.	1012324	Woodstock Farmers Market	2016-07-07
8585.	1006276	Woodstock Farmers Market	2014-05-19
8586.	1000241	Woodstock Market on the Green	2009-01-01
8587.	1005884	Woodstock Summer Farmers Market	2016-06-29
8588.	1005885	Woodstock Winter Farmers Market	2016-06-29
8589.	1011337	Woodward Farmers' Market Association	7/6/2016 12:40:36
8590.	1006781	Woonsocket Farmers Market	2016-10-27
8591.	1009982	Wooster Downtown Farmers Market	2016-06-27
8592.	1007624	Worcester Northeast Side Farmers Market	2009-01-01
8593.	1002569	Worcester/Main South	2012-05-24
8594.	1007728	Worcester/UMass Medical	2011-01-01
8595.	1003473	Worland - Ten Sleep Farmer's Market	5/23/2012 12:41:12
8596.	1010127	Worth County Farmers Market	2014-05-12
8597.	1009979	Worthington Farmers Market	2014-04-19
8598.	1010064	WRC-WCRC Farmers' Market	2014-04-29
8599.	1007309	Wrightstown Farmers Market	2015-03-21
8600.	1005860	Wrightsville Beach Farmers' Market	2014-03-14

Figure 13 *updateTime* after removing all instances of "PM"

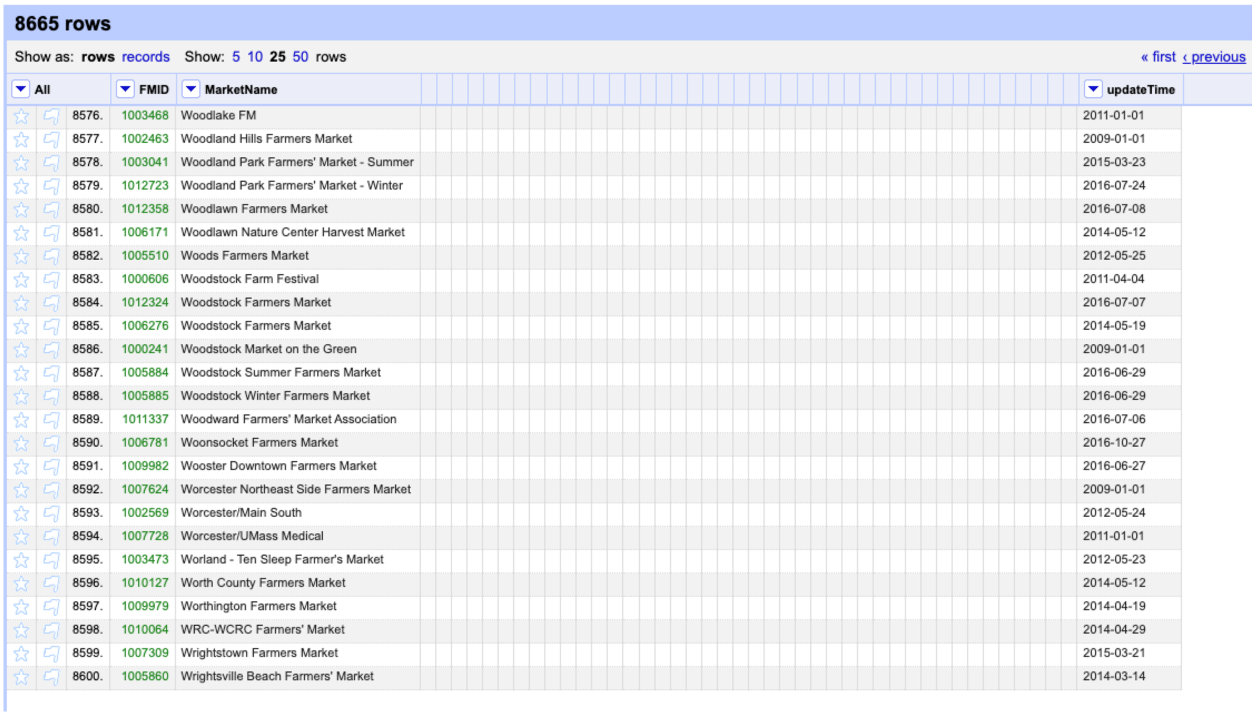
After the change, we can apply both, Date data type and GREL command toString(toDate(value, “YYYY-MM-DD”) again on the *updateTime* attribute which resolves the issue and converts all updateTime to YYYY-MM-DD format.


8665 rows

Show as: **rows** records Show: 5 10 **25** 50 rows « first ‹ previo

All	FMID	MarketName	updateTime
8576.	1003468	Woodlake FM	2011-01-01T00:00:00Z
8577.	1002463	Woodland Hills Farmers Market	2009-01-01T00:00:00Z
8578.	1003041	Woodland Park Farmers' Market - Summer	2015-03-23T00:00:00Z
8579.	1012723	Woodland Park Farmers' Market - Winter	2016-07-24T00:00:00Z
8580.	1012358	Woodlawn Farmers Market	2016-07-08T12:08:46Z
8581.	1006171	Woodlawn Nature Center Harvest Market	2014-05-12T00:00:00Z
8582.	1005510	Woods Farmers Market	2012-05-25T00:00:00Z
8583.	1000606	Woodstock Farm Festival	2011-04-04T00:00:00Z
8584.	1012324	Woodstock Farmers Market	2016-07-07T00:00:00Z
8585.	1006276	Woodstock Farmers Market	2014-05-19T00:00:00Z
8586.	1000241	Woodstock Market on the Green	2009-01-01T00:00:00Z
8587.	1005884	Woodstock Summer Farmers Market	2016-06-29T00:00:00Z
8588.	1005885	Woodstock Winter Farmers Market	2016-06-29T00:00:00Z
8589.	1011337	Woodward Farmers' Market Association	2016-07-06T12:40:36Z
8590.	1006781	Woonsocket Farmers Market	2016-10-27T00:00:00Z
8591.	1009982	Wooster Downtown Farmers Market	2016-06-27T00:00:00Z
8592.	1007624	Worcester Northeast Side Farmers Market	2009-01-01T00:00:00Z
8593.	1002569	Worcester/Main South	2012-05-24T00:00:00Z
8594.	1007728	Worcester/UMass Medical	2011-01-01T00:00:00Z
8595.	1003473	Worland - Ten Sleep Farmer's Market	2012-05-23T12:41:12Z
8596.	1010127	Worth County Farmers Market	2014-05-12T00:00:00Z
8597.	1009979	Worthington Farmers Market	2014-04-19T00:00:00Z
8598.	1010064	WRC-WCRC Farmers' Market	2014-04-29T00:00:00Z
8599.	1007309	Wrightstown Farmers Market	2015-03-21T00:00:00Z
8600.	1005860	Wrightsville Beach Farmers' Market	2014-03-14T00:00:00Z

