# Project Report –CS513 Summer 2020
# Dataset: US Farmers Markets
**Saurav Chetry (schetry2@illiois.edu)**

1. **Overview & Initial Assessment of Dataset:**

Data Description:

The data is about farmers market, the produce available at different seasons and the location co-ordinates to the markets for customers to plan their visit for purchases. Besides, there is also information about what types of cash transactions are offered by individual markets. In short it looks like a Google Maps review page of a map where such information is available for end users.

There are total Rows: 8687, Columns: 59 and contains the market name, website, address, social media information, products, season times, and updated time for the entry.

Feasible Use-Case and Data Cleaning Goals:

The data can be used for an online search tool for customers looking to purchase local produce. With the name, location, season data and time, produce details, purchasing options and location co-ordinates, customers can use the tool to navigate to the market.

For the navigation use case, the markets must offer at least either the latitude longitude co-ordinates or a combination of any of street, state, county or zip. Besides information about social media can be used to collate the reviews and photos of the markets in the search use case.

Social Media and Location information must be cleaned and made ready to use for Navigation and Search use-case.
This will involve correcting Social Media URLs, hyperlinks and market websites.
The ZIP code needs to be correct by removing special characters which are not required.
Season Date/Time must be correct to match the produce availability and to appear on search based on relevance.
Unique Market name and City name will in the use case.
Correct "UpdateTime" is useful for customers to see the latest information.
Purchasing options and produce options are useful for convenience to the customers.

Data Ready for following Use-Cases:
1. Offline database about markets, location and season details. This is a look-up use case good for record-keeping or auditing.

Partially ready Data for following Use-Cases
2. Navigation apps which can locate markets based on latitudes and longitudes.
3. Online application for search, review of markets – data cleaning is required.

Data Not Suitable for following Use-Cases:
1. Contact and Appointment Scheduling apps. No customer contact details available.
2. Legal, administrative solutions as there is no Owner details available.

List of data quality issues:

| **Fields** | **Issues** |
|---|---|
| **FMID** | They are unique values. They have whitespaces and  trailing/leading  zeros. |
| **Market Names** | They have different name variations for the same markets. Same markets do open at different seasons or years. The names can be clustered to a more suitable name. |
| **SocialMedia (All)** | There issues are incorrect URLs, hyperlinks etc.<br>For "website", some fields are blank, and some have both https and http format.<br>For "Twitter", some have https, and some have @ format.<br>For "Facebook", some do not have the URLs but simple<br>For "Youtube", most fields are empty.<br>For "OtherMedia", most fields are blank. Incorrect URLs can be removed. |
| **Season1 Date** | There are different formats of dates. MMM, MM-DD-YYYY, YYYY-MM-DD etc.<br>Some Season Start Dates are Older  than the Season End Dates.<br>Some markets do not have year values for seasonal data, and some are missing end data values.<br>Some markets do not have year values for seasonal data, and some are missing end data values.<br>Most products offered in markets have availability for single season. For them other Season Date and Time are missing. |
| **Other Season Dates** | Same issues as Season1 Date |
| **Season Time** | They have whitespaces and  trailing/leading zeros. |
| **Street** | They have whitespaces and  trailing/leading zeros. |
| **State** | They have whitespaces and  trailing/leading zeros. |
| **County** | They have whitespaces and  trailing/leading zeros.<br>There are multiple markets for same city, this can be clustered. |
| **ZIP** | They have non-numeric entries.<br>Some values are missing. Not a big problem if other details like street and county data are available. |
| **City** | They have whitespaces and  trailing/leading zeros. |
| **Location** | Most markets do not have this information. Others have whitespaces and  trailing/leading zeros. |
| **Produce (All)** | "Organic" products field has special characters |
| **Update Time** | They have whitespaces and  trailing/leading zeros.<br>There is a mix of "MM/DD/YYYY Time" and "Month DD Year Time" format. |

Overall the data is not useful as-is and would lead to questions and confusions for the end users. The data needs to be cleaned to make it consumption ready.

## 2. Data Cleaning with OpenRefine:

The goal is to implement the use case of the search and navigation application. With that in mind, the following is the overview of the OpenRefine steps carried out on the raw data.

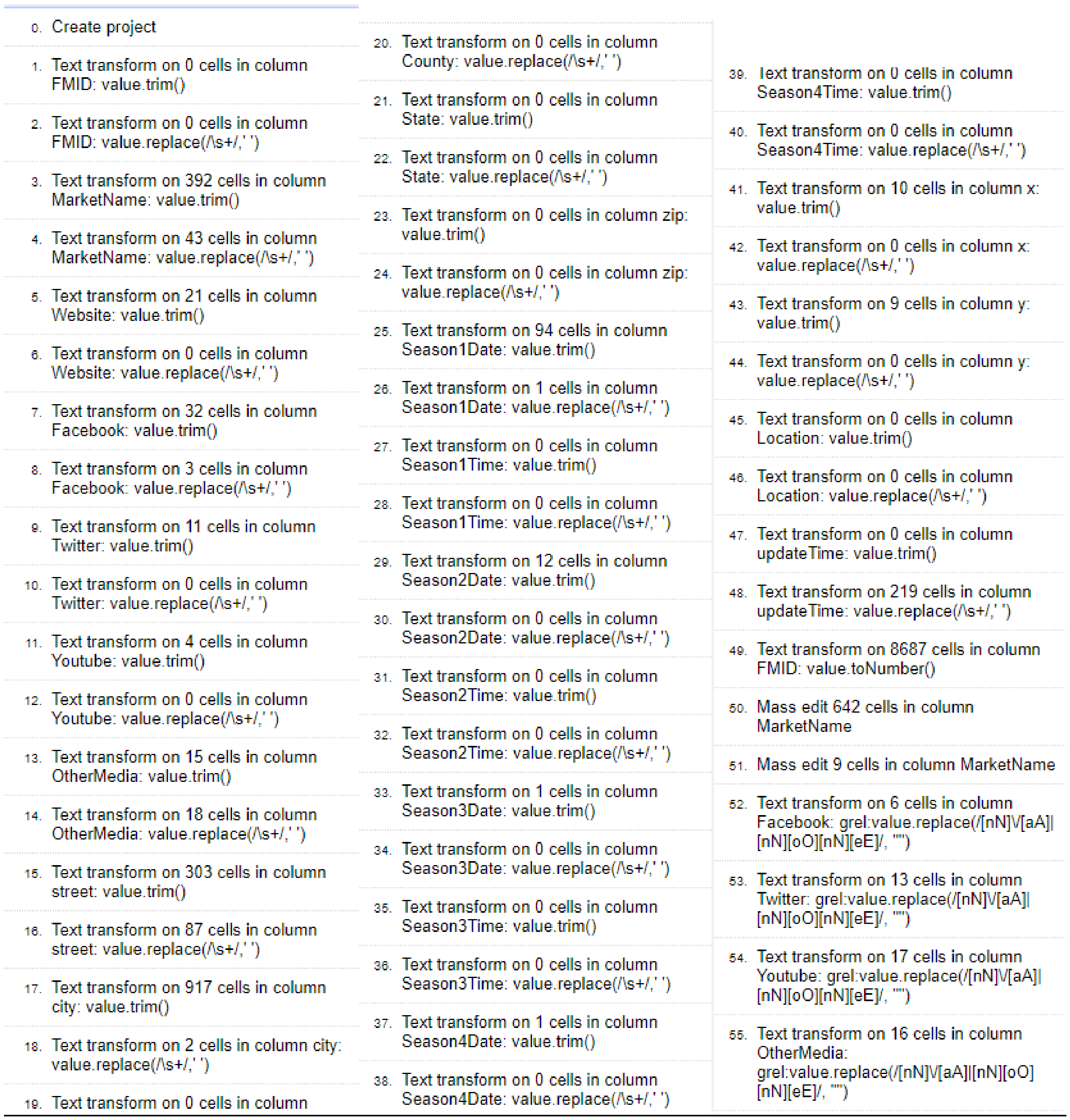
| Operations / Column | Trim | Whitespace | Duplicates | ToDate | Split Columns | Value Replace | Cluster | DataValidation Python/Jython |
|---|---|---|---|---|---|---|---|---|
| **FMID** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Market Names** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ |
| **SocialMedia All** | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ |
| **Season Date** | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| **Season Time** | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Street** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **State** | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **County** | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **ZIP** | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| **City** | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| **Organic** | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| **Update Time** | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |

| Fields | OpenRefine Operation |
|---|---|
| **All Fields** | All fields had leading, trailing and intermediate whitespaces. Trimming and collapsing the whitespaces was done on all fields. |
| **FMID,Market Names, Social Media fields, Season Date and Street** | Check for Duplicates. |
| **Market Name** | Market Name had multiple variant names with separate unique FMIDs and other details. The names were clustered, and a common name was chosen for the market. |

