CS513- Theory and Practice of Data Cleaning

Final Project Report

Viney Kharbanda-vkk2

Cleaned Data: LINK

Dirty Data: LINK

Data cleaning: The process of detecting and correcting inaccurate data records is one of the most important steps in any project that makes use of data. This process of cleaning is to be done prior to any actual analysis. The cleaner the data, the higher its value. Thus, the purpose of this document is to outline steps performed to make data more suitable for accurate and meaningful analysis. Before cleaning the data, it was assessed and use cases were developed to determine the type of cleaning and the tools to be used to perform this cleaning. In this project the tools used included OpenRefine, followed by some cleaning via Python script, and then SQLite checked for integrity constraints. Additionally, to document this workflow, the YesWorkflow tool was used.

Initial Assessment of Data

For this project we used Farmers Market directory dataset maintained by Agricultural Marketing Service. The directory is designed to provide the customers with convenient access to information about farmers market listings and included: market locations, directions, operating time, product offering, accepted forms of payment etc.

The initial dataset had 8771 observations and 59 columns. The columns in the dataset provide some structure in grouping. The first group was an Identifier which had FMID and MarketName columns. FMID identifies each farmers market by a unique 7-digit number. MarketName represent name of the Market and is free text. This column has noticeable quality issues, as the same names had different spellings, punctuation and cases. The second group appears to be Web/Social which has 5 columns (Website, Facebook, Twitter, YouTube, OtherMedia) appears to identify markets web and social media presence. These columns are just free text columns and support blank or null values. The third group of columns provides location information which has these columns as (Street, City, County, State, zip, x, y and Location). The X and Y columns represent latitude and longitude. The fourth group of columns provide market availability information such as the time of the year when farmers markets are active over various seasons, via columns such as Season1Date, Season1Time etc. An Inconsistent format

was visible in these columns. Most noticeable is the use of the text “to” in between two dates to signify date range in the same column. The next group of columns provide mode of payment information. These columns are meant to be Boolean Y-Yes and N-No, however, some of the columns have missing data. The next group of columns provide information about what goods are carried. The columns are separated by various item names such as Organic, Cheese, eggs etc. These columns are meant to be Boolean as well, but also have some missing information.

The main quality issues seen from the dataset in above columns were inconsistent format punctuation, spelling and missing data.

Current Use case of data

With the current state of data, it can be useful for the following use cases:

1. Which farmer’s markets do not accept credit card?
2. Identifying which farmers markets in the US participate in acceptance of WIC? Note: WIC is a program for low income women, infants and children
3. Which item is a commodity at the farmers markets (i.e. can be found at almost all the farmers market)?
4. Which item is carried the least at farmers market?

Hypothetical Use cases after cleaning

After applying cleaning methods to be discussed later in this report, cleaned data can be used for the following use cases:

1. Mapped Geolocation of the data identifying the location of farmers market in a certain area based on user’s location.
2. Identify which season is the most popular to find a farmer’s market in the US.
3. Identify which city/state/county has the most farmers markets.
4. Identify most common name of a farmer’s market.

Data Cleaning Methods and Process

The data provided **here** (note data used was latest downloaded data from USDA website), was cleaned in the 3 different stages with 3 different tools. OpenRefine, Python Script and SQLite were used to clean this data. In addition to data cleaning, a workflow model of this process was created in a visual form to illustrate the overall workflow using YesWorkflow tool.

