

Assignment 4 - Group Project Final Report

Group Members: Mingwei Gao, Yuwei Chen, Hui Lyu

1 Overview and Initial Assessment of Dataset

1.1 Overview

Farmer markets is an important part of local economies. It not only serves as a way for people to purchase locally grown produce, but also as an opportunity for them to connect with others within their communities. Such experience helps to make individuals feel tied to their communities and promoted a sense of belonging.

In our project, we pick US Farmers Markets as our dataset.

(<https://www.ams.usda.gov/local-food-directories/farmersmarkets>) It contains 8665 rows (instances) and 59 columns. For each row, it represents information of a single farmer market with a unique FMID. The information includes market names, websites, social medias, market locations, season dates and times, directions, operating times, product offerings, accepted forms of payment, and more.

1.2 Quality Issues

From an initial inspection, we find the following quality issues.

Problem Type	Dirty Data	Reasons
Missing values	County field is null	Unavailable values during data entry
Misspelling words	City = "Amhersts"	Typos and phonetic errors
Mislocated values	County = "Missouri"	Human errors
Varying representations for null value	Including "no", "none", "n/a", "-", etc	Different representations
Varying representations for same value	Different URLs for the same website of the same market	Typos
Inconsistency of data (in different columns)	Zipcode does not match latitude and longitude	The same real world entity is described by different values
Inconsistency of format (in the same column)	Different descriptions in updateTime, Season1Date, etc	Different representations in different circumstances

1.3 Use Case of Dataset

Strengthening local food system is important to every government. Particularly, food distribution by local agriculture producer, namely farmer markets, is an important aspect of food distribution.

In particular, for users who need exact location information (especially zip code) of farmer markets, this dataset may not provide precise enough information. While, a reference postal code is offered accompanied with its origin given zip code for validation.

In addition, for users who need precise dates and times, not all the farmers markets have corresponding exact information.

1.4 Cleaning Goals

There are several cleaning goals we attempt to realize:

1. Eliminate errors
2. Eliminate redundancy
3. Increase data reliability
4. Deliver accuracy
5. Ensure consistency and completeness
6. Provide feedback for improvement

In essence, the goal of data cleaning is to minimize errors, prevent dirty data and keep data up to date. It helps to make following workflows smooth and accurate and increase efficiency of transaction.

2 Data Cleaning with OpenRefine

2.1 Complete Cleaning Steps

1. Create a new project and load the csv file into it (first row as header)

OpenRefine A power tool for working with messy data.

Project name: Final-farmers-markets

FMID	MarketName	Website	Facebook	Twitter	Youtube
1. 1012063	Caledonia Farmers Market Association - Danville	https://sites.google.com/site/caledoniafarmersmarket/	https://www.facebook.com/Danville.VT.Farmers.Market/		
2. 1011871	Stearns Homestead Farmers' Market	http://StearnsHomestead.com			
3. 1011878	100 Mile Market	http://www.pfcmarkets.com	https://www.facebook.com/100MileMarket/?fref=ts		
4. 1009364	106 S. Main Street Farmers	http://thetownofsixmile.wordpress.com/			

Parse data as

Character encoding:

CSV / TSV / separator-based files

Line-based text files

Fixed-width field text files

PC-Axis text files

JSON files

MARC files

RDF/N3 files

XML files

Open Document Format spreadsheets (.ods)

Columns are separated by

☒ commas (CSV)

☐ tabs (TSV)

☐ custom .

Escape special characters with \

☐ Ignore first 0 line(s) at beginning of file

☒ Parse next 1 line(s) as column headers

☐ Discard initial 0 row(s) of data

☐ Load at most 0 row(s) of data

☐ Parse cell text into numbers, dates, ...

☒ Quotation marks are used to enclose cells containing column separators

☒ Store blank rows

☒ Store blank cells as nulls

☐ Store file source (file names, URLs) in each row

Update Preview

Version 2.6-rc2 [TRUNK]

Help About

2. Transform text values of “x” and “y” columns into number values

The screenshot shows the Refine tool interface for a dataset named "Final-farmer-markets". The left sidebar displays a list of transformations: "0. Create project", "1. Text transform on 8636 cells in column x: value.toNumber()", and "2. Text transform on 8636 cells in column y: value.toNumber()". The main table shows 8665 rows. The columns include "Season3Time", "Season4Date", "Season4Time", "x", "y", "Location", "Credit", "WIC", "WICcash", "SFMNP", and "SNAP". A context menu is open over the "x" and "y" columns, showing options like "Facet", "Text filter", "Edit cells", "Edit column", "Transpose", "Sort...", "View", "Reconcile", "Cluster and edit...", "Common transforms", "Fill down", "Blank down", "Split multi-valued cells...", "Join multi-valued cells...", "Trim leading and trailing whitespace", "Collapse consecutive whitespace", "Unescape HTML entities", "To titlecase", "To uppercase", "To lowercase", "To number", "To date", "To text", and "Blank out cells".

3. Make a scatter plot of values of “x” and “y” columns

The screenshot shows the Refine tool interface with a scatter plot of "x (x) vs. y (y)". The left sidebar displays the scatter plot visualization, which shows a map of the United States with orange dots representing the data points. The main table shows 8665 rows. The columns include "Season4Date", "Season4Time", "x", "y", "Location", "Credit", "WIC", "WICcash", "SFMNP", and "SNA". A context menu is open over the "x" and "y" columns, showing options like "Facet", "Text facet", "Text filter", "Numeric facet", "Timeline facet", "Scatterplot facet", "Custom text facet...", "Custom Numeric Facet...", "Customized facets", "Edit cells", "Edit column", "Transpose", "Sort...", "View", "Reconcile", "Custom text facet...", "Custom Numeric Facet...", and "Customized facets".

4. Develop numeric facet of “x” and “y” to check number of blank (null) values

Open... Export ▼ Help

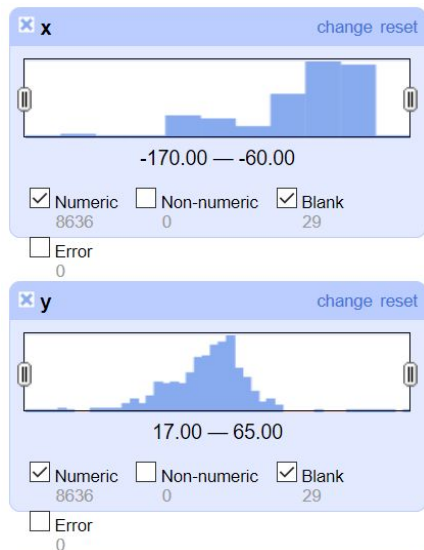
Extension

« first < previous 1 - 10 next > last

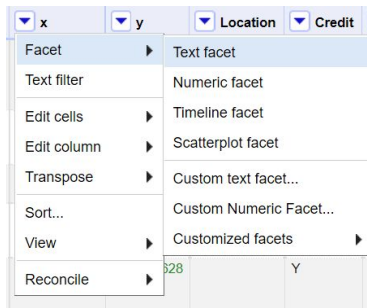
x	y	Location	Credit	WIC
Facet	Text facet			
Text filter	Numeric facet			
Edit cells	Timeline facet			
Edit column	Scatterplot facet			
Transpose	Custom text facet...			
Sort...	Custom Numeric Facet...			
View	Customized facets			
Reconcile				
-73.9493	40.7939	Private business parking lot	N	N
-86.790709	36.11837		Y	N
-73.9482477	40.8089533	Federal/State government building grounds	Y	Y
-75.53446	39.742117	On a farm from: a barn, a	N	N

Result:

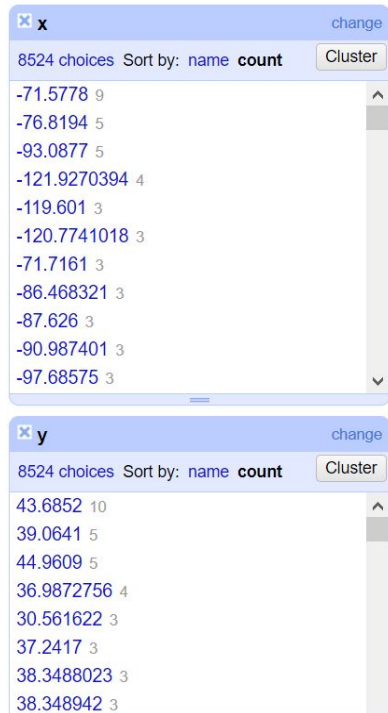
- 29 null values for each
- The ranges of latitude and longitude are reasonable



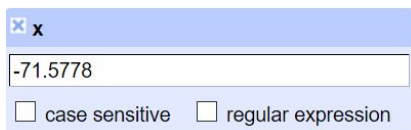
5. Develop text facet of “x” and “y” to check duplicates



Result: There are duplicate values in both “x” and “y”.