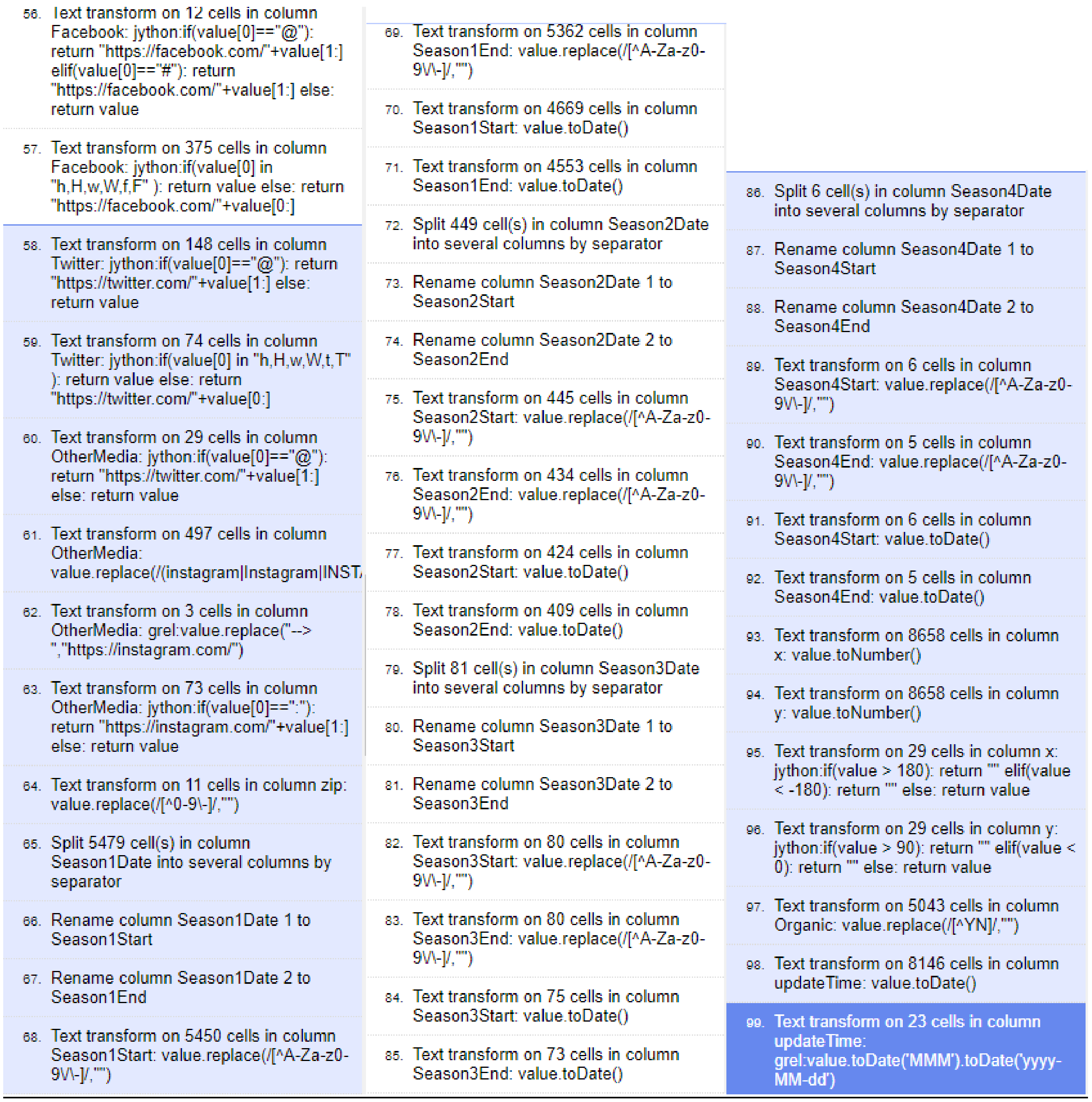
| | |
|---|---|
| | Make a facet and perform the cluster operation using the *key-collison* method and *fingerprint* function. Merge the relevant clusters. |
| **City** | same operation as Market Name |
| **All SeasonDate** | Column split into two with separator “to” to create start and end Season dates.<br>Text changed to Date format yyyy-MM-dd HH:MM:SS<br>Remove non-numeric and not allowed characters using **value.replace(/[^A-Za-z0-9\\/\\-]/,\"\")** |
| **All Social Media** | None or NA values were removed using **value.replace(/[nN]\\/[aA]\|[nN][oO][nN][eE]/, \"\")**<br><br>Texts starting with “@” were replaced with appropriate URLs using Pyhton E.g:<br>**if(value[0]==\"@\"): return \"https://facebook.com/\"+value[1:]\nelif(value[0]==\"#\"): return \"https://facebook.com/\"+value[1:]\nelse: return value**<br><br>Texts not starting with www, WWW, http, Http etc. were replaced with appropriate URLs using Python, e.g.<br>**if(value[0] in \"h,H,w,W,f,F\" ): return value \nelse: return \"https://facebook.com/\"+value[0:]"**<br><br>**if(value[0] in \"h,H,w,W,t,T\" ): return value \nelse: return \"https://twitter.com/\"+value[0:]** |
| **Other Media** | None or NA values were removed using **value.replace(/[nN]\\/[aA]\|[nN][oO][nN][eE]/, \"\")**<br><br>Text starting with @ was replaced with twitter URL:<br>**if(value[0]==\"@\"): return \"https://twitter.com/\"+value[1:]\nelse: return value**<br><br>Text having mention of Instagram, but no clickable links were replaced as:<br>**value.replace(/(instagram\|Instagram\|INSTAGRAM)/,\"\")**<br>**value.replace(\"--> \",\"https://instagram.com/\")**<br>**if(value[0]==\":\"): return \"https://instagram.com/\"+value[1:]\nelse: return value** |
| **ZIP** | Invalid and not allowed characters were removed as **value.replace(/[^0-9\\-]/,\"\")** |
| **X, Y** | Converted all text to Numbers<br>Validated and removed values of **180<X<-180** as:<br>**if(value > 180): return \"\"\nelif(value < -180): return \"\"\nelse: return value**<br><br>Validated and removed values of **0<Y<90** as:<br>**if(value > 90): return \"\"\nelif(value < 0): return \"\"\nelse: return value** |
| **Organic** | Replaced characters other than Y or N as:<br>**value.replace(/[^YN]/,\"\")** |
| **UpdateTime** | Text changed to Date format yyyy-MM-dd HH:MM:SS |

Quantification of above changes:

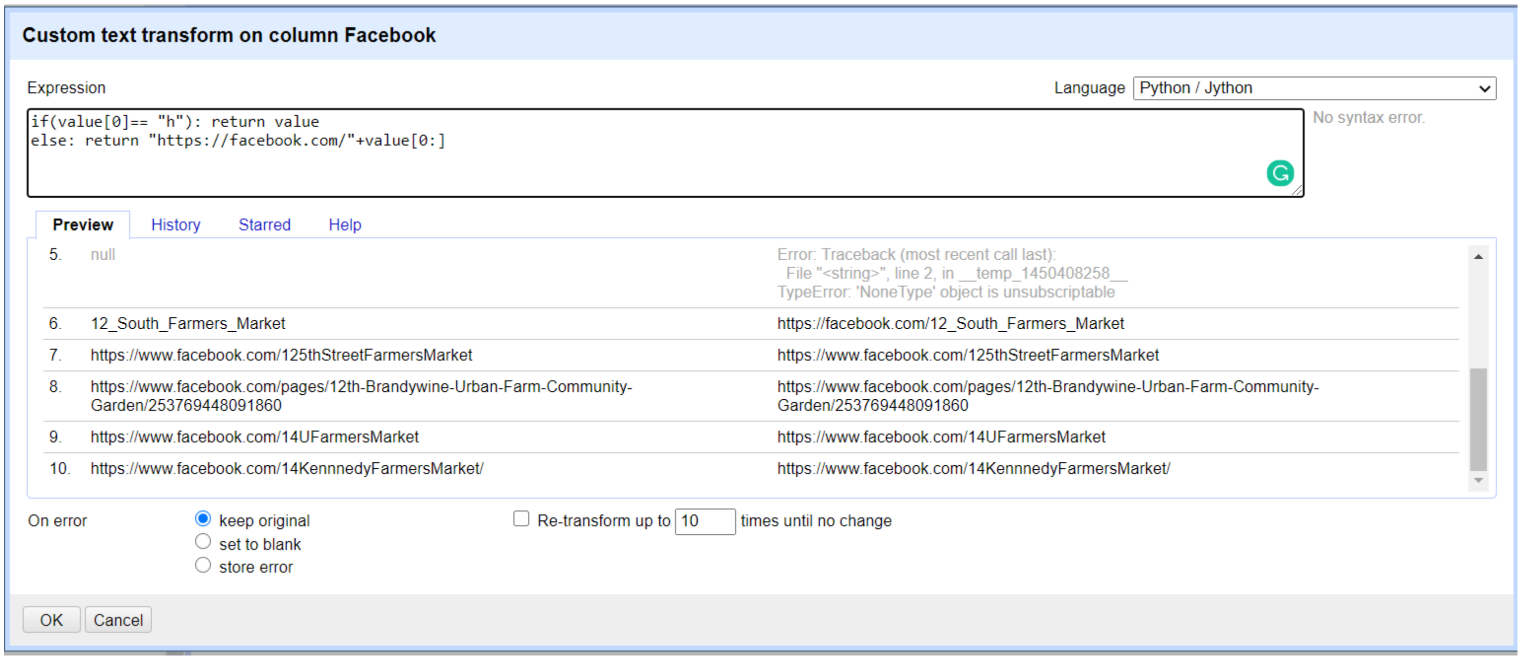
Below are the snapshots of changes effected to the raw data by each OpenRefine operation mentioned above.