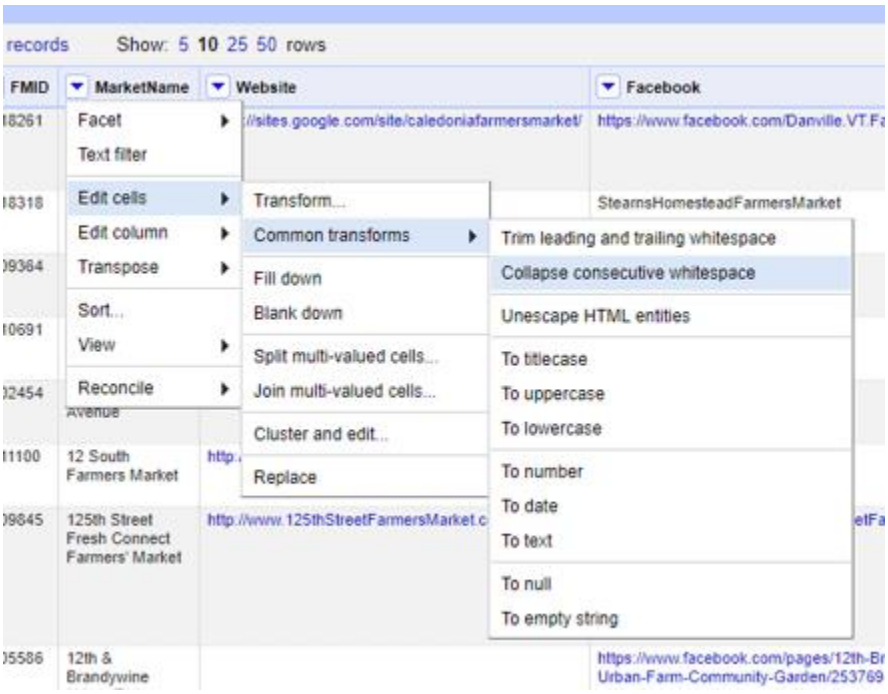
Cleaning using OpenRefine

OpenRefine is an open source tool with a web user interface for data cleanup and transformation to other formats.

Before any cleaning is done, two fundament things must be done. The things are:

1. Trim leading and trailing whitespaces
2. Collapse consecutive whitespaces

Trim leading and trailing whitespaces returns a copy of the string with the leading and trailing whitespace removed. This can be easily done in OpenRefine using the steps shown in the picture below:

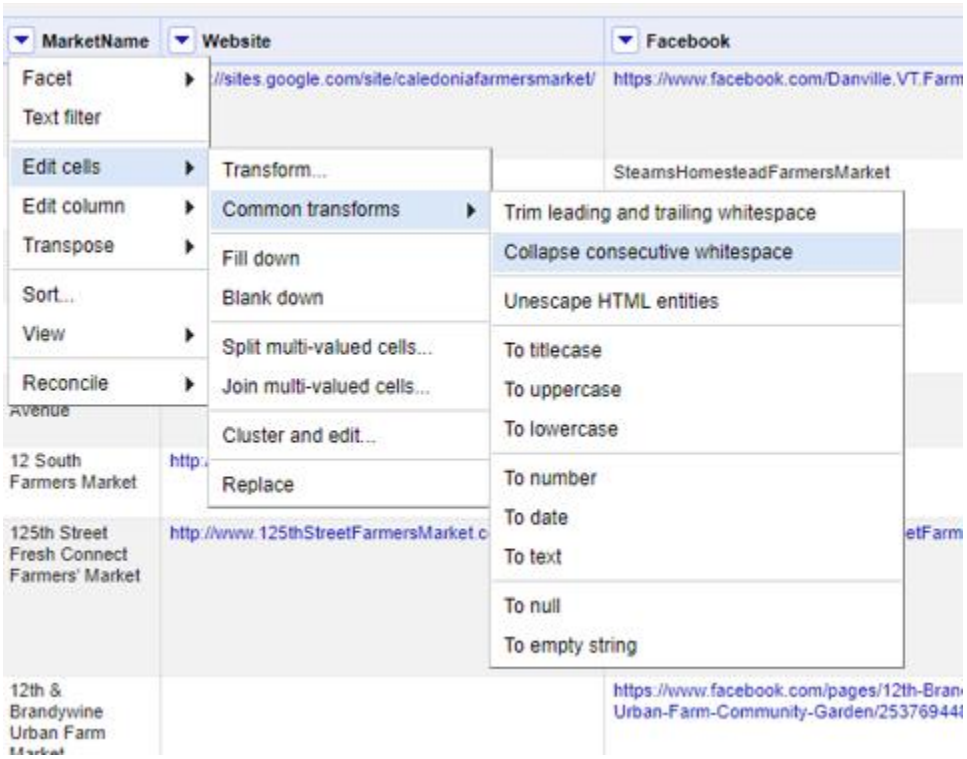


Below are the cell counts in the column which were affected by this cleaning:

Column Name	**Number of Cells affected by trimming leading and trailing whitespace**
MarketName	412
Website	26
Facebook	39
Twitter	10
Youtube	4