6. Filter by “x=-71.5778”. We find 9 rows with the same value of both “x” and “y”, but of different addresses.



street	city	County	State	zip
Academy at Gilmanton 4 Corners	Gilmanton		New Hampshire	
	Goshen		New Hampshire	
	Hampton		New Hampshire	
1292 Hooksett Road	Hooksett		New Hampshire	
Main Street	Bethlehem		New Hampshire	
1864 Chocorua Mt Highway (Route 16)	Silver Lake		New Hampshire	
River Road and Route 28	Pittsfield		New Hampshire	
145 Main Street	Plaistow		New Hampshire	
on the Commons	Whitefield		New Hampshire	

7. Implement `value.trim()` to each column to delete the leading and trailing whitespace of the values
8. Add a new column called “coordinates” to concatenate latitude and longitude in order to use Google Map API reverse geocoding
 - a. We assume the not blank coordinates values are basically correct since they have been validated on Google Map visualization (no unreasonable points).
 - b. Then we use the assumed correct coordinates to get the corresponding postal code through reverse geocoding.

Add column based on column y

New column name

☐ set to blank ☐ store error ☒ copy value from original column

Expression
Language General Refine Expression Language (GREL) ▾

No syntax error.

Preview History Starred Help

row	value	cells[\"y\"].value + \",\" + cells[\"x\"].value
1.	44.411013	44.411013,-72.140305
2.	41.375118	41.375118,-81.7285969
3.	42.296024	42.296024,-85.574887
4.	34.8042	34.8042,-82.8187
5.	37.495628	37.495628,-94.2746191
6.	40.7939	40.7939,-73.9493

OK Cancel

Facet by blank: 29 blank rows

coordinates [change](#) [invert](#) [reset](#)

2 choices Sort by: **name** count

false 8636

true 29 [exclude](#)

[Facet by choice counts](#)

9. Add a new column by fetching URLs with json responses of Google Map API reverse geocoding

Add column by fetching URLs based on column coordinates

New column name Throttle delay milliseconds

On error ☒ set to blank ☐ store error

Formulate the URLs to fetch:

Expression Language General Refine Expression Language (GREL)

No syntax error.

Preview History Starred Help

row	value	reverseGeocoding
1.	44.411013,-72.140305	https://maps.googleapis.com/maps/api/geocode/json?latlng=44.411013,-72.140305&result_type=postal_code&key=AlzaSyAQME9fbTXb7UmkvLwLcNt9w8yMjOeDGks
2.	41.375118,-81.7285969	https://maps.googleapis.com/maps/api/geocode/json?latlng=41.375118,-81.7285969&result_type=postal_code&key=AlzaSyAQME9fbTXb7UmkvLwLcNt9w8yMjOeDGks
3.	42.296024 -	https://maps.googleapis.com/maps/api/geocode/json?latlng=42.296024 -

OK Cancel

10. Parse JSON and extract the postal code using
 “value.parseJson().results[0].address_components[0].short_name”

Add column based on column reverseGeocoding

New column name ☒ set to blank ☐ store error ☐ copy value from original column

Expression Language General Refine Expression Language (GREL)

No syntax error.

Preview History Starred Help

row	value	postalCodeGeocoding
1.	{ "results": [{ "address_components": [{ "long_name": "05828", "short_name": "05828", "types": ["postal_code"] }, { "long_name": "Danville", "short_name": "Danville", "types": ["locality", "political"] }, { "long_name": "Caledonia County", "short_name": "Caledonia County", "types": ["administrative_area_level_2", "political"] }] }] }	05828

OK Cancel

11. Repeat step 8 & 9 for several times using different API Keys, and then generate multiple columns
12. Transform blank cells in these columns to a whitespace for future use

Custom text transform on column Geo3

Expression Language General Refine Expression Language (GREL) ▾

`" "` No syntax error.

13. Merge all the columns of postal codes to a newly added column called “geoPostalCode”, and then transform `value.trim()` to all the cells

Add column based on column postalCodeGeocoding

New column name

☒ set to blank ☐ store error ☐ copy value from original column

Expression Language General Refine Expression Language (GREL) ▾

`cells["Geo3"].value + cells["Geo2"].value +
cells["reverseGeo"].value + cells["postalCodeGeocoding"].value` No syntax error.

Preview History Starred Help

row	value	cells["Geo3"].value + cells["Geo2"].value + cells["reverseGeo"].value + cells["postalCodeGeocoding"].value
1.	05828	05828
2.	44129	44129
3.	49007	49007
4.	29682	29682
5.	64759	64759
6.	10029	10029

Consequently, “geoPostalCode” column includes the data coming from Google Map reverse geocoding result based on latitude and longitude.

14. Move column “geoPostalCode” to position 12, right after “zip” column for comparison
15. Collapse consecutive whitespaces for “MarketName” column

8665 rows Extens

Show as: **rows** records Show: 5 10 25 50 rows « first < previous 1 - 50 next > |

	All	FMID	MarketName	Website	Facebook	Twitter
position	8.	1009845	Facet	www.125thStreetFarmersMarket.com	https://www.facebook.com/125thStreetFarmersMarket	https://twitter.com/FarmMarket125
position	9.	1005586	Text filter			
position	10.	1008071	Edit cells			
position	11.	1012710	Transform...			
position	12.	1010284	Common transforms			
			Trim leading and trailing whitespace			
			Collapse consecutive whitespace			
			Unescape HTML entities			
			To titlecase			
			To uppercase			
			To lowercase			
			To number			
			To date			
			To text			
			Blank out cells			

16. Consider about clustering and merging cells in column “MarketName”

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 collisionKeying Function fingerprint

214 clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value	# Choices in Cluster
4	12	<ul style="list-style-type: none"> 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) 	<input type="checkbox"/>	Main Street Farmers Market	
3	4	<ul style="list-style-type: none"> Columbus Farmers' Market (2 rows) Columbus Farmers Market (1 rows) columbus farmers market (1 rows) 	<input type="checkbox"/>	Columbus Farmers' Market	
3	3	<ul style="list-style-type: none"> WATERTOWN FARMERS MARKET (1 rows) Watertown Farmers market (1 rows) Watertown Farmers' Market (1 rows) 	<input type="checkbox"/>	WATERTOWN FARMERS M	
3	5	<ul style="list-style-type: none"> Rochester Downtown Farmers Market (3 rows) Downtown Rochester Farmers Market (1 rows) Downtown Rochester Farmers' Market (1 rows) 	<input type="checkbox"/>	Rochester Downtown Farme	
3	3	<ul style="list-style-type: none"> Harrison Farmer's Market (1 rows) Harrison Farmers Market (1 rows) Harrison Farmers' Market (1 rows) 	<input type="checkbox"/>	Harrison Farmer's Market	

After browsing these clusters, we find that they are different markets located in different places. Therefore, we decide not to merge anything.

17. Modify potential typo by replacing "//www" with "http://www"

Custom text transform on column Website

Expression value.replace('//www', 'http://www') Language General Refine Expression Language (GREL) No syntax error.