0. Create project
1. Text transform on 0 cells in column FMID: value.trim()
2. Text transform on 0 cells in column FMID: value.replace(/\s+/,' ')
3. Text transform on 392 cells in column MarketName: value.trim()
4. Text transform on 43 cells in column MarketName: value.replace(/\s+/,' ')
5. Text transform on 21 cells in column Website: value.trim()
6. Text transform on 0 cells in column Website: value.replace(/\s+/,' ')
7. Text transform on 32 cells in column Facebook: value.trim()
8. Text transform on 3 cells in column Facebook: value.replace(/\s+/,' ')
9. Text transform on 11 cells in column Twitter: value.trim()
10. Text transform on 0 cells in column Twitter: value.replace(/\s+/,' ')
11. Text transform on 4 cells in column Youtube: value.trim()
12. Text transform on 0 cells in column Youtube: value.replace(/\s+/,' ')
13. Text transform on 15 cells in column OtherMedia: value.trim()
14. Text transform on 18 cells in column OtherMedia: value.replace(/\s+/,' ')
15. Text transform on 303 cells in column street: value.trim()
16. Text transform on 87 cells in column street: value.replace(/\s+/,' ')
17. Text transform on 917 cells in column city: value.trim()
18. Text transform on 2 cells in column city: value.replace(/\s+/,' ')
19. Text transform on 0 cells in column
20. Text transform on 0 cells in column County: value.replace(/\s+/,' ')
21. Text transform on 0 cells in column State: value.trim()
22. Text transform on 0 cells in column State: value.replace(/\s+/,' ')
23. Text transform on 0 cells in column zip: value.trim()
24. Text transform on 0 cells in column zip: value.replace(/\s+/,' ')
25. Text transform on 94 cells in column Season1Date: value.trim()
26. Text transform on 1 cells in column Season1Date: value.replace(/\s+/,' ')
27. Text transform on 0 cells in column Season1Time: value.trim()
28. Text transform on 0 cells in column Season1Time: value.replace(/\s+/,' ')
29. Text transform on 12 cells in column Season2Date: value.trim()
30. Text transform on 0 cells in column Season2Date: value.replace(/\s+/,' ')
31. Text transform on 0 cells in column Season2Time: value.trim()
32. Text transform on 0 cells in column Season2Time: value.replace(/\s+/,' ')
33. Text transform on 1 cells in column Season3Date: value.trim()
34. Text transform on 0 cells in column Season3Date: value.replace(/\s+/,' ')
35. Text transform on 0 cells in column Season3Time: value.trim()
36. Text transform on 0 cells in column Season3Time: value.replace(/\s+/,' ')
37. Text transform on 1 cells in column Season4Date: value.trim()
38. Text transform on 0 cells in column Season4Date: value.replace(/\s+/,' ')
39. Text transform on 0 cells in column Season4Time: value.trim()
40. Text transform on 0 cells in column Season4Time: value.replace(/\s+/,' ')
41. Text transform on 10 cells in column x: value.trim()
42. Text transform on 0 cells in column x: value.replace(/\s+/,' ')
43. Text transform on 9 cells in column y: value.trim()
44. Text transform on 0 cells in column y: value.replace(/\s+/,' ')
45. Text transform on 0 cells in column Location: value.trim()
46. Text transform on 0 cells in column Location: value.replace(/\s+/,' ')
47. Text transform on 0 cells in column updateTime: value.trim()
48. Text transform on 219 cells in column updateTime: value.replace(/\s+/,' ')
49. Text transform on 8687 cells in column FMID: value.toNumber()
50. Mass edit 642 cells in column MarketName
51. Mass edit 9 cells in column MarketName
52. Text transform on 6 cells in column Facebook: grel:value.replace(/[nN]\/[aA]|[nN][oO][nN][eE]/, "")
53. Text transform on 13 cells in column Twitter: grel:value.replace(/[nN]\/[aA]|[nN][oO][nN][eE]/, "")
54. Text transform on 17 cells in column Youtube: grel:value.replace(/[nN]\/[aA]|[nN][oO][nN][eE]/, "")
55. Text transform on 16 cells in column OtherMedia: grel:value.replace(/[nN]\/[aA]|[nN][oO][nN][eE]/, "")



56. Text transform on 12 cells in column Facebook: jython:if(value[0]=="@"): return "https://facebook.com/"+value[1:] elif(value[0]=="#"): return "https://facebook.com/"+value[1:] else: return value
57. Text transform on 375 cells in column Facebook: jython:if(value[0] in "h,H,w,W,f,F" ): return value else: return "https://facebook.com/"+value[0:]
58. Text transform on 148 cells in column Twitter: jython:if(value[0]=="@"): return "https://twitter.com/"+value[1:] else: return value
59. Text transform on 74 cells in column Twitter: jython:if(value[0] in "h,H,w,W,t,T" ): return value else: return "https://twitter.com/"+value[0:]
60. Text transform on 29 cells in column OtherMedia: jython:if(value[0]=="@"): return "https://twitter.com/"+value[1:] else: return value
61. Text transform on 497 cells in column OtherMedia: value.replace(/(instagram|Instagram|INST
62. Text transform on 3 cells in column OtherMedia: grel:value.replace("--> ","https://instagram.com/")
63. Text transform on 73 cells in column OtherMedia: jython:if(value[0]==":"): return "https://instagram.com/"+value[1:] else: return value
64. Text transform on 11 cells in column zip: value.replace(/[^0-9\-]/,"")
65. Split 5479 cell(s) in column Season1Date into several columns by separator
66. Rename column Season1Date 1 to Season1Start
67. Rename column Season1Date 2 to Season1End
68. Text transform on 5450 cells in column Season1Start: value.replace(/[^A-Za-z0-9\/\-]/,"")
69. Text transform on 5362 cells in column Season1End: value.replace(/[^A-Za-z0-9\/\-]/,"")
70. Text transform on 4669 cells in column Season1Start: value.toDate()
71. Text transform on 4553 cells in column Season1End: value.toDate()
72. Split 449 cell(s) in column Season2Date into several columns by separator
73. Rename column Season2Date 1 to Season2Start
74. Rename column Season2Date 2 to Season2End
75. Text transform on 445 cells in column Season2Start: value.replace(/[^A-Za-z0-9\/\-]/,"")
76. Text transform on 434 cells in column Season2End: value.replace(/[^A-Za-z0-9\/\-]/,"")
77. Text transform on 424 cells in column Season2Start: value.toDate()
78. Text transform on 409 cells in column Season2End: value.toDate()
79. Split 81 cell(s) in column Season3Date into several columns by separator
80. Rename column Season3Date 1 to Season3Start
81. Rename column Season3Date 2 to Season3End
82. Text transform on 80 cells in column Season3Start: value.replace(/[^A-Za-z0-9\/\-]/,"")
83. Text transform on 80 cells in column Season3End: value.replace(/[^A-Za-z0-9\/\-]/,"")
84. Text transform on 75 cells in column Season3Start: value.toDate()
85. Text transform on 73 cells in column Season3End: value.toDate()
86. Split 6 cell(s) in column Season4Date into several columns by separator
87. Rename column Season4Date 1 to Season4Start
88. Rename column Season4Date 2 to Season4End
89. Text transform on 6 cells in column Season4Start: value.replace(/[^A-Za-z0-9\/\-]/,"")
90. Text transform on 5 cells in column Season4End: value.replace(/[^A-Za-z0-9\/\-]/,"")
91. Text transform on 6 cells in column Season4Start: value.toDate()
92. Text transform on 5 cells in column Season4End: value.toDate()
93. Text transform on 8658 cells in column x: value.toNumber()
94. Text transform on 8658 cells in column y: value.toNumber()
95. Text transform on 29 cells in column x: jython:if(value > 180): return "" elif(value < -180): return "" else: return value
96. Text transform on 29 cells in column y: jython:if(value > 90): return "" elif(value < 0): return "" else: return value
97. Text transform on 5043 cells in column Organic: value.replace(/[^YN]/,"")
98. Text transform on 8146 cells in column updateTime: value.toDate()
99. Text transform on 23 cells in column updateTime: grel:value.toDate('MMM').toDate('yyyy-MM-dd')



**Custom text transform on column Facebook**

Expression

Language: Python / Jython

```
if(value[0]== "h"): return value
else: return "https://facebook.com/"+value[0:]
```

No syntax error.