Figure 14 *updateTime* after applying the Date data type


8665 rows

Show as: **rows** records Show: 5 10 **25** 50 rows « first ‹ previous

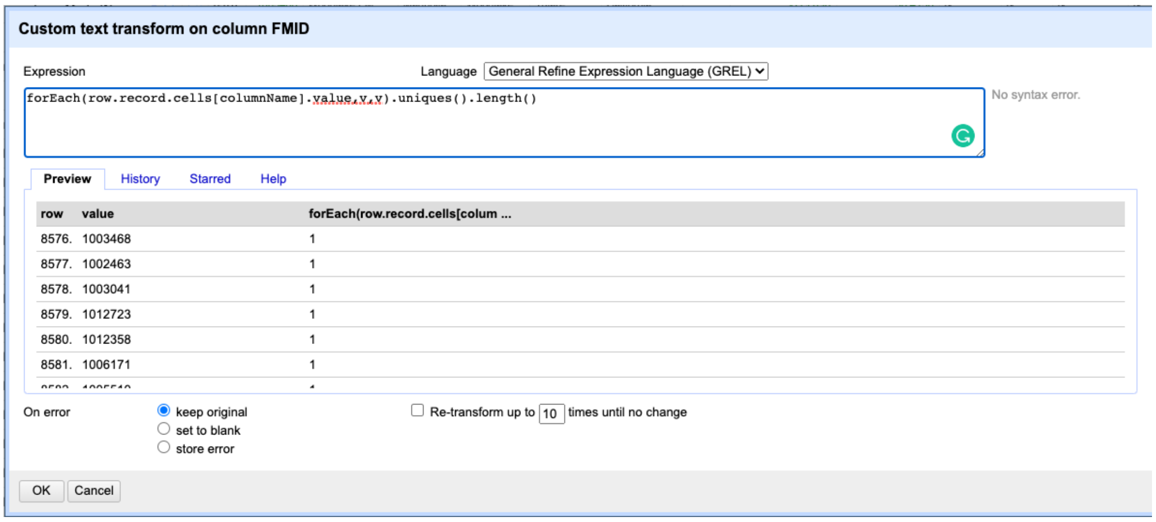
All	FMID	MarketName	updateTime
8576.	1003468	Woodlake FM	2011-01-01
8577.	1002463	Woodland Hills Farmers Market	2009-01-01
8578.	1003041	Woodland Park Farmers' Market - Summer	2015-03-23
8579.	1012723	Woodland Park Farmers' Market - Winter	2016-07-24
8580.	1012358	Woodlawn Farmers Market	2016-07-08
8581.	1006171	Woodlawn Nature Center Harvest Market	2014-05-12
8582.	1005510	Woods Farmers Market	2012-05-25
8583.	1000606	Woodstock Farm Festival	2011-04-04
8584.	1012324	Woodstock Farmers Market	2016-07-07
8585.	1006276	Woodstock Farmers Market	2014-05-19
8586.	1000241	Woodstock Market on the Green	2009-01-01
8587.	1005884	Woodstock Summer Farmers Market	2016-06-29
8588.	1005885	Woodstock Winter Farmers Market	2016-06-29
8589.	1011337	Woodward Farmers' Market Association	2016-07-06
8590.	1006781	Woonsocket Farmers Market	2016-10-27
8591.	1009982	Wooster Downtown Farmers Market	2016-06-27
8592.	1007624	Worcester Northeast Side Farmers Market	2009-01-01
8593.	1002569	Worcester/Main South	2012-05-24
8594.	1007728	Worcester/UMass Medical	2011-01-01
8595.	1003473	Worland - Ten Sleep Farmer's Market	2012-05-23
8596.	1010127	Worth County Farmers Market	2014-05-12
8597.	1009979	Worthington Farmers Market	2014-04-19
8598.	1010064	WRC-WCRC Farmers' Market	2014-04-29
8599.	1007309	Wrightstown Farmers Market	2015-03-21
8600.	1005860	Wrightsville Beach Farmers' Market	2014-03-14

Figure 15 *updateTime* after applying the GREL command again

Next, let's verify that all FMID values are unique.

To do this we will use a GREL expression:
forEach(row.record.cells[columnName].value,v,v).uniques().length() where we create an array for each record, remove all duplicates within the array and count the number of elements in the array. The result of the expression 0 would mean that there are no values in the record, 1 would mean that there is only one unique value assigned to each entry in the record, and 2 or more means that there is more than one value assigned, therefore it is not unique.

When running this on the FMID attribute, we can observe that all FMID values turn into 1, meaning they are all unique.



Custom text transform on column FMID

Expression   Language: General Refine Expression Language (GREL)

```
forEach(row.record.cells[columnName].value,v,v).uniques().length()
```

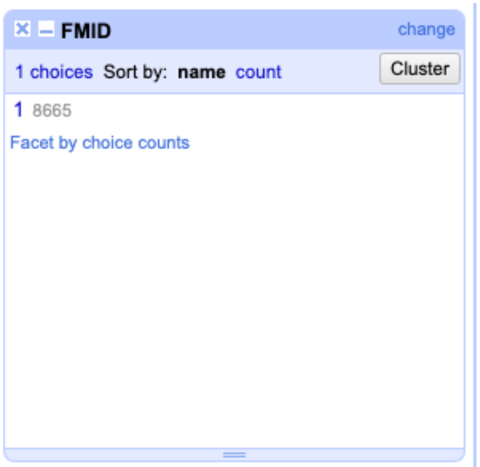
No syntax error.

row	value	forEach(row.record.cells[colum ...
8576.	1003468	1
8577.	1002463	1
8578.	1003041	1
8579.	1012723	1
8580.	1012358	1
8581.	1006171	1

On error: [✓ keep original] set to blank / store error

Re-transform up to 10 times until no change

Figure 16 Applying GREL command to FMID



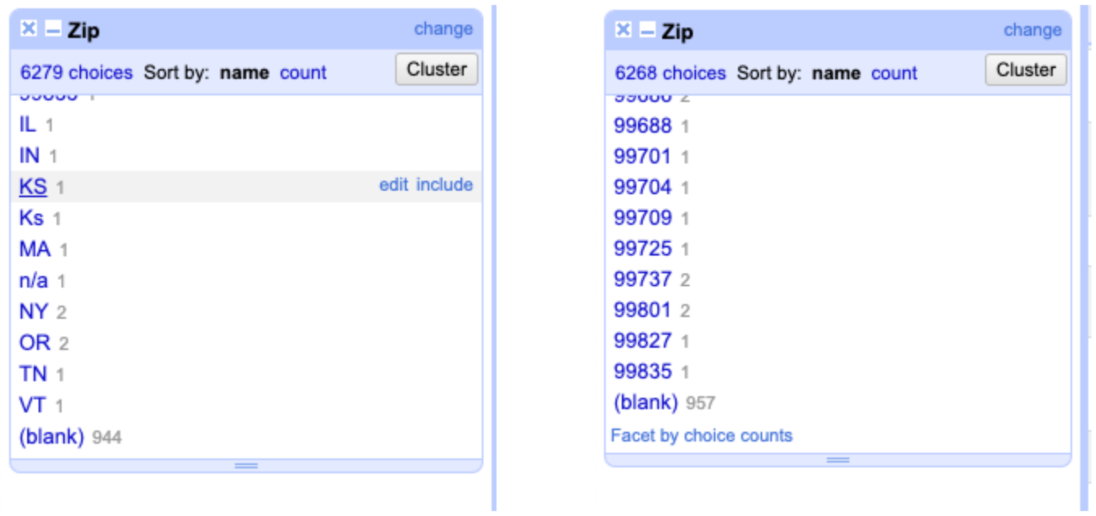
FMID

1 choices Sort by: name count

1	8665

Figure 17 Confirming all FMID values are 1 (hence unique)

Detailed observation of the *Zip* data values shows that there are text entries present. As the data type of *Zip* is numeric, it makes sense here to change the text values into blanks for consistency. As we are not actually utilizing the *Zip* attribute for U1 queries, assigning the values as blanks will not have any substantial impact. However, if in the future, we need to reference these attribute values, it will be ready. Manually remove the text entries using a text facet as shown below.



