CS 513: Theory & Practice of Data Cleaning Final Project

Phase-II Report

Wei Dai <weidai6@illinois.edu>
Xiaoyu Wen <xwen20@illinois.edu>
Felicia Liu <liu318@illinois.edu>

Abstract:

The Chicago Department of Public Health has performed sanitation inspections for more than 15,000 food facilities across the City of Chicago each year, and has systematically collected the results of over 150,000 sanitation inspections from 2010 to 2017. These food inspections are beneficial to promote public health in areas of food safety and prevent the occurrence of food-borne illness. This project aims at cleaning the Chicago_Food_Inspection dataset and further analyzing the relationship between the food establishment types and locations in Chicago and the critical violations so as to assist the consumers in selecting the least risky food facilities and areas in Chicago.

1. Overview and initial assessment of the dataset

1.1 Technological tools for cleaning and analyzing the dataset

The following technological tools are adapted to perform the data cleaning and complete this report:

- Python 3 with Jupyter Notebook
- Tableau

1.2 Dataset description

The Chicago food inspection dataset can be publicly accessed via Kaggle and City of Chicago. The dataset contains 153,810 entries and 17 columns starting from 2010 to 2017. Overall, the dataset provides information regarding the facility's inspection characteristics, identification and location. For further detail, the below table provided a brief description in each column of the dataset.

Table 1 - Brief description of the dataset

Name	Type	Description
Inspection ID	int	Unique inspection ID of the facility
DBA Name	String	'Doing business as'. This is the legal name of the establishment.
AKA Name	String	'Also known as'. This is the name the public would know the establishment as.
License #	int	This is a unique number assigned to the establishment for the purposes of licensing by the Department of Business Affairs and Consumer Protection.
Facility Type	String	Each establishment is described by one of the following: bakery, banquet hall, candy store, caterer, coffee shop, day care center (for ages less than 2), day care center (for ages 2 – 6), day care center (combo, for ages less than 2 and 2 – 6 combined), gas station, Golden Diner, grocery store, hospital, long term care center(nursing home), liquor store, mobile food dispenser, restaurant, paleteria, school, shelter, tavern, social club, wholesaler, or Wrigley Field Rooftop.
Risk	String	Each establishment is categorized as to its risk of adversely affecting the public's health, with 1 being the highest and 3 the lowest. The frequency of inspection is tied to this risk, with risk 1 establishments inspected most frequently and risk 3 least frequently.
Address	String	Address of the facility according to Chicago food inspection dataset
City	String	City of the facility according to Chicago food inspection dataset

State	String	State of the facility according to Chicago food inspection dataset
Zip	int	Zip Code of the facility according to Chicago food inspection dataset
Inspection Date	date	This is the date the inspection occurred. A particular establishment is likely to have multiple inspections which are denoted by different inspection dates.
Inspection Type	String	An inspection can be one of the following types: canvass, the most common type of inspection performed at a frequency relative to the risk of the establishment; consultation, when the inspection is done at the request of the owner prior to the opening of the establishment; complaint, when the inspection is done in response to a complaint against the establishment; license, when the inspection is done as a requirement for the establishment to receive its license to operate; suspect food poisoning, when the inspection is done in response to one or more persons claiming to have gotten ill as a result of eating at the establishment (a specific type of complaint- based inspection); task-force inspection, when an inspection of a bar or tavern is done. Re-inspections can occur for most types of these inspections and are indicated as such.
Results	String	An inspection can pass, pass with conditions or fail. Establishments receiving a ‘pass’ were found to have no critical or serious violations (violation number 1-14 and 15- 29, respectively). Establishments receiving a ‘pass with conditions’ were found

		to have critical or serious violations, but these were corrected during the inspection. Establishments receiving a ‘fail’ were found to have critical or serious violations that were not correctable during the inspection. An establishment receiving a ‘fail’ does not necessarily mean the establishment’s license is suspended. Establishments found to be out of business or not located are indicated as such.
Violations	String	An establishment can receive one or more of 45 distinct violations (violation numbers 1-44 and 70). For each violation number listed for a given establishment, the requirement the establishment must meet in order for it to NOT receive a violation is noted, followed by a specific description of the findings that caused the violation to be issued.
Latitude	float	Latitude of the facility
Longitude	float	Longitude of the facility
Location	float	Location including latitude and longitude information of the facility

1.3 Dataset before cleaning and after cleaning

1.3.1 Dataset before cleaning

Input dataset: Raw_data_Food_Inspections.csv

	Inspection ID	DBA Name	AKA Name	License #	Facility Type	Risk	Address	City	State	Zip	Inspection Date	Inspection Type	Results	Violations	Latitude	Longitude	Location
0	2079132	MARRIOT MARQUIS CHICAGO	MARRIOT MARQUIS CHICAGO	2517328.0	Restaurant	Risk 1 (High)	2121 S PRAIRIE AVE	CHICAGO	IL	60616.0	08/28/2017	License	Pass	NaN	41.853651	-87.620534	(41.853650885040594, -87.62053358114167)
1	2079129	JET'S PIZZA	JET'S PIZZA	2522268.0	Restaurant	Risk 2 (Medium)	1025 W MADISON ST	CHICAGO	IL	60607.0	08/28/2017	License	Not Ready	NaN	41.881572	-87.653052	(41.88157249576794, -87.65305233593274)
2	2079125	ROOM 1520	ROOM 1520	2446638.0	Special Event	Risk 3 (Low)	301 N JUSTINE ST	CHICAGO	IL	60607.0	08/28/2017	License Re-Inspection	Not Ready	8. SANITIZING RINSE FOR EQUIPMENT AND UTENSILS...	41.886577	-87.665328	(41.8865765215085 4, -87.66532812140231)
3	2079123	MARRIOT MARQUIS CHICAGO	MARRIOT MARQUIS CHICAGO	2517338.0	Restaurant	Risk 1 (High)	2121 S PRAIRIE AVE	CHICAGO	IL	60616.0	08/28/2017	License	Pass	35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTR...	41.853651	-87.620534	(41.853650885040594, -87.62053358114167)
4	2079105	CHARTWELLS	CICS WEST BELDEN CAMPUS	2549079.0	CHARTER SCHOOL	Risk 1 (High)	2245 N MCVICKER AVE	CHICAGO	IL	60639.0	08/28/2017	License Re-Inspection	Pass	18. NO EVIDENCE OF RODENT OR INSECT OUTER OPEN...	41.921675	-87.776711	(41.921675488910864, -87.7767113569357)

