CS513 Data Cleaning Group Project

Github Repository: https://github.com/ruoyu-qian/cs513-data-cleaning/

Phase-II

Team32: Ruoyu QIAN (ruoyuq2@illinois.edu), Yasmin YIN (yueyin8@illinois.edu)

Description of Data Cleaning Performed

Data Cleaning Steps

Data Quality Check in Python

We first load the dataset into Python and do some data quality checks on a few columns:

1. We first validate that FMID is the unique identifier of farmer's markets and does not have any missing data:

```
data = pd.read_csv('./data/farmersmarkets-2017-01-10.csv')
```

```
data.shape
```

```
(8665, 59)
```

FMID

```
sum(data.FMID.isnull())
```

```
0
```

```
data.FMID.nunique()
```

```
8665
```

2. We then check all columns related to payment types and confirm that we don't have missing data and only have two unique values in each of those columns, 'Y' and 'N', which is consistent among all payment related columns and we don't need to do any data cleaning.

Payment

```
for i in ['Credit', 'WIC', 'WICcash', 'SFMNP', 'SNAP']:
    print("Number of missing values: {}".format(sum(data[i].isnull())))
    print(data[i].unique())
```

```
Number of missing values: 0
['Y' 'N']
Number of missing values: 0
['Y' 'N']
Number of missing values: 0
['N' 'Y']
Number of missing values: 0
['Y' 'N']
Number of missing values: 0
['N' 'Y']
```

3. We also check all columns related to products and find that all columns have ‘Y’, ‘N’ as possible values, except for the column ‘Organic’, which also has ‘-’. Under the context, we can treat ‘-’ as missing values.

Products

```
for i in [
    'Organic', 'Bakedgoods','Cheese', 'Crafts', 'Flowers', 'Eggs', 'Seafood', 'Herbs',
    'Vegetables', 'Honey', 'Jams', 'Maple', 'Meat', 'Nursery', 'Nuts', 'Plants',
    'Poultry', 'Prepared', 'Soap', 'Trees', 'Wine', 'Coffee', 'Beans', 'Fruits',
    'Grains', 'Juices', 'Mushrooms', 'PetFood', 'Tofu'
]:
    print(data[i].unique())
```

```
['Y' '-' 'N']
['Y' nan 'N']
['Y' 'N' nan]
['Y' 'N' nan]
['Y' nan 'N']
['Y' nan 'N']
['N' nan 'Y']
['Y' nan 'N']
['Y' nan 'N']
['Y' nan 'N']
['Y' nan 'N']
['Y' nan 'N']
['Y' nan 'N']
['N' nan 'Y']
['N' nan 'Y']
['Y' 'N' nan]
['Y' 'N' nan]
['Y' 'N' nan]
['Y' 'N' nan]
['Y' 'N' nan]
['N' nan 'Y']
['Y' 'N' nan]
['Y' 'N' nan]
['Y' nan 'N']
['N' nan 'Y']
['Y' 'N' nan]
['N' nan 'Y']
['Y' 'N' nan]
['N' nan 'Y']
```

4. Lastly, we check column ‘x’ and ‘y’, which represent longitude and latitude. We can see that ‘x’ in the data ranges between [-166.54, -64.7043], and ‘y’ ranges between [17.7099, 64.86275], which are reasonable.

Longitude and Latitude

```
data[['x', 'y']].describe()
```

	x	y
count	8636.000000	8636.000000
mean	-90.972803	39.164435
std	17.491602	5.294078
min	-166.540000	17.709900
25%	-97.296663	36.276390
50%	-86.230908	40.050650
75%	-77.535994	42.449438
max	-64.704300	64.862750

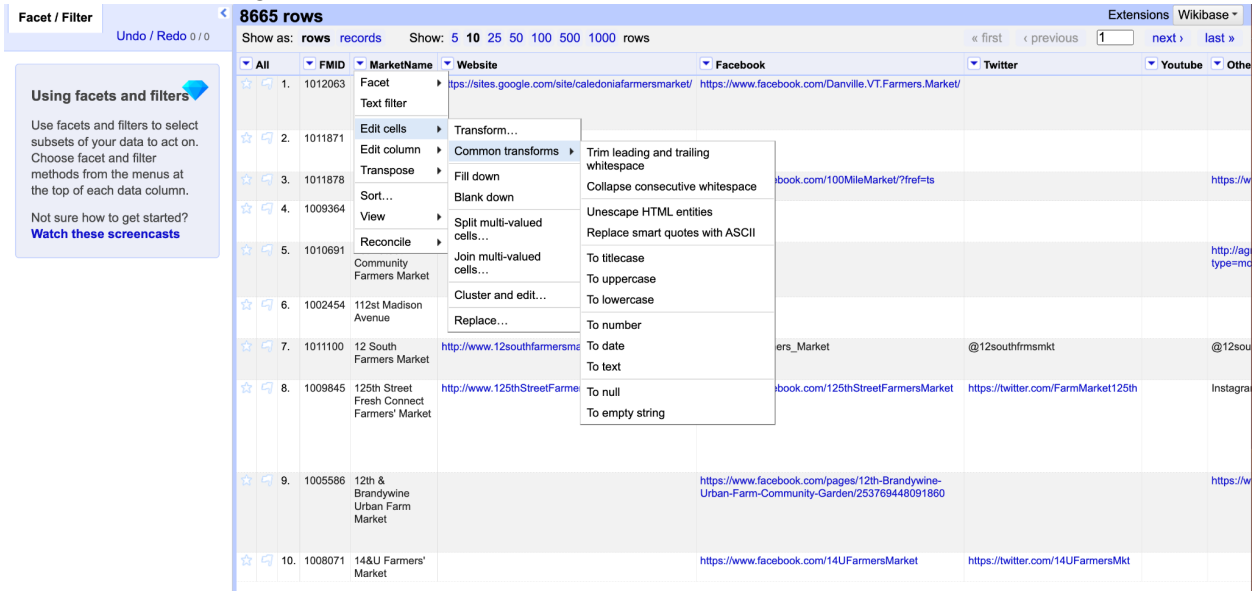
Data Cleaning in OpenRefine

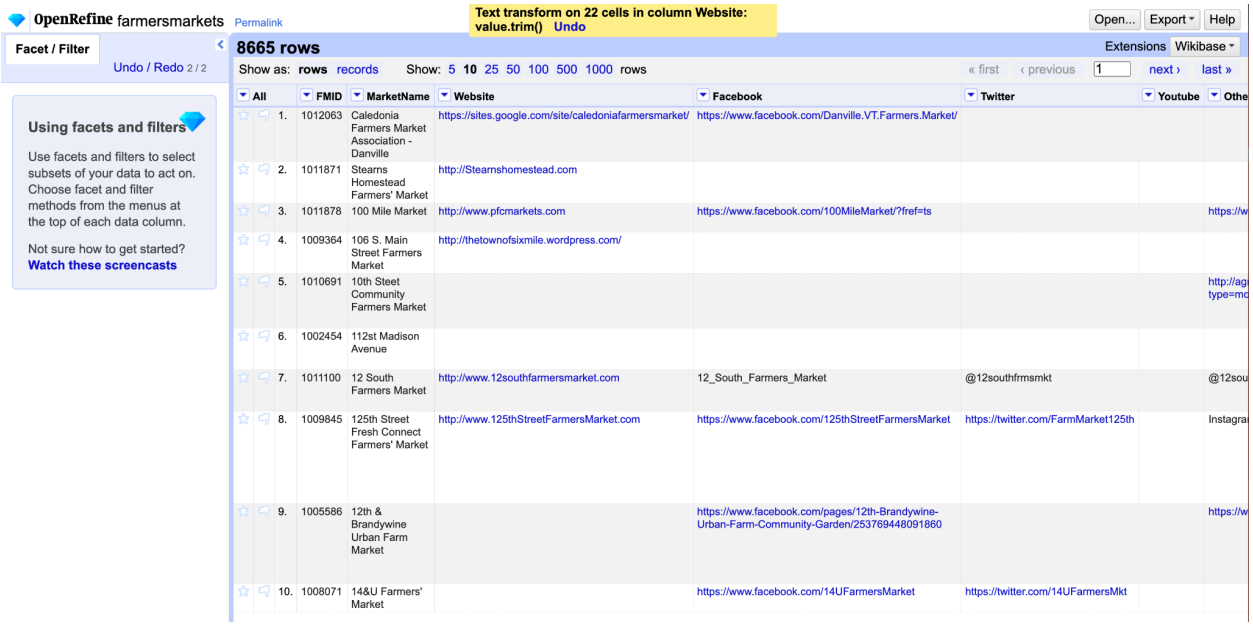
After those sanity checks, we load the original dataset into OpenRefine and take the following data cleaning steps.