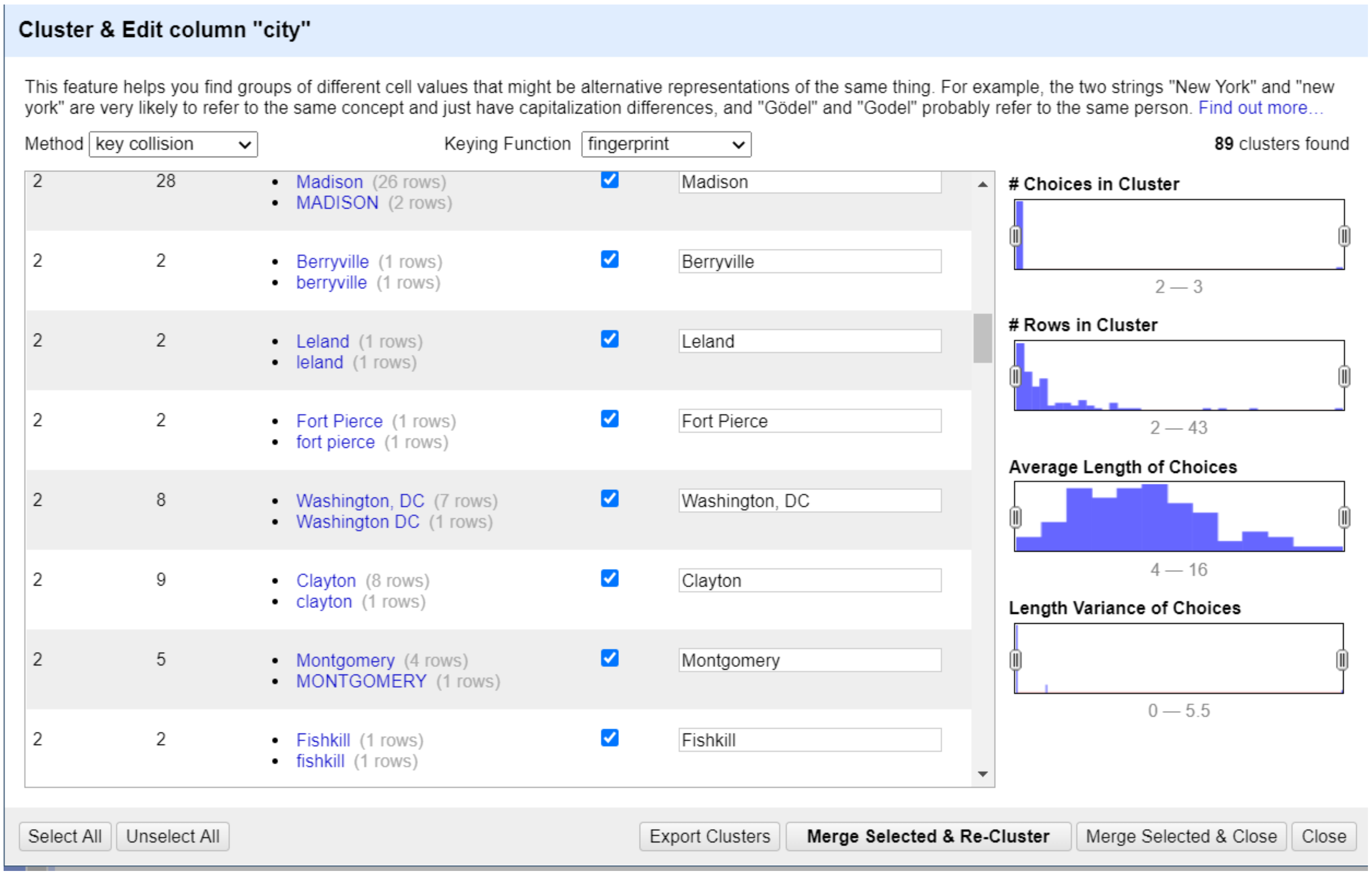
Preview | History | Starred | Help

| | | |
|---|---|---|
| 5. | null | Error: Traceback (most recent call last): File "<string>", line 2, in __temp_1450408258__ TypeError: 'NoneType' object is unsubscriptable |
| 6. | 12_South_Farmers_Market | https://facebook.com/12_South_Farmers_Market |
| 7. | https://www.facebook.com/125thStreetFarmersMarket | https://www.facebook.com/125thStreetFarmersMarket |
| 8. | https://www.facebook.com/pages/12th-Brandywine-Urban-Farm-Community-Garden/253769448091860 | https://www.facebook.com/pages/12th-Brandywine-Urban-Farm-Community-Garden/253769448091860 |
| 9. | https://www.facebook.com/14UFarmersMarket | https://www.facebook.com/14UFarmersMarket |
| 10. | https://www.facebook.com/14KennnedyFarmersMarket/ | https://www.facebook.com/14KennnedyFarmersMarket/ |