Preview History Starred Help

row	value	value.replace('//www', 'http://www')
1.	https://sites.google.com/site/caledoniafarmersmarket/	https://sites.google.com/site/caledoniafarmersmarket/
2.	http://Stearnshomestead.com	http://Stearnshomestead.com
3.	http://www.pfcmarkets.com	http://www.pfcmarkets.com
4.	http://thetownofsixmile.wordpress.com/	http://thetownofsixmile.wordpress.com/
5.	null	Error: replace expects 3 strings, or 1 string, 1 regex, and 1 string

On error ☒ keep original ☐ set to blank ☐ store error ☐ Re-transform up to 10 times until no change

OK Cancel

18. Cluster and merge cells in "Website" after manual browsing and checking

Cluster & Edit column "Website"

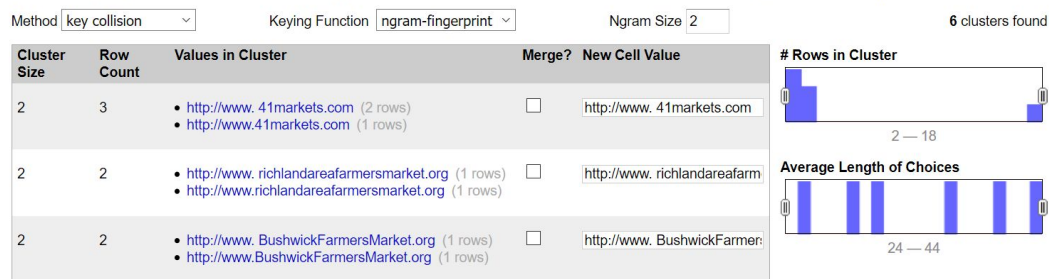
This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer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

Method key collisionKeying Function fingerprint

56 clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value	# Choices in Cluster
3	3	<ul style="list-style-type: none"> http://www.smgov.net/Portals/Farmers_Market/ (1 rows) http://www.smgov.net/portals/farmersmarket (1 rows) http://www.smgov.net/portals/farmersmarket/ (1 rows) 	<input type="checkbox"/>	http://www.smgov.net/Portals	
2	9	<ul style="list-style-type: none"> http://www.portlandfarmersmarket.org/ (6 rows) http://www.portlandfarmersmarket.org (3 rows) 	<input type="checkbox"/>	http://www.portlandfarmersm	
2	3	<ul style="list-style-type: none"> http://www.growersmarket.org (2 rows) http://www.Growers-Market.org (1 rows) 	<input type="checkbox"/>	http://www.growersmarket.or	
2	2	<ul style="list-style-type: none"> http://www.RaleighEatLocal.com (1 rows) http://www.raleigheatlocal.com (1 rows) 	<input type="checkbox"/>	http://www.RaleighEatLocal.c	
2	2	<ul style="list-style-type: none"> http://www.peacham.net/market (1 rows) http://www.peacham.net/market/ (1 rows) 	<input type="checkbox"/>	http://www.peacham.net/mar	
2	3	<ul style="list-style-type: none"> http://www.ithacamarket.com (2 rows) http://www.ithacamarket.com/ (1 rows) 	<input type="checkbox"/>	http://www.ithacamarket.com	

19. Modify cluster keying function to “ngram-fingerprint” and set a value for ngram size



20. Modify typo by replacing “:// ” with “:./”

Custom text transform on column Website

Expression: `value.replace('/:// ', ':./')` Language: General Refine Expression Language (GREL) No syntax error.

Preview

row	value	value.replace('/:// ', ':./')
1.	https://sites.google.com/site/caledoniafarmersmarket/	https://sites.google.com/site/caledoniafarmersmarket/
2.	http://StearnsHomestead.com	http://StearnsHomestead.com
3.	http://www.pfcmarkets.com	http://www.pfcmarkets.com
4.	http://thetownofsixmile.wordpress.com/	http://thetownofsixmile.wordpress.com/
5.	null	Error: replace expects 3 strings, or 1 string, 1 regex, and 1 string

On error: ☒ keep original ☐ set to blank ☐ store error ☐ Re-transform up to 10 times until no change

OK Cancel

21. Repeat step 16 through step 19 for “Facebook”, “Twitter”, “Youtube”, and “OtherMedia” columns

22. Dealing with NULL values:

23. Replace “none”, “no” and “n/a” with blank values in “Facebook”, “Twitter”, “Youtube”, and “OtherMedia” columns

Twitter

Refresh Reset All Remove All

none ☐ case sensitive ☐ regular expression

Apply Apply to All Identical Cells Cancel

FMID	MarketName	Website	Facebook	Twitter	Youtube	OtherMedia	str
			none	None	None		Route 4
			none	none	none		719 NORTH FIRST STREET
			none	none	none		315 Old Conestoga Rd
6030	Rancho Bernardo Certified Farmers Market and Specialties		none	none	Bernardo Winery		13330 Paseo T Verno Norte
2316	Springdale Farmers' Market	http://springdalefarmersmarket.org	none	none	none		

24. Collapse consecutive whitespaces, cluster and merge values for “city”, “street”, “County”, “State” columns

25. Replace “none”, “n/a”, and “no” values with blank values in “city”, “street”, “County”, “State” and “zip” columns

26. Replace ‘-’ with blank values in “Organic” column

Custom text transform on column Organic

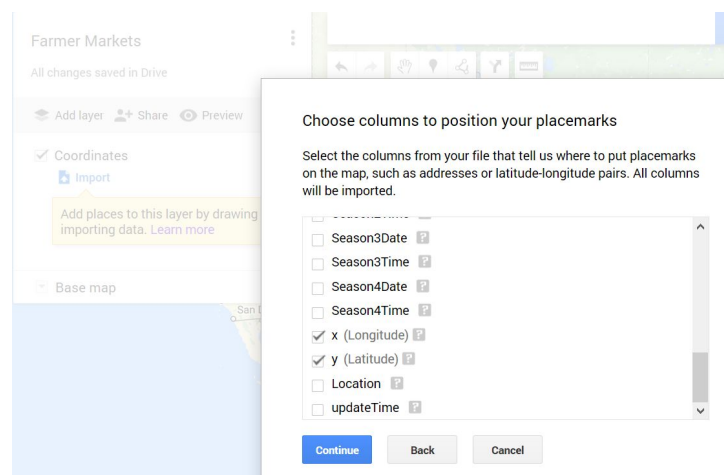
Expression Language General Refine Expression Language (GREL)

`value.replace('-', '')` No syntax error.

27. We decide not to transform date or datetime in OpenRefine since it will automatically add unexpected information. For instance, “2007” will be transformed to a format similar to “2007-01-01T00:00:00”. The detailed datetime information which may not be correct will be added. Therefore, formatting of date and time will be done in SQLite.

2.2 Google Map Visualization of Coordinates

1. Sort by FMID in Excel
2. Separate the whole sorted csv file into 5 files in Excel
 - a. 1-2000 rows
 - b. 2001-4000 rows
 - c. 4001-6000 rows
 - d. 6001-8000 rows
 - e. 8001-8665 rows
3. Create new map called “Farmer Markets” in Google Map
4. Add new layer for each individual csv files (because Google Map can display at most 2000 items per layer)
5. Import data “google-map-2000.csv”
 - a. Remove columns from “Credit” to “WildHarvested” (because Google Map allow to import csv with no more than 50 columns)
 - b. Choose columns to position the placemarks
 - i. X as longitude
 - ii. Y as latitude



- c. Choose columns to title the markers

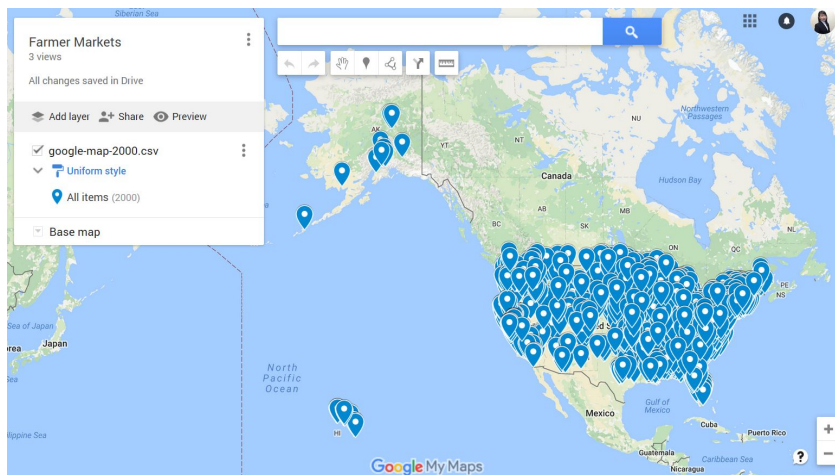
Choose a column to title your markers

Pick a column to use as the title for the placemarks, such as the name of the location or person.

☐ FMID ?
 ☒ MarketName ?
 ☐ Website ?
 ☐ Facebook ?
 ☐ Twitter ?
 ☐ Youtube ?
 ☐ OtherMedia ?
 ☐ street ?