1. Trim whitespaces in a few columns

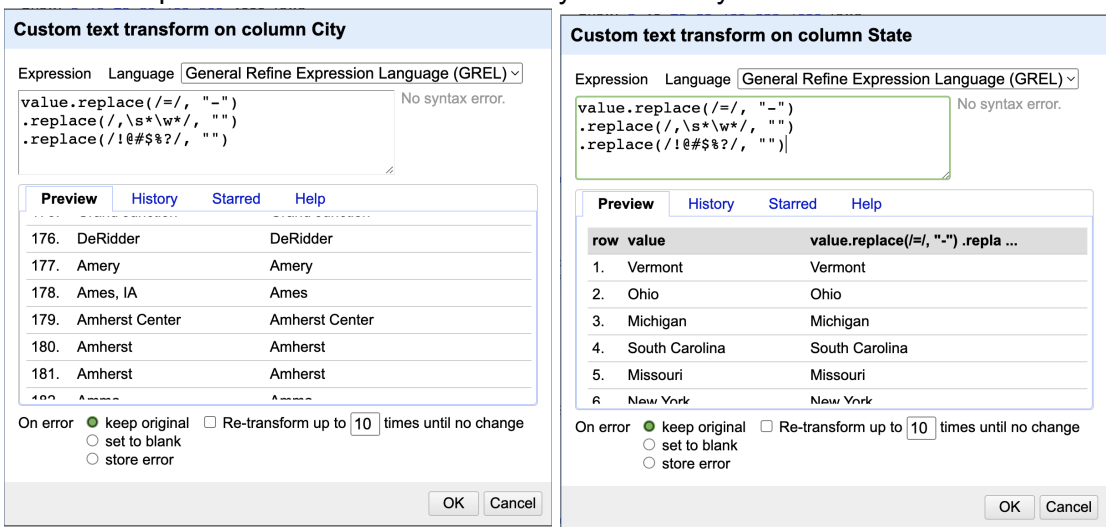
   Trim all leading and trailing whitespaces of columns MarketName, Website, Facebook, Twitter, Youtube, OtherMedia, Street, City, County, State, Zip, updateTime.

   The following screenshot is an example of this step for column MarketName:

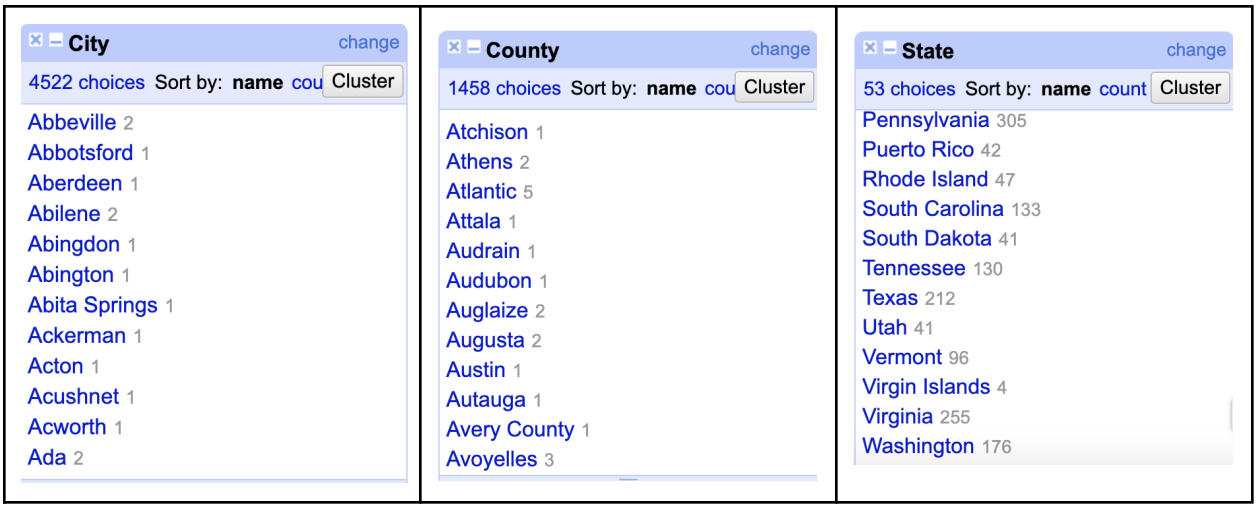




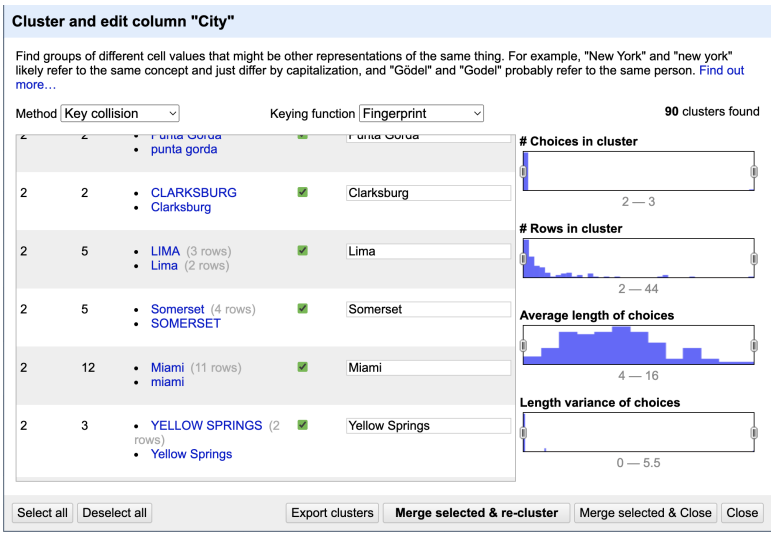
2. Remove special characters in column City and County

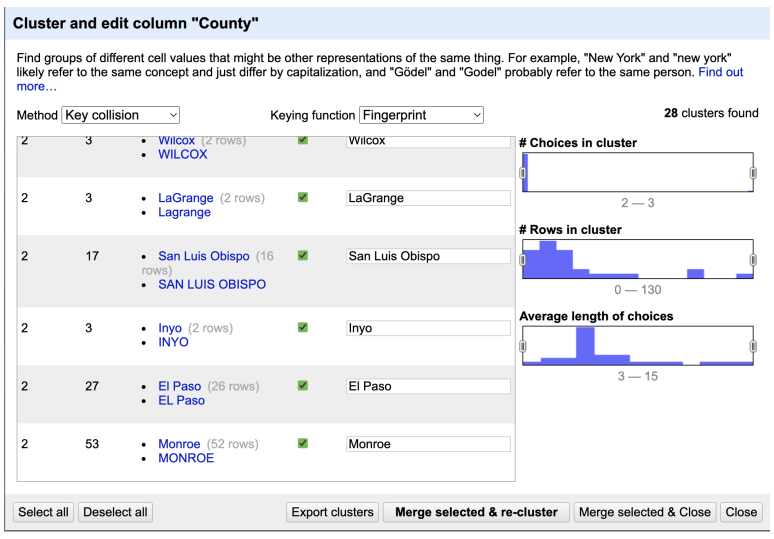


3. Facet and cluster
Make facet and perform clustering with Key collision method and Fingerprint keying function over columns City, County and State, and then merge the relevant ones for City and County.