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

☐ Re-transform up to 10 times until no change

OK Cancel



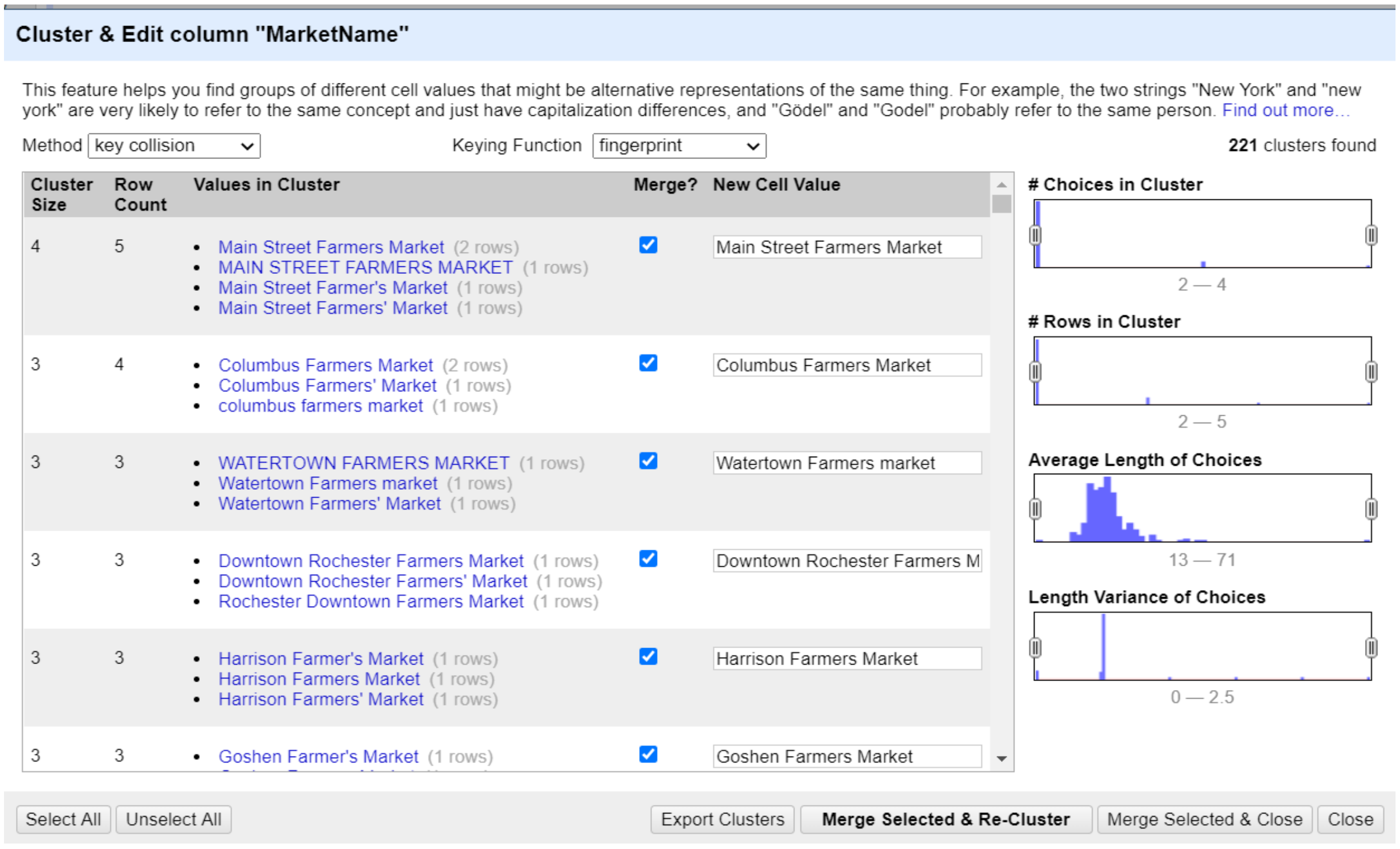
**Cluster & Edit column "city"**

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 **89** clusters found

| | | | | |
|---|---|---|---|---|
| 2 | 28 | • Madison (26 rows) • MADISON (2 rows) | ✓ | Madison |
| 2 | 2 | • Berryville (1 rows) • berryville (1 rows) | ✓ | Berryville |
| 2 | 2 | • Leland (1 rows) • leland (1 rows) | ✓ | Leland |
| 2 | 2 | • Fort Pierce (1 rows) • fort pierce (1 rows) | ✓ | Fort Pierce |
| 2 | 8 | • Washington, DC (7 rows) • Washington DC (1 rows) | ✓ | Washington, DC |
| 2 | 9 | • Clayton (8 rows) • clayton (1 rows) | ✓ | Clayton |
| 2 | 5 | • Montgomery (4 rows) • MONTGOMERY (1 rows) | ✓ | Montgomery |
| 2 | 2 | • Fishkill (1 rows) • fishkill (1 rows) | ✓ | Fishkill |

Select All Unselect All Export Clusters **Merge Selected & Re-Cluster** Merge Selected & Close Close

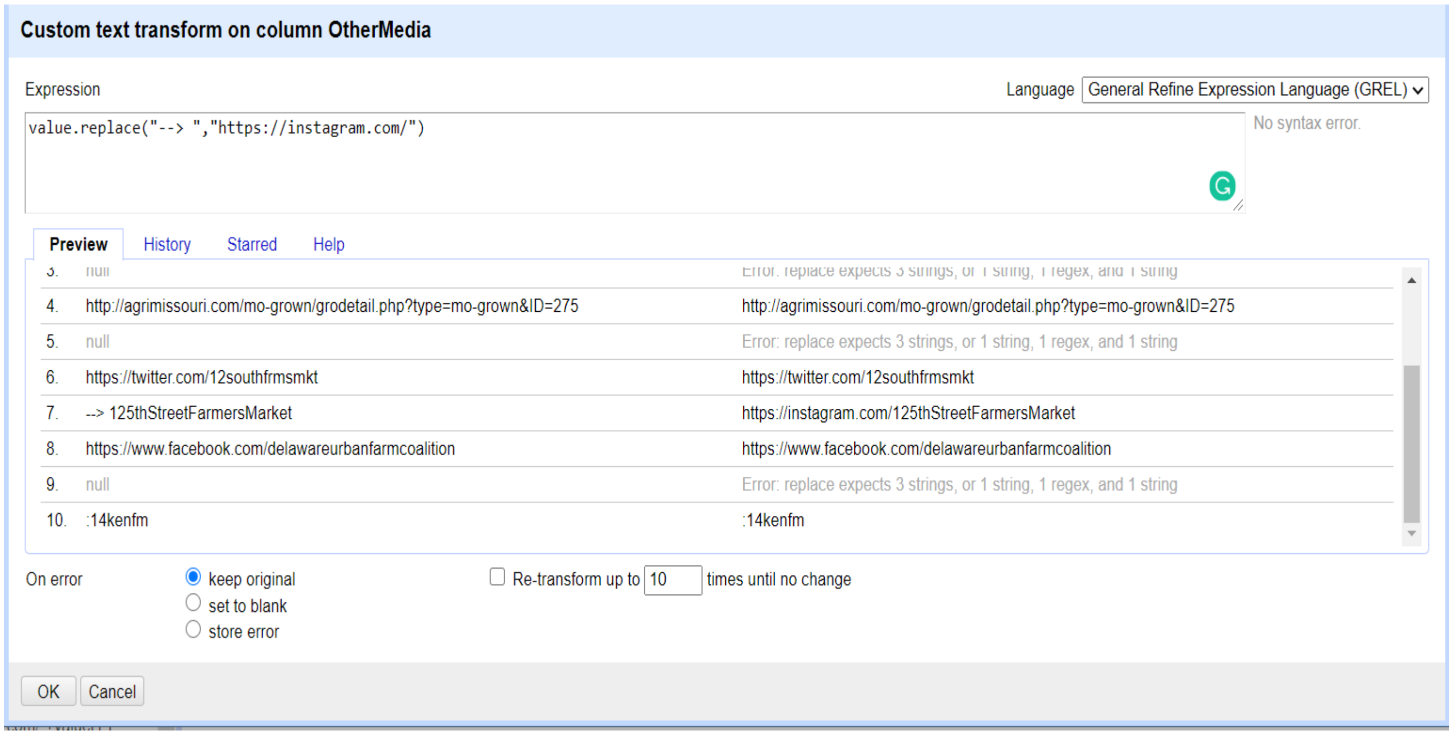


**Cluster & Edit column "MarketName"**

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 — **221** clusters found

| Cluster Size | Row Count | Values in Cluster | Merge? | New Cell Value |
|---|---|---|---|---|
| 4 | 5 | Main Street Farmers Market (2 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 | 3 | Downtown Rochester Farmers Market (1 rows); Downtown Rochester Farmers' Market (1 rows); Rochester Downtown Farmers Market (1 rows) | ✓ | Downtown Rochester Farmers M |
| 3 | 3 | Harrison Farmer's Market (1 rows); Harrison Farmers Market (1 rows); Harrison Farmers' Market (1 rows) | ✓ | Harrison Farmers Market |
| 3 | 3 | Goshen Farmer's Market (1 rows) | ✓ | Goshen Farmers Market |



**Custom text transform on column OtherMedia**

Expression — Language: General Refine Expression Language (GREL)

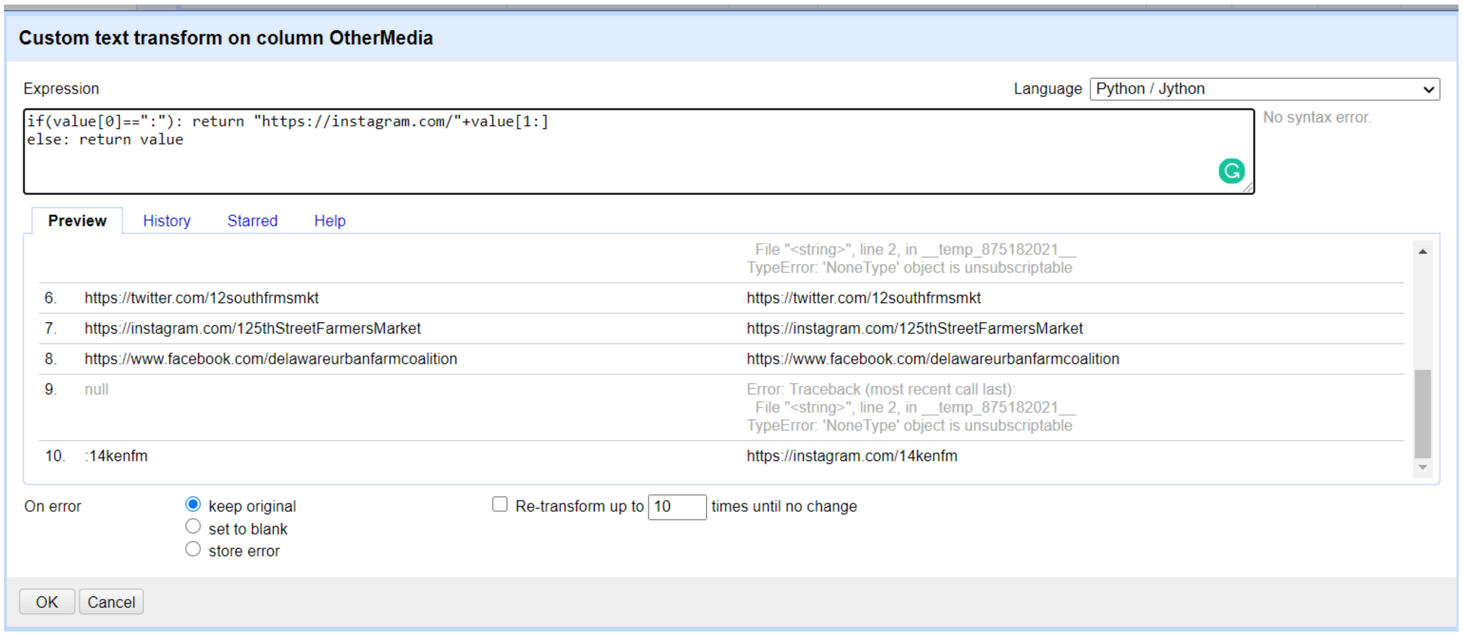
```
value.replace("--> ","https://instagram.com/")
```

No syntax error.