Zip change
6279 choices Sort by: name count Cluster

IL	1
IN	1
KS	1
Ks	1
MA	1
n/a	1
NY	2
OR	2
TN	1
VT	1
(blank)	944

Zip change
6268 choices Sort by: name count Cluster

99688	1
99701	1
99704	1
99709	1
99725	1
99737	2
99801	2
99827	1
99835	1
(blank)	957

Facet by choice counts

Figure 18 *Zip* entries using a text facet before (L) and after (R)

This concludes the cleaning performed using OpenRefine. We can export the cleaned data as a .csv file to perform further organization, run queries and visualizations using SQLite, Jupyter notebook, and Python.

SQLite

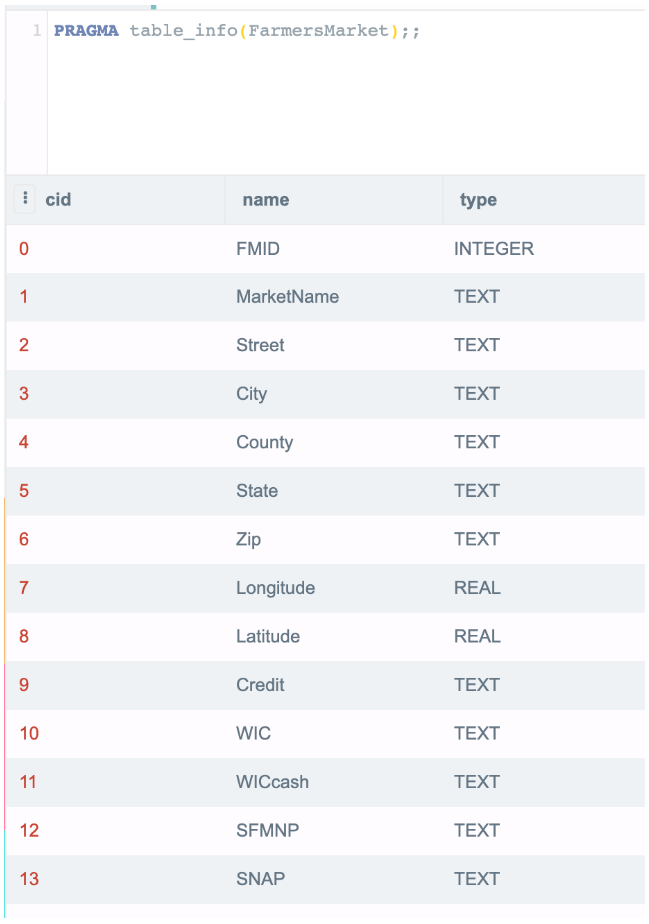
In the second part cleaning the Farmers' Market data, we use SQLite to further organize our data and run queries to satisfy our U1 goals and get supporting data to visualize the U1 in Jupyter notebook.

SQLite runs SQL, which is defined as **S**tructured **Q**uery Language, is a standardized programming language that is used for managing relational databases and carrying out different operations on them. The reason for using SQLite is that it is built for swiftly querying large datasets, potentially millions of rows and columns. It will also provide an avenue to build queries so we can easily search our database or subsets within our database. SQLite will also allow us to further organize our data into multiple easy to read relations.

Load into SQLite. For this project, we are using sqliteonline.com, a browser-based SQLite software.

To begin, import the downloaded .csv file from OpenRefine into the SQLite software. This opens the cleaned Farmers' Market dataset.

To organize the data into the schema that we created earlier in *Figure 1* and to verify the data types are correct, we can run the following command on the database: PRAGMA table_info(FarmersMarket);



```
PRAGMA table_info(FarmersMarket);;
```

cid	name	type
0	FMID	INTEGER
1	MarketName	TEXT
2	Street	TEXT
3	City	TEXT
4	County	TEXT
5	State	TEXT
6	Zip	TEXT
7	Longitude	REAL
8	Latitude	REAL
9	Credit	TEXT
10	WIC	TEXT
11	WICcash	TEXT
12	SFMNP	TEXT
13	SNAP	TEXT

Figure 19 Verifying the datatypes of imported Farmers' Market database

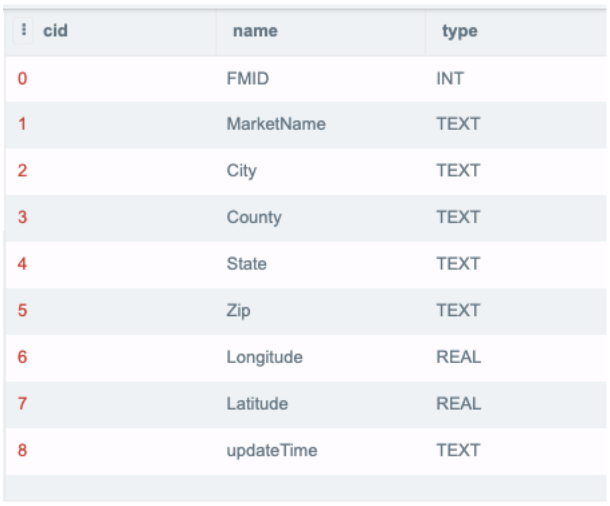
Once the data types are correctly verified, we can break apart the imported FarmersMarket dataset into multiple relations. As shown in *Figure 1*, the Farmers' Market dataset will be divided into *Market*, *Payment*, and *Products*. These different relations are to contain all definitive information pertaining to its category, connected to other relations using the FMID as PK.

To create the relation *Market*:

```
CREATE TABLE Market AS

    SELECT fmid, marketname, city, county, state, zip, longitude, latitude, updatetime

    FROM FarmersMarket;
```



cid	name	type
0	FMID	INT
1	MarketName	TEXT
2	City	TEXT
3	County	TEXT
4	State	TEXT
5	Zip	TEXT
6	Longitude	REAL
7	Latitude	REAL
8	updateTime	TEXT

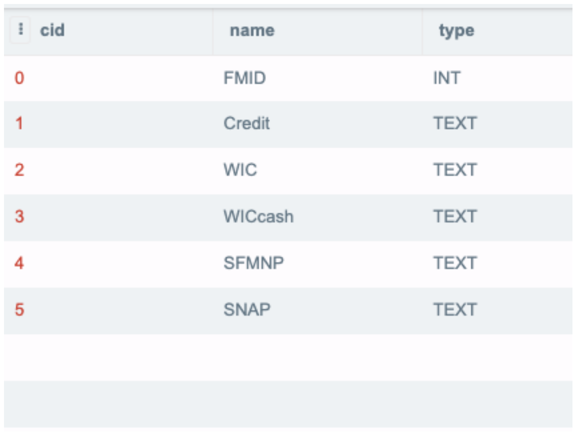
Figure 20 *Market* Relation in SQLite

To create the relation *Payment*:

```
CREATE TABLE Payment AS

    SELECT fmid, credit, wic, wiccash, sfmnp, snap

    FROM FarmersMarket;
```



cid	name	type
0	FMID	INT
1	Credit	TEXT
2	WIC	TEXT
3	WICcash	TEXT
4	SFMNP	TEXT
5	SNAP	TEXT