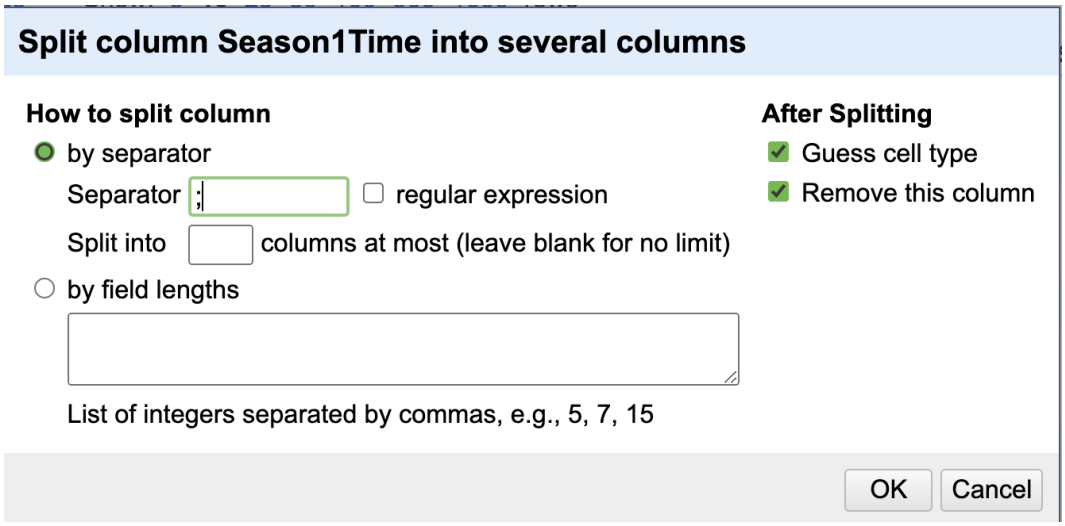


.

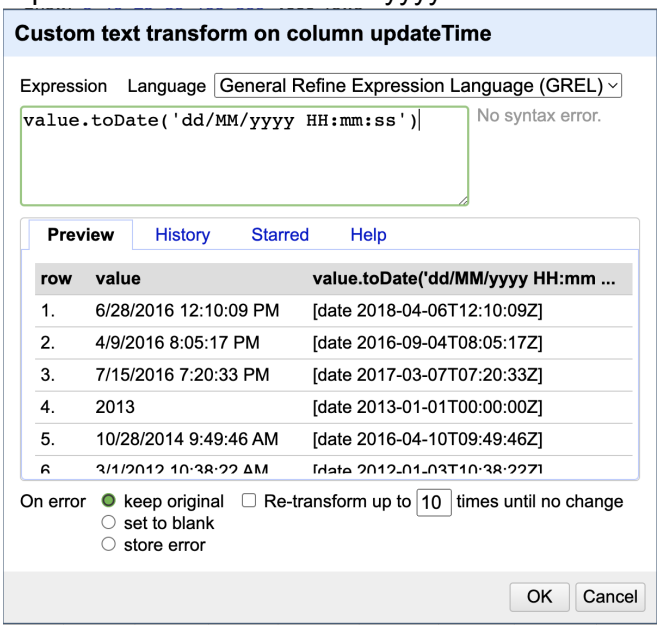




4. Split columns seasonDate and seasonTime based on ‘;’



5. Convert column updateTime to Date ‘dd/mm/yyyy HH:mm:ss’



Data Cleaning in Python

After some basic data cleaning in OpenRefine, we load the intermediate dataset into Python again and finish up the remaining cleaning.

```
data = pd.read_csv('./farmersmarkets.csv', low_memory=False)
```

1. Select only columns related to seasons

```
seasons = data[['FMID', 'Season1Date 1', 'Season2Date 1', 'Season3Date 1', 'Season4Date 1']]
seasons
```

	FMID	Season1Date 1	Season2Date 1	Season3Date 1	Season4Date 1
0	1012063	06/08/2016 to 10/12/2016	NaN	NaN	NaN
1	1011871	06/25/2016 to 10/01/2016	NaN	NaN	NaN
2	1011878	05/04/2016 to 10/12/2016	NaN	NaN	NaN
3	1009364	NaN	NaN	NaN	NaN
4	1010691	04/02/2014 to 11/30/2014	NaN	NaN	NaN
...	...	...	...	...	...
8660	1004767	07/04/2014 to 10/24/2014	NaN	NaN	NaN
8661	1000778	06/07/2016 to 10/04/2016	NaN	NaN	NaN
8662	1012380	05/07/2016 to 10/15/2016	NaN	NaN	NaN
8663	1004686	NaN	NaN	NaN	NaN
8664	1011418	06/01/2015 to 09/30/2015	NaN	NaN	NaN

8665 rows × 5 columns

2. Consolidate all season dates into one column

```
seasons = seasons.melt(
    id_vars=["FMID"],
    var_name="SeasonNum",
    value_name="SeasonDate"
)[['FMID', 'SeasonDate']].dropna().reset_index(drop=True)

seasons
```

	FMID	SeasonDate
0	1012063	06/08/2016 to 10/12/2016
1	1011871	06/25/2016 to 10/01/2016
2	1011878	05/04/2016 to 10/12/2016
3	1010691	04/02/2014 to 11/30/2014
4	1002454	July to November
...	...	...
5896	1012508	01/15/2017 to 01/15/2017
5897	1005991	05/01/2015 to
5898	1010118	04/14/2016 to 04/14/2016
5899	1001875	05/28/2016 to 09/24/2016
5900	1009023	12/10/2016 to 12/17/2016

5901 rows × 2 columns

3. Trim all leading and trailing whitespaces of the column, and split string into a list of start date and end date by ‘ to’

```
seasons['SeasonDate'] = [s.strip() for s in seasons['SeasonDate']]
seasons['SeasonDateList'] = [[d for d in s if d] for s in seasons['SeasonDate'].str.split(' to')]
seasons
```

	FMID	SeasonDate	SeasonDateList
0	1012063	06/08/2016 to 10/12/2016	[06/08/2016, 10/12/2016]
1	1011871	06/25/2016 to 10/01/2016	[06/25/2016, 10/01/2016]
2	1011878	05/04/2016 to 10/12/2016	[05/04/2016, 10/12/2016]
3	1010691	04/02/2014 to 11/30/2014	[04/02/2014, 11/30/2014]
4	1002454	July to November	[July, November]
...	...	...	...
5896	1012508	01/15/2017 to 01/15/2017	[01/15/2017, 01/15/2017]
5897	1005991	05/01/2015 to	[05/01/2015]
5898	1010118	04/14/2016 to 04/14/2016	[04/14/2016, 04/14/2016]
5899	1001875	05/28/2016 to 09/24/2016	[05/28/2016, 09/24/2016]
5900	1009023	12/10/2016 to 12/17/2016	[12/10/2016, 12/17/2016]

5901 rows × 3 columns

4. Extract the first date out as SeasonStart and for SeasonEnd, use the second date if it's available; otherwise, fill in with the first date.

```
seasons['SeasonStart'] = [s[0] for s in seasons['SeasonDateList']]
seasons['SeasonEnd'] = [s[1] if len(s) > 1 else s[0] for s in seasons['SeasonDateList']]
```

```
seasons
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd
0	1012063	06/08/2016 to 10/12/2016	[06/08/2016, 10/12/2016]	06/08/2016	10/12/2016
1	1011871	06/25/2016 to 10/01/2016	[06/25/2016, 10/01/2016]	06/25/2016	10/01/2016
2	1011878	05/04/2016 to 10/12/2016	[05/04/2016, 10/12/2016]	05/04/2016	10/12/2016
3	1010691	04/02/2014 to 11/30/2014	[04/02/2014, 11/30/2014]	04/02/2014	11/30/2014
4	1002454	July to November	[July, November]	July	November
...	...	...	...	...	...
5896	1012508	01/15/2017 to 01/15/2017	[01/15/2017, 01/15/2017]	01/15/2017	01/15/2017
5897	1005991	05/01/2015 to	[05/01/2015]	05/01/2015	05/01/2015
5898	1010118	04/14/2016 to 04/14/2016	[04/14/2016, 04/14/2016]	04/14/2016	04/14/2016
5899	1001875	05/28/2016 to 09/24/2016	[05/28/2016, 09/24/2016]	05/28/2016	09/24/2016
5900	1009023	12/10/2016 to 12/17/2016	[12/10/2016, 12/17/2016]	12/10/2016	12/17/2016

5901 rows × 5 columns

5. Convert SeasonEnd to datetime format.