Finish
 Back
 Cancel

6. Result for the first layer



- Repeat step 5 for the other csv files. There are 5 layers eventually.
- Find 29 rows in total which do not have value in (x,y). **The null values of “x” and “y” are in pairs. In other words, if one of “x” or “y” is null, the other is also null.**

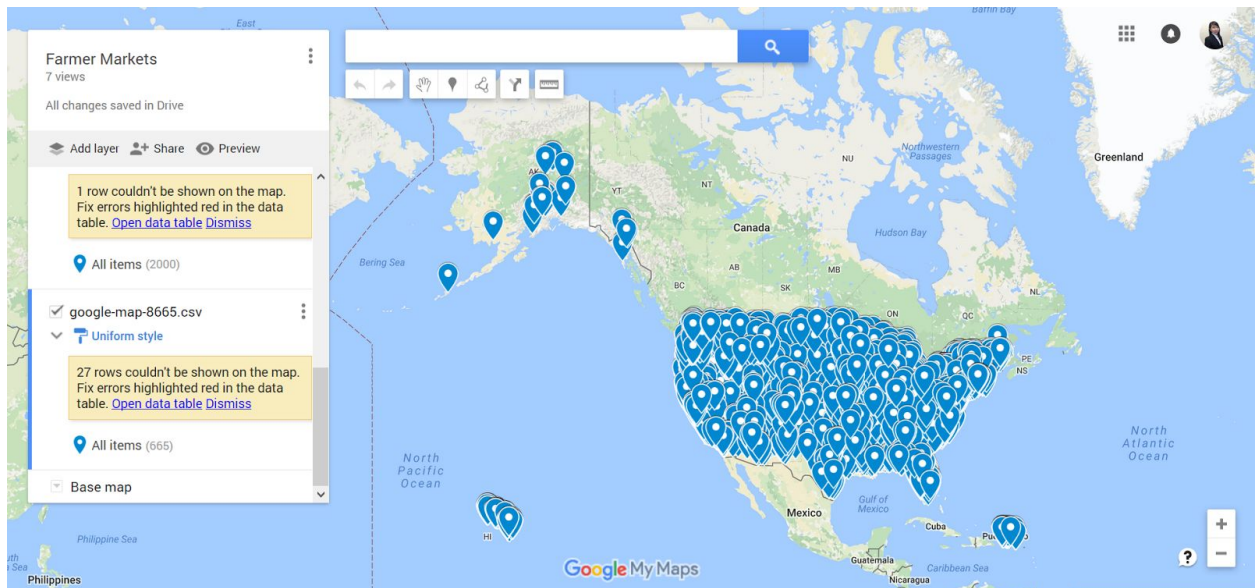
<input checked="" type="checkbox"/> google-map-6000.csv <input checked="" type="checkbox"/> Uniform style <input checked="" type="checkbox"/> All items (2000)		
<input checked="" type="checkbox"/> google-map-4000.csv <input checked="" type="checkbox"/> Uniform style <div>1 row couldn't be shown on the map. Fix errors highlighted red in the data table. Open data table Dismiss</div> <input checked="" type="checkbox"/> All items (2000)	<input checked="" type="checkbox"/> google-map-8000.csv <input checked="" type="checkbox"/> Uniform style <div>1 row couldn't be shown on the map. Fix errors highlighted red in the data table. Open data table Dismiss</div> <input checked="" type="checkbox"/> All items (2000)	<input checked="" type="checkbox"/> google-map-8665.csv <input checked="" type="checkbox"/> Uniform style <div>27 rows couldn't be shown on the map. Fix errors highlighted red in the data table. Open data table Dismiss</div> <input checked="" type="checkbox"/> All items (665)

google-map-8665.csv

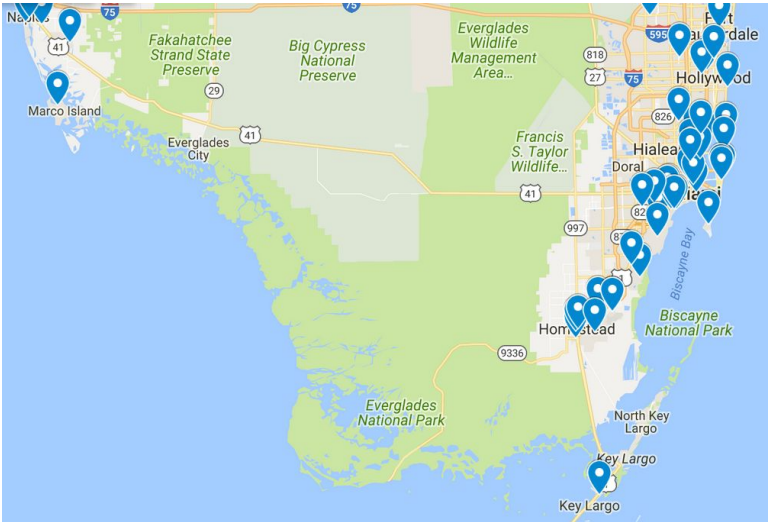
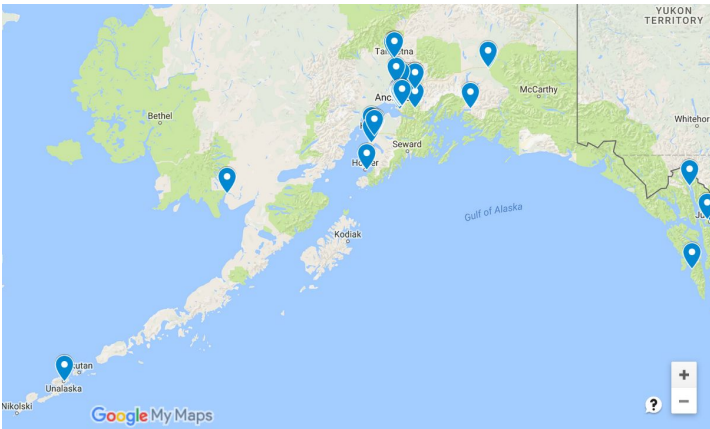
Find in table 1–200 of 665

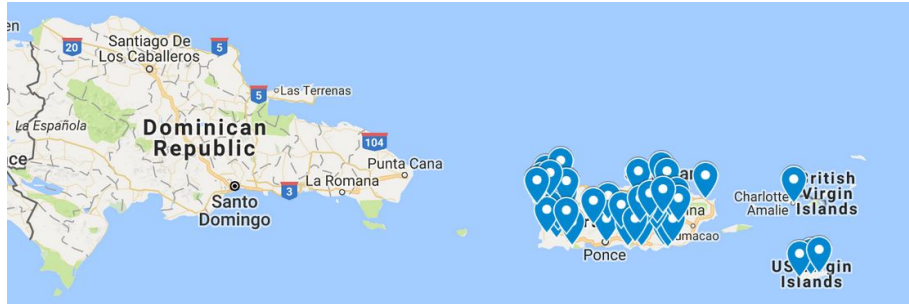
	Season4Time	x	y	Location	updateTime
1		⚠	⚠		2013
2					
3		⚠	⚠		2013
4		⚠	⚠		2013
5		⚠	⚠		2013
6		⚠	⚠		2013
7					
8		⚠	⚠		2013
9		⚠	⚠		2013

9. Final result



Zoom in:

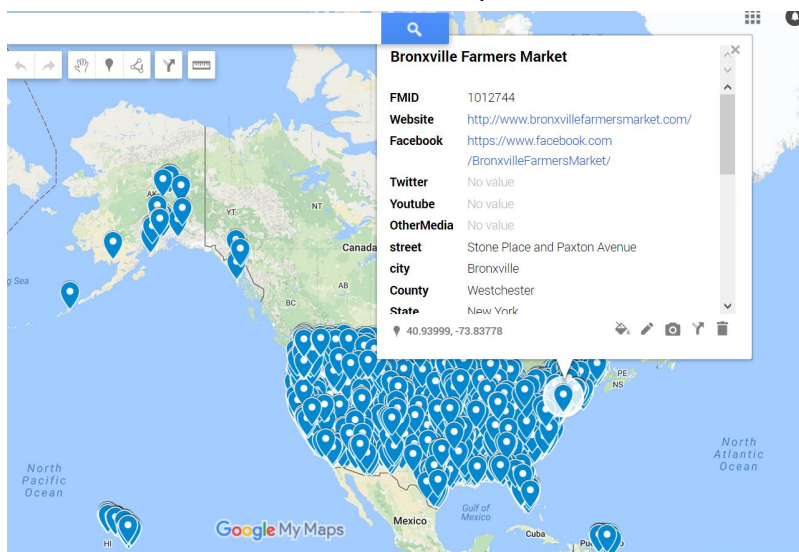




10. Validate the coordinates on map

- a. no unreasonable (x,y) values in the given dataset
- b. no point is located in the ocean