1.3.2 Dataset after cleaning

Output dataset: Clean_food_inspection_data.csv

	Inspection_Date	Inspection_ID	DBA_Name	License_#	Facility_Type	Risk	Address	City	Zip	Inspection_Type	Results	Latitude	Longitude	Year	Month	Day
0	01/02/2013	1146224	EL CASTILLO MEAT MARKET	66267	GROCERY STORE	Risk 1 (High)	3704 W 55TH ST	CHICAGO	60632	Canvass	FAIL	41.7934	-87.7160	2013	1	2
1	01/02/2013	1202588	CHURCH'S CHICKEN # 1053	1273275	RESTAURANT	Risk 2 (Medium)	1151 S INDEPENDENCE BLVD	CHICAGO	60624	Complaint	FAIL	41.8665	-87.7194	2013	1	2
2	01/02/2013	1202587	ADAM & ARAM CORPORATION	2137725	GROCERY STORE	Risk 3 (Low)	5145 W DIVISION ST	CHICAGO	60651	Complaint	PASS	41.9021	-87.7552	2013	1	2
3	01/02/2013	1146223	LE PARIS BAKERY NO 2 INC	1095974	RESTAURANT	Risk 1 (High)	5434 S PULASKI RD	CHICAGO	60632	Canvass	FAIL	41.7940	-87.7233	2013	1	2
4	01/02/2013	1286217	A MOTHER'S TOUCH DAYCARE	1738081	DAYCARE (2 - 6 YEARS)	Risk 1 (High)	1505 N PULASKI RD	CHICAGO	60651	Canvass	PASS	41.9083	-87.7263	2013	1	2

1.4 Project Use Case U1

Food safety has brought itself into notice with an increasing number of customers paying more attention to the violations that the food establishments may have. As a consequence, the governments devote themselves to collecting relevant information and providing safety ratings for customers. Therefore, this report directs at understanding and analyzing the relationship between food facility types and locations and the inspection result.

2. Data cleaning procedures

2.1 High-level steps of cleaning data

The data cleaning procedures are mainly completed in the Python Jupyter Notebook and generally, the following steps are conducted to clean the data in order to further analyze U1.

- Redundant columns with respect to the use case got removed: After studying and comprehending the dataset and data points, column [State], [Violations], [Locations], [AKA Name] are moved since we do not need these information to come up with the correlation between food establishments' types and locations and its corresponding inspection results.
- Handled with missing values: After checking the missing values, we noticed that there are 4,560 [Facility_Type] values missing, 544 [Longitude] and [Latitude] values missing, 159 [City] values missing, 98 [Zip] values missing, 66 [Risk] values missing, 15 [License_#] values missing, 1 [Inspection_Type] values missing. Since the dataset contains more than 150,000 entries, we decided to drop all these missing values due to the reason that we will still have sufficient data points for analysis even if we dropped all the missing values.
- Checked the duplicates and dropped the duplicates if applicable: After checking the dataset, we noticed that there are no duplicates in this dataset

- Data types and decimals of the integers were corrected: To better organize the dataset, we performed some procedures including rounding the column [Latitude] and [Longitude] to 4 decimals, replaced np.nan format to integer 0 for column [License_#], divided the column [Inspection_Date] into year, month, day and etc.,
- Similar clusters were clustered as a group: For certain columns like [DBA_Name] and [City], we combined the different values since it should point to the same value in the dataset. For instance, after combining the similar groups, we changed city like “CCHICAGO”, “CHCHICAGO”, “CHICAGOI” to “CHICAGO”.

2.2 Detailed level steps of cleaning data and rationale adapted

2.2.1 Step 1. Import libraries and dataset, and understand the data set.

2.2.1.1 By carefully read each columns, some information caught our eyes:

- Risk category: Each establishment is categorized as to its risk of adversely affecting the public’s health, with 1 being the highest and 3 the lowest.

This column tells us there are three risk levels, any rows without risk level or not showing the right risk level should be removed from our dataset.

- Results: An inspection can pass, pass with conditions or fail. Establishments receiving a ‘pass’ were found to have no critical or serious violations. Establishments receiving a ‘pass with conditions’ were found to have critical or serious violations, but these were corrected during the inspection.

This column tells us there are three types of results, any results not included should be removed from our dataset. We will show it later on.

- Inspection date: This is the date the inspection occurred. A particular establishment is likely to have multiple inspections which are denoted by different inspection dates.

This column shows that there may be duplicates in License number columns, since one particular facility may be inspected multiple times.

```
df.head(5)
```

	Inspection ID	DBA Name	License #	Facility Type	Risk	Address	City	Zip	Inspection Date	Inspection Type	Results	Latitude	Longitude
0	2079132	MARRIOT MARQUIS CHICAGO	2517328.0	Restaurant	Risk 1 (High)	2121 S PRAIRIE AVE	CHICAGO	60616.0	08/28/2017	License	Pass	41.853651	-87.620534
1	2079129	JET'S PIZZA	2522268.0	Restaurant	Risk 2 (Medium)	1025 W MADISON ST	CHICAGO	60607.0	08/28/2017	License	Not Ready	41.881572	-87.653052
2	2079125	ROOM 1520	2446638.0	Special Event	Risk 3 (Low)	301 N JUSTINE ST	CHICAGO	60607.0	08/28/2017	License Re-Inspection	Not Ready	41.886577	-87.665328
3	2079123	MARRIOT MARQUIS CHICAGO	2517338.0	Restaurant	Risk 1 (High)	2121 S PRAIRIE AVE	CHICAGO	60616.0	08/28/2017	License	Pass	41.853651	-87.620534
4	2079105	CHARTWELLS	2549079.0	CHARTER SCHOOL	Risk 1 (High)	2245 N MCVICKER AVE	CHICAGO	60639.0	08/28/2017	License Re-Inspection	Pass	41.921675	-87.776711

2.2.1.2 By using df.info() command, we carefully inspected each column. We noted that there are some missing data and some wrong data types. We will deal with them later on.

```
In [4]: df.info(verbose = True)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153810 entries, 0 to 153809
Data columns (total 17 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   Inspection ID    153810 non-null  int64
 1   DBA Name         153810 non-null  object
 2   AKA Name         151267 non-null  object
 3   License #        153795 non-null  float64
 4   Facility Type    149250 non-null  object
 5   Risk             153744 non-null  object
 6   Address          153810 non-null  object
 7   City             153651 non-null  object
 8   State            153802 non-null  object
 9   Zip              153712 non-null  float64
 10  Inspection Date  153810 non-null  object
 11  Inspection Type  153809 non-null  object
 12  Results          153810 non-null  object
 13  Violations       123012 non-null  object
 14  Latitude         153266 non-null  float64
 15  Longitude        153266 non-null  float64
 16  Location         153266 non-null  object
dtypes: float64(4), int64(1), object(12)
memory usage: 19.9+ MB
```

2.2.2 Step 2. Remove redundant columns, regularize column names, clean missing values.

2.2.2.1 Remove redundant columns

- Column ‘State’: All facilities are in IL, except 8 missing data, so the ‘State’ column is not necessary to our case U1. Column removed.