```
seasons['MostRecentOpeningDate'] = pd.to_datetime(seasons['SeasonEnd'], errors = 'coerce')
seasons
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
1397	1000003	05/31/2014 to 10/31/2014	[05/31/2014, 10/31/2014]	05/31/2014	10/31/2014	2014-10-31
5642	1000008	05/21/2016 to 10/29/2016	[05/21/2016, 10/29/2016]	05/21/2016	10/29/2016	2016-10-29
3123	1000008	05/23/2015 to 11/01/2015	[05/23/2015, 11/01/2015]	05/23/2015	11/01/2015	2015-11-01
5865	1000008	11/05/2016 to 04/29/2017	[11/05/2016, 04/29/2017]	11/05/2016	04/29/2017	2017-04-29
4390	1000009	06/14/2014 to 10/11/2014	[06/14/2014, 10/11/2014]	06/14/2014	10/11/2014	2014-10-11
...	...	...	...	...	...	...
2266	1012844	05/06/2017 to 10/28/2017	[05/06/2017, 10/28/2017]	05/06/2017	10/28/2017	2017-10-28
3356	1016768	05/07/2016 to 10/01/2016	[05/07/2016, 10/01/2016]	05/07/2016	10/01/2016	2016-10-01
2539	1016770	10/02/2016 to 02/04/2017	[10/02/2016, 02/04/2017]	10/02/2016	02/04/2017	2017-02-04
1577	2000005	01/01/2014 to 12/31/2014	[01/01/2014, 12/31/2014]	01/01/2014	12/31/2014	2014-12-31
3545	2000021	07/01/2014 to 07/31/2014	[07/01/2014, 07/31/2014]	07/01/2014	07/31/2014	2014-07-31

5901 rows × 6 columns

6. Group by FMID and find the maximum value from MostRecentOpeningDate for each farmer's market

```
last_season = seasons.groupby('FMID')['MostRecentOpeningDate'].max().reset_index()
last_season
```

	FMID	MostRecentOpeningDate
0	1000003	2014-10-31
1	1000008	2017-04-29
2	1000009	2014-10-11
3	1000010	NaT
4	1000011	NaT
...	...	...
5387	1012844	2017-10-28
5388	1016768	2016-10-01
5389	1016770	2017-02-04
5390	2000005	2014-12-31
5391	2000021	2014-07-31

5392 rows × 2 columns

```
last_season.to_csv('./data/seasons.csv', index=False)
```

Data Cleaning Rationale

Our use case is as follows:

U1: build a recommendation system that given a buyer's preferred payment type, the list of the items they need to buy, the county they live in, and their gps location (i.e., longitude and latitude), returns the list of the farmer's markets in the same county that satisfy all the needs, along with their most recent opening date, rank ordered by the distance between user and the farmer.

Data Quality Check in Python

1. Checking the data quality of the column FMID helps us confirm that it can be used as the unique identifier of farmer's market, which will be used as the key of the group by function when we do the data cleaning and processing to get the **most recent opening date** of each market.

2. Checking the data quality of all the columns related to payment types will help when we build the recommendation system to test if the user's **preferred payment method** is accepted at a given farmer's market. It also informs us that the data is already clean and no further cleaning is needed.

3. Checking the data quality of all the columns related to products will help when we build the recommendation system to test if **all the items the user needs to buy** are available at a given farmer's market. Similarly, it also informs us that the data is clean enough and ready for use.

4. Checking the data quality and value range of x and y columns are also necessary because we want to be confident that we can use those columns to **get the distance between the user and each eligible farmer's market**.

Data Cleaning in OpenRefine

1. Cleaning up the County column, including trimming whitespaces, removing special characters and clustering, is definitely necessary because we need to use that column to filter farmer's markets down to those in the same county as where the user lives.

2. Cleaning up all other address related columns is also necessary because we want to provide users with accurate addresses of farmer's markets as much as we can.

3. Splitting columns seasonDate and seasonTime are also necessary because it will be helpful when we work on getting the **most recent opening date** of each market.

4. It's not required to clean up all other columns for U1, but still useful because we want to provide users with high quality information with the recommendation system – even though Website, Facebook, Twitter, Youtube, OtherMedia, and updateTime are probably nice-to-have information, but cleaning those columns won't hurt. This includes –
    a. Trimming whitespaces of those columns.
    b. Converting column updateTime to Date.

Data Cleaning in Python

The main objective of the further data cleaning in Python is to get the **most recent opening date** of each market. We have a few issues we need to deal with:

1. There are 4 columns including up to 4 seasons when each farmer's market was open, but not every farmer's market has information in all 4 columns, and they may not be ordered by time either. Therefore, we need to consolidate them into one column and later on do group by with maximum function to get the **most recent opening date** of each market.

2. We have different formats when it comes to dates, so we need to make the format consistent across records. The following are two examples that showcase how they got changed from original format to the final format:

```
seasons.loc[seasons['FMID'] == 1000788, ]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
5135	1000788	July 9, 2012 to October 29, 2012	[July 9, 2012, October 29, 2012]	July 9, 2012	October 29, 2012	2012-10-29

```
seasons.loc[seasons['FMID'] == 1001139, ]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
2196	1001139	April to Sept 24, 2011	[April, Sept 24, 2011]	April	Sept 24, 2011	2011-09-24

3. In some cases, we only have the start date of the season but miss the end date. We fill in the most recent opening date with the start date in this case, because that's the best data we have:

```
seasons.loc[seasons['FMID'] == 1000961, ]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
5114	1000961	05/01/2015 to	[05/01/2015]	05/01/2015	05/01/2015	2015-05-01

4. For some markets, we only have month information regarding seasons. Missing year information makes it impossible to guess the dates even roughly. Therefore, we fill in NaT as the most recent opening date.

```
seasons.loc[seasons['FMID'] == 1000165, ]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
4452	1000165	March to November	[March, November]	March	November	NaT

Document data quality changes

Summary table of changes

Column Name	Change	Amount of Cells changed
Market Name	value.trim()	387
Website	value.trim()	22
Facebook	value.trim()	31
Twitter	value.trim()	10
Youtube	value.trim()	3
OtherMedia	value.trim()	14
Street	value.trim()	286
City	value.trim()	805
City	value.replace()	52
County	value.replace()	0
State	value.replace()	1
updateTime	value.toDate()	8352
Season1Date	Split by ‘;’	88
Season2Date Season3Date Season4Date	Split by ‘;’	0
Season1Time	Split by ‘;’	5525
Season2Time	Split by ‘;’	414
Season3Time	Split by ‘;’	75
Season4Time	Split by ‘;’	7
City	Cluster	584
County	Cluster	736
State	Cluster	0

MostRecentOpeningDate	Newly created column with data cleaning and manipulation using Python	5392

Data Quality Improved

Demonstrate that data quality has been improved, e.g., by devising IC-violation reports (answers to denial constraints) and showing the difference between number of IC violations reported before and after cleaning. (10 points)

OpenRefine Cleaning

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

Text transform on 22 cells in column Website: value.trim()

Text transform on 31 cells in column Facebook: value.trim()

Text transform on 10 cells in column Twitter: value.trim()

Text transform on 0 cells in column Twitter: value.trim()

Text transform on 3 cells in column Youtube: value.trim()

Text transform on 14 cells in column OtherMedia: value.trim()

Text transform on 286 cells in column street: value.trim()

Text transform on 805 cells in column city: value.trim()

Text transform on 8,352 cells in column updateTime: grel:value.toDate('dd/MM/yyyy HH:mm:ss')

Split 5525 cell(s) in column Season1Time into several columns by separator

Split 7 cell(s) in column Season4Time into several columns by separator

Split 75 cell(s) in column Season3Time into several columns by separator

Split 414 cell(s) in column Season2Time into several columns by separator

Text transform on 52 cells in column City: grel:value.replace(/=/, "-") .replace(/,\s*\w*/, "") .replace(/!@#$%?/, "")

Text transform on 1 cells in column County: grel:value.replace(/=/, "-") .replace(/,\s*\w*/, "") .replace(/!@#$%?/, "")

Season Dates

1. When splitting SeasonDate into start date and end date, we use IC violation queries to test –
    a. whether the results have more than three items in the list
    b. whether we have only 1 item in the list that look suspicious
    c. whether the results have ‘ ’ (string with only whitespace) in the list

Approach #1

```
seasons['SeasonDateList'] = seasons['SeasonDate'].str.split(' to ')
seasons
```

	FMID	SeasonDate	SeasonDateList
0	1012063	06/08/2016 to 10/12/2016	[06/08/2016, 10/12/2016]
1	1011871	06/25/2016 to 10/01/2016	[06/25/2016, 10/01/2016]
2	1011878	05/04/2016 to 10/12/2016	[05/04/2016, 10/12/2016]
3	1010691	04/02/2014 to 11/30/2014	[04/02/2014, 11/30/2014]
4	1002454	July to November	[July, November]
...	...	...	...
5896	1012508	01/15/2017 to 01/15/2017	[01/15/2017, 01/15/2017]
5897	1005991	05/01/2015 to	[05/01/2015,]
5898	1010118	04/14/2016 to 04/14/2016	[04/14/2016, 04/14/2016]
5899	1001875	05/28/2016 to 09/24/2016	[05/28/2016, 09/24/2016]
5900	1009023	12/10/2016 to 12/17/2016	[12/10/2016, 12/17/2016]