11. Labels can also be shown on the map



2.3 Google Map API Reverse Geocoding

API: <https://developers.google.com/maps/documentation/geocoding/intro>

API Key:

- AlzaSyAQME9fbTXb7UmkvLwLCnT9w8yMjOeDGks
- AlzaSyCO3shtN23zrr6LjENrt1IQ1611myysis4
- AlzaSyDUB0N3Xlr2LvYuWlsX3DZa49Gw0hdcJY0
- AlzaSyAs_QgNHBLIjykmR-rNPnKcmHMOwh2vHgE

Use latitude and longitude to get its postal code:

- result_type=postal_code

For example:

https://maps.googleapis.com/maps/api/geocode/json?latlng=44.41101,-72.1403&result_type=postal_code&key=AlzaSyAQME9fbTXb7UmkvIwLCnT9w8yMjOeDGks

3 Data Cleaning with SQLite

3.1 Relational Database Schema

Table: fmv2

(FMID, MarketName, Website, Facebook, Twitter, Youtube, OtherMedia, street, city, County, State, zip, geoPostalCode, Season1Date, Season1Time, Season2Date, Season2Time, Season3Date, Season3Time, Season4Date, Season4Time, x, y, Location, Credit, WIC, WICcash, SFMNP, SNAP, 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, updateTime)

3.2 Integrity Constraints Checking and Fixing

1. Primary key checking

Check whether FMID could act as the primary key in the table (whether there is duplicate FMID):

```
SELECT T1.FMID FROM fmv2 T1, fmv2 T2
WHERE T1.FMID=T2.FMID
AND T1.x != T2.x
```

Since we got 0 result after running the query above, FMID could be assigned as the primary key for the table.

2. Further processing for zip code validation

We noticed that there are records whose geoPostalCode value is NULL. That means zip code cannot be found through Reverse Geocoding API according to x and y values, which indicates that these zip codes may be wrong.

```
SELECT FMID, zip, geoPostalCode
FROM fmv2
WHERE zip is not null
AND geoPostalCode is null;
```

In total, there are 19 records whose zip code may be wrong. Here is the screenshot of the results:

FMID	city	County	State	zip	geoPostalCode	x	y
1000291	Standord	Santa Clara	California	94305		-122.171	37.4244
1001100	Altus	Jackson	Oklahoma	73521		-99.3336	34.6491
1001225	Hayfork	Humboldt	California	96041		-123.38	40.4157
1001311	San Rafael	Alameda	California	94901		-122.455	37.9367
1002155	Rancho Santa Fe	San Diego	California	92067		-117.196585536003	32.9898123443127
1002854	West Chester	Chester	Pennsylvania	19380			
1002953	Cayucos	San Luis Obispo	California	93430		-120.912	35.4454
1003684	Rancho Cucamonga	San Bernardino	California	91739		-117.532318	34.110872
1004251	El Cerrito	Contra Costa	California	94596		-122.3	37.8997
1005086	Athens	Athens	Ohio	45701		-82.0584387	39.3359163
1005743	Federal Way	King	Washington	98003		-122.313022	47.313385
1008861	Langley	Adams	Washington	98260		-122.461312	48.006879
1010049	Rochester	Olmsted	Minnesota	55902		-92.478747	44.001971
1010316	Macon	Bibb	Georgia	31201		-83.646551	32.833797
1010364	Orlando, FL	Orange	Florida	32803		-81.3462829	28.5537904
1010617	Boynton Beach	Palm Beach	Florida	33473		-80.221052	26.489351
1011689	Charlotte	Mecklenburg	North Carolina	28217			
1012182	Missoula	Missoula	Montana	59812		-113.983083	46.86023
1012201	Olympia	Thurston	Washington	98501		-122.902506	47.050307

Among these 19 records, two of them are without latitude and longitude values. For the rest of them, since it is not a huge amount of work, we decide to manually validate the zip code.

The website we use:

<http://www.melissadata.com/lookups/latlngzip4.asp?lat=-122.171&lng=37.4244>

Through manual check, we found that the zip code of whose FMID are 1000291, 1001100, 1002953, 1003684, 1005743, 1008861, 1010049, 1010316, 1010364, 1010617, 1012201 are exactly the same with the zip code we got according to latitude and longitude from the website. The zip code of whose FMID are 1001225, 1001311, 1002155, 1004251, 1012182, are different from what we got from the website.

Therefore, we update those rows whose zip codes are exactly the same with what we got from the website.

```
UPDATE fmv2 SET geoPostalCode=zip
WHERE FMID=1000291
OR FMID=1001100
OR FMID=1002953
OR FMID=1003684
OR FMID=1005743
OR FMID=1008861
OR FMID=1010049
OR FMID=1010316
OR FMID=1010364
OR FMID=1010617
OR FMID=1012201;
```

Besides, there are some records whose zip code are not the same with what we got based on latitude and longitude. Similarly, there may be problems for those zip codes.

```
SELECT zip, geoPostalCode
FROM fmv2
WHERE zip <> geoPostalCode;
```

We got 1164 results after running the script above. That is too much. Do we really need to clean all of them? Are those zip codes all wrong? After the discussion, our conclusion is: small differences between zip in dataset and zip we got from the third party should be tolerated. (Think about zip code 61820 (in Champaign) and 61801 (in Urbana). There is a difference of 19 between the zip code but in fact the distance between geographic locations of these two zip codes is not that far.) Our group feel that the difference of 50 could be a value to determine whether there is significant problem with the zip code. Thus, we ran the following script.

```
SELECT zip, geoPostalCode, cast(zip as numeric)-cast(geoPostalCode as numeric)
FROM fmv2
WHERE cast(zip as numeric)-cast(geoPostalCode as numeric)>50
OR cast(zip as numeric)-cast(geoPostalCode as numeric)<=-50;
```

This time we got 335 results. However, we found that some of them, the values in zip column are not 5 digits, but 9 digits, while what we got from Reverse Geocoding would always be 5 digits. That would be one of the reasons leading to big differences. Another possible reason would be that there are non-numeric values in zip column, such as 'IL', 'FL'. Due to those facts, we found that we need to clean those who cells whose contents are not 5-digit numbers.

Moreover, we found that there are some zip codes who are not 5 digits.

```
SELECT FMID, zip, geoPostalCode
FROM fmv2
WHERE length(zip)<>5;
```

After running the script above, we got 29 results in total. Some of them, as the screenshot (1) shows, are 9 digits, and the first 5 digits are the them with what we got from the Reverse Geocoding. We believe those zip codes are highly possibly with no problem.

FMID	zip	geoPostalCode
1001493	20015-2541	20015
1001791	57201-3645	57201
1002349	57501-3504	57501

```
UPDATE fmv2 SET geoPostalCode=zip
WHERE FMID=1001493
OR FMID=1001791
OR FMID=1002349
```

```
OR FMID=1005109
OR FMID=1006828
OR FMID=1007541
OR FMID=1007758;
```

The reason why we put value in zip column to geoPostalCode, rather than copying from geoPostalCode to zip is that the information contained in 9-digit zip code would definitely more than in 5-digit zip code. We do not want to lose them.

Then we ran script below after updating:

```
SELECT FMID, zip, geoPostalCode
FROM fmv2
WHERE length(zip)<>5
AND zip<>geoPostalCode;
```

We got 22 results this time. Some of them, as shown below, obviously has typo problem, or they lose the leading zero. We believe that we could update those to values we got from Reverse Geocoding.

1005771	513338	51338
1006166	550424	50424
1007506	6106	06106
1007674	6412	06412
1007830	6105	06105

```
UPDATE fmv2 SET zip=geoPostalCode
WHERE FMID=1002457
OR FMID=1005771
OR FMID=1006166
OR FMID=1007506
OR FMID=1007674
OR FMID=1007830;
```

For those using short names of states, such as IL, MA, in zip column, we decide to put the value as NULL, if the short name in zip column matches the State name in State column, since it is duplicative information. The reason why we do not copy the zip code from geoPostalCode directly is that we feel that may be too arbitrary because the value in geoPostalCode column are from latitude and longitude, but we did not check whether those latitudes and longitudes match the addresses.