- Column ‘Violations’ : Our main target is to find the risk levels and inspection results corresponding with the zip code. The details of each violation are not important to our study. Column removed.
- Column ‘Location’ : The column in ‘Location’ is the combination of column ‘Latitude’ and ‘Longitude’, Column removed.

```
# All facilities are in IL, except 8 missing data (153810 total entries - 153802 entries with state)
# so 'State' column is not very necessary
print(df['State'].value_counts())
df.drop(columns = ['State'], inplace=True)

# Our main target is to find the risk levels corresponding with the zip code.
# The details of each violation are not important to our study.
df.drop(columns = ['Violations'], inplace=True)

# The information in 'Location' is duplicated with ‘Latitude’ and ‘Longitude’
df.drop(columns='Location', inplace=True)

# 'AKA name' is relevant to our use cases.
df.drop(columns='AKA Name', inplace=True)
```

2.2.2.2 Regularize column names

After inspecting the columns’ names, we noticed that there is some space in the columns' names. We removed the space and we changed each name to ‘firstword_secondword’ form. Also we split column ‘Inspection_Date’ into three columns ‘Year’, ‘Month’ and ‘Day’ for better usage later on.

```
# After inspecting the colomn name, we noticed that there are some space in the columns' name.
# Clean up the columns' name and remove the spaces
df.columns = [each.split()[0] + "_" + each.split()[1] if len(each.split()) > 1 else each for each in df.columns]

# Separate the inspection_date into Year, Month and Day
df['Year'] = pd.DatetimeIndex(df['Inspection_Date']).year
df['Month'] = pd.DatetimeIndex(df['Inspection_Date']).month
df['Day'] = pd.DatetimeIndex(df['Inspection_Date']).day

df.columns
```

```
Index(['Inspection_ID', 'DBA_Name', 'License_#', 'Facility_Type', 'Risk',
       'Address', 'City', 'Zip', 'Inspection_Date', 'Inspection_Type',
       'Results', 'Latitude', 'Longitude', 'Year', 'Month', 'Day'],
      dtype='object')
```

2.2.2.3 Check and clean missing values

We use isnull() command there are a few missing values in column ‘Facility_Type’, ‘Longitude’, ’Latitude’, ’City’, ‘Zip’, ‘Risk’, ‘License_#’, ‘Inspection_Type’. Since it is only very small numbers compared with our large amount of data, we believe removing them will not affect our result, so we dropped those rows directly, by using command dropna().

```
# Drop rows missing Facility_Type, Longitude, Latitude, Zip, Risk and Inspection_Type
df.dropna(subset=['Facility_Type'], inplace=True)
df.dropna(subset=['Longitude'], inplace=True)
df.dropna(subset=['Latitude'], inplace=True)
df.dropna(subset=['Zip'], inplace=True)
df.dropna(subset=['Risk'], inplace=True)
df.dropna(subset=['Inspection_Type'], inplace=True)
df.dropna(subset=['License_#'], inplace=True)
df.dropna(subset=['City'], inplace=True)

# Check missing values after handling the missing values
print('Missing values for each column in the Dataframe:')
df.isnull().sum().sort_values(ascending=False)
```

```
Missing values for each column in the Dataframe:

Day                 0
Month               0
Year                0
Longitude           0
Latitude            0
Results             0
Inspection_Type     0
Inspection_Date     0
Zip                 0
City                0
Address             0
Risk                0
Facility_Type       0
License_#           0
DBA_Name            0
Inspection_ID       0
dtype: int64
```

2.2.3 Step 3. Check duplicates, correct data type and decimal, trim strings and clustering, remove outliers

2.2.3.1 Check duplicates

After removing unnecessary columns and rows, there are 148538 entries left. By using nunique() command, we noticed that inspection ID is unique to the total entries, whereas there are many duplicates in other columns such as ‘DBA_Name’ ‘Longitude’ ‘Latitude’ , etc. Those columns need to be cleaned further.

```
# Column 'Inspection ID' is clean, with unique numbers each, and correct data type.
print('Total entries:', len(df))
print('Number of unique values in each column: ')
print(df.nunique().sort_values(ascending=False))
```

```
Total entries: 148538
Number of unique values in each column:
Inspection_ID      148538
License_#           28521
DBA_Name            21151
Address             15851
Longitude           15062
Latitude            15062
Inspection_Date      1945
Facility_Type         435
Inspection_Type       105
Zip                    63
Day                    31
City                   17
Month                  12
Year                    8
Results                 7
Risk                    4
dtype: int64
```

```
# There is no duplicate since the number of unique inspection ID is equal to the total entries.
duplicates = df[df.duplicated()]

duplicates
```

2.2.3.2 Correct data types and decimal

- Column ‘Latitude’ and ‘Longitude’: they are in correct data type (float). But we noticed that they are too precise for us to use. We rounded them to four decimals.

- Column ‘License_#’ and ‘Zip’: Data Type changed from ‘float’ to ‘int’, using command astype()

- Column ‘Inspection Date’ : Data Type changed from ‘object’ to ‘datetime’. Then we sort the data frame by inspection date to make it clearer.

```
# ‘Latitude’ and ‘Longitude’ should be rounded to four decimals.
df['Latitude'] = round(df['Latitude'], 4)
df['Longitude'] = round(df['Longitude'], 4)

# Column ‘zip’ and ‘License #’ should be converted to integer.
df['License_#'] = df['License_#'].replace(np.nan, 0)
df['License_#'] = df['License_#'].astype(int)

# If the facility's zipcode is missing, we are not able to locate it,
# thus all rows without  zip code are removed, along with wrong zipcodes.
df['Zip'] = df['Zip'].astype(int)

# 'Inspection Date' Column should be datetime
pd.set_option('display.max_rows', 10)
pd.to_datetime(df['Inspection_Date'])

# Sort the dataframe by inspection date to make it more clear.
df.sort_values("Inspection_Date", inplace=True)
df.insert(0, 'Inspection_Date', df.pop('Inspection_Date'))
df.head(5)
```

	Inspection_Date	Inspection_ID	DBA_Name	License_#	Facility_Type	Risk	Address	City	Zip	Inspection_Type	Results	Latitude	L
98202	01/02/2013	1146224	EL CASTILLO MEAT MARKET	66267	GROCERY STORE	Risk 1 (High)	3704 W 55TH ST	CHICAGO	60632	Canvass	FAIL	41.7934	
98201	01/02/2013	1301079	NUMBER 1 CHOP SUEY	1677434	RESTAURANT	Risk 1 (High)	7342 S STONY ISLAND AVE	CHICAGO	60649	Complaint	PASS	41.7611	
98204	01/02/2013	1286218	RESTAURANT & POZOLERIA SAN JUAN	80690	RESTAURANT	Risk 1 (High)	1523 N PULASKI RD	CHICAGO	60651	Canvass Re-Inspection	PASS	41.9088	
98187	01/02/2013	1151048	NICK'S GYROS	1403378	RESTAURANT	Risk 1 (High)	2011 W 63RD ST	CHICAGO	60636	Complaint	PASS	41.7792	
98190	01/02/2013	1235577	MCDONALD'S	1225984	RESTAURANT	Risk 2 (Medium)	6125 N MILWAUKEE AVE	CHICAGO	60646	Canvass Re-Inspection	PASS	41.9929	