5901 rows × 3 columns

```
seasons.loc[[len(d) > 2 for d in seasons.SeasonDateList],]
```

FMID	SeasonDate	SeasonDateList

```
seasons.loc[[len(d) < 2 for d in seasons.SeasonDateList],]
```

	FMID	SeasonDate	SeasonDateList
58	1004762	10/25/2014 to	[10/25/2014 to]
66	1011322	05/19/2016	[05/19/2016]
79	1004846	05/21/2013 to	[05/21/2013 to]
90	1011910	04/18/2016 to	[04/18/2016 to]
250	1011092	06/07/2016 to	[06/07/2016 to]
...	...	...	...
5286	1005677	07/28/2016 to	[07/28/2016 to]
5303	1010125	12/01/2014 to	[12/01/2014 to]
5391	1011180	06/07/2015	[06/07/2015]
5416	1001403	10/03/2015	[10/03/2015]
5501	1006151	06/07/2013	[06/07/2013]

117 rows × 3 columns

```
seasons.loc[[' ' in d for d in seasons.SeasonDateList],]
```

FMID	SeasonDate	SeasonDateList

Results from approach # 1 have some season dates that still contain ‘to’.

Approach #2

```
seasons['SeasonDateList'] = seasons['SeasonDate'].str.split('to')
seasons
```

	FMID	SeasonDate	SeasonDateList
0	1012063	06/08/2016 to 10/12/2016	[06/08/2016 , 10/12/2016]
1	1011871	06/25/2016 to 10/01/2016	[06/25/2016 , 10/01/2016]
2	1011878	05/04/2016 to 10/12/2016	[05/04/2016 , 10/12/2016]
3	1010691	04/02/2014 to 11/30/2014	[04/02/2014 , 11/30/2014]
4	1002454	July to November	[July , November]
...	...	...	...
5896	1012508	01/15/2017 to 01/15/2017	[01/15/2017 , 01/15/2017]
5897	1005991	05/01/2015 to	[05/01/2015 ,]
5898	1010118	04/14/2016 to 04/14/2016	[04/14/2016 , 04/14/2016]
5899	1001875	05/28/2016 to 09/24/2016	[05/28/2016 , 09/24/2016]
5900	1009023	12/10/2016 to 12/17/2016	[12/10/2016 , 12/17/2016]

5901 rows × 3 columns

```
seasons.loc[[len(d) > 2 for d in seasons.SeasonDateList],]
```

	FMID	SeasonDate	SeasonDateList
43	1001961	May to October	[May , Oc, ber]
53	1004598	May to October	[May , Oc, ber]
60	1003469	June 19, 2012 to October 21, 2012	[June 19, 2012 , Oc, ber 21, 2012]
102	1002836	April to October	[April , Oc, ber]
115	1001100	June to October	[June , Oc, ber]
...	...	...	...
5380	1003731	April to October	[April , Oc, ber]
5618	1001431	October to December	[Oc, ber , December]
5817	1002222	October to November	[Oc, ber , November]
5837	1002854	September to October	[September , Oc, ber]
5849	1004979	September to October	[September , Oc, ber]

386 rows × 3 columns

```
seasons.loc[[len(d) < 2 for d in seasons.SeasonDateList],]
```

	FMID	SeasonDate	SeasonDateList
66	1011322	05/19/2016	[05/19/2016]
548	1012171	06/25/2016	[06/25/2016]
807	1002353	05/23/2014	[05/23/2014]
1285	1011887	04/17/2016	[04/17/2016]
1552	1011719	12/05/2015	[12/05/2015]
...	...	...	...
5228	1003961	07/13/2013	[07/13/2013]
5250	1011897	01/30/2016	[01/30/2016]
5391	1011180	06/07/2015	[06/07/2015]
5416	1001403	10/03/2015	[10/03/2015]
5501	1006151	06/07/2013	[06/07/2013]

29 rows × 3 columns

```
seasons.loc[[' ' in d for d in seasons.SeasonDateList],]
```

	FMID	SeasonDate	SeasonDateList
5400	1008251	06/03/2014 to	[06/03/2014 ,]
5465	1000158	12/05/2015 to	[12/05/2015 ,]
5512	1010414	03/01/2015 to	[03/01/2015 ,]
5516	1010802	11/17/2015 to	[11/17/2015 ,]
5537	1009921	10/10/2014 to	[10/10/2014 ,]
...	...	...	...
5745	1012351	09/16/2016 to	[09/16/2016 ,]
5775	1006826	11/01/2012 to	[11/01/2012 ,]
5876	1011004	09/24/2016 to	[09/24/2016 ,]
5887	1008842	11/27/2016 to	[11/27/2016 ,]
5897	1005991	05/01/2015 to	[05/01/2015 ,]

17 rows × 3 columns

Results from approach # 2 have string with only whitespace in the list and also list with more than 3 items.

Approach #3

```
seasons['SeasonDateList'] = [[d for d in s if d] for s in seasons['SeasonDate'].str.split(' to')]
seasons
```

	FMID	SeasonDate	SeasonDateList
0	1012063	06/08/2016 to 10/12/2016	[06/08/2016, 10/12/2016]
1	1011871	06/25/2016 to 10/01/2016	[06/25/2016, 10/01/2016]
2	1011878	05/04/2016 to 10/12/2016	[05/04/2016, 10/12/2016]
3	1010691	04/02/2014 to 11/30/2014	[04/02/2014, 11/30/2014]
4	1002454	July to November	[July, November]
...	...	...	...
5896	1012508	01/15/2017 to 01/15/2017	[01/15/2017, 01/15/2017]
5897	1005991	05/01/2015 to	[05/01/2015,]
5898	1010118	04/14/2016 to 04/14/2016	[04/14/2016, 04/14/2016]
5899	1001875	05/28/2016 to 09/24/2016	[05/28/2016, 09/24/2016]
5900	1009023	12/10/2016 to 12/17/2016	[12/10/2016, 12/17/2016]

5901 rows × 3 columns

```
seasons.loc[[len(d) > 2 for d in seasons.SeasonDateList],]
```

FMID	SeasonDate	SeasonDateList

```
seasons.loc[[len(d) < 2 for d in seasons.SeasonDateList],]
```

	FMID	SeasonDate	SeasonDateList
58	1004762	10/25/2014 to	[10/25/2014]
66	1011322	05/19/2016	[05/19/2016]
79	1004846	05/21/2013 to	[05/21/2013]
90	1011910	04/18/2016 to	[04/18/2016]
250	1011092	06/07/2016 to	[06/07/2016]
...	...	...	...
5286	1005677	07/28/2016 to	[07/28/2016]
5303	1010125	12/01/2014 to	[12/01/2014]
5391	1011180	06/07/2015	[06/07/2015]
5416	1001403	10/03/2015	[10/03/2015]
5501	1006151	06/07/2013	[06/07/2013]

117 rows × 3 columns

```
seasons.loc[[' ' in d for d in seasons.SeasonDateList],]
```

	FMID	SeasonDate	SeasonDateList
5400	1008251	06/03/2014 to	[06/03/2014,]
5465	1000158	12/05/2015 to	[12/05/2015,]
5512	1010414	03/01/2015 to	[03/01/2015,]
5516	1010802	11/17/2015 to	[11/17/2015,]
5537	1009921	10/10/2014 to	[10/10/2014,]
...	...	...	...
5745	1012351	09/16/2016 to	[09/16/2016,]
5775	1006826	11/01/2012 to	[11/01/2012,]
5876	1011004	09/24/2016 to	[09/24/2016,]
5887	1008842	11/27/2016 to	[11/27/2016,]
5897	1005991	05/01/2015 to	[05/01/2015,]

17 rows × 3 columns

Results from approach # 3 also have string with only whitespace in the list.