Preview | History | Starred | Help

| | | |
|---|---|---|
| 3. | null | Error: replace expects 3 strings, or 1 string, 1 regex, and 1 string |
| 4. | http://agrimissouri.com/mo-grown/grodetail.php?type=mo-grown&ID=275 | http://agrimissouri.com/mo-grown/grodetail.php?type=mo-grown&ID=275 |
| 5. | null | Error: replace expects 3 strings, or 1 string, 1 regex, and 1 string |
| 6. | https://twitter.com/12southfrmsmkt | https://twitter.com/12southfrmsmkt |
| 7. | --> 125thStreetFarmersMarket | https://instagram.com/125thStreetFarmersMarket |
| 8. | https://www.facebook.com/delawareurbanfarmcoalition | https://www.facebook.com/delawareurbanfarmcoalition |
| 9. | null | Error: replace expects 3 strings, or 1 string, 1 regex, and 1 string |
| 10. | :14kenfm | :14kenfm |

On error: [✓ keep original] set to blank / store error — ☐ Re-transform up to 10 times until no change


Custom text transform on column OtherMedia

Expression — Language: Python / Jython

```
if(value[0]==":"): return "https://instagram.com/"+value[1:]
else: return value
```

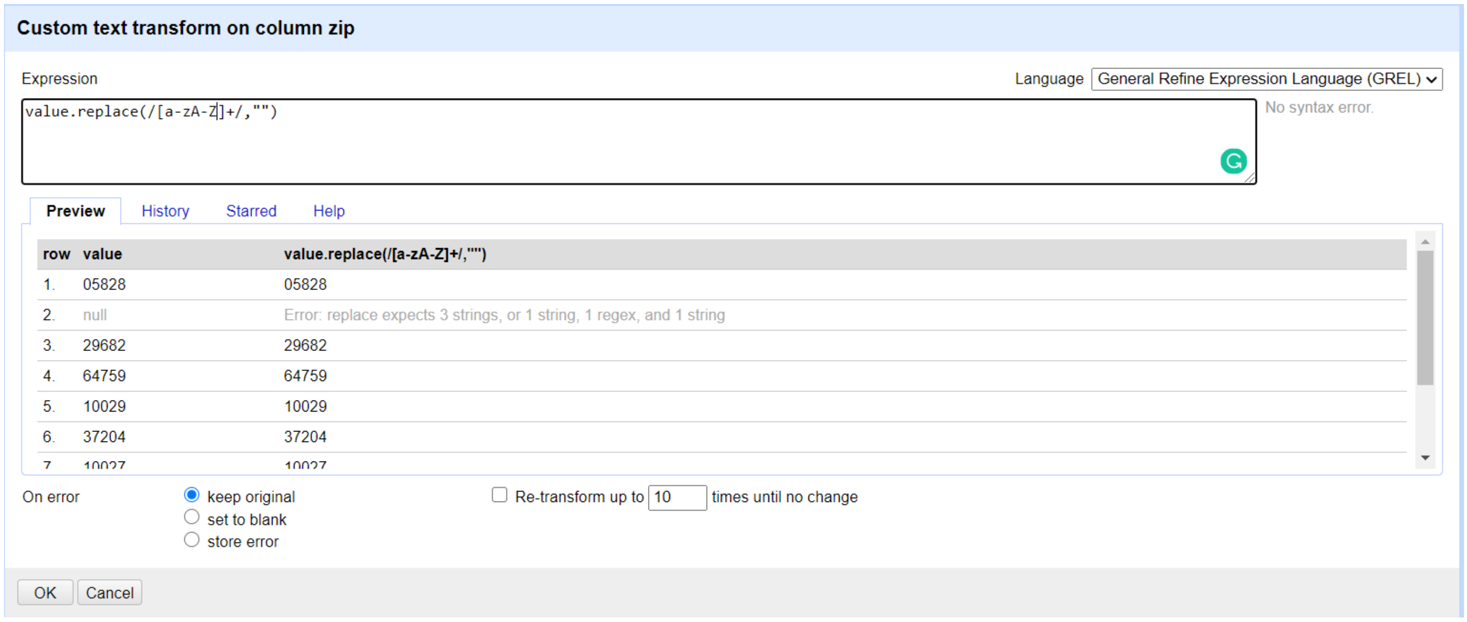
No syntax error.

Preview | History | Starred | Help

```
         File "<string>", line 2, in __temp_875182021__
       TypeError: 'NoneType' object is unsubscriptable
6.   https://twitter.com/12southfrmsmkt                 https://twitter.com/12southfrmsmkt
7.   https://instagram.com/125thStreetFarmersMarket     https://instagram.com/125thStreetFarmersMarket
8.   https://www.facebook.com/delawareurbanfarmcoalition   https://www.facebook.com/delawareurbanfarmcoalition
9.   null                                               Error: Traceback (most recent call last):
                                                          File "<string>", line 2, in __temp_875182021__
                                                        TypeError: 'NoneType' object is unsubscriptable
10.  :14kenfm                                           https://instagram.com/14kenfm
```

On error: [✓ keep original] set to blank / store error — ☐ Re-transform up to 10 times until no change

OK Cancel


Custom text transform on column zip

Expression — Language: General Refine Expression Language (GREL)

```
value.replace(/[a-zA-Z]+/,"")
```

No syntax error.

Preview | History | Starred | Help

| row | value | value.replace(/[a-zA-Z]+/,"") |
|---|---|---|
| 1. | 05828 | 05828 |
| 2. | null | Error: replace expects 3 strings, or 1 string, 1 regex, and 1 string |
| 3. | 29682 | 29682 |
| 4. | 64759 | 64759 |
| 5. | 10029 | 10029 |
| 6. | 37204 | 37204 |
| 7 | 10027 | 10027 |

On error: [✓ keep original] set to blank / store error — ☐ Re-transform up to 10 times until no change

OK Cancel

<u>Provenance information from OpenRefine:</u>

The complete series of steps is provided as a JSON file names schetry2_final.json as supplementary project material.

## 3. **Developing a relational schema:**

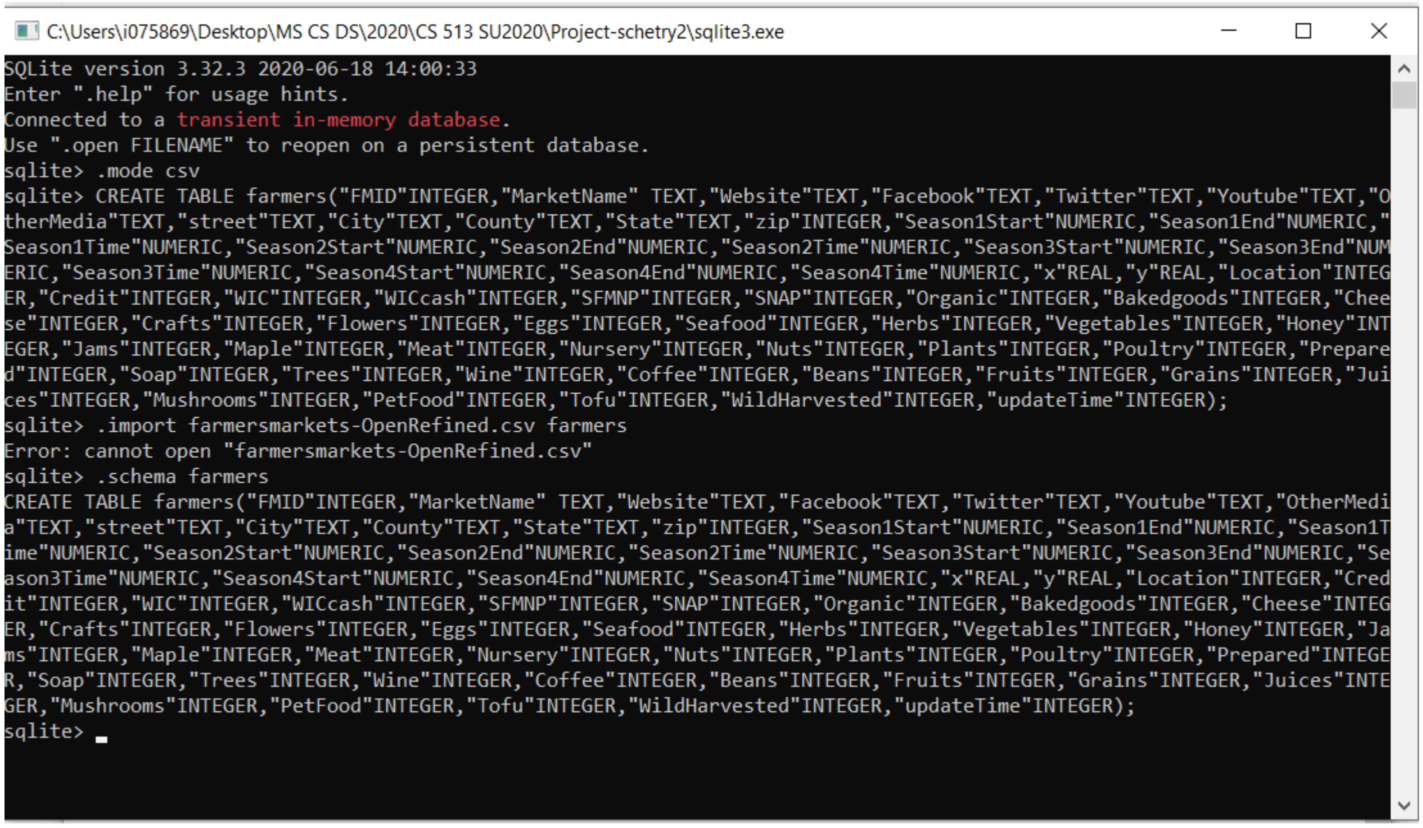
SQLite was used to create schema, load data from cleaned data from OpenRefine. SQL scripts were run to check Integrity Constraints and the violation cases were updated or removed using SQL. Finally, the SQL cleaned data was saved as csv.