2.2.3.3 Trim strings and clustering

We noticed that there are a lot of similar strings in column ‘DBA_Name’ are actually has the same meaning, such as “mcdonald's","McDONALD'S","MCDONALDS","MCDONALD'S RESTAURANT”, those names should be clustered. So we firstly trim the strings and change them to upper case. Then, we clustered them by using replace() command. Then the same as the column ‘City’.

```
# Trim the strings
df['DBA_Name'] = df['DBA_Name'].str.strip()
df['Facility_Type'] = df['Facility_Type'].str.strip()

# Change to upper cases
df['DBA_Name'] = df['DBA_Name'].str.upper()
df['Facility_Type'] = df['Facility_Type'].str.upper()
df['Address'] = df['Address'].str.upper()
df['City'] = df['City'].str.upper()
df['Results'] = df['Results'].str.upper()
df['DBA_Name'].value_counts()

# Many names can be clustered, e.g: 'Subway' & 'Subway Sandwiches', 'MCDONALD'S' & 'MC DONALD'S'
df['DBA_Name'] = df['DBA_Name'].replace(["MC DONALD'S","McDONALD'S","MCDONALDS ","MCDONALD'S RESTAURANT","MCDONALDS"
df['DBA_Name'] = df['DBA_Name'].replace(['SUBWAY SANDWICHES',"SUBWAY SANDWICH","SUBWAY RESTAURANT",'SUBWAY SANDWICHE
df['DBA_Name'] = df['DBA_Name'].replace(['DUNKIN DONUTS/BASKIN ROBBINS','DUNKIN DONUTS BASKIN ROBBINS','DUNKIN DONUT
df['DBA_Name'] = df['DBA_Name'].replace(['KENTUCKY FRIED CHICKEN'],'KFC')
df['DBA_Name'] = df['DBA_Name'].replace(["POPEYES CHICKEN","POPEYE'S CHICKEN","POPEYE'S FRIED CHICKEN"],'POPEYES')
df['DBA_Name'] = df['DBA_Name'].replace(['7 - ELEVEN'],'7-ELEVEN')
df['DBA_Name'] = df['DBA_Name'].replace(['POTBELLY SANDWICH WORKS LLC'],'POTBELLY SANDWICH WORKS')
df['DBA_Name'] = df['DBA_Name'].replace(['STARBUCKS COFFEE'],'STARBUCKS')
df['DBA_Name'] = df['DBA_Name'].replace(["JIMMY JOHN'S","JIMMY JOHNS SANDWICH SHOPS"],"JIMMY JOHNS")

# City
df['City'] = df['City'].replace(["CCHICAGO","CHCHICAGO","CHCICAGO",'CHICAGOI','CHICAGOCHICAGO'],"CHICAGO")

print('Result after clustering:')
print(df['DBA_Name'].value_counts())
print(' ')
print('========================')
print(df['City'].value_counts())
```

```
Result after clustering:
SUBWAY                      2520
DUNKIN DONUTS               1428
MCDONALD'S                  1253
7-ELEVEN                     415
POTBELLY SANDWICH WORKS      408
                            ...
LA  BABY  RESTAURANT           1
TIENDITA GUTIERREZ             1
R & S MINI MART                1
EDENS LIQUOR & FOOD II         1
4JS                            1
Name: DBA_Name, Length: 20185, dtype: int64

========================
CHICAGO            133558
CHESTNUT STREET         8
SUMMIT                  4
CHARLES A HAYES         3
BLUE ISLAND             1
312CHICAGO              1
BEDFORD PARK            1
LOMBARD                 1
Name: City, dtype: int64
```

2.2.3.4 Remove outliers

- Column ‘Results’: As discussed in step 1. We are only looking at facilities that are still open now. So we remove the facilities that are ‘out of business’, ‘Not ready’, ‘no entry’ and ‘business not located’, only keep ‘pass’,’fail’ and ‘pass w/ conditions’.
- Column ‘Risk’: one outlier removed with content ‘All’.

- Column ‘Latitude’ and ‘Longitude’: by using describe() command, we noticed that the standard deviations are pretty small for both of them, so there is no outlier for these two numerical columns. We left them unchanged.

```
# Remove invalid results and only consider successful inspections
print('Original Results: ')
print(df['Results'].value_counts())
df = df[~df.Results.isin(["OUT OF BUSINESS", "NO ENTRY", "NOT READY", "BUSINESS NOT LOCATED"])]
print()
print('Results after removing outliers: ')
print(df['Results'].value_counts())

# Remove invalid risk type
print()
print('Original Risk type: ')
print(df['Risk'].value_counts())
df = df[df['Risk'] != 'All']
print()
print('Risk type after removing outlier: ')
print(df['Risk'].value_counts())
```

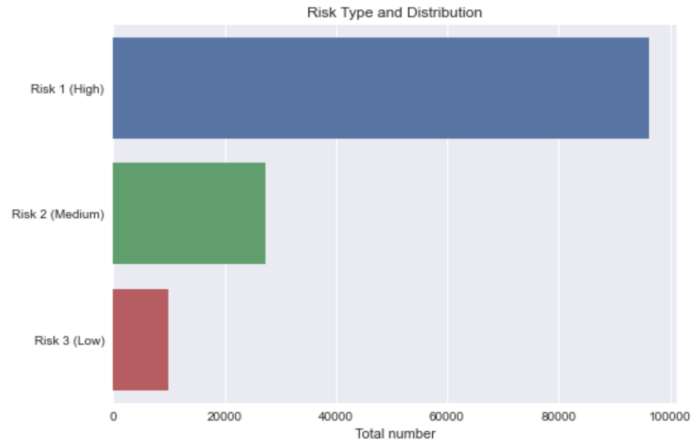
```
Original Results:
PASS                    89797
FAIL                    29321
PASS W/ CONDITIONS      14460
OUT OF BUSINESS          9968
NO ENTRY                 4200
NOT READY                 756
BUSINESS NOT LOCATED       36
Name: Results, dtype: int64

Results after removing outliers:
PASS                   89797
FAIL                   29321
PASS W/ CONDITIONS     14460
Name: Results, dtype: int64

Original Risk type:
Risk 1 (High)      96324
Risk 2 (Medium)    27408
Risk 3 (Low)        9845
All                    1
Name: Risk, dtype: int64
```

2.2.4 Step4 Data Visualization to explain use case

2.2.4.1 Check Risk Type

We use bar charts to examine risk types. More than 80,000 facilities are in low risk level, whereas only less than 20,000 are in low risk level.

```
# Plot Risk Type
c2=sns.barplot(x=df['Risk'].value_counts()[:4],y=df['Risk'].value_counts()[:4].index)
c2.set_title('Risk Type and Distribution')
c2.set_ylabel('')
c2.set_xlabel('Total number')
plt.show()
```