The code we use to update here:

```
UPDATE fmv2 SET zip=""
WHERE length(zip)=2;
```

Then we ran the code below again:

```
SELECT zip, geoPostalCode, cast(zip as numeric)-cast(geoPostalCode as numeric) AS
DIFFERENCE
FROM fmv2
WHERE cast(zip as numeric)-cast(geoPostalCode as numeric)>50
OR cast(zip as numeric)-cast(geoPostalCode as numeric)<-50
ORDER BY DIFFERENCE DESC;
```

This time, we got a result of 331. Besides, accidentally, we found an obvious typo problem:

726o1

72601

We choose to update this error using the following query:

```
UPDATE fmv2 SET zip=geoPostalCode
WHERE zip='726o1';
```

For the rest of them, we choose to leave them there. But for the sake of potential dataset users, we recommend a warning here to alert users that these zip codes are not that reliable.

3. Uniforming format for updateTime column

For updateTime column, we found that some are using words like 'May', 'Apr' to represent months, rather than using numbers. The reason for we choose to uniform the format of date is that we found that there is comma when using letters to represent months and it caused problems when we try to import data into MySQL. The original dataset is a csv file (Comma Separated Values). Thus, it is hard for the MySQL to really distinguish which comma is for separating columns and which are within the format of dates.

We runs codes below:

```
SELECT UpdateTime FROM fmv2
WHERE UpdateTime like 'Jan%';
```

```
SELECT UpdateTime FROM fmv2
WHERE UpdateTime like 'Jan%'
OR UpdateTime like 'Feb%';
```

...

Every time we adding a new 'OR' statement and write down the total number of results we get. And we found that there are only May, Apr and Jun used for months in updateTime columns. Therefore we ran code below:

```
SELECT UpdateTime FROM fmv2
```

```
WHERE UpdateTime like 'Jun%'
OR UpdateTime like 'May%'
OR UpdateTime like 'Apr%';
```

And we got 312 results in total. Then we ran codes below to update those rows.

```
UPDATE fmv2 SET UpdateTime=replace(UpdateTime,'Apr ','4/')
WHERE UpdateTime like 'Apr %';
```

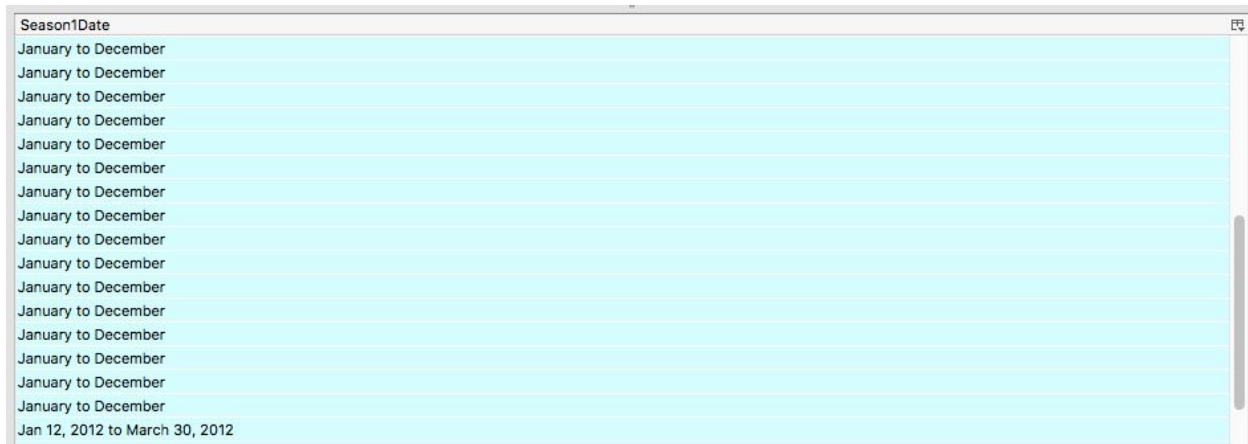
```
UPDATE fmv2 SET UpdateTime=replace(UpdateTime,'May ','5/')
WHERE UpdateTime like 'May %';
```

```
UPDATE fmv2 SET UpdateTime=replace(UpdateTime,'Jun ','6/')
WHERE UpdateTime like 'Jun %';
```

```
UPDATE fmv2 SET UpdateTime=replace(UpdateTime,' 20','/20')
WHERE UpdateTime like '% 20%';
```

4. Uniforming format for Season1Date, Season2Date and Season3Date

For Season1Date, similarly, there are also some cells in which months are represented using words, rather than numbers. See the last row in the following screenshot.



Season1Date
January to December
January to December
January to December
January to December
January to December
January to December
January to December
January to December
January to December
January to December
January to December
January to December
January to December
January to December
January to December
January to December
January to December
Jan 12, 2012 to March 30, 2012

For cells like what is shown in the last row of the screenshot above, we decide to use the similar SQL scripts to update it to the same format as the rest of cells. For those who only have month values, we choose to keep them unchanged since if we change them into numbers, it will be confused for users to understand whether it means days or months.

```
SELECT Season1Date
FROM fmv2
WHERE Season1Date like 'Jan %';
```

Season1Date
Jan 12, 2012 to March 30, 2012

```
UPDATE fmv2 SET Season1Date=replace(Season1Date,'Jan ', '01/')
WHERE Season1Date like 'Jan %';
```

```
SELECT Season1Date
FROM fmv2
WHERE Season1Date like '%to Oct %';
```

Season1Date
May 5, 2012 to Oct 6, 2012
June 5, 2012 to Oct 30, 2012

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'Oct ', '10/')
WHERE Season1Date like "% Oct %";
```

After finishing steps above, we found a problem. Since we cannot use Regular Expression in SQLite Manager, how could we distinguish, for example, “May” in “May 5, 2012 to Oct 6, 2012” from “May” in “May to October” ? We want to change the format of the previous one but do not want to change the latter one. Therefore, we went back to our initial concern: removing the comma, and we ran codes below to fulfil the task:

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'May ', '05/')
WHERE Season1Date like "%, 2012";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'June ', '06/')
WHERE Season1Date like "%, 2012";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'April ', '04/')
WHERE Season1Date like "%, 2012";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'July ', '07/')
WHERE Season1Date like "%, 2012";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'September ', '09/')
WHERE Season1Date like "%, 2012";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'October ', '10/')
WHERE Season1Date like "%, 2012";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'November ', '11/')
WHERE Season1Date like "%, 2012";
```



```
UPDATE fmv2 SET Season1Date=replace(Season1Date, ', 2012','/2012')
WHERE Season1Date like "%, 2012";
```

```
SELECT FMID, Season1Date
FROM fmv2
WHERE Season1Date like "%, 2011";
```

FMID	Season1Date
1001137	April to November 4, 2011
1001139	April to Sept 24, 2011
1001166	June 25, 2011 to September 24, 2011
1004852	June 30, 2011 to Oct. 13, 2011
1005772	May 7, 2011 to October 15, 2011

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'June ','06/')
WHERE Season1Date like "%, 2011";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'November ','11/')
WHERE Season1Date like "%, 2011";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'May ','05/')
WHERE Season1Date like "%, 2011";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'September ','09/')
WHERE Season1Date like "%, 2011";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'Sept ','09/')
WHERE Season1Date like "%, 2011";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'Oct. ','10/')
WHERE Season1Date like "%, 2011";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'October ','10/')
WHERE Season1Date like "%, 2011";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, ', 2011','/2011')
WHERE Season1Date like "%, 2011";
```

Due to codes we run above, in fact we missed some rows in which only the part before “to” contains comma. So we ran the code below:

```
SELECT FMID, Season1Date
FROM fmv2
WHERE Season1Date like "%,%";
```

FMD	Season1Date
1001722	June 2, 2012 to September
1002713	May 21, 2011 to October
1003827	June 11, 2012 to October
1004736	July 1, 2012 to December
1004753	July 11, 2012 to November

And then we used codes below to fix them:

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'June ','06/')
WHERE Season1Date like "%,%";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'May ','05/')
WHERE Season1Date like "%,%";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, 'July ','07/')
WHERE Season1Date like "%,%";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, ', 2011','/2011')
WHERE Season1Date like "%,%";
```

```
UPDATE fmv2 SET Season1Date=replace(Season1Date, ', 2012','/2012')
WHERE Season1Date like "%,%";
```

We ran the similar queries for Season2Date and Season3Date:

Season2Date:

```
SELECT Season2Date
FROM fmv2
WHERE Season2Date like "%,%";
```

Result:

Season2Date
August 18, 2012 to August 18, 2012
Aug 23, 2012 to Nov 1, 2012
June 15, 2012 to June