OtherMedia	19
Street	323
City	964
Season1Date	106
Season2Date	11
Season3Date	3

Collapse consecutive whitespace removes extra spaces between individual strings within a cell value in the selected column. This can be easily done in OpenRefine using the steps shown in picture below:

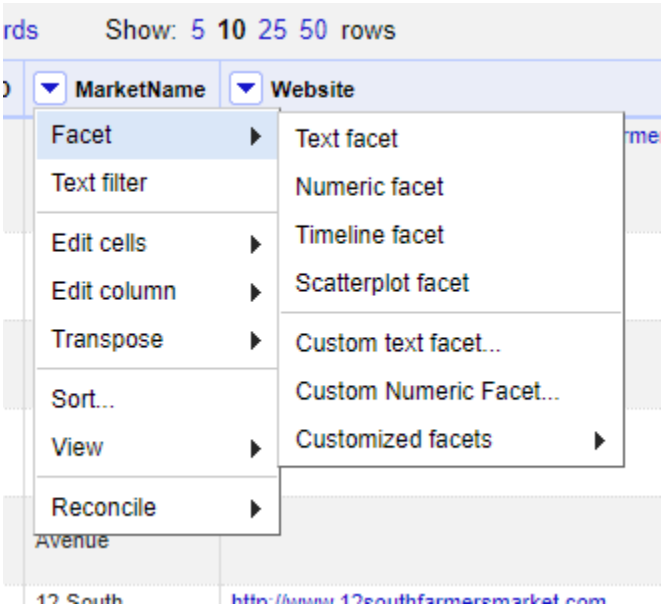


Below are the cell counts in the column which were affected by this cleaning:

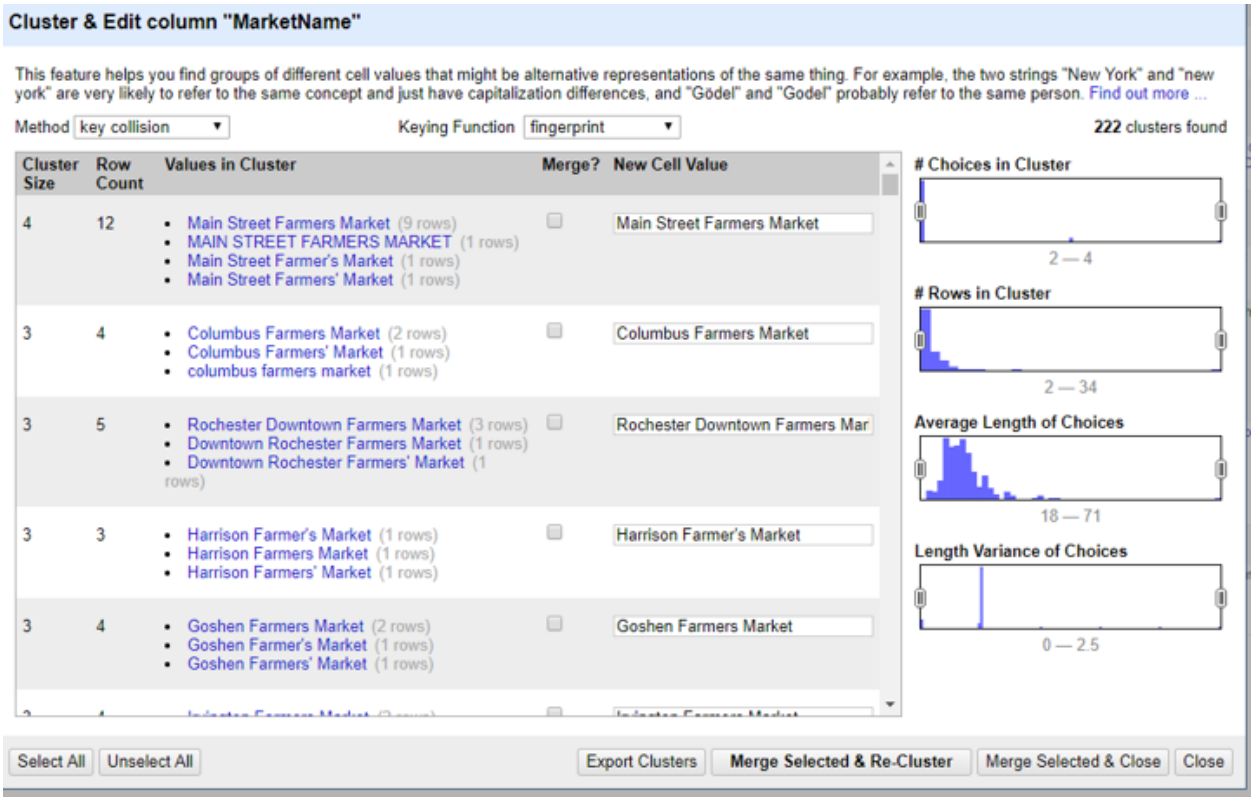
Column Name	**Number of Cells affected by Collapse Consecutive whitespace**
MarketName	46
Facebook	3
Twitter	1
OtherMedia	22
Street	88
City	2