Integrity Constraints identified:

a. Check for null or non-unique FMIDs
b. Check for markets which do not have ZIP codes but have X, Y co-ordinates
c. Check for markets which have X,Y co-ordinates but none of ZIP,State,City,County,Street,Social Media
d. Check for markets which do not have X,Y co-ordinates but none of ZIP,State,City,County,Street,Social Media
e. Check for duplicate social media URLs in OtherMedia field. For example Facebook URL available in both Facebook field and OtherMedia field.
f. Check for Season Start Date is older than Season End Date.
g. Check for Latitude and Longitude range values to validate.

Loading OpenRefined data into SQLite:

SQLite CLI was used for creating schema and loading the csv. SQL script was used to create schema.



C:\Users\i075869\Desktop\MS CS DS\2020\CS 513 SU2020\Project-schetry2\sqlite3.exe

```
SQLite version 3.32.3 2020-06-18 14:00:33
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .mode csv
sqlite> CREATE TABLE farmers("FMID"INTEGER,"MarketName" TEXT,"Website"TEXT,"Facebook"TEXT,"Twitter"TEXT,"Youtube"TEXT,"O
therMedia"TEXT,"street"TEXT,"City"TEXT,"County"TEXT,"State"TEXT,"zip"INTEGER,"Season1Start"NUMERIC,"Season1End"NUMERIC,"
Season1Time"NUMERIC,"Season2Start"NUMERIC,"Season2End"NUMERIC,"Season2Time"NUMERIC,"Season3Start"NUMERIC,"Season3End"NUM
ERIC,"Season3Time"NUMERIC,"Season4Start"NUMERIC,"Season4End"NUMERIC,"Season4Time"NUMERIC,"x"REAL,"y"REAL,"Location"INTEG
ER,"Credit"INTEGER,"WIC"INTEGER,"WICcash"INTEGER,"SFMNP"INTEGER,"SNAP"INTEGER,"Organic"INTEGER,"Bakedgoods"INTEGER,"Chee
se"INTEGER,"Crafts"INTEGER,"Flowers"INTEGER,"Eggs"INTEGER,"Seafood"INTEGER,"Herbs"INTEGER,"Vegetables"INTEGER,"Honey"INT
EGER,"Jams"INTEGER,"Maple"INTEGER,"Meat"INTEGER,"Nursery"INTEGER,"Nuts"INTEGER,"Plants"INTEGER,"Poultry"INTEGER,"Prepare
d"INTEGER,"Soap"INTEGER,"Trees"INTEGER,"Wine"INTEGER,"Coffee"INTEGER,"Beans"INTEGER,"Fruits"INTEGER,"Grains"INTEGER,"Jui
ces"INTEGER,"Mushrooms"INTEGER,"PetFood"INTEGER,"Tofu"INTEGER,"WildHarvested"INTEGER,"updateTime"INTEGER);
sqlite> .import farmersmarkets-OpenRefined.csv farmers
Error: cannot open "farmersmarkets-OpenRefined.csv"
sqlite> .schema farmers
CREATE TABLE farmers("FMID"INTEGER,"MarketName" TEXT,"Website"TEXT,"Facebook"TEXT,"Twitter"TEXT,"Youtube"TEXT,"OtherMedi
a"TEXT,"street"TEXT,"City"TEXT,"County"TEXT,"State"TEXT,"zip"INTEGER,"Season1Start"NUMERIC,"Season1End"NUMERIC,"Season1T
ime"NUMERIC,"Season2Start"NUMERIC,"Season2End"NUMERIC,"Season2Time"NUMERIC,"Season3Start"NUMERIC,"Season3End"NUMERIC,"Se
ason3Time"NUMERIC,"Season4Start"NUMERIC,"Season4End"NUMERIC,"Season4Time"NUMERIC,"x"REAL,"y"REAL,"Location"INTEGER,"Cred
it"INTEGER,"WIC"INTEGER,"WICcash"INTEGER,"SFMNP"INTEGER,"SNAP"INTEGER,"Organic"INTEGER,"Bakedgoods"INTEGER,"Cheese"INTEG
ER,"Crafts"INTEGER,"Flowers"INTEGER,"Eggs"INTEGER,"Seafood"INTEGER,"Herbs"INTEGER,"Vegetables"INTEGER,"Honey"INTEGER,"Ja
ms"INTEGER,"Maple"INTEGER,"Meat"INTEGER,"Nursery"INTEGER,"Nuts"INTEGER,"Plants"INTEGER,"Poultry"INTEGER,"Prepared"INTEGE
R,"Soap"INTEGER,"Trees"INTEGER,"Wine"INTEGER,"Coffee"INTEGER,"Beans"INTEGER,"Fruits"INTEGER,"Grains"INTEGER,"Juices"INTE
GER,"Mushrooms"INTEGER,"PetFood"INTEGER,"Tofu"INTEGER,"WildHarvested"INTEGER,"updateTime"INTEGER);
sqlite> _
```

Writing Queries to Check Constraints and Data Cleaning:

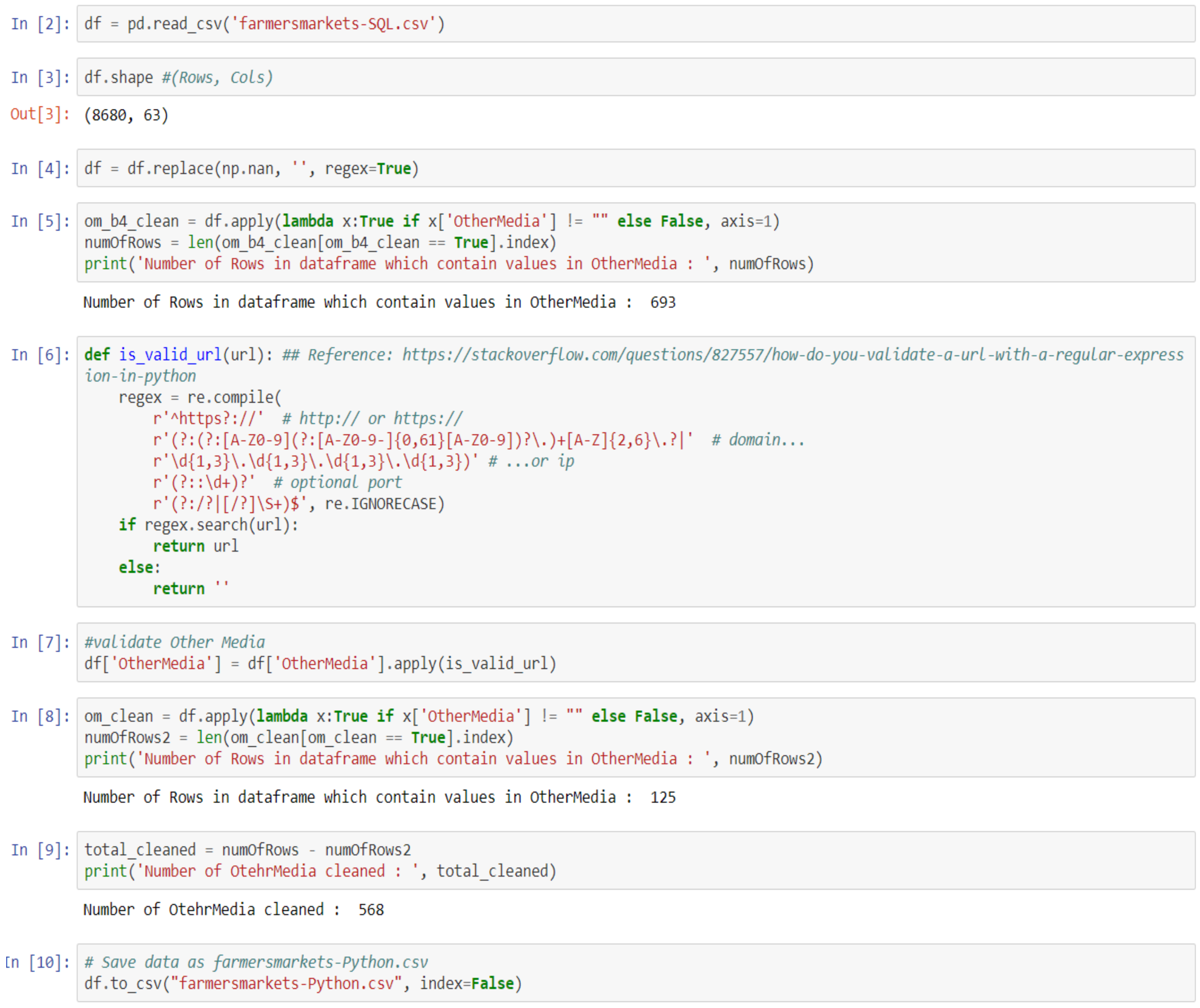
The list of SQL queries to check IC and update data for IC violations are provided as supplementary file.