Queries to fix:

```
UPDATE fmv2 SET Season2Date=replace(Season2Date, 'Nov ','11/')
WHERE Season2Date like "%,%";
```

```
UPDATE fmv2 SET Season2Date=replace(Season2Date, 'Aug ','08/')
WHERE Season2Date like "%,%";
```

```
UPDATE fmv2 SET Season2Date=replace(Season2Date, 'August ','08/')
WHERE Season2Date like "%,%";
```

```
UPDATE fmv2 SET Season2Date=replace(Season2Date, 'June ', '06/')  
WHERE Season2Date like "%,%";
```

```
UPDATE fmv2 SET Season2Date=replace(Season2Date, ', 2012', '/2012')  
WHERE Season2Date like "%,%";
```

Season3Date:

```
SELECT Season3Date  
FROM fmv2  
WHERE Season3Date like "%,%";
```

Result:

Season3Date
July 1, 2012 to September 1, 2012

Queries to fix:

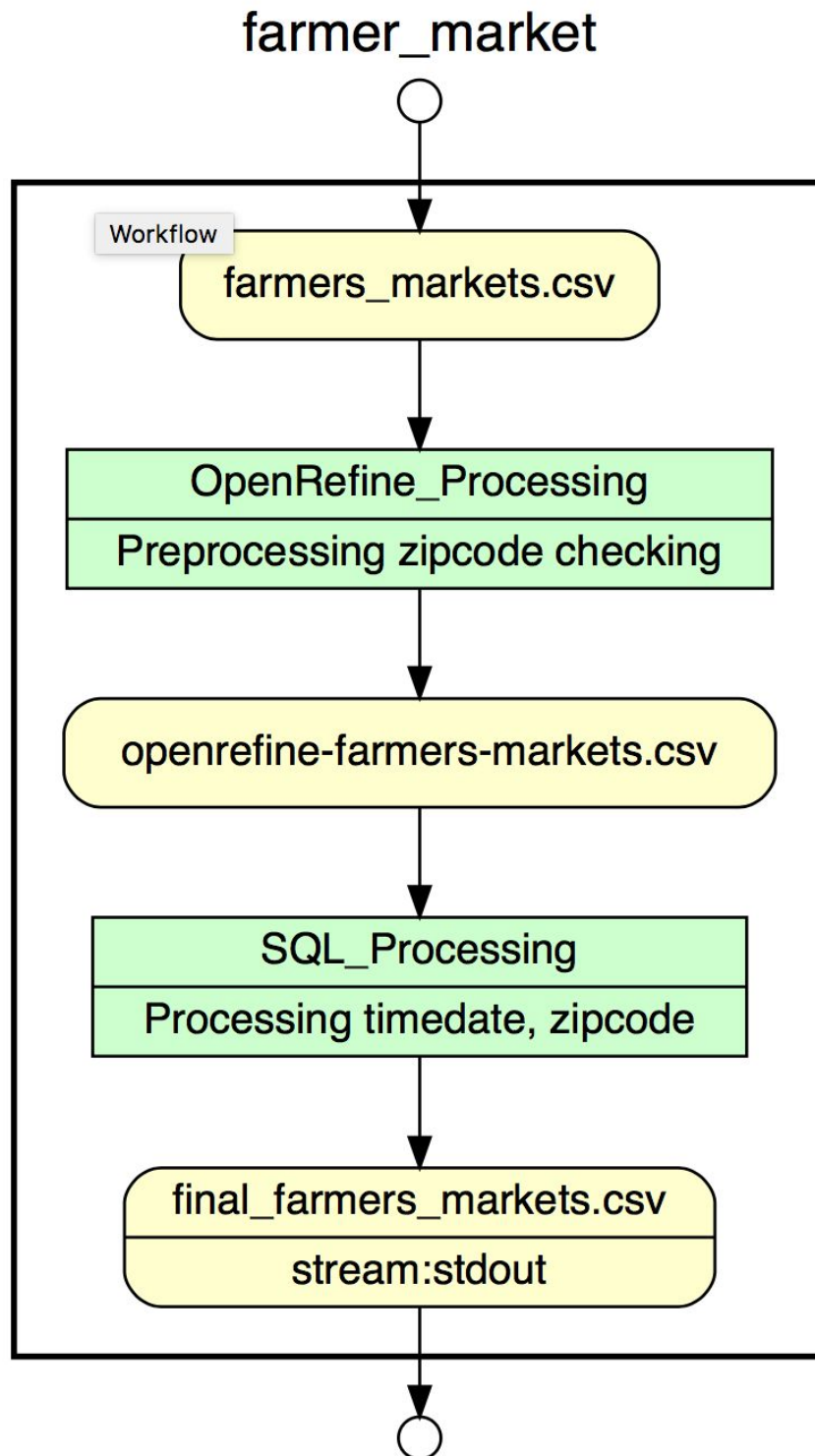
```
UPDATE fmv2 SET Season3Date=replace(Season3Date, 'July ', '07/')  
WHERE Season3Date like "%,%";
```

```
UPDATE fmv2 SET Season3Date=replace(Season3Date, 'September ', '09/')  
WHERE Season3Date like "%,%";
```

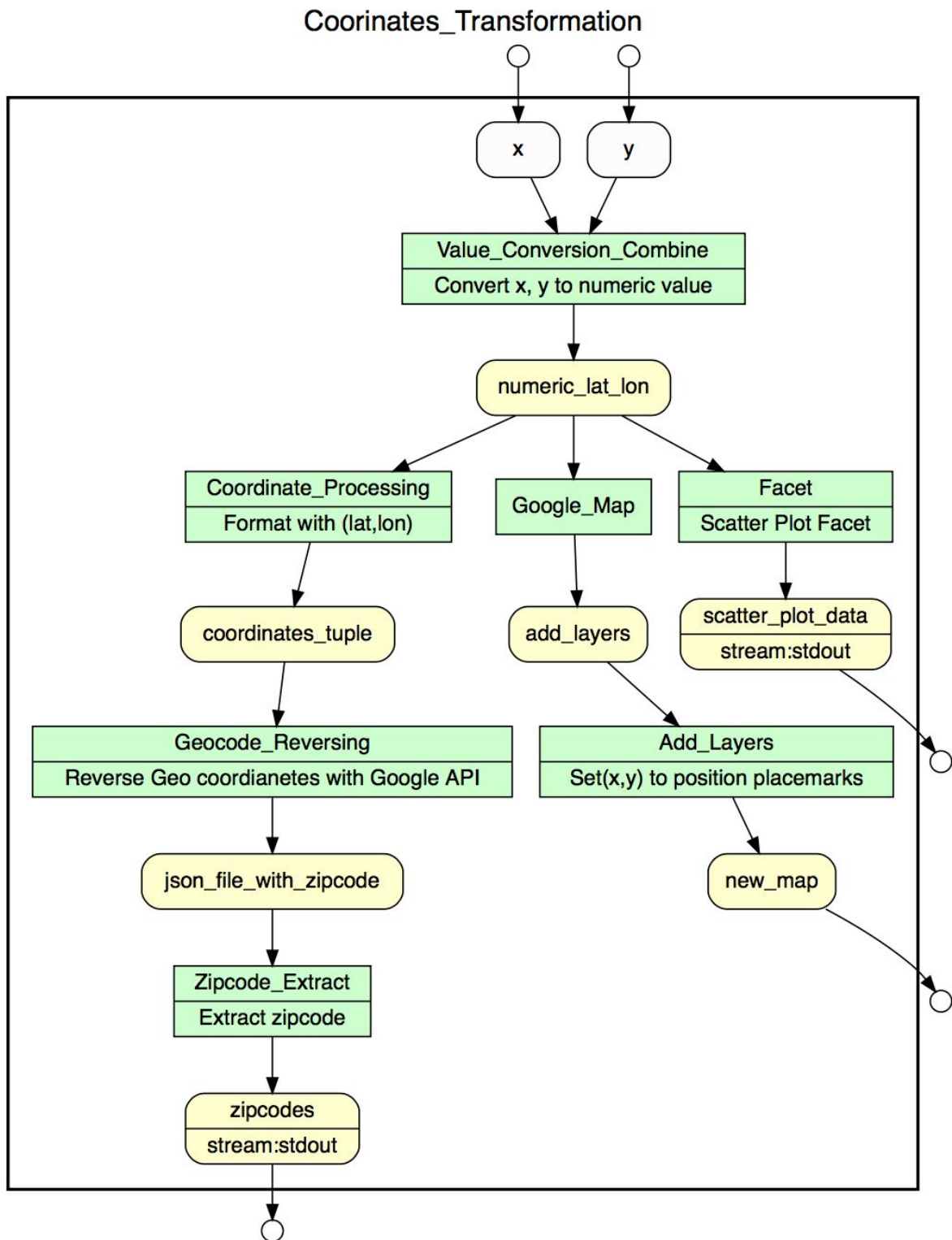
```
UPDATE fmv2 SET Season3Date=replace(Season3Date, ', 2012', '/2012')  
WHERE Season3Date like "%,%";
```