Season1Date	1

Now that the basic cleaning was done, we used “Clustering”, which is one of OpenRefine’s most powerful features. With the support of several types of key collision and nearest neighbor algorithms, the Clustering function helps identify misspellings, inconsistent format in the data. Clustering can be done on any column by selecting “Text facet” after selecting “Facet” from any column name as shown below:



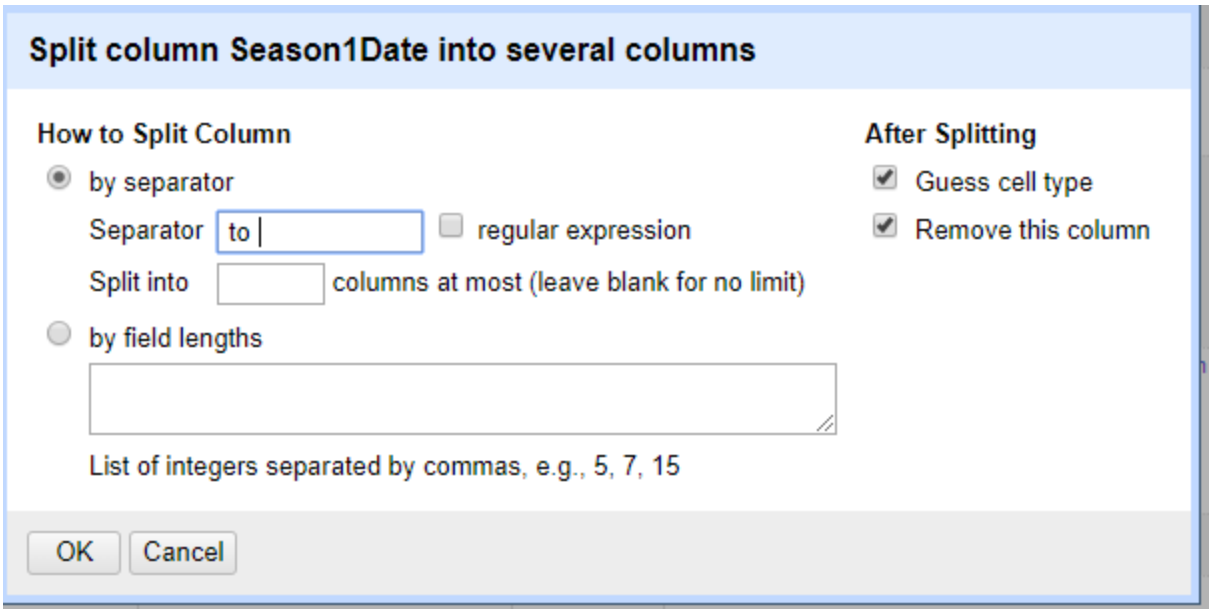
For this data, Key collision method and fingerprint keying function was used. In our data most of the clustering involved case matching and punctuations matching. Below is an illustration of clustering done on MarketName of the data:



Listed below are the column names and the cells affected by clustering:

Column Name	Number of Cells affected by Clustering
MarketName	646
City	594
County	738

