

CS-513 Theory and Practice of Data Cleaning

Yanislav Shterev

The purpose of this document is to summarize the steps that were performed in order to clean and reorder the data from farmersmarkets datasets. After performing all of these operations the use cases included in the initial data assessment shall be easily to be implemented and executed.

1. **Initial data and assessment**- includes all the initial overview of the data and what are some of main issues and useful use cases that can be done based on the data. More information can be found in "1. InitialAssesment" folder "Overview & Initial Assessment.docx".
2. **OpenRefine**- Given all the data present in .csv format here are some of the columns and the transformation that were done on them. OpenRefine has been used mainly to remove bad/special characters, clustering based on similarity function, splitting the data based on some criteria and converting data to specific ISO format for consistency.

The consolidated farmersmarkets.csv was split into 11 sub csv files which contain the same data having some relations between the different tuples. This was done to reduce the data repetitions. Now counties, cities, states etc. have their separate data csv files. On top of this the following several data cleaning operations were performed on some of the columns. The provenance information is saved in "provenance.json" file which is in the "2. OpenRefine" directory.

Transformations over **FMID**:

- Verify FMID contains only unique values:
 - Sort the values in FMID.
 - Blank down and verify that 0 cells were blanked down.

Transformations over **City, Country, State**:

- Remove the special characters - `*% @ # ! \ [] () ? ' *` using `value.replace(/% @ # ! \ [] () ? ' * /, "")`

Custom text transform on column city

Expression Language **General Refine Expression Language (GREL)** ▼

`value.replace(/%#@!\\[\]\(\)\?/, "")` No syntax error.

Preview History Starred Help

row	value	value.replace(/%#@!\\[\]\(\)\?/, "...)
1.	Danville	Danville
2.	Parma	Parma
3.	Six Mile	Six Mile
4.	Lamar	Lamar
5.	New York	New York
6.	Nashville	Nashville
7.	New York	New York

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

OK Cancel

- Trim leading and trailing white spaces and collapse consecutive white spaces.
- Make a facet and perform the cluster operation using the **key-collison** method and **fingerprint** function. Merge the relevant clusters.

Facet / Filter Undo / Redo 12

Refresh Reset All Remove All

city change

4577 choices Sort by: **name** count Cluster

- Atavista 1
- Alton 2
- Altoona 2
- Altus 1
- Alva 1
- Amarillo 1
- Ambridge 1
- Amelia Court House 1
- Amelia Island 1
- American Fork 1
- Americus 1
- Amery 1

County change

1469 choices Sort by: **name** count Cluster

- Anasco 1
- Anchorage 8
- Anderson 10
- Androscoggin 5
- Angelina 1
- Anne Arundel 9
- Anoka 6
- Anson 2
- Antelope 3
- Antrim 5
- Apache 1
- Annanose 1

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

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value	# Choices in Cluster
3	8	<ul style="list-style-type: none"> Woodstock (5 rows) woodstock (2 rows) WOODSTOCK (1 rows) 	<input checked="" type="checkbox"/>	Woodstock	<p>2 — 3</p>
2	43	<ul style="list-style-type: none"> Brooklyn (41 rows) BROOKLYN (2 rows) 	<input checked="" type="checkbox"/>	Brooklyn	<p>2 — 43</p>
2	7	<ul style="list-style-type: none"> Akron (6 rows) AKRON (1 rows) 	<input checked="" type="checkbox"/>	Akron	<p>4 — 16</p>
2	2	<ul style="list-style-type: none"> FOUNTAIN (1 rows) Fountain (1 rows) 	<input checked="" type="checkbox"/>	FOUNTAIN	<p>0 — 5</p>
2	32	<ul style="list-style-type: none"> Los Angeles (30 rows) LOS ANGELES (2 rows) 	<input checked="" type="checkbox"/>	Los Angeles	
2	10	<ul style="list-style-type: none"> St. Louis (8 rows) St Louis (2 rows) 	<input checked="" type="checkbox"/>	St. Louis	
2	14	<ul style="list-style-type: none"> Canton (13 rows) canton (1 rows) 	<input checked="" type="checkbox"/>	Canton	

Select All Unselect All

Export Clusters

Merge Selected & Re-Cluster

Merge Selected & Close

Close

Transformations over **Season1Date**:

- Remove the special characters - *% @ # ! \ [] () ? ' * using `value.replace(/% @ # ! \ [] () ? ' * /, "")`
- Split the column values using * to * as separator, we get 2 columns as a result.

Split column Season1Time into several columns

How to Split Column

☒ by separator

Separator ☐ regular expression

Split into columns at most (leave blank for no limit)

☐ by field lengths

List of integers separated by commas, e.g., 5, 7, 15

After Splitting

☒ Guess cell type

☒ Remove this column

OK

Cancel

- Rename the first column as *Season1DateStart* and second column as *Season1DateEnd*.
- Convert the values in the column to ISO 'd/M/y' format using value.toDate('d/M/y').

Custom text transform on column updateTime

Expression Language General Refine Expression Language (GREL) ▼

`value.toDate('d/M/y H:m:s')` No syntax error.