4 Data Cleaning Workflow

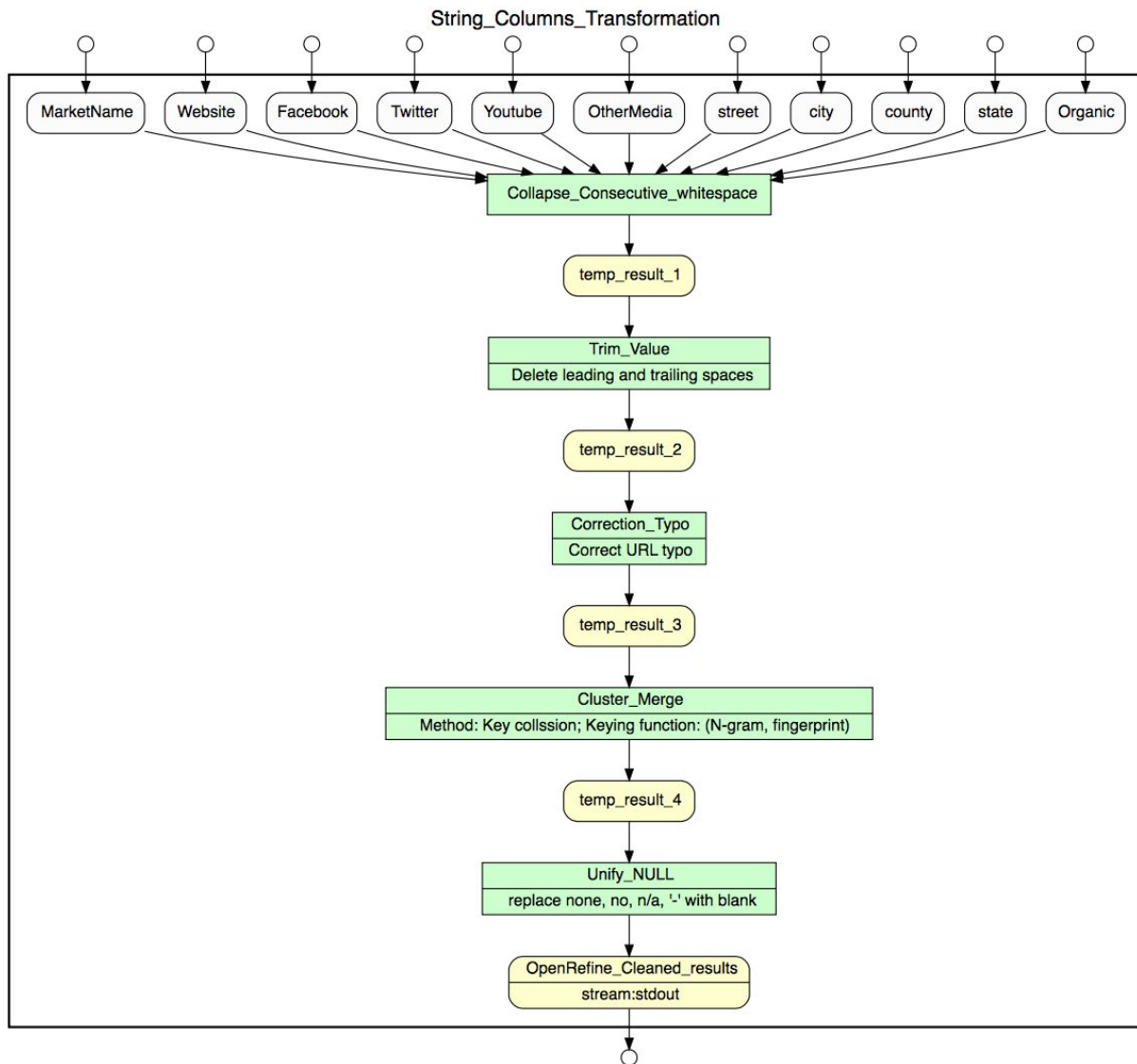
4.1 Overall Cleaning Processing



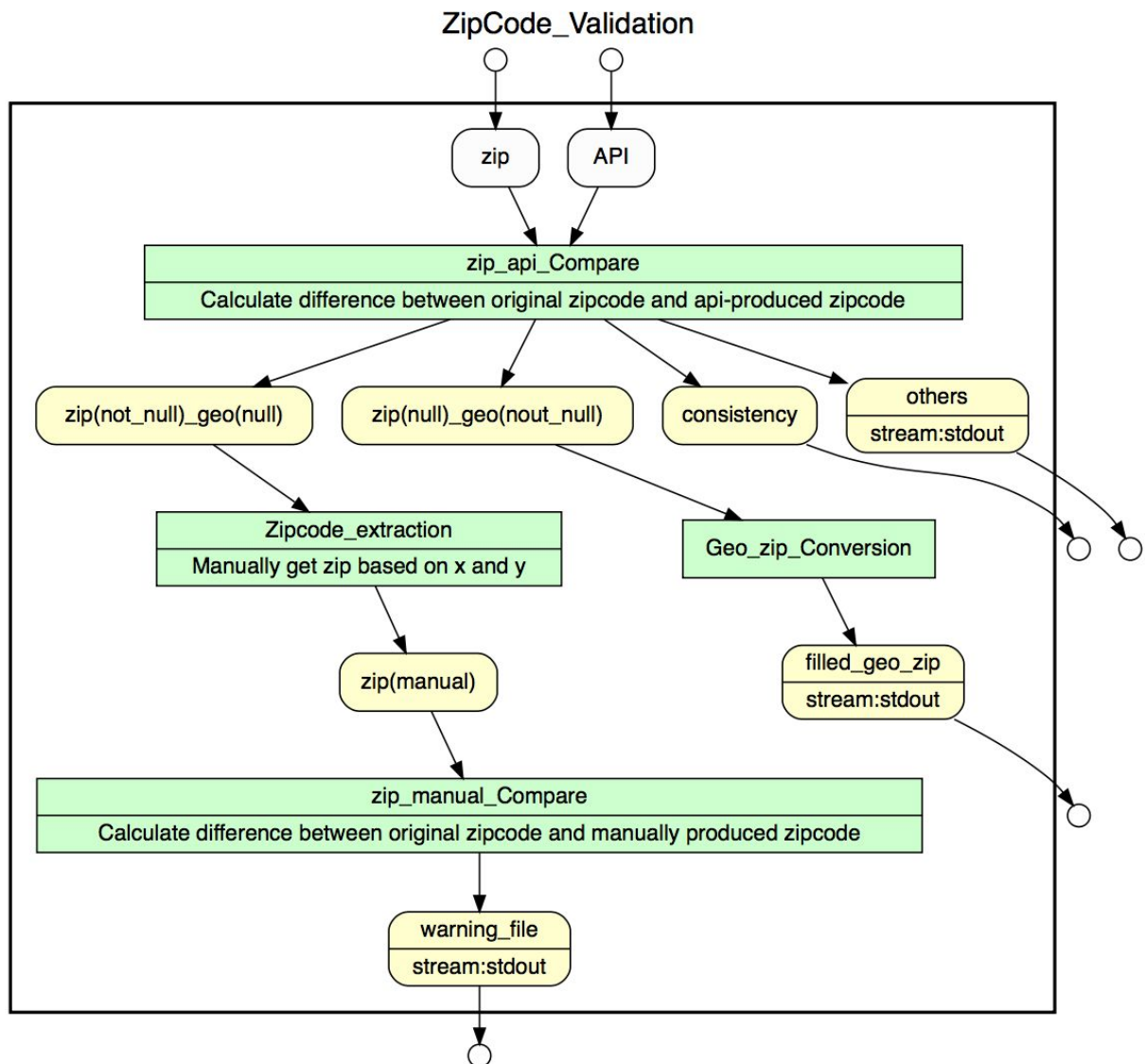
4.2 Coordinates Transformation in OpenRefine and Google Map



4.3 String Columns Transformation in OpenRefine



4.4 Zip Code Validation in SQLite



4.5 Date Format Transformation in SQLite