Approach #4

```
seasons['SeasonDate'] = [s.strip() for s in seasons['SeasonDate']]
seasons['SeasonDateList'] = [[d for d in s if d] for s in seasons['SeasonDate'].str.split(' to')]
seasons
```

	FMID	SeasonDate	SeasonDateList
0	1012063	06/08/2016 to 10/12/2016	[06/08/2016, 10/12/2016]
1	1011871	06/25/2016 to 10/01/2016	[06/25/2016, 10/01/2016]
2	1011878	05/04/2016 to 10/12/2016	[05/04/2016, 10/12/2016]
3	1010691	04/02/2014 to 11/30/2014	[04/02/2014, 11/30/2014]
4	1002454	July to November	[July, November]
...	...	...	...
5896	1012508	01/15/2017 to 01/15/2017	[01/15/2017, 01/15/2017]
5897	1005991	05/01/2015 to	[05/01/2015]
5898	1010118	04/14/2016 to 04/14/2016	[04/14/2016, 04/14/2016]
5899	1001875	05/28/2016 to 09/24/2016	[05/28/2016, 09/24/2016]
5900	1009023	12/10/2016 to 12/17/2016	[12/10/2016, 12/17/2016]

5901 rows × 3 columns

```
seasons.loc[[len(d) > 2 for d in seasons.SeasonDateList],]
```

FMID	SeasonDate	SeasonDateList

```
seasons.loc[[len(d) < 2 for d in seasons.SeasonDateList],]
```

	FMID	SeasonDate	SeasonDateList
58	1004762	10/25/2014 to	[10/25/2014]
66	1011322	05/19/2016	[05/19/2016]
79	1004846	05/21/2013 to	[05/21/2013]
90	1011910	04/18/2016 to	[04/18/2016]
250	1011092	06/07/2016 to	[06/07/2016]
...	...	...	...
5745	1012351	09/16/2016 to	[09/16/2016]
5775	1006826	11/01/2012 to	[11/01/2012]
5876	1011004	09/24/2016 to	[09/24/2016]
5887	1008842	11/27/2016 to	[11/27/2016]
5897	1005991	05/01/2015 to	[05/01/2015]

134 rows × 3 columns

```
seasons.loc[[' ' in d for d in seasons.SeasonDateList],]
```

	FMID	SeasonDate	SeasonDateList

Results from approach # 4 do not have any violations with more than three items or ‘ ’ (string with only whitespace) in the list. We still have some cases where we only have 1 item in the list but that’s because they don’t have an end date in the original SeasonDate column.

2. After all the cleaning, we do some validation on special cases:

Validation

```
seasons.loc[seasons['FMID'] == 1000165, ]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
4452	1000165	March to November	[March, November]	March	November	NaT

```
seasons.loc[seasons['FMID'] == 1000788, ]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
5135	1000788	July 9, 2012 to October 29, 2012	[July 9, 2012, October 29, 2012]	July 9, 2012	October 29, 2012	2012-10-29

```
seasons.loc[seasons['FMID'] == 1000961, ]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
5114	1000961	05/01/2015 to	[05/01/2015]	05/01/2015	05/01/2015	2015-05-01

```
seasons.loc[seasons['FMID'] == 1001139, ]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
2196	1001139	April to Sept 24, 2011	[April, Sept 24, 2011]	April	Sept 24, 2011	2011-09-24

We also use IC-violation queries to check number of records we cleaned for each type:

```
seasons.loc[[',' in s for s in seasons.SeasonEnd],]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
1476	1000107	May 5, 2012 to Oct 6, 2012	[May 5, 2012, Oct 6, 2012]	May 5, 2012	Oct 6, 2012	2012-10-06
5135	1000788	July 9, 2012 to October 29, 2012	[July 9, 2012, October 29, 2012]	July 9, 2012	October 29, 2012	2012-10-29
5134	1000789	July 12, 2012 to October 25, 2012	[July 12, 2012, October 25, 2012]	July 12, 2012	October 25, 2012	2012-10-25
2198	1001137	April to November 4, 2011	[April, November 4, 2011]	April	November 4, 2011	2011-11-04
2196	1001139	April to Sept 24, 2011	[April, Sept 24, 2011]	April	Sept 24, 2011	2011-09-24
...	...	...	...	...	...	...
4306	1005772	May 7, 2011 to October 15, 2011	[May 7, 2011, October 15, 2011]	May 7, 2011	October 15, 2011	2011-10-15
2130	1005993	May 25, 2012 to August 31, 2012	[May 25, 2012, August 31, 2012]	May 25, 2012	August 31, 2012	2012-08-31
1513	1006135	August 2, 2012 to September 27, 2012	[August 2, 2012, September 27, 2012]	August 2, 2012	September 27, 2012	2012-09-27
1849	1006688	June 30, 2012 to September 1 , 2012	[June 30, 2012, September 1 , 2012]	June 30, 2012	September 1 , 2012	2012-09-01
140	1007647	May 1, 2012 to October 30, 2012	[May 1, 2012, October 30, 2012]	May 1, 2012	October 30, 2012	2012-10-30

30 rows × 6 columns

```
seasons.loc[[',' in str(s) for s in seasons.MostRecentOpeningDate],]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate

Result: # of violations: 30 → 0

```
seasons.loc[['/' in s for s in seasons.SeasonEnd],]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
1397	1000003	05/31/2014 to 10/31/2014	[05/31/2014, 10/31/2014]	05/31/2014	10/31/2014	2014-10-31
5642	1000008	05/21/2016 to 10/29/2016	[05/21/2016, 10/29/2016]	05/21/2016	10/29/2016	2016-10-29
3123	1000008	05/23/2015 to 11/01/2015	[05/23/2015, 11/01/2015]	05/23/2015	11/01/2015	2015-11-01
5865	1000008	11/05/2016 to 04/29/2017	[11/05/2016, 04/29/2017]	11/05/2016	04/29/2017	2017-04-29
4390	1000009	06/14/2014 to 10/11/2014	[06/14/2014, 10/11/2014]	06/14/2014	10/11/2014	2014-10-11
...	...	...	...	...	...	...
2266	1012844	05/06/2017 to 10/28/2017	[05/06/2017, 10/28/2017]	05/06/2017	10/28/2017	2017-10-28
3356	1016768	05/07/2016 to 10/01/2016	[05/07/2016, 10/01/2016]	05/07/2016	10/01/2016	2016-10-01
2539	1016770	10/02/2016 to 02/04/2017	[10/02/2016, 02/04/2017]	10/02/2016	02/04/2017	2017-02-04
1577	2000005	01/01/2014 to 12/31/2014	[01/01/2014, 12/31/2014]	01/01/2014	12/31/2014	2014-12-31
3545	2000021	07/01/2014 to 07/31/2014	[07/01/2014, 07/31/2014]	07/01/2014	07/31/2014	2014-07-31

5024 rows × 6 columns

```
seasons.loc[['/' in str(s) for s in seasons.MostRecentOpeningDate],]
```

FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate

Result: # of violations: 5024 → 0

```
seasons.loc[[re.search(r'\d', s) is None for s in seasons.SeasonEnd],]
```

	FMID	SeasonDate	SeasonDateList	SeasonStart	SeasonEnd	MostRecentOpeningDate
3643	1000010	April to October	[April, October]	April	October	NaT
1860	1000011	April to October	[April, October]	April	October	NaT
4629	1000016	May to October	[May, October]	May	October	NaT
1447	1000022	January to December	[January, December]	January	December	NaT
483	1000023	January to December	[January, December]	January	December	NaT
...	...	...	...	...	...	...
769	1007672	May to October	[May, October]	May	October	NaT
601	1007741	April to October	[April, October]	April	October	NaT
834	1007752	May to November	[May, November]	May	November	NaT
4836	1008037	August to August	[August, August]	August	August	NaT
2079	1010153	June to Octobsr	[June, Octobsr]	June	Octobsr	NaT

846 rows × 6 columns

```
seasons.loc[[re.search(r'\d', s) is None for s in seasons.SeasonEnd], 'MostRecentOpeningDate'].unique()
```

```
array(['NaT'], dtype='datetime64[ns]')
```

Result: # of violations: 846 → 0

Address

All markets should not have empty city, state and county. We use following query to check if there is any empty value:

```
empty_address = '''
SELECT FMID, MarketName, city, state, county
FROM farmersmarkets
WHERE city IS NULL
OR state IS NULL
OR county IS NULL;
'''
data = pd.read_sql(empty_address, conn_ori)
data
```

	FMID	MarketName	city	State	County
0	1009364	106 S. Main Street Farmers Market	Six Mile	South Carolina	None
1	1006234	4th Street Farmers Market	Larimer	Colorado	None
2	1006494	52 & Shadeland Avenue Farmers Market	Indianapolis	Indiana	None
3	1009543	ABV Farm Market	Mount Bethe	Pennsylvania	None
4	1001300	Algonquin Farmers Market	Algonquin	Illinois	None
...	...	...	...	...	...
546	1009255	Wood Dale Farmer's Market	Wood Dale	Illinois	None
547	1007624	Worcester Northeast Side Farmers Market	Worcester	Massachusetts	None
548	1009336	Yazoo Farmers Market	Yazoo City	Mississippi	None
549	1009531	Year-Round Cedar City Farmer's Market	Cedar City	Utah	None
550	2000036	YMCA Farmers Market and Veggie Van	None	Michigan	None

551 rows × 5 columns

Original dataset (before)