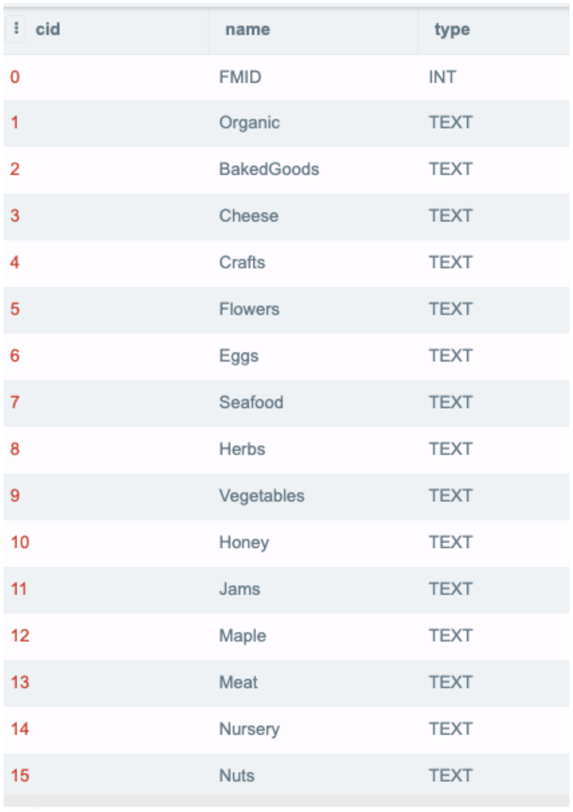
Figure 21 *Payment* Relation in SQLite

To create the relation *Products*:

```
CREATE TABLE Products AS

  SELECT     fmid, organic, bakedgoods, cheese, crafts, flowers, eggs, seafood, herbs,
             vegetables, honey, jams, maple, meat, nursery, nuts, plants, poultry, soap,
             trees, wine, coffee, beans, fruits, grains, juices, mushrooms, petfood, tofu,
             wildharvested

    FROM FarmersMarket;
```



cid	name	type
0	FMID	INT
1	Organic	TEXT
2	BakedGoods	TEXT
3	Cheese	TEXT
4	Crafts	TEXT
5	Flowers	TEXT
6	Eggs	TEXT
7	Seafood	TEXT
8	Herbs	TEXT
9	Vegetables	TEXT
10	Honey	TEXT
11	Jams	TEXT
12	Maple	TEXT
13	Meat	TEXT
14	Nursery	TEXT
15	Nuts	TEXT

Figure 21 *Products* Relation in SQLite

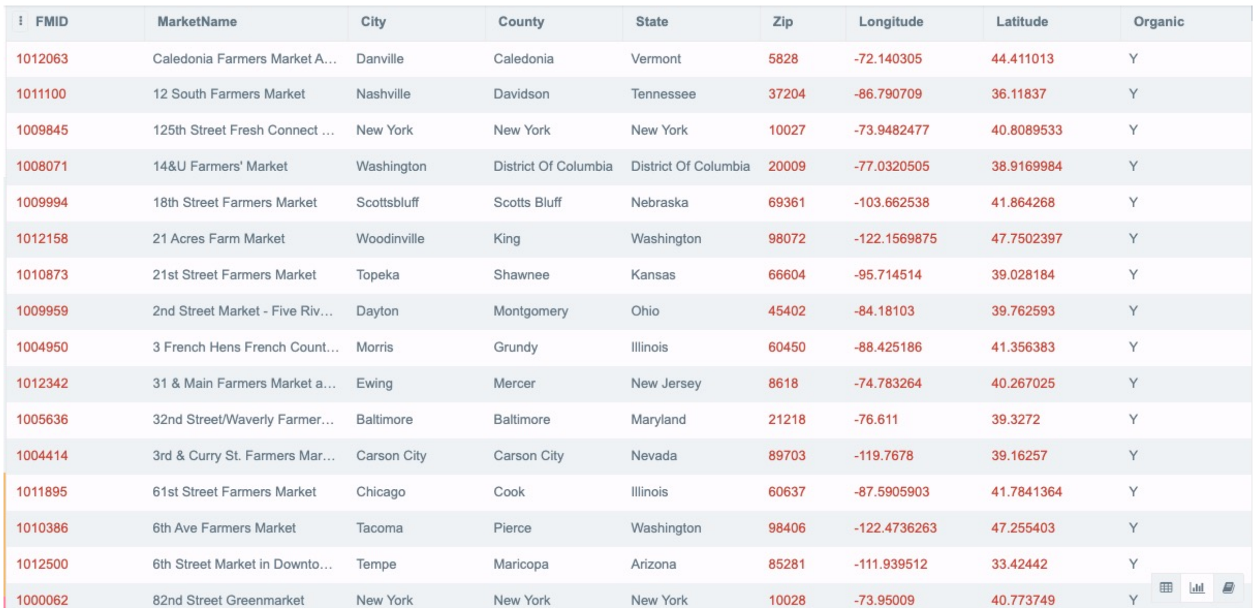
After the all the data is organized into its own relation, we can remove the imported *FamersMarket* by using:

```
DROP TABLE FarmersMarket;
```

At this time, the data is fully cleaned and organized. Hence, we can begin running queries to answer U1 questions

U1 Question: "Which Farmer's Markets sell organic produce?"

```
SELECT      Market.fmid, Market.MarketName, Market.City, Market.County,
            Market.State, Market.Zip, Market.Longitude, Market.Latitude,
            Products.organic
FROM Market
INNER JOIN Products
ON Market.fmid = Products.fmid
WHERE Products.organic = 'Y';
```