After visually inspecting the data we noticed text “to” in Season1Date, Season2Date, Season3Date and Season4Date. Since this signifies the starting and ending period of the Season date when a market is available, it made sense to separate this information into to two different columns. To do this we used the feature of splitting a column by a separator. This was accomplished by selecting “Split into several columns”. In our data Season1Date, Season2Date, Season3Date and Season4Date were separated by using “ to “ as a separator as shown below:



Below are the numbers of cells that were affected by this split:

Original Column Name	**Number of cells split using " to " separator**
Season1Date	5643
Season2Date	457
Season3Date	79
Season4Date	1

After the split Season1Date was split to 2 columns named Season1DateStart and Season1DateEnd, Season2Date was split to 2 columns named Season2DateStart and Season2DateEnd, Season3Date was split to 2 columns named Season3DateStart and Season3DateEnd, and Season4Date was split to 2 columns named Season4DateStart and Season4DateEnd.

Lastly, we converted the entire column x and y to numeric, since they represent longitude and latitude respectively. Below is a summary of the number of cells affected:

Column Name	**Number of cells affected by converting to numeric**
x	8743
y	8743

Latitude/Longitude to zipcode in Python

One of the hypothetical cases we wanted to use this data for was to be able to locate farmers markets in a particular zip code, so to do this we needed to ensure validity of the zip code in the data. So, we will used Python script to provide zip code based data on latitude/longitude that can be used to crosscheck the zip code provided in the data. One way to do this in python is to use Google Maps API to extract zip code from latitude and longitude. However, the use of API requires API key and has associated cost with using the key. More information regarding this can be found in reference [1]. So, we had to explore another alternative way to do this. For this project we used uszipcode database in python to locate zip code based on longitude and latitude information. More information regarding this is in the reference [2]. To provide the zip code we used radius of 30miles for a specific longitude/latitude. In python we searched to see if the zip code in the data was in this 30mile radius. If the zip code was in the 30mile radius we added it to the ZipcodeCalc column to the data, and if it was not, or if the original zip code was blank, we added the closest zip code at its respective longitude/latitude. Since this database is not as complete as Google Maps, there were few longitudes/latitudes where there was no zip code provided. In those cases, we left the zip code in ZipcodeCalc column blank. Python code for this is provided in the file "getzipcodefromlatitude.py"

In summary, column ZipcodeCalc was added to the end of the data with 8086 zip code and 685 were left blank for which either latitude/longitude were missing or there was no information for this in uszipcode database.

Integrity Checks in Database

In the next stage of cleaning we will be using output from python script as described above to use database technologies for data profiling, checking for integrity constraints and repairing. For this project SQLite was used. We considered just one Entity/table for this dataset. The schema for this database is in the file "farmersmarket_schema.sql". Below is the ER diagram for our dataset.

farmersmarket	
FMID	INT PRIMARY KEY
MarketName	TEXT
Website	TEXT
Facebook	TEXT
Twitter	TEXT
Youtube	TEXT
OtherMedia	TEXT
street	TEXT
city	TEXT

County	TEXT
State	TEXT
zip	INTEGER
Season1DateStart	NUMERIC
Season1DateEnd	NUMERIC
Season1Time	NUMERIC
Season2DateStart	NUMERIC
Season2DateEnd	NUMERIC
Season2Time	NUMERIC
Season3DateStart	NUMERIC
Season3DateEnd	NUMERIC
Season3Time	NUMERIC
Season4DateStart	NUMERIC
Season4DateEnd	NUMERIC
Season4Time	NUMERIC
x	REAL
y	REAL
Location	TEXT
Credit	CHAR(1)
WIC	CHAR(1)
WICcash	CHAR(1)
SFMNP	CHAR(1)
SNAP	CHAR(1)
Organic	CHAR(1)
Bakedgoods	CHAR(1)
Cheese	CHAR(1)
Crafts	CHAR(1)
Flowers	CHAR(1)
Eggs	CHAR(1)
Seafood	CHAR(1)
Herbs	CHAR(1)
Vegetables	CHAR(1)
Honey	CHAR(1)
Jams	CHAR(1)
Maple	CHAR(1)
Meat	CHAR(1)
Nursery	CHAR(1)
Nuts	CHAR(1)
Plants	CHAR(1)
Poultry	CHAR(1)
Prepared	CHAR(1)
Soap	CHAR(1)
Trees	CHAR(1)

Wine	CHAR(1)
Coffee	CHAR(1)
Beans	CHAR(1)
Fruits	CHAR(1)
Grains	CHAR(1)
Juices	CHAR(1)
Mushrooms	CHAR(1)
PetFood	CHAR(1)
Tofu	CHAR(1)
WildHarvested	CHAR(1)
updateTime	NUMERIC
ZipcodeCalc	NUMERIC

Below are the integrity constraints which were checked.