```
empty_address = '''
SELECT FMID, MarketName, city, state, county
FROM cleanfarmersmarkets
WHERE city IS NULL
OR state IS NULL
OR county IS NULL;
'''
data = pd.read_sql(empty_address, conn_clean)
data
```

FMID	MarketName	City	State	County

Cleaned dataset (after)

Result: # of violations: 551 → 0

Two markets should not be in the same location. When separating address from farmermarket file, we use `groupby` to see if there is any duplicated street, city, county, state and zip for different FMID and keep the latest record. We use following query to check if there is any empty value:

```
same_address = '''
SELECT *
FROM
(
SELECT FMID, MarketName, zip, street, city, county, state
FROM farmersmarkets
) AS m1,
(
SELECT FMID, MarketName, zip, street, city, county, state
FROM farmersmarkets
) AS m2
WHERE  m1.zip = m2.zip
AND m1.county = m2.county
AND m1.city = m2.city
AND m1.street = m2.street
AND m1.state = m2.state
AND m1.fmid <> m2.fmid;
'''
data = pd.read_sql(same_address, conn_ori)
data
```

28	1012325	Vashon Farmers Market - Saturday	98070	17519 Vashon Highway SW	Vashon	King	Washington	1012329	Vashon Farmers Market - Wednesday	98070	17519 Vashon Highway SW	Vashon	Kin
29	1012329	Vashon Farmers Market - Wednesday	98070	17519 Vashon Highway SW	Vashon	King	Washington	1012325	Vashon Farmers Market - Saturday	98070	17519 Vashon Highway SW	Vashon	Kin
30	1006893	Woburn Farmers Market	01801	41 Wyman Street	Woburn	Middlesex	Massachusetts	1004024	Farmer's Market @ Woburn's Spence Farm	01801	41 Wyman Street	Woburn	Middlese
31	1006893	Woburn Farmers Market	01801	41 Wyman Street	Woburn	Middlesex	Massachusetts	1008931	Woburn Indoor Farmers Market	01801	41 Wyman Street	Woburn	Middlese
32	1008931	Woburn Indoor Farmers Market	01801	41 Wyman Street	Woburn	Middlesex	Massachusetts	1004024	Farmer's Market @ Woburn's Spence Farm	01801	41 Wyman Street	Woburn	Middlese
33	1008931	Woburn Indoor Farmers Market	01801	41 Wyman Street	Woburn	Middlesex	Massachusetts	1006893	Woburn Farmers Market	01801	41 Wyman Street	Woburn	Middlese

Original dataset (before)