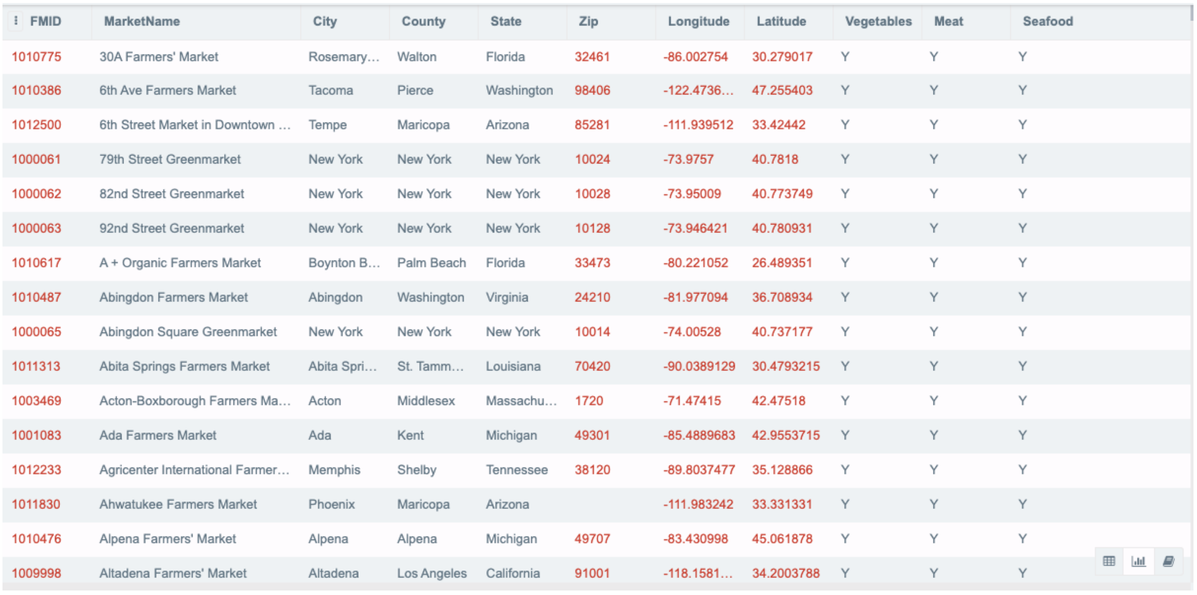


FMID	MarketName	City	County	State	Zip	Longitude	Latitude	Organic
1012063	Caledonia Farmers Market A...	Danville	Caledonia	Vermont	5828	-72.140305	44.411013	Y
1011100	12 South Farmers Market	Nashville	Davidson	Tennessee	37204	-86.790709	36.11837	Y
1009845	125th Street Fresh Connect ...	New York	New York	New York	10027	-73.9482477	40.8089533	Y
1008071	14&U Farmers' Market	Washington	District Of Columbia	District Of Columbia	20009	-77.0320505	38.9169984	Y
1009994	18th Street Farmers Market	Scottsbluff	Scotts Bluff	Nebraska	69361	-103.662538	41.864268	Y
1012158	21 Acres Farm Market	Woodinville	King	Washington	98072	-122.1569875	47.7502397	Y
1010873	21st Street Farmers Market	Topeka	Shawnee	Kansas	66604	-95.714514	39.028184	Y
1009959	2nd Street Market - Five Riv...	Dayton	Montgomery	Ohio	45402	-84.18103	39.762593	Y
1004950	3 French Hens French Count...	Morris	Grundy	Illinois	60450	-88.425186	41.356383	Y
1012342	31 & Main Farmers Market a...	Ewing	Mercer	New Jersey	8618	-74.783264	40.267025	Y
1005636	32nd Street/Waverly Farmer...	Baltimore	Baltimore	Maryland	21218	-76.611	39.3272	Y
1004414	3rd & Curry St. Farmers Mar...	Carson City	Carson City	Nevada	89703	-119.7678	39.16257	Y
1011895	61st Street Farmers Market	Chicago	Cook	Illinois	60637	-87.5905903	41.7841364	Y
1010386	6th Ave Farmers Market	Tacoma	Pierce	Washington	98406	-122.4736263	47.255403	Y
1012500	6th Street Market in Downto...	Tempe	Maricopa	Arizona	85281	-111.939512	33.42442	Y
1000062	82nd Street Greenmarket	New York	New York	New York	10028	-73.95009	40.773749	Y

Figure 22 Query showing all Farmers' Markets that sell organic produce

U1 Question: "Which Farmer's Markets sell vegetables, meats, and seafood?"

```
SELECT        Market.fmid, Market.MarketName, Market.City, Market.County,
              Market.State, Market.Zip, Market.Longitude, Market.Latitude,
              Products.organic
FROM Market
INNER JOIN Products
ON Market.fmid = Products.fmid
WHERE Products.vegetables = 'Y' AND Products.meat = 'Y' AND Products.seafood = 'Y';
```



FMID	MarketName	City	County	State	Zip	Longitude	Latitude	Vegetables	Meat	Seafood
1010775	30A Farmers' Market	Rosemary…	Walton	Florida	32461	-86.002754	30.279017	Y	Y	Y
1010386	6th Ave Farmers Market	Tacoma	Pierce	Washington	98406	-122.4736…	47.255403	Y	Y	Y
1012500	6th Street Market in Downtown …	Tempe	Maricopa	Arizona	85281	-111.939512	33.42442	Y	Y	Y
1000061	79th Street Greenmarket	New York	New York	New York	10024	-73.9757	40.7818	Y	Y	Y
1000062	82nd Street Greenmarket	New York	New York	New York	10028	-73.95009	40.773749	Y	Y	Y
1000063	92nd Street Greenmarket	New York	New York	New York	10128	-73.946421	40.780931	Y	Y	Y
1010617	A + Organic Farmers Market	Boynton B…	Palm Beach	Florida	33473	-80.221052	26.489351	Y	Y	Y
1010487	Abingdon Farmers Market	Abingdon	Washington	Virginia	24210	-81.977094	36.708934	Y	Y	Y
1000065	Abingdon Square Greenmarket	New York	New York	New York	10014	-74.00528	40.737177	Y	Y	Y
1011313	Abita Springs Farmers Market	Abita Spri…	St. Tamm…	Louisiana	70420	-90.0389129	30.4793215	Y	Y	Y
1003469	Acton-Boxborough Farmers Ma…	Acton	Middlesex	Massachu…	1720	-71.47415	42.47518	Y	Y	Y
1001083	Ada Farmers Market	Ada	Kent	Michigan	49301	-85.4889683	42.9553715	Y	Y	Y
1012233	Agricenter International Farmer…	Memphis	Shelby	Tennessee	38120	-89.8037477	35.128866	Y	Y	Y
1011830	Ahwatukee Farmers Market	Phoenix	Maricopa	Arizona		-111.983242	33.331331	Y	Y	Y
1010476	Alpena Farmers' Market	Alpena	Alpena	Michigan	49707	-83.430998	45.061878	Y	Y	Y
1009998	Altadena Farmers' Market	Altadena	Los Angeles	California	91001	-118.1581…	34.2003788	Y	Y	Y

Figure 23 Query showing all Farmers' Markets that sell vegetables, meat, and seafood

U1 Question: "Could I map the density of Farmer's Market which accept Credit/WIC/WICcash/SFMNAP/SNAP across the U.S.?"

For this U1 question, we must run the query to get the necessary data in SQLite and then we move to Jupyter notebook and use python to visualize the data. We will run the following queries to gather the data and export the resulting subset as .csv files.

All payment methods accepted:

```
SELECT         Market.fmid, Market.MarketName, Market.City, Market.County,
Market.State,  Market.Zip, Market.Longitude, Market.Latitude, Payment.credit,
Payment.wic,   Payment.wiccash, Payment.sfmnp, Payment.snap

FROM Market
INNER JOIN Payment
ON Market.fmid = Payment.fmid
WHERE          Payment.credit = 'Y' AND Payment.WIC = 'Y' AND Payment.wiccash =
'Y'            AND Payment.sfmnp = 'Y' AND Payment.snap = 'Y';
```

Credit accepted:

```
SELECT         Market.fmid, Market.MarketName, Market.City, Market.County,
               Market.State, Market.Zip, Market.Longitude, Market.Latitude, Payment.snap
FROM Market
INNER JOIN Payment
ON Market.fmid = Payment.fmid
WHERE Payment.credit = 'Y';
```

WIC accepted:

```
SELECT         Market.fmid, Market.MarketName, Market.City, Market.County,
               Market.State, Market.Zip, Market.Longitude, Market.Latitude, Payment.snap
FROM Market
INNER JOIN Payment
ON Market.fmid = Payment.fmid
WHERE Payment.wic = 'Y';
```

WICCash accepted:

```
SELECT         Market.fmid, Market.MarketName, Market.City, Market.County,
               Market.State, Market.Zip, Market.Longitude, Market.Latitude, Payment.snap
FROM Market
INNER JOIN Payment
ON Market.fmid = Payment.fmid
WHERE Payment.wiccash = 'Y';
```