4. <u>Data Cleaning with Python:</u>

   There were some values in the field OtherMedia not having a valid URL format. These values were removed using Python as shown below.

   The OtherMedia values were missing domain names, which could be either facebook/twitter/Instagram. But without the domain names – the values do not add any information for locating the market. The below Python step was done to identify these non-meaningful URLs and remove them.

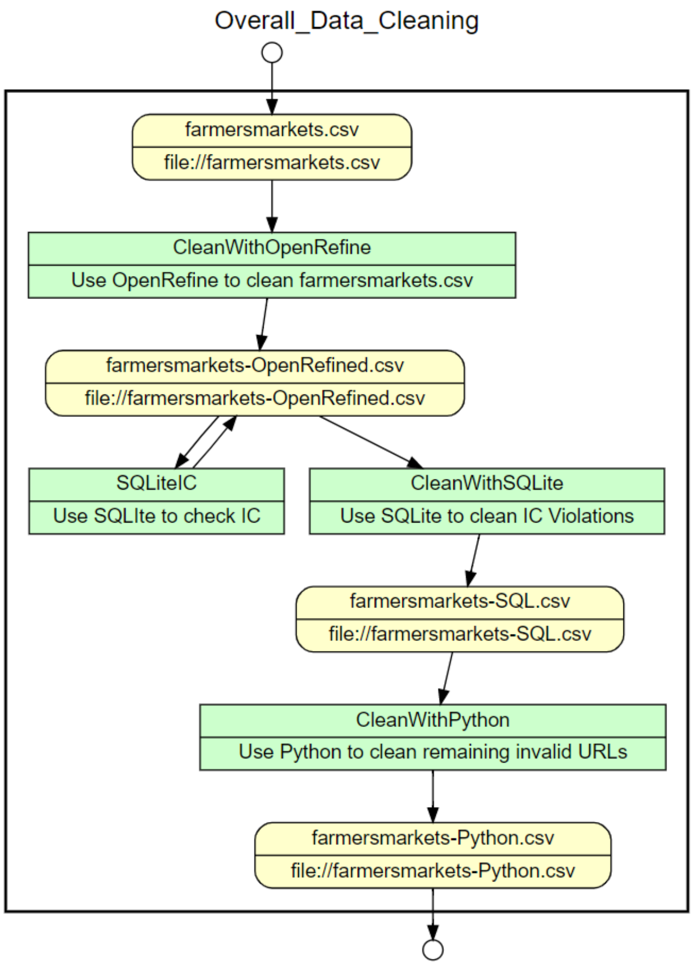
   568 entries in OtherMedia violated correct URL and  were cleaned with this operation.



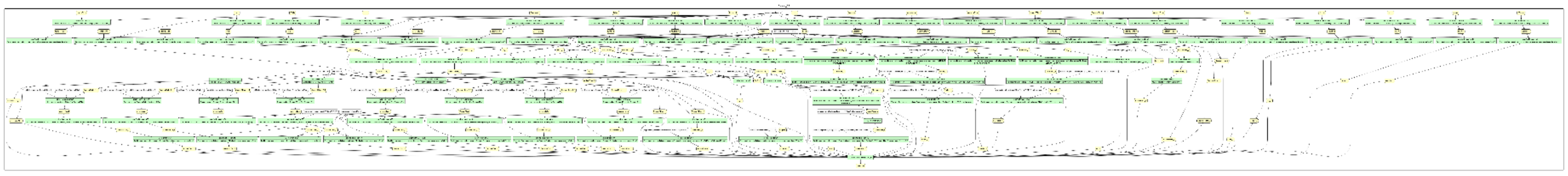
```
In [2]: df = pd.read_csv('farmersmarkets-SQL.csv')

In [3]: df.shape #(Rows, Cols)

Out[3]: (8680, 63)

In [4]: df = df.replace(np.nan, '', regex=True)

In [5]: om_b4_clean = df.apply(lambda x:True if x['OtherMedia'] != "" else False, axis=1)
        numOfRows = len(om_b4_clean[om_b4_clean == True].index)
        print('Number of Rows in dataframe which contain values in OtherMedia : ', numOfRows)

        Number of Rows in dataframe which contain values in OtherMedia :  693

In [6]: def is_valid_url(url): ## Reference: https://stackoverflow.com/questions/827557/how-do-you-validate-a-url-with-a-regular-express
        ion-in-python
            regex = re.compile(
                r'^https?://'  # http:// or https://
                r'(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+[A-Z]{2,6}\.?|'  # domain...
                r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})' # ...or ip
                r'(?::\d+)?'  # optional port
                r'(?:/?|[/?]\S+)$', re.IGNORECASE)
            if regex.search(url):
                return url
            else:
                return ''

In [7]: #validate Other Media
        df['OtherMedia'] = df['OtherMedia'].apply(is_valid_url)

In [8]: om_clean = df.apply(lambda x:True if x['OtherMedia'] != "" else False, axis=1)
        numOfRows2 = len(om_clean[om_clean == True].index)
        print('Number of Rows in dataframe which contain values in OtherMedia : ', numOfRows2)

        Number of Rows in dataframe which contain values in OtherMedia :  125

In [9]: total_cleaned = numOfRows - numOfRows2
        print('Number of OtehrMedia cleaned : ', total_cleaned)

        Number of OtehrMedia cleaned :  568

In [10]: # Save data as farmersmarkets-Python.csv
         df.to_csv("farmersmarkets-Python.csv", index=False)
```

5. Creating a Workflow Model:

YesWorkFlow web interface(try.yesworkflow.org) was used for prepare the overall workflow model. Appropriate inputs and outputs from the aforementioned steps were identified and captured. The YW script and the GZ script are provided as supplementary material.


Overall_Data_Cleaning
farmersmarkets.csv
file://farmersmarkets.csv
CleanWithOpenRefine
Use OpenRefine to clean farmersmarkets.csv
farmersmarkets-OpenRefined.csv
file://farmersmarkets-OpenRefined.csv
SQLiteIC
Use SQLIte to check IC
CleanWithSQLite
Use SQLite to clean IC Violations
farmersmarkets-SQL.csv
file://farmersmarkets-SQL.csv
CleanWithPython
Use Python to clean remaining invalid URLs
farmersmarkets-Python.csv
file://farmersmarkets-Python.csv

OR2YW tool was used to create the YW script. Yesworkflow web interface was used to create the OpenRefine detailed steps. The GZ file and YW script is provided as supplementary materials.



6. Further Analysis/Challenges:

   Missing ZIP Codes could be replaced using Google APIs which can take state, county, city, street as input and return a JSON with ZIP code.

   For entries which no ZIP codes or any location details or social media, they are not good for end user. They can be best for record keeping use.

   Some values in OtherMedia had missing domain names in the URL string. This could be populated via trial and error using either facebook,twitter,youtube or instagram as probable domains.

   Parsing error was encountered when converting OpenRefine JSON steps to a Yesworkflow script using OR2YW tool.

   With the current cleaned data, it is now possible to build a Search and Navigation App, integrated with Map App for directions and route recommendation.