```
same_address = '''
SELECT *
FROM
(
SELECT FMID, MarketName, zip, street, city, county, state
FROM cleanfarmersmarkets
) AS m1,
(
SELECT FMID, MarketName, zip, street, city, county, state
FROM cleanfarmersmarkets
) AS m2
WHERE  m1.zip = m2.zip
AND m1.county = m2.county
AND m1.city = m2.city
AND m1.street = m2.street
AND m1.state = m2.state
AND m1.fmid <> m2.fmid;
'''
data = pd.read_sql(same_address, conn_clean)
data
```

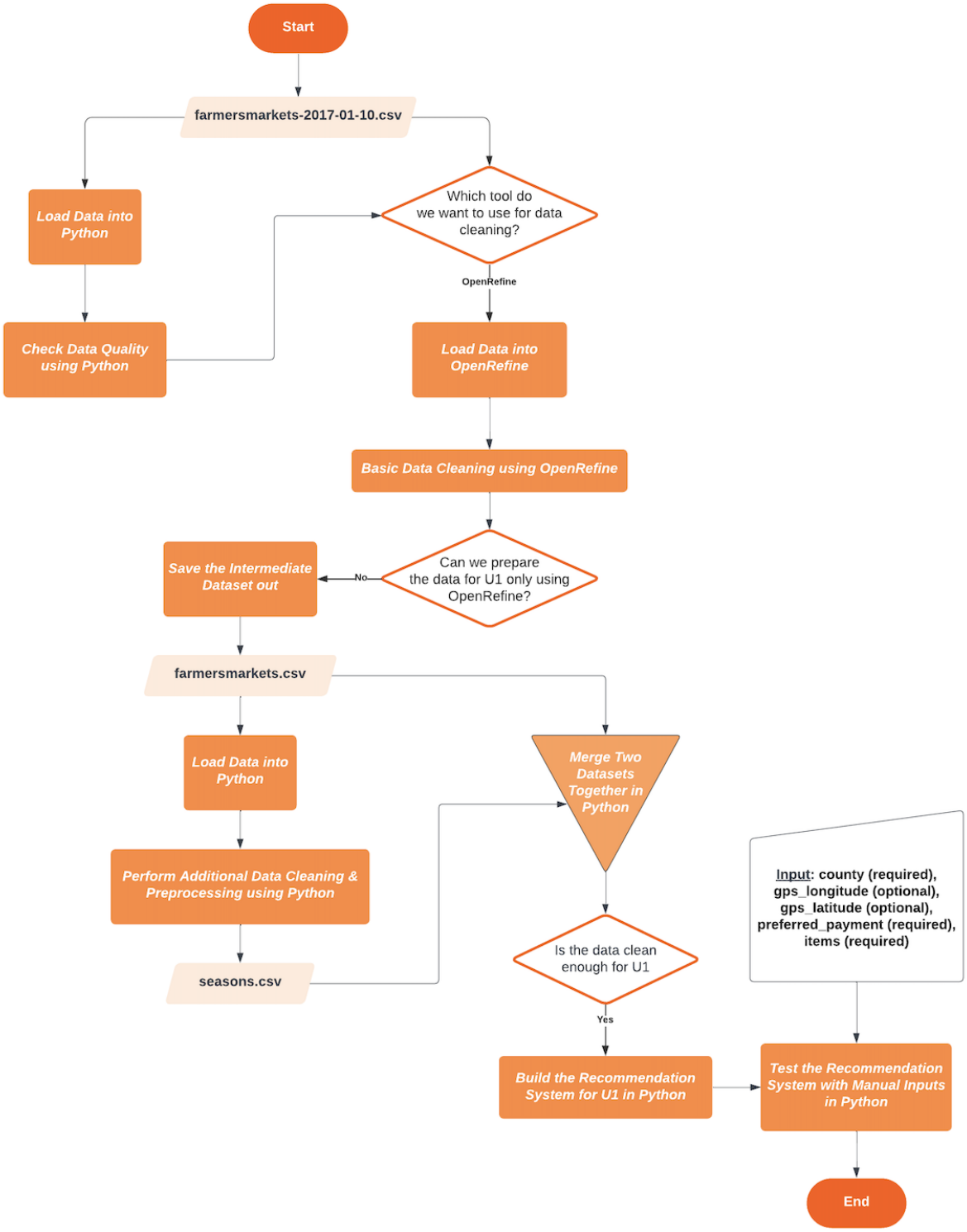
FMID	MarketName	zip	street	city	county	state	FMID	MarketName	zip	street	city	county	state

Cleaned dataset (after)

Result: # of violations: 34 → 0

Workflow Model

Outer Workflow

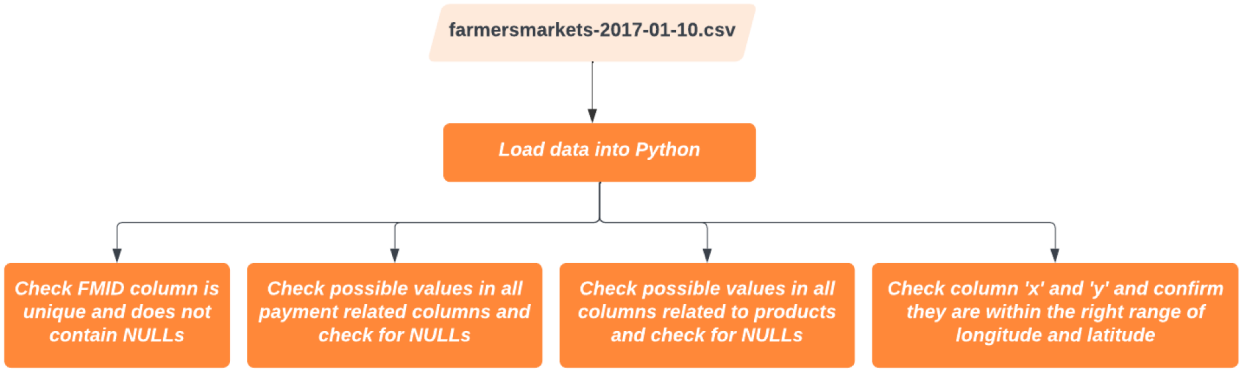


Explanation and Reasoning:

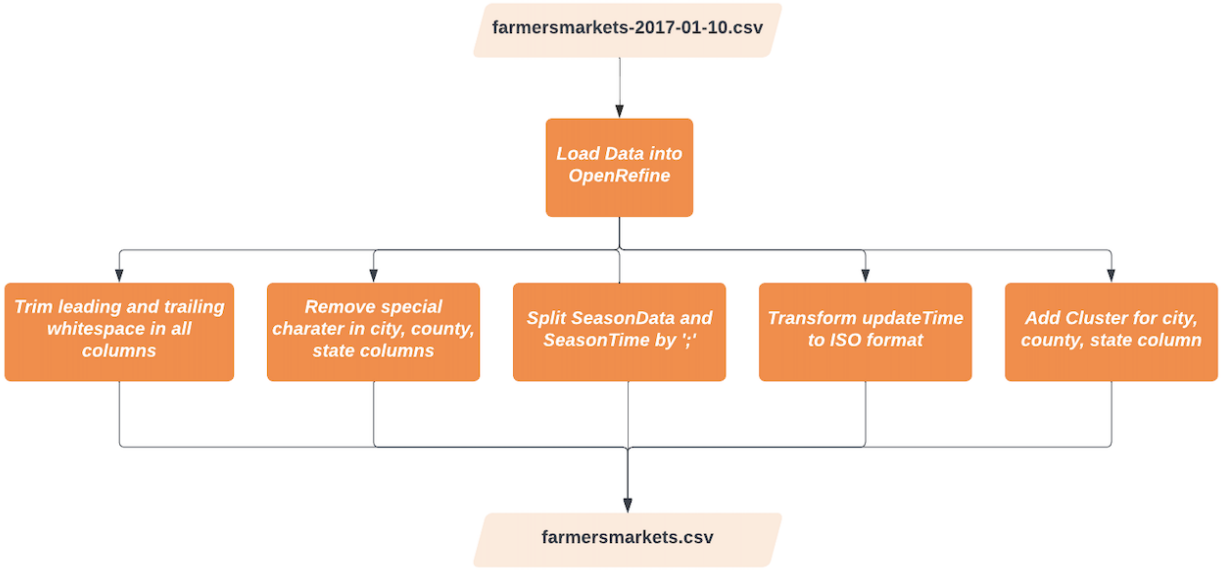
We first use Python to do a data profiling and quality check because it's a relatively more flexible tool. We then use OpenRefine to do some basic data cleaning and clustering, as it has some built-in functions that we can directly leverage. As of the third step, we load the intermediate results from OpenRefine back to Python, and finish the consolidation and preprocessing for information related to seasons. We use Python again because this step requires more customized data cleaning and processing. Finally, we merge the cleaned season dataset with the rest of the dataset from OpenRefine, and do IC violation checks using Python, which allows us to write more customized queries.

Inner Workflow

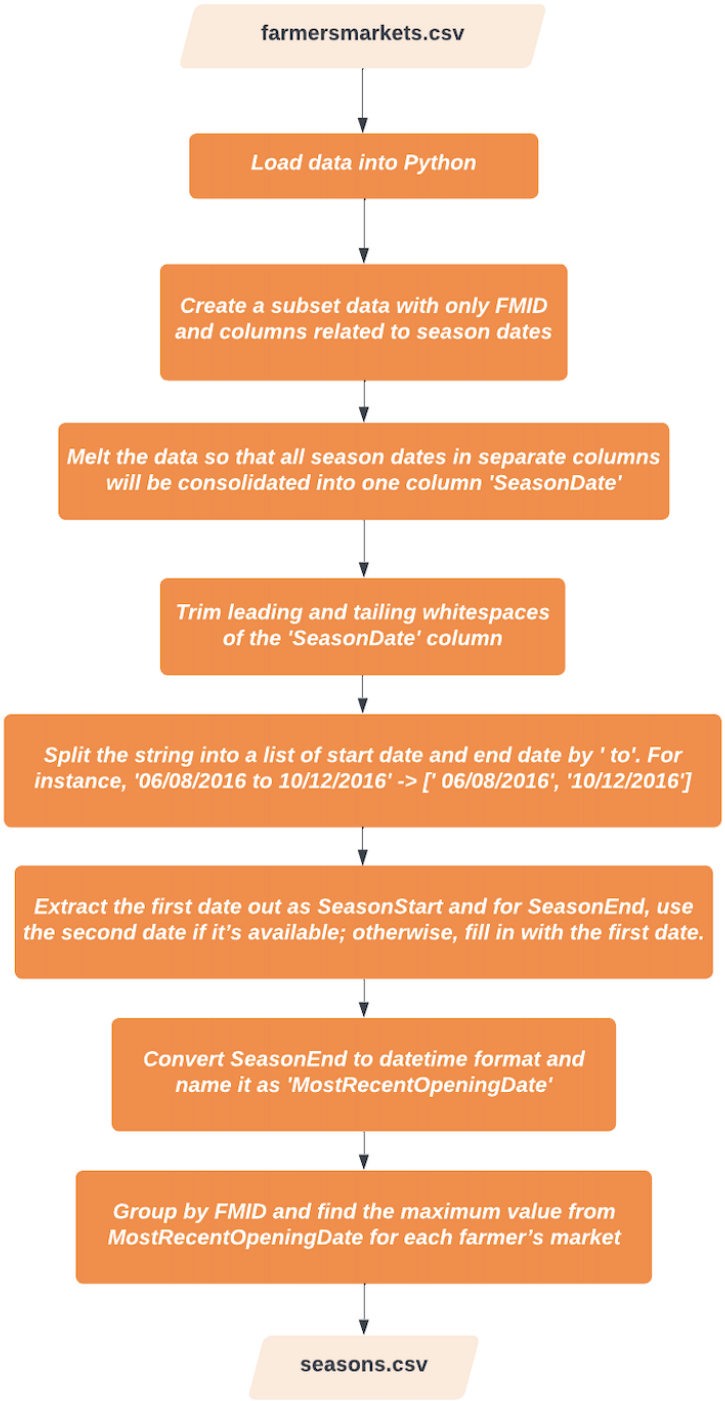
Data Quality Check in Python



Data Cleaning in OpenRefine



Data Cleaning in Python



Conclusions & Summary

Summary

We have gained valuable insights into various data cleaning techniques and tools which are introduced in the course, including OpenRefine and Python. These new skills have equipped us to effectively tackle data cleaning challenges in our workflow. We now understand how to apply these techniques to improve the quality of our data and help implement our main use case. In this project, we have a comprehensive understanding of data cleaning steps, such as removing useless whitespace or special characters, addressing inconsistent data format, handling missing values or typos, dealing with duplicates, and etc.

The biggest challenge we encountered is how to decide which tool to use and check whether the cleaning has covered all different IC violation cases. We learned that OpenRefine is probably more suitable for simple cleaning tasks and is very efficient because it has a lot of powerful built-in functions, while Python is more flexible and useful for special cases and more advanced cleaning purposes.

By integrating these techniques and tools into our data cleaning workflow, we cleaned our datasets thoroughly to some degree. However, for main use case U1, we could incorporate proximity search algorithms or other distance algorithms to determine the closest markets to the user's location and improve the filtering of seasonDate, seasonTime and payment type based on user’s need in the future to better support the recommendation system.

Contributions

Team Member	Contributions
Ruoyu QIAN	● Review and update the use case description ● Profile and check data quality in Python ● Further data cleaning for season dates in Python ● Build recommendation system and test use cases in Python ● Create outer workflow and inner workflow for steps using Python ● Report writeup
Yasmin YIN	● Data cleaning in OpenRefine ● Document and quantify changes ● Further data cleaning for address in Python ● Create inner workflow for steps using OpenRefine ● Report writeup