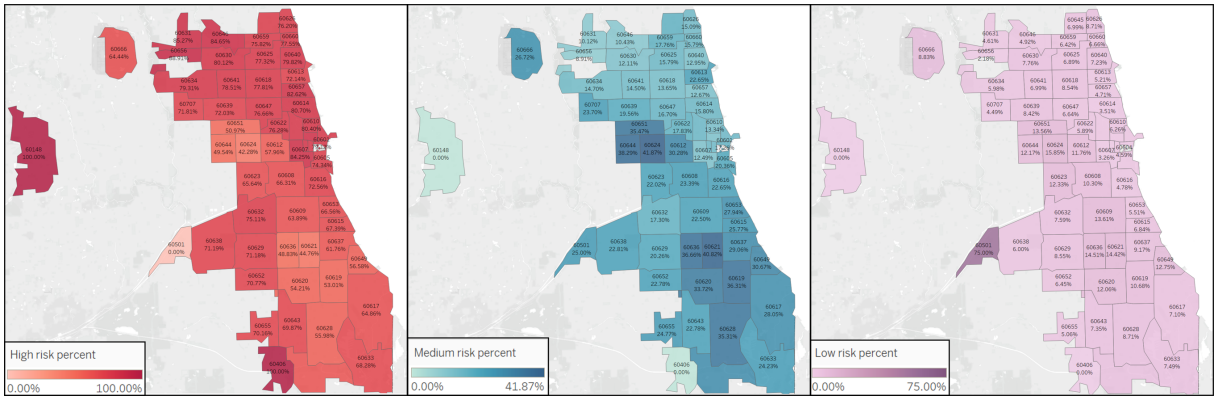
2.2.4.2 Create separate data frame for visualizing risk percentage distributions by Zipcode

Since our target is to find the relationship between zip code and risk level. We separate the data frame by risk level. Then we export the risk distribution dataset and plot choropleth map in Tableau.

```
# Create separete dataframe for visualizing risk percentage distributions by Zipcode
risk_df = df.groupby(['Zip','Risk']).count()
risk_df.reset_index(level='Risk',inplace=True)
risk_dis = risk_df[['Risk','Inspection_ID']]
risk_dis['High_risk'] = risk_dis[risk_dis['Risk'] == 'Risk 1 (High)']['Inspection_ID']
risk_dis['Medium_risk'] = risk_dis[risk_dis['Risk'] == 'Risk 2 (Medium)']['Inspection_ID']
risk_dis['Low_risk'] = risk_dis[risk_dis['Risk'] == 'Risk 3 (Low)']['Inspection_ID']
risk_dis.reset_index(inplace=True)
risk_dis = risk_dis.groupby(['Zip']).mean()
risk_dis.drop(columns='Inspection_ID', inplace=True)
risk_dis.reset_index(inplace=True)
risk_dis.sort_values('High_risk', ascending=False)
risk_dis.fillna(0,inplace=True)
risk_dis['risk_sum'] = risk_dis['High_risk'] + risk_dis['Medium_risk'] + risk_dis['Low_risk']
risk_dis['High_risk_percent'] = risk_dis['High_risk'] / (risk_dis['risk_sum'])
risk_dis['Medium_risk_percent'] = risk_dis['Medium_risk'] / (risk_dis['risk_sum'])
risk_dis['Low_risk_percent'] = risk_dis['Low_risk'] / (risk_dis['risk_sum'])

risk_dis.head()
```

	Zip	High_risk	Medium_risk	Low_risk	risk_sum	High_risk_percent	Medium_risk_percent	Low_risk_percent
0	60148	1.0	0.0	0.0	1.0	1.000000	0.000000	0.000000
1	60406	1.0	0.0	0.0	1.0	1.000000	0.000000	0.000000
2	60501	0.0	1.0	3.0	4.0	0.000000	0.250000	0.750000
3	60601	1898.0	414.0	87.0	2399.0	0.791163	0.172572	0.036265
4	60602	787.0	204.0	50.0	1041.0	0.756004	0.195965	0.048031

Graph 1 - Choropleth map of risk distributions by Zip Code

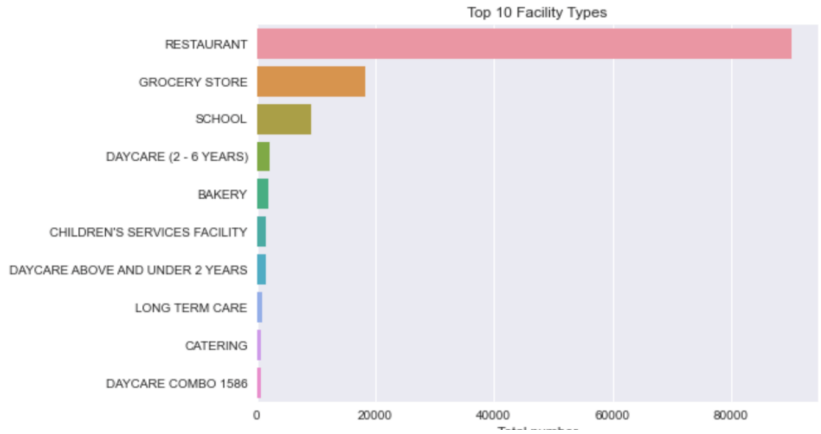


Graph 1 shows the distribution of high, medium and low risk level by percentage of each risk level by Zip code in the map. Each area is labeled by Zip code and risk percentage. The proportion of high risk is more than 50% in most areas, and medium risk is about 10% ~ 20% correspondingly. The proportion of low risk is less than 10% in most areas. Only in the 60501 area, the low risk percentage reaches 75% which represents the most safe inspection area.

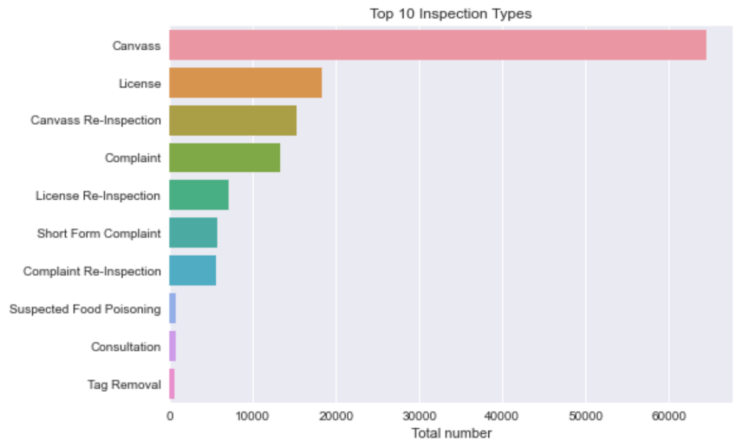
2.2.4.3 Check top 10 facility types and top 10 inspection types

Restaurant, grocery store and school are the top 3 facility types; canvass, License and canvass re-inspection are the top 3 inspection types.

```
# Check top 10 facility types
c1=sns.barplot(x=df['Facility_Type'].value_counts()[:10],y=df['Facility_Type'].value_counts()[:10].index)
c1.set_title('Top 10 Facility Types')
c1.set_ylabel('')
c1.set_xlabel('Total number')
plt.show()
```



```
# Check top 10 inspection types
c3=sns.barplot(x=df['Inspection_Type'].value_counts()[:10],y=df['Inspection_Type'].value_counts()[:10].index)
c3.set_title('Top 10 Inspection Types')
c3.set_ylabel('')
c3.set_xlabel('Total number')
plt.show()
```



3. Data quality changes

3.1 Data quality issues summary

After investigating the dataset in Python Jupyter Notebook, we identified the below potential data quality issues:

- Leading / trailing spaces in the string type values
- Special characters like '#$^&*()[]
- Blank values for certain fields and shown as np.nan
- Inaccurate data format
- Inconsistent date format for date type values
- Poor formatting
- Missing values
- Type errors for string type fields
- Outliers affecting the data analysis

3.2 Data quality issue change measures

To better analyze the use case, we performed several data cleaning steps to clean the dataset. The below table elaborates on the details of the data quality issue descriptions and the data quality change measures we have taken by using Python.