Preview History Starred Help

row	value	value.toDate('d/M/y H:m:s')
1.	6/20/2017 10:43:57 PM	[date 2018-08-06T10:43:57Z]
2.	6/21/2017 5:15:01 PM	[date 2018-09-06T05:15:01Z]
3.	2013	Error: Unable to parse as date
4.	10/28/2014 9:49:46 AM	[date 2016-04-10T09:49:46Z]
5.	3/1/2012 10:38:22 AM	[date 2012-01-03T10:38:22Z]
6.	5/1/2015 10:40:56 AM	[date 2015-01-05T10:40:56Z]
7.	4/7/2014 4:32:01 PM	[date 2014-07-04T04:32:01Z]

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

OK Cancel

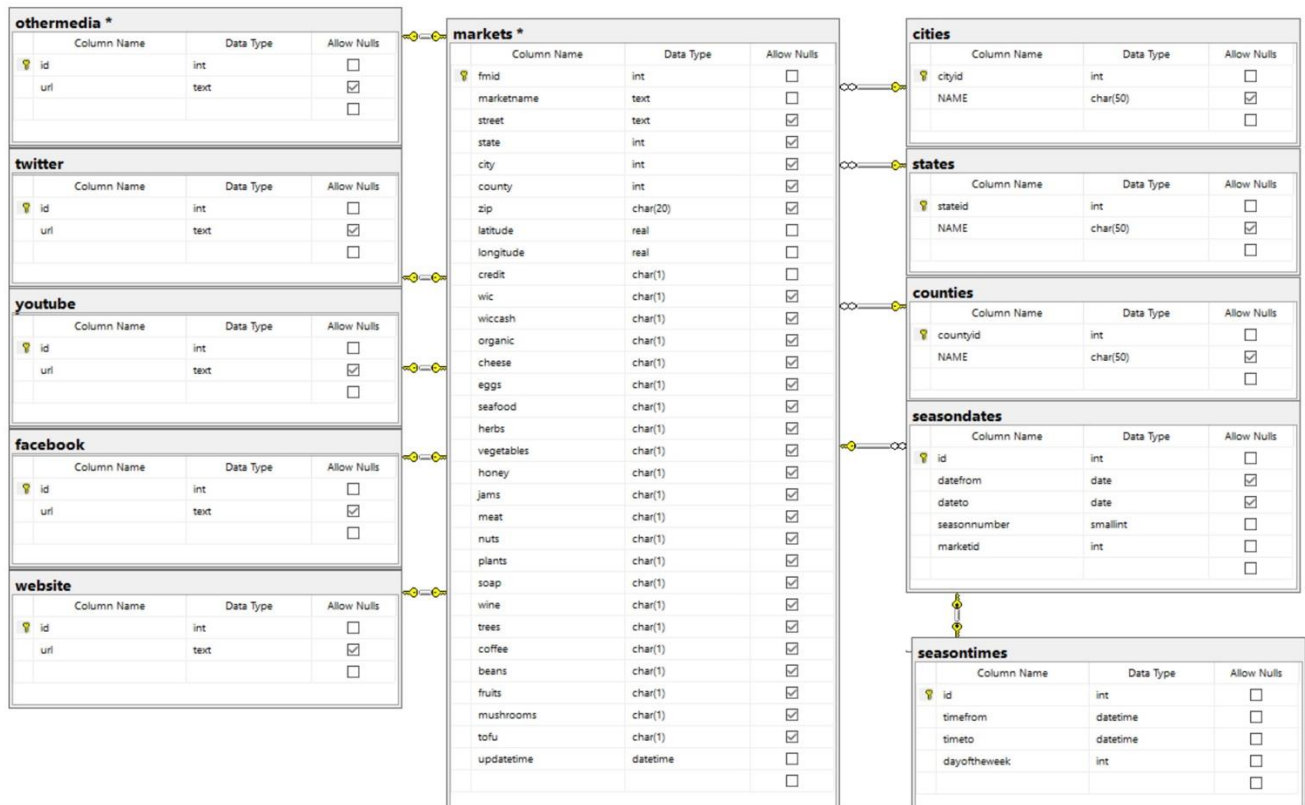
- Repeat 1, 2 and 3 for *Season2Date*, *Season3Date* and *Season4Date*.

Transformations over Season1Time:

- Remove the special characters - *% @ # ! \ [] () ? ' * using `value.replace(/% @ # ! \ [] () ? ' * /, '')`
- Split the column values using *,* as separator. Each of the new columns will represent values for given day of the week.
- Repeat 1 for *Season2Time*, *Season3Time* and *Season4Time*.

Transformations over updateTime:

- Convert the values in the column to ISO 'd/M/y H:m:s' format using value.toDate('d/M/y H:m:s').
3. **SQL**- the following ER diagram represents the data structure after cleaning with OpenRefine and importing the data into SQLite database. In order my database schema to satisfy the forth normal form a lot of manipulations for removing the data redundancy and improving the data reusability were done. Having the schema below it allows adaptability, expandability and high performance once the data becomes part of a real system which interacts with it. The main idea behind the division is to use the concept of dimensions and facts.



All of the scripts related to the SQL part of the project were included in `sqlite-commands.txt` under “3. SQL” directory. The scripts were run in a terminal to create schema, import the data from csv and export it to SQL based insert queries. “farmersmarket_schema.sql” is the file containing the plain schema implementing the relations between the tables including foreign and primary keys and other constraints like field types.

Additionally, SQL queries for integrity constraints validation were created (`IC_sql_queries.sql`):

- Latitude and longitude should be in the interval [0,90], [-180,180]
- There are no inactive markets. There should be at least one datetime per market in seasoondates. And the datefrom and dateto values should not be empty
- Select all the markets that have seafood and vegetables and are opened in November
- There should not be two markets that are in the same location during the same time
- All the markets should have non-empty city, state and county
- No empty names for cities, states, counties

4. YesWorkflow – Represents the workflow of the data and how it changed during the different steps of the process. Firstly, the data root is the `farmersmarkets.csv` which got split into multiple files and cleaning using OpenRefine was performed over each of these files. After the cleaning the diagram represents how each of the files got imported into corresponding SQL table and constraints check was executed over all the tables to verify

whether the data has the needed quality to satisfy the use cases mentioned in the Initial Assessment document.