SFMNP accepted:

```
SELECT        Market.fmid, Market.MarketName, Market.City, Market.County,
              Market.State, Market.Zip, Market.Longitude, Market.Latitude, Payment.snap
FROM Market
INNER JOIN Payment
ON Market.fmid = Payment.fmid
WHERE Payment.sfmnp = 'Y';
```

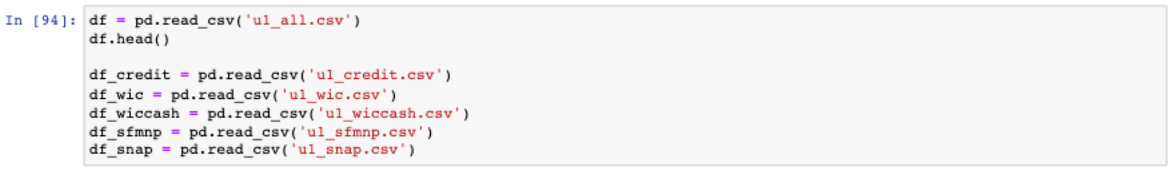
SNAP accepted:

```
SELECT        Market.fmid, Market.MarketName, Market.City, Market.County,
              Market.State, Market.Zip, Market.Longitude, Market.Latitude, Payment.snap
FROM Market
INNER JOIN Payment
ON Market.fmid = Payment.fmid
WHERE Payment.snap = 'Y';
```

Now, we can go to Jupyter notebook to use a scatter plot to map latitude and longitude from datasets in Python using Pandas Dataframe. We use libraries: GeoPandas, Shapely, Matplotlib, and Plotly.

Findings

To analyze the data, load CSV files exported by queries in SQLite into Jupyter notebook and create a Dataframe.

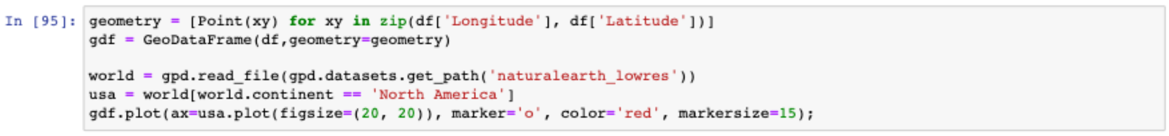


```
In [94]: df = pd.read_csv('ul_all.csv')
         df.head()

         df_credit = pd.read_csv('ul_credit.csv')
         df_wic = pd.read_csv('ul_wic.csv')
         df_wiccash = pd.read_csv('ul_wiccash.csv')
         df_sfmnp = pd.read_csv('ul_sfmnp.csv')
         df_snap = pd.read_csv('ul_snap.csv')
```

Figure 24 CSV files to Dataframe

Below is the visualization of Farmer's Markets that accept all payment methods plotted over North America.



```
In [95]: geometry = [Point(xy) for xy in zip(df['Longitude'], df['Latitude'])]
         gdf = GeoDataFrame(df,geometry=geometry)

         world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
         usa = world[world.continent == 'North America']
         gdf.plot(ax=usa.plot(figsize=(20, 20)), marker='o', color='red', markersize=15);
```

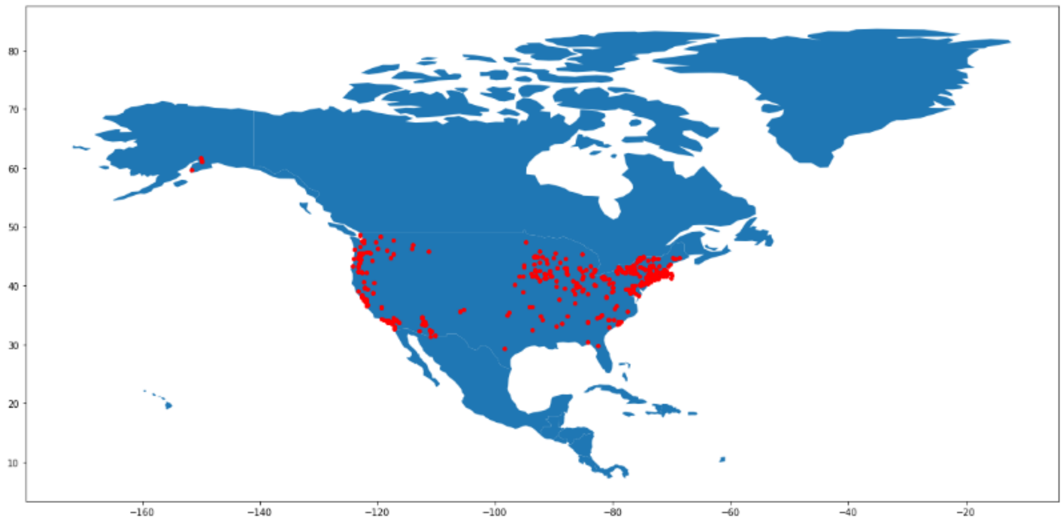


Figure 25 "All Payments Accepted" over North America

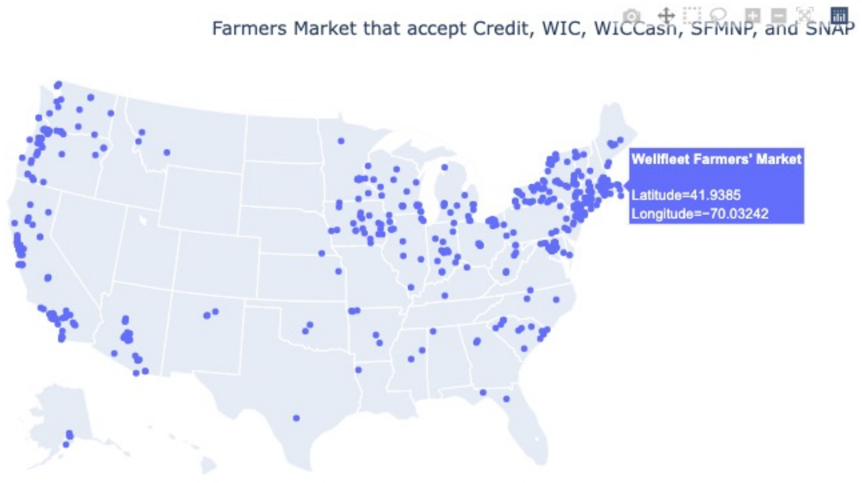
This map is useful in showing the density of Farmer's Market across the United States that accept all forms of payment. We can note that these are most prevalent on the east and west coast as well as the Midwest. We can also observe that the southern region has a relatively low to moderate density of Farmer's Market that accept all forms of payments.

This is great for overall visualization, however, for this to be useful we need to be able to zoom in and hover over each datapoint to distinct each Farmer's Market. For the next visualization, let's use a geographical scatter plot using Plotly and Plotly Express, which will give us those functionalities in our map.

Figure 26 shows the Plotly Express function using GeoPandas Dataframe to show the geographical scatter plot for all payment methods accepted plot mapped over the United States. Note that we can hover over and zoom in to the map. Please view the Jupyter notebook for a demo.