Table 2 - Data Quality Changes Summary

Data Quality Issue Type	Column with issues	Data Quality Remediation Action
There are redundant columns that are irrelevant to the analysis	State	Used Python Pandas Library function .drop to get rid of redundant columns
	Violations	
	Location	
	AKA Name	
Column titles are not well-organized	Inspection ID	Removed the white spaces and replaced by underscore by using Python Pandas Library function .split()
	DBA Name	
	AKA Name	
	License #	
	Facility Type	
	Inspection Date	
	Inspection Type	
Column contains multiple information	inspection_date	Separated the column inspection_date into Year, Month and Day for further analysis by using Python Pandas Library function .DatetimeIndex
Missing values exist in multiple columns	Facility_Type	1. Filtered out all the columns that have missing values in Python Jupyter Notebook by using .isnull().sum().sort_values;
	Longitude	
	Latitude	
	City	

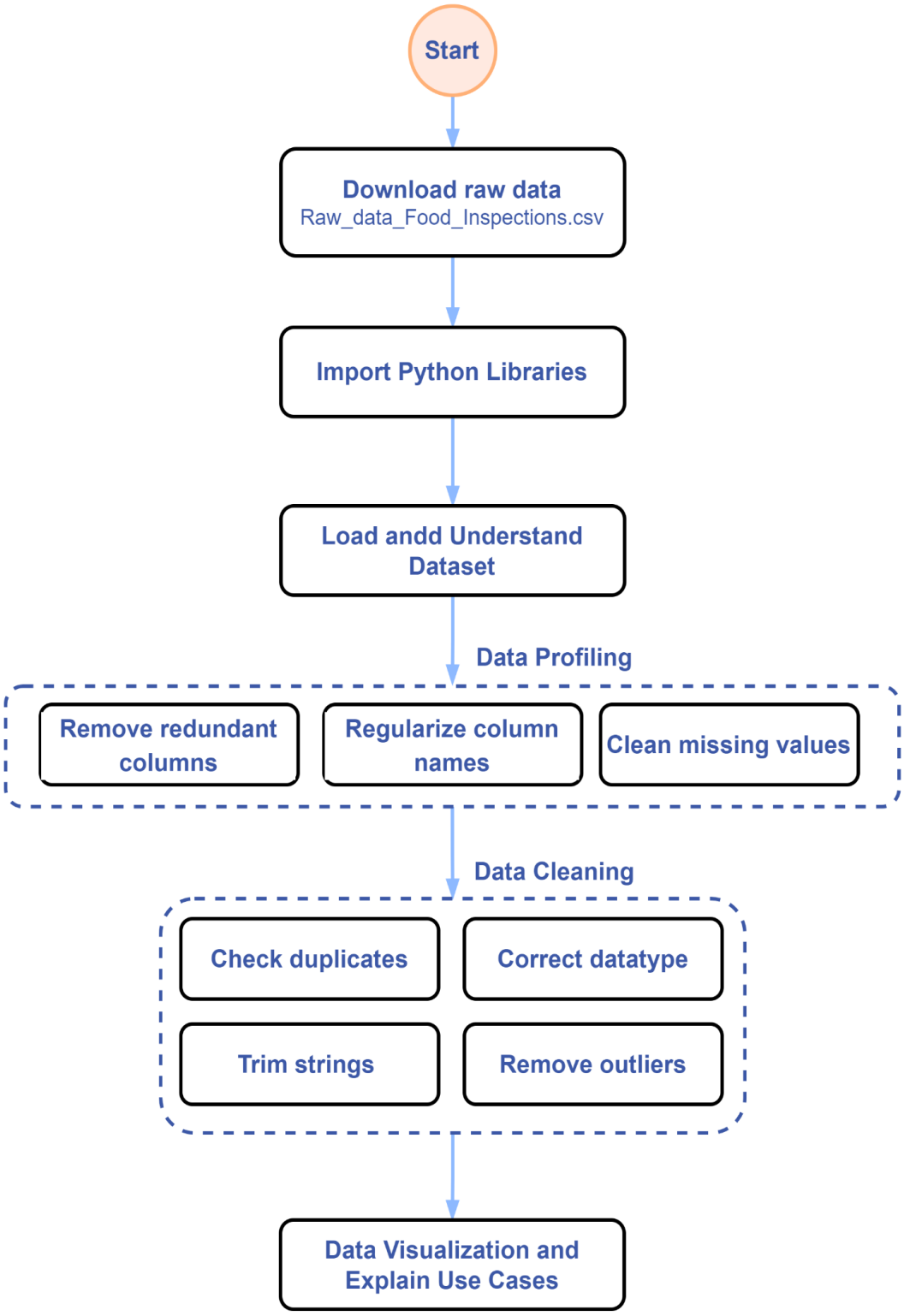
	Zip	2. Dropped rows with missing values for all the columns by using Python Pandas Library function .dropna 3. Confirmed that all the rows with missing values were removed by using .isnull().sum().sort_values
	Risk	
	License_#	
	Inspection_Type	
Wrong data types are saved for certain columns in the original dataset	License_#	Used Python Pandas Library function .astype(int) to change the textual data type (String) to numerical data type (integer)
	Zip	
	Inspection_Date	Used Python Pandas Library function to_datetime to change the column [Inspection_Date] to datetime
There are white spaces in the beginning and at the end of the string type values, which may affect the analysis	DBA_Name	Removed the white spaces by using Python Pandas Library function .str.strip()
	Facility_Type	
There are both upper and lower letters for the string values, which may affect the analysis	DBA_Name	Changed all the letters to upper letter by using Python Pandas Library function .str.upper()
	Facility_Type	
	Address	
	City	
	Results	

Some values are reorganized as different values due to typo errors	DBA_Name City	1. Used Python Pandas Library function .value_counts() to check if there exists similar clusters 2. Used Python Pandas Library function .replace to cluster the similar groups into the same value
There are some outliers existing in certain columns, which may affect the analysis	Results Risk	Got rid of outliers that are not suitable for further analysis by using .isin and ~.isin function