1. Check FMID is unique for each observation
2. Check FMID is not NULL
3. Check Longitude is in the range of -180 and 180 and Latitude is in the range of -90 and 90
4. Check if zip column matches ZipcodeCalc column

Below is an evidence of each of integrity constraints checked and repaired if needed.

Evidence of Check of FMID is unique for each observation:

```
sqlite> SELECT COUNT(DISTINCT FMID) as Count
   ...> FROM farmersmarket;
Count
8771
```

Evidence of check that FMID is not NULL:

```
sqlite> SELECT COUNT(1) FROM farmersmarket
   ...> WHERE FMID is NULL;
COUNT(1)
0
```

Evidence of check the Longitude is in the range of -180 and 180 and Latitude is in the range of -90 and 90:

```
sqlite> SELECT COUNT(1)
   ...> FROM   farmersmarket
   ...> WHERE(Cast(y AS FLOAT) < 0
   ...> OR Cast(y AS FLOAT) > 90
   ...> OR Cast(x AS FLOAT) > 180
   ...> OR Cast(x AS FLOAT) <- 180 );
COUNT(1)
0
```

Evidence of check zip column should match the ZipcodeCalc:

```
sqlite> SELECT COUNT(1) FROM farmersmarket
WHERE zip <> ZipcodeCalc;
COUNT(1)
1573
```

From the above Integrity Constraint check we saw that 1573 zip codes did not match zip code calculated in Python from latitude/longitude using the uszipcode database. Some of these 1573 difference were due to the fact that the zipcode was missing in the original database, zipcode was incorrect or there was a minor difference between uszipcode database and actual zip code. As discussed earlier, this was restricted since google maps required money to use their API which is far more accurate and up to date. For further enhancement this is recommended. For this project these 1573 cells were updated from ZipcodeCalc column as shown below:

```
sqlite> UPDATE farmersmarket
   ...> SET zip=ZipcodeCalc
   ...> WHERE zip <> ZipcodeCalc;
sqlite> SELECT COUNT(1) FROM farmersmarket
WHERE zip <> ZipcodeCalc;
COUNT(1)
0
```

After the update above zip and ZipcodeCalc matched as shown above.