All the various maps from *Figure 26* to *Figure 31* provide a great tool for individuals who have the different means of payment and helps them figure out which Farmer's Market they can visit in their nearest vicinity to enjoy the experience and make health and budget conscious decisions.

```
In [103]: fig = px.scatter_geo(df,lat='Latitude',lon='Longitude', hover_name="MarketName", scope='usa')
          fig.update_layout(title = 'Farmers Market that accept Credit, WIC, WICCash, SFMNP, and SNAP', title_x=1)
          fig.show()
```



Farmers Market that accept Credit, WIC, WICCash, SFMNP, and SNAP
Wellfleet Farmers' Market
Latitude=41.9385
Longitude=-70.03242

Figure 26 Farmers Market that accept Credit, WIC, WICCash, SFMNP, and SNAP

```
In [104]: fig = px.scatter_geo(df_credit,lat='Latitude',lon='Longitude', hover_name="MarketName", scope='usa', color_discrete_seq
          fig.update_layout(title = 'Farmers Market that accept Credit', title_x=1)
          fig.show()
```

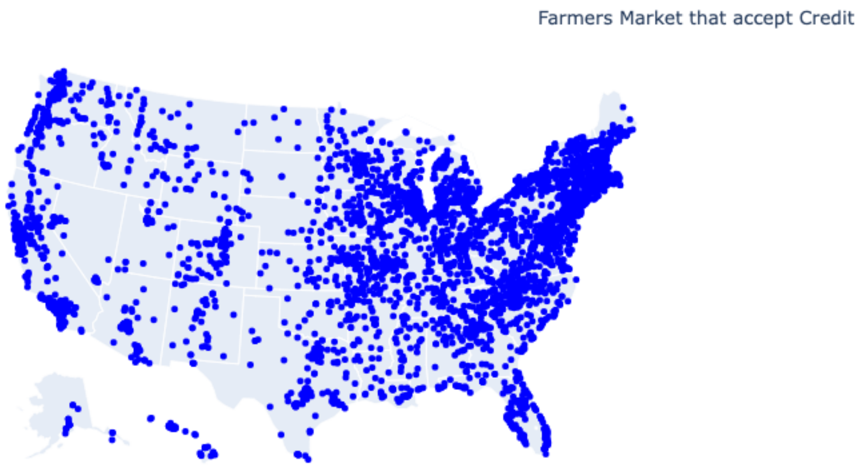


Figure 27 Farmers Market that accept Credit

```
In [105]: fig = px.scatter_geo(df_wic,lat='Latitude',lon='Longitude',hover_name="MarketName", scope='usa', color_discrete_sequenc
          fig.update_layout(title = 'Farmers Market that accept WIC', title_x=1)
          fig.show()
```

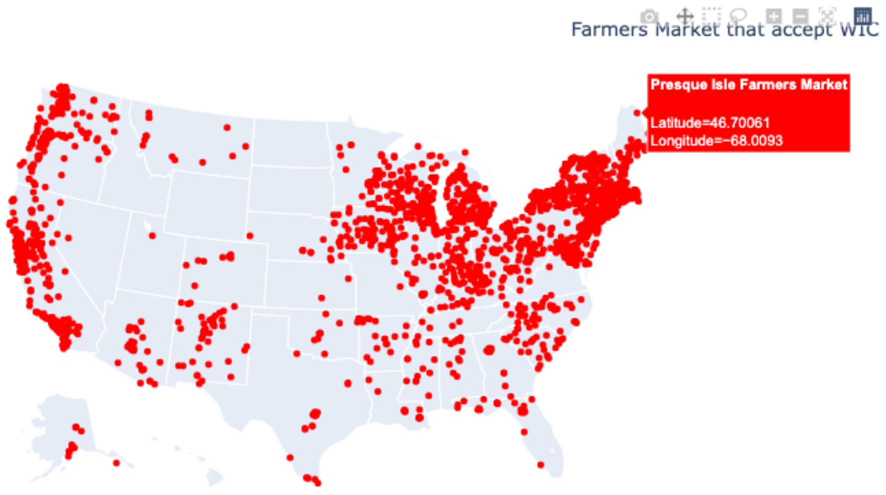


Figure 28 Farmers Market that accept WIC

```
In [106]: fig = px.scatter_geo(df_wiccash,lat='Latitude',lon='Longitude', hover_name="MarketName", scope='usa', color_discrete_se
          fig.update_layout(title = 'Farmers Market that accept WICCash', title_x=1)
          fig.show()
```

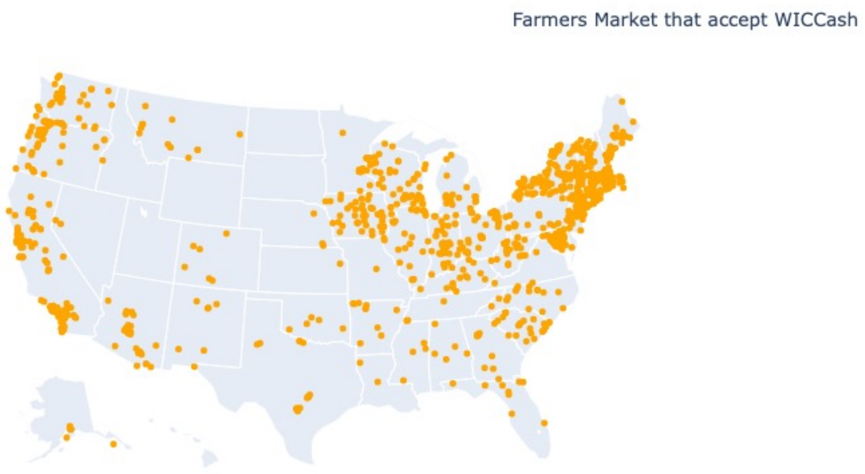
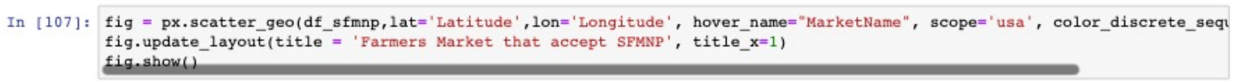


Figure 29 Farmers Market that accept WICCash



```
In [107]: fig = px.scatter_geo(df_sfmnp,lat='Latitude',lon='Longitude', hover_name="MarketName", scope='usa', color_discrete_sequ
          fig.update_layout(title = 'Farmers Market that accept SFMNP', title_x=1)
          fig.show()
```

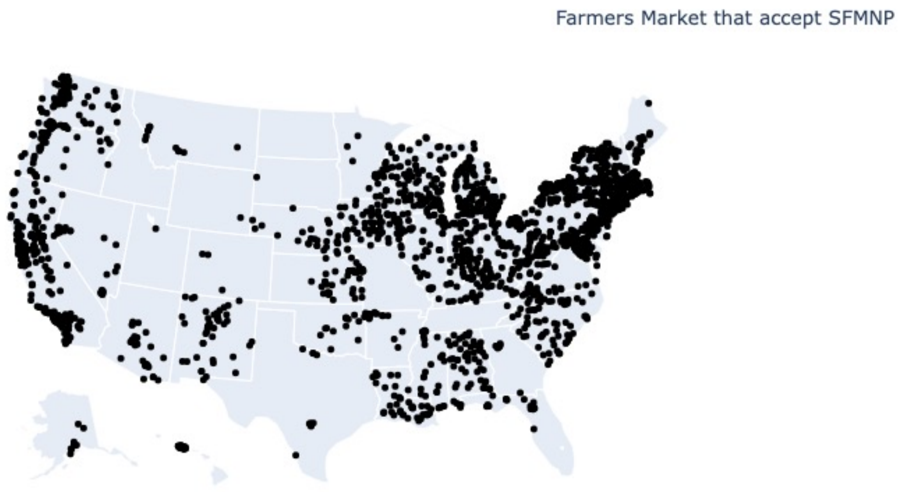
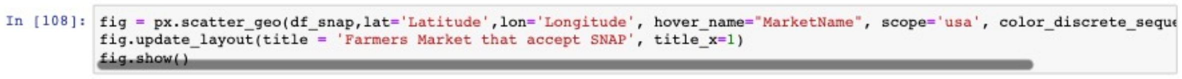