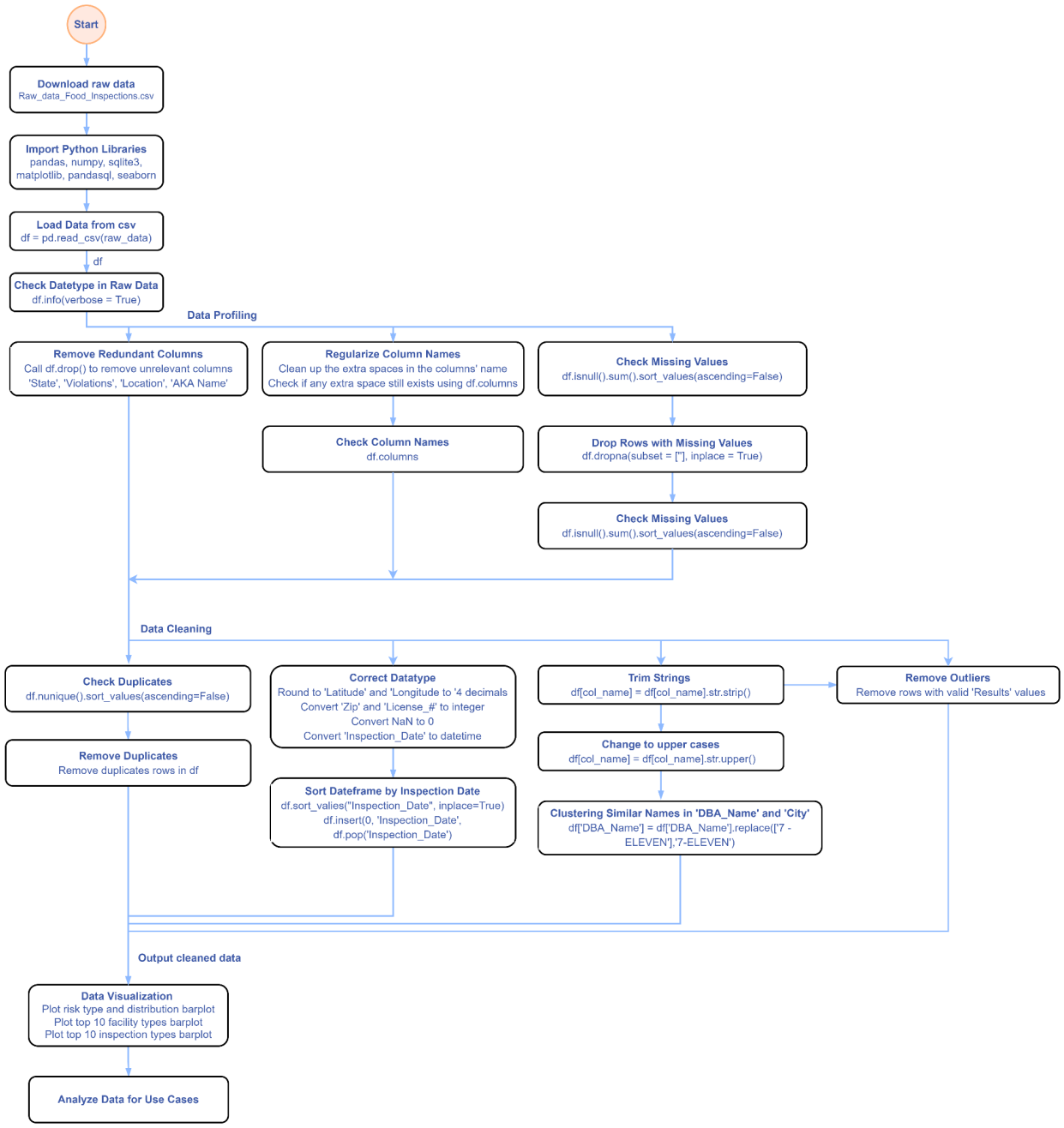
4. Create a workflow model

4.1 A visual representation of the overall workflow

In this project, we create the workflow model by using online workflow tool Nutstore (jianguoyun.com).


Start
Download raw data
Raw_data_Food_Inspections.csv
Import Python Libraries
Load andd Understand Dataset
Data Profiling
Remove redundant columns
Regularize column names
Clean missing values
Data Cleaning
Check duplicates
Correct datatype
Trim strings
Remove outliers
Data Visualization and Explain Use Cases

4.2 A detailed representation of the inner data cleaning workflow



5. Conclusions & Summary

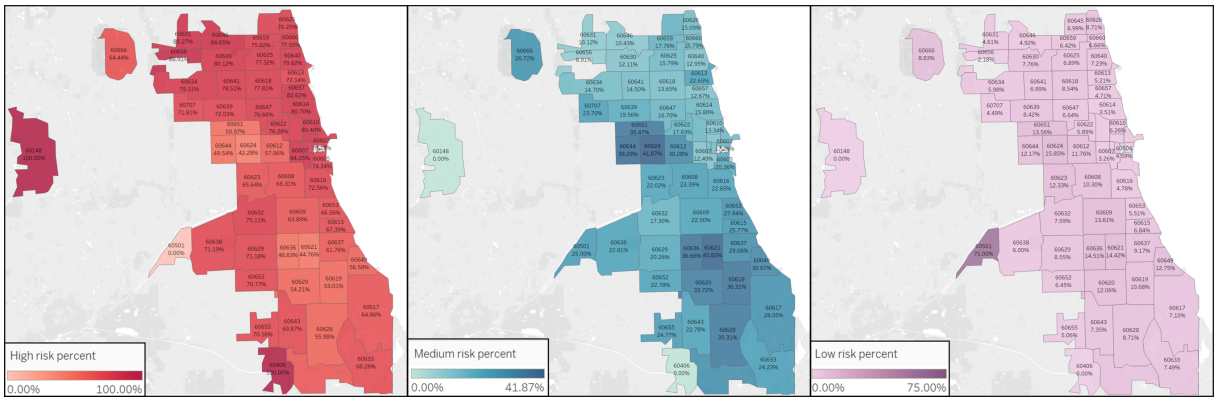
5.1 Summary and conclusions of the project, including the lessons learned

In the Phase-I report, an assumption indicating that the facility's types and locations may have an impact on the inspection results has been put forward. Nevertheless, before cleaning the Chicago_Food_Inspection dataset, it is difficult to draw the conclusion that the facility's types/locations have a positive/negative influence on the inspection results, and to what extent the facility's types/locations can affect the inspection results. After performing the data cleaning using Python and Tableau, it provides clearer insights to better understand use case U1.

For use case U1, we intend to leverage the cleaned data to analyze the relationship between the facility's types, locations and the inspection results. By deep diving into this use case, we hope to achieve the goals that assist the consumers to select the least risky food establishments in Chicago, and help the investor choose the least risky locations in Chicago.