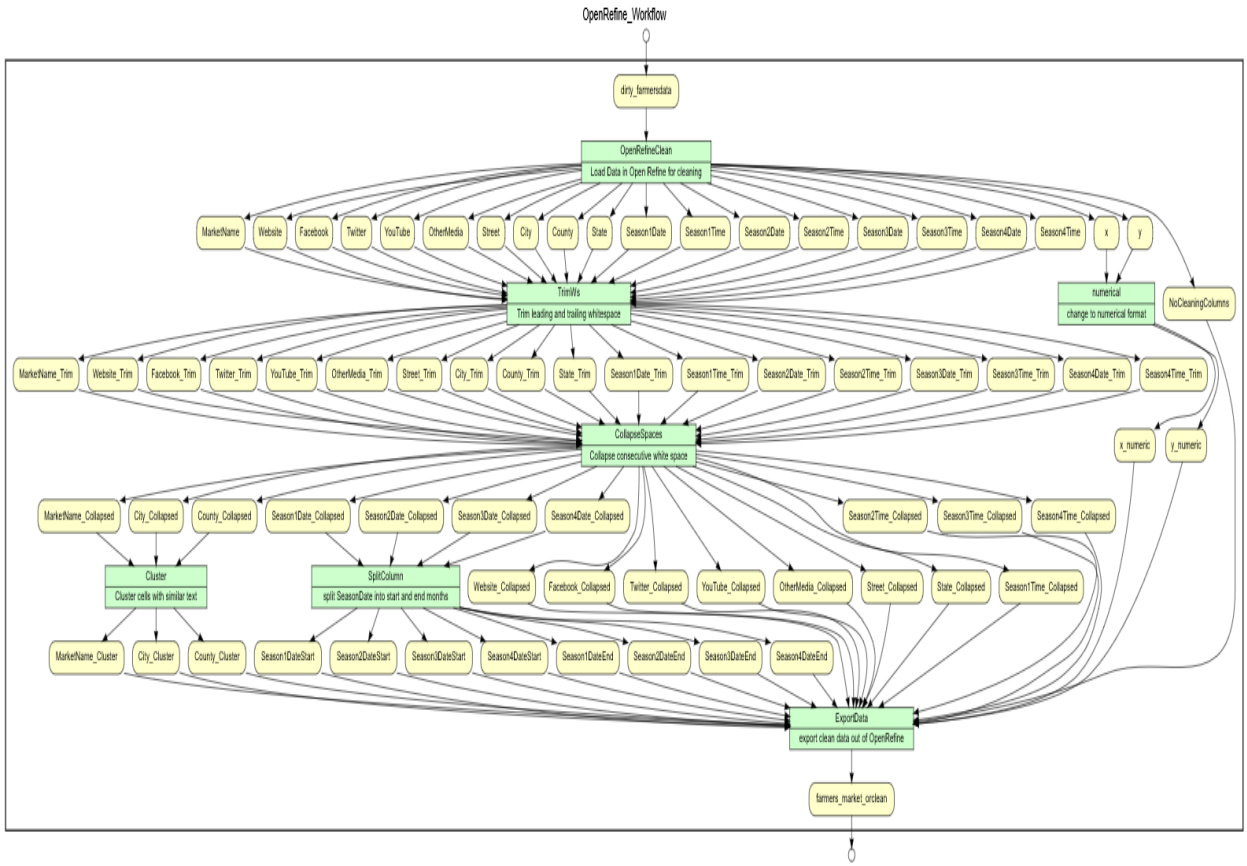
Column Name	Number of Cells modified in SQLite
Zip	1573

All the queries done are listed in the file “sqlqueries.txt”.

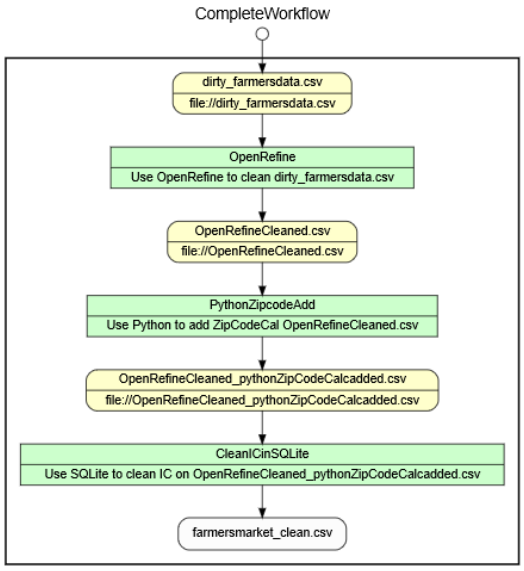
From the database a csv file was generated with the same column name as the original file. This cleaned file is located here.

Workflow Model

After our dataset was cleaned, we saw the procedure was quite involved. To summarize the procedure, we used YesWorkflow to represent the workflow of the data and how it changed during different steps of the process. The picture below shows the YesWorkflow diagram for the cleaning done in OpenRefine. This is optional for individual submission of the project but it was done as an exercise. This was generated manually on workflow website. The file named “yesworkflow_openrefine.yw” has the content used.



Below is the complete workflow showing all the cleaning steps taken with various tools. Similar to openrefine workflow this was generated manually and the content is in the file “CompleteWorkflow.yw”.



Conclusion

The final cleaned data is good enough for the hypothetical use cases identified earlier. However, further cleaning can be done to make the data more useful. Some of the cleaning may also require human intervention. If there was more time, more work with regular expression in OpenRefine or scripting in python, could be done to format certain columns which have the Date, Time in a particular format. Additional work could also be done to social site links provided (e.g. correct the link to direct Facebook/Twitter/Youtube site instead of just username etc). Overall, this project confirms that data cleaning is a daunting yet useful task.

Reference

[1]. Google Maps Geocoding API. Link

[2]. USzipcode. Link