Figure 30 Farmers Market that accept SFMNP



```
In [108]: fig = px.scatter_geo(df_snap,lat='Latitude',lon='Longitude', hover_name="MarketName", scope='usa', color_discrete_seque
          fig.update_layout(title = 'Farmers Market that accept SNAP', title_x=1)
          fig.show()
```

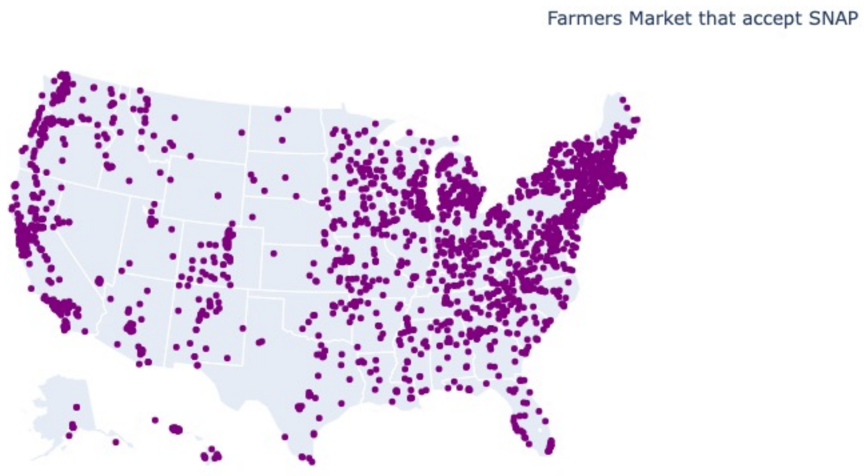


Figure 31 Farmers Market that accept SNAP

Data Cleaning Changes (from Phase-I to Phase-II):

Overall, the initial plan drafted in Phase-I was all in all very useful in cleaning the data. Hence for Phase-II, not much change was required. We did however make some important changes as discussed below that were discovered while performing the data cleaning.

In Phase-I, we had made the decision to keep the attribute data for the season date and time and to just combine them into one respective column for each to organize the data. However, after deliberating this, it makes more sense to remove the attributes all together as the extra work to clean the data that is not being used in U1 is not astute.

In Phase-I we had also discussed changing the data values of *Products* attributes from Y to 1 and N to 0 and to make the data type a BIT value in SQL to be conservative about space. The argument was that even though CHAR(1) and BIT data type only consume 1 byte to store, when large data is considered in SQL server, we can add up to 8 columns of a BIT into a single byte, while each column of type CHAR(1) will take up one byte. This would optimize space in the case our dataset continues to grow. Though, when performing the data cleaning, it was discovered that many data values were blank in the dataset, which was not accounted for when we had decided to use BIT data type. To remedy this, we kept the Y and N values for products' data values and blank values were changed to 0, so when we queried the data for U1 we would get accurate results.

Even though, we are not using the data values of the *Zip* attribute in U1, to clean it up we chose to remove the text data values and to convert the attribute to an INT. This will be helpful in the future when and if the data is needed for additional and to build-upon U1.3 queries regarding the location of the Farmer's Markets.

Instead of using Python to check the uniqueness of each FMID, as plan in Phase-I, we were able to achieve this in OpenRefine. In OpenRefine, we used a GREL expression forEach(row.record.cells[columnName].value,v,v).uniques().length() where we create an array for each record, remove all duplicate within the array, and then count how many elements there are in the array. Result of 0 would mean there is no value in the record. Result of 1 would mean that there is only one value assigned per entry per record, and therefore it is unique. 2 or more means that there is more than one value assigned and therefore the value is not unique. When the change was made FMID had 8665 rows of 1, which means all FMID values are unique.

Problems Encountered and Lessons Learned

In the *updateTime* attribute, we were able to change all *updateTime* values to the OpenRefine's default Date format, YYYY-MM-DDTHH:MM:SSZ. Though as we need to convert the dates into the ISO-format YYYY-MM-DD, this only gets us partly there. It was anticipated, as done in the OpenRefine Homework, that using the GREL command would resolve the issue.

However, when manually combing through the date data we realized that cell values with time ranging from 12 PM to 1 PM would not change. As we were getting an invalid date error every time any GREL command was applied, it was clear that we needed a different approach. After finding patterns and troubleshooting we were able to debug the issue. Once we were able to figure out the issue was with times not the dates but the "PM" part of the string, the solution was simple. We used a GREL command to remove all instances of the work "PM" from the *updateTime* attribute. When time was no longer displayed, we were able to re-apply the date format and GREL toString to toDate command and the issue was fixed.

Another issue we encountered while data cleaning was with making the correct decision for the Y/N values for the Products' attributes. In the Phase-I, it was clear to us that changing these Y/N values to a bit value would be clear and provide a space optimizing solution should the database grow. Though, when performing the data cleaning we realized that many of the Product values had blank cell values and/or "-" in place of Y/N. As there was missing values present in the dataset, it does not make sense to assume anything about the blank cells and move forward with the bit value change. Hence, we chose the best way forward was to leave the Y/N values as is and change any miscellaneous symbols such as "-" or "NA" into blank cell values.

Lastly, in Phase-I we had also planned to keep the season dates and times values and to create two new attributes, *SeasonDate* and *SeasonTime*, to display this information concisely. Though, as these columns are not needed for the target U1 goals, it was best to remove the unwanted and unnecessary information to make the database utmost functional.

Possible Next Steps

As the Farmers' Market dataset is cleaned and properly organized, there are numerous trends that can be run to gain more insights with the dataset.

A possible next step could be running visualization against another dataset. For example, we can retrieve a U.S median household income database; once cleaned and standardized, we can use it to make observations and to find any correlation between the household income versus the number of Farmer's Market present in the area. We can also query the number of Farmers' Market per state and compare it with the average household income per state to draw interesting conclusions.

Conclusion

Data cleaning is important to achieve scientific accuracy, efficiency, optimization of the data, and to communicate the results effectively. By creating a workflow and plan before embarking upon refining the data provides a critical step of exploration of the dataset and mental analysis before refining. Documenting the cleaning process also provides a clear understanding of the steps that need to be performed, improves the quality of work, and opens up a means of communication for the work.

Using the methodologies and concepts learned in CS513: Data Cleaning to practically clean the Farmers' Market data has solidified the understanding of the course material.

Resources & References

K, John D. "PLOT Latitude and Longitude from Pandas DataFrame in Python." *Data Science Guides, Data Science Guides*, 1 Sept. 2021, https://datascientyst.com/plot-latitude-longitude-pandas-dataframe-python/.

"How to Check If Value within a Record Are Unique in OpenRefine." How to Check If Value within a Record Are Unique in OpenRefine ~, https://kb.refinepro.com/2015/12/check-if-value-in-one-record-are-unique.html.

"Remove or Replace a Specific Character in a Column." *Remove or Replace a Specific Character in a Column* ~, https://kb.refinepro.com/2011/07/remove-specific-character-in-column.html.