5.1.1 Risk Analysis in different Zip Codes

Graph 1 - Choropleth map of risk distributions by Zip Code



By leveraging the cleaned dataset, graph 1 "Choropleth map of risk distributions by Zip Code" was generated in Tableau. The three graphs show the distribution of high, medium and low risk level by percentage of each risk level by Zip code in the map. Each area is labeled by Zip code and risk percentage. The proportion of high risk is more than 50% in most areas, and medium risk is about 10% ~ 20% correspondingly. The proportion of low

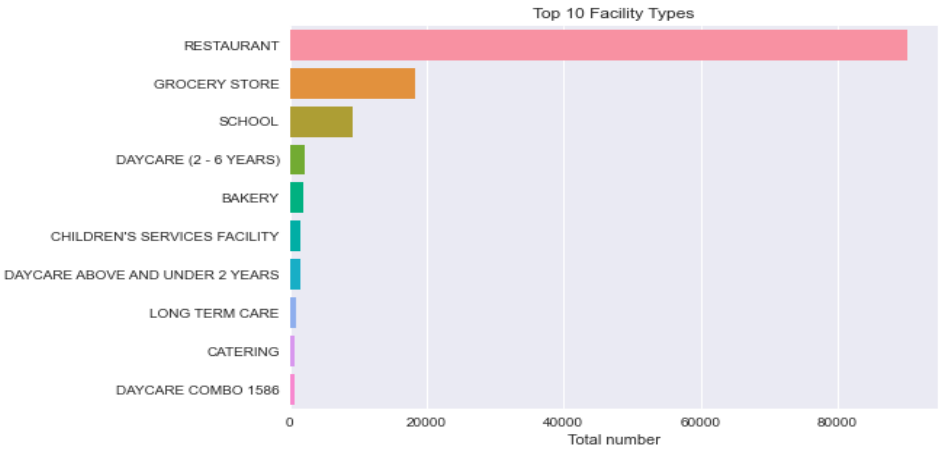
risk is less than 10% in most areas. Only in the 60501 area, the low risk percentage reaches 75% which represents the safest inspection area.

As a consequence, the U1 assumption has been proved that the facility's locations will have an impact on the inspection results, and based on the observations, if the customers visit the food establishments in the 60501 area, and the investors invest in the food facilities in the 60510 area, they are able to lower down the possibilities to be exposed to riskier food establishments in Chicago.

5.1.2 Risk Analysis for different food establishments

By leveraging the cleaned dataset, graph 2 "Top 10 Facility Types" was generated in Python. From the graph 2, it can be easily seen that "Restaurant", "Grocery Store", "School" constitute the majority of food establishments in Chicago.

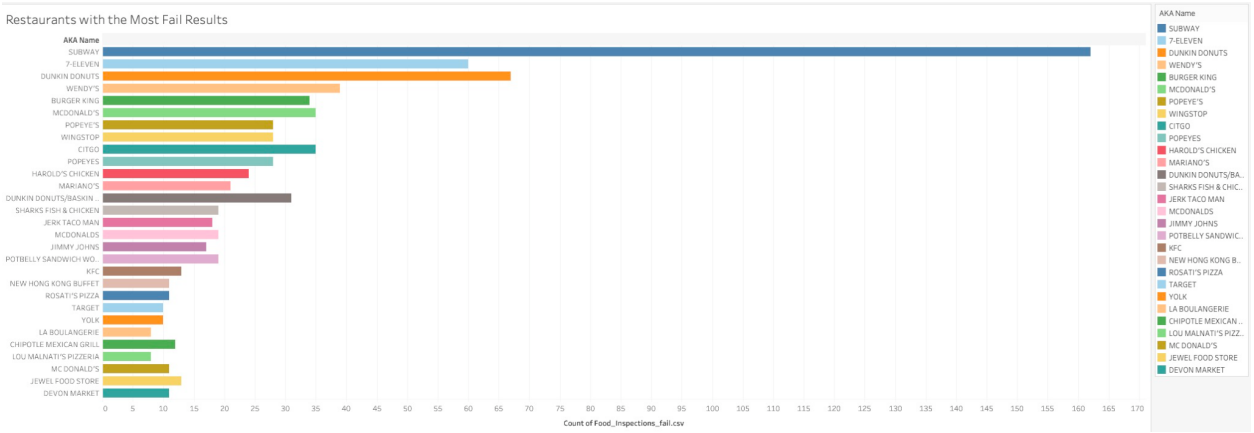
Graph 2 - Top 10 Facility Types



After checking the top 10 facility types, we noticed that facility type "Restaurant" took up a large portion of facility types in Chicago, resulting in the fact that customers are more likely to visit "Restaurant" facility types. Hence, graph 3 "Restaurants with the most fail results" was created via Tableau. By viewing the bar chart presented, we are able to conclude that fast food restaurants like "Subway", "Dunkin Donuts", "7-ELEVEN" are more likely to fail the inspections and are the riskiest facility types.

Consequently, if the consumers want to seek the facility establishments that are not frequently failed the inspections, they should attempt not to select fast food restaurants.

Graph 3 - Restaurants with the most fail results



5.1.3 Conclusions of lessons learned

There is no doubt that data cleaning plays a critical role and consists of an incredible part when we are eager to identify a pattern or draw a conclusion from the massive data. In reality, most of the datasets are not cleaned and require the data scientists or engineers to devote a lot of time and human resources to cleaning the datasets. If the dataset is not cleaned properly, it is highly likely that we are not capable of drawing the correct conclusions, and even worse, it may lead to an inaccurate result.

From this project, we have learned that for some use cases, we may be able to draw the conclusions without cleaning the dataset, on the contrary, for some use cases, we will never be able to draw the conclusions even if we apply ourselves to cleaning and exploring the dataset. However, in most cases, the dataset needs to be cleaned and it is cleaning the dataset in a proper method that helps us come to the correct statements.

In this project, we mainly used Python to clean the dataset, used Tableau to visualize the cleaned dataset and used Nutstore to create workflows due to time constraints. There are all varieties of magnificent technological tools that work outstandingly when cleaning the dataset like SQLite, OpenRefine, Yes WorkFlow etc. For instance, for the Chicago_Food_Inspection dataset, we have identified a great number of typo errors in

the string type values, however, it is difficult to identify all the typo errors and cluster them as a group. If we have time to improve our project in the future, we hope to adapt SQLite and OpenRefine to assist us improve the data quality.

5.2 Summary of the contributions of each team member

Wei Dai:

- Clean raw data in Python
- Participate in drafting Phase 1 and Phase 2 reports
- Perform initial assessment of the dataset
- Document data cleaning steps and screenshots

Xiaoyu Wen:

- Clean raw data in Python and organize Python script
- Participate in drafting Phase 1 and Phase 2 reports
- Data visualization in Tableau
- Create overall and detailed inner workflows
- Organize the supplementary materials

Felicia Liu

- Clean raw data in Python
- Participate in drafting Phase 1 and Phase 2 reports
- Perform exploratory data analysis and visualization in Python and Tableau
- Summarize the data quality issues and final conclusions
- Document dataset assessment and cleaning step statements

6. Appendix - Supplementary materials

All the supplementary materials are provided via the Zip file.