

# CS513-DataCleaning: Project Proposal

Amrutha Gujjar  
agujjar2@illinois.edu

Dixon Liang  
dixonl2@illinois.edu

Megan Masanz  
mjneuman@illinois.edu

## 1. FINAL REPORT

Points received for initial Phase-1 submission : 98/100 Our team chooses the following Phase-1 option:(A) No change to Phase-1 report

## 2. PHASE I ANSWERS

### 2.1 Data Set

The data set that we have chosen to use is the AirBnb data set. The AirBnb set is a dirty data set of listings from the Chicago area that is provided as a project reference data set.<sup>1</sup>

### 2.2 Target Use Case

Let's explore major and minor target use-cases for the chosen AirBNB dataset. The main use-case where data-cleaning will be both (1) necessary, and (2) sufficient such that resulting cleaned dataset D' is fit for-for-purpose in answering a question. Furthermore, there are minor use-cases where the dataset is already sufficient for a given purpose, or where the dataset will never be sufficient for a given purpose.

- Main Use Case ( $U_1$ ):

**What is the average price of an AirBNB in each neighborhood in Chicago?**

With the dataset in the current uncleaned state, we can attempt to answer this question by constructing a query that finds unique neighborhoods, all associated listings in that neighborhood, and the average price for that grouping. While the structure of the data gives us the capability to attempt to answer this question, the inconsistencies in the neighborhood name field (misspellings, typos, slight variations in phrasing) largely increase the true number of unique neighborhoods in the first place, thereby throwing off our average computations. Data cleaning through a tool like OpenRefine can allow us to group together the unique neighborhood names that are truly referring to the same

neighborhood, thereby allowing us to compute a more accurate result for the client asking this question.

- Minor Use Case 1 ( $U_0$ ): Requires zero data cleaning. D is good enough as it is.

**How many AirBNB listings exist in Chicago?**

We can answer this question without any data cleaning because each unique ID corresponds to a single AirBNB listing, and we simply want an understanding of the volume of total listings regardless of the other features of the data, we do not need to do any cleaning to answer the question. A select count(\*) from airbnb.csv alone would yield the correct answer for the given question. Any refinement of the data would not have an effect on the final result to the given question.

- Minor Use Case 2 ( $U_2$ ): Dataset D is never good enough. No data cleaning is enough.

**How have the AirBNB prices in Chicago changed over time?**

Even though there is information on annual availability for the given data, there is no column corresponding to time-series data, so we cannot come up with a conclusion on changes in average pricing over time simply because of the schema of the data. Additionally, any data cleaning we would do in terms of removing logical inconsistencies through Datalog, or fixing syntactic errors through regex, would not yield any particular benefit when it comes to answer a time series question. We would need to join it with another table, or find a different dataset for this minor use-case.

### 2.3 Dataset Description

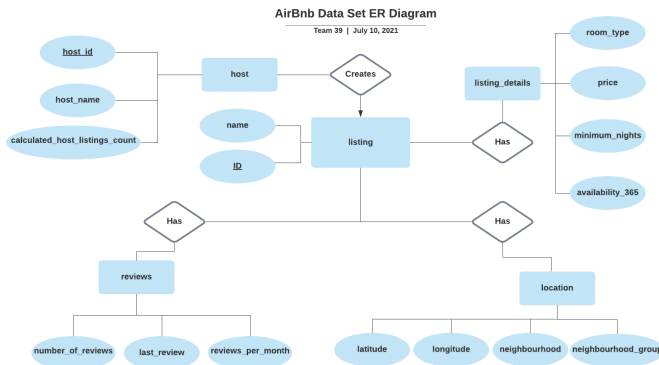
The data set we have chosen to use is taken directly from AirBnb listings. The period of time for the listings is for the year 2018, specifically for the Chicago area. Below we have arranged the columns into relational categories. There are 16 columns in total, and 7594 total listings.

Each rental is identified by the name and id number. Hosts are identified by an id, their names, and how many listings they have active. The rental location is given by the neighborhood, latitude, and longitude. Details of the rentals include the room type and minimum nights. Finally, information about reviews are provided in three columns. The data is not cleaned and data within columns are often not

---

1. Lan Li, "CS513-Data Cleaning Reference Datasets," 2021, <https://uofi.app.box.com/s/y7123ku1dpzd4m3126sqvpa2agvv08hv/file/82526727413>.

Figure 1: ERD Chart



consistent. We will discuss quality issues in the following section.

Figure 1. ERD Chart provides a high level view of the data set that will be cleansed through this project ensuring automation, scaling, abstraction and provenance are applied.

## 2.4 Dataset Quality

The use case analysis will benefit from applying quality controls to the data set. Nearly each column in the data set provides a set of unique challenges. Several columns include special characters which should be removed, in addition several columns would benefit from trimming of start and ending spaces, as well as removal of consecutive spacing.

## 2.5 Dataset Columns Quality Known Issues

The information provided below highlights obvious data quality issues that are required to be resolved during the data cleansing process. The rules provided below were broken within the data set, and are provided to illustrate the need for data cleansing to ensure a valuable data set is established for analysis.

**name** - requires standardization for the use case. It appears there are parenthesis around each name, which itself is actually not a data quality issue, but does make it more difficult to read. These can be removed from the value given they are not providing value and reduce readability of the value. Some rows have the name in single quotes, some have it in double quotes which should be standardized as well. In addition to benefiting from space trimming, separating values based on capitalization would also be ideal, as an example: HomeChicago should be changed to Home Chicago. The name and the id's don't match up. There are 7594 unique ids, however there are only 7524 unique names. This indicates that the id column does not pertain to the name column which indicates there should not be a uniqueness constraint placed on the name column, however, there should be a uniqueness constraint on a listing name for a given host to ensure integrity in the data set.

**host\_id** - also does not match up with the column **host\_name** as there are 4647 ids and only 2219 unique **host\_name** values. This indicates there is not a uniqueness constrained placed on names, which in itself is not a data quality issue, as 2 people can certainly have the same name, but we do

need to ensure that the `host_id` aligns to the `host_name` accordingly.

**host\_name** - includes first name only, first and last names, as well as multiple people's names. There is also an example of an address appearing in the name, which would benefit from a default value as an example being - 'Not Available'. This could also be applied in the case when '(Email hidden by Airbnb)' was entered as the name, and an additional column could provide if an email was hidden by Airbnb.

**neighbourhood\_group** - has no valuable information in the data set.

**neighbourhood** - has values that without data cleansing would not be grouped together due to typos, as well as missing spaces. (An example of this would be the neighborhood West Garfield Park which applying basic data cleansing described previously would resolve.

Another example of this would be with the location OHare is represented several different ways.

In addition there are values L??p that perhaps were caused by exporting and importing encoding issues.

**latitude** - as well as the column **longitude** have different levels of precision.

**minimum\_nights** - should not have a value of 365. That would mean a person would be required to rent for an entire year, and as an example, given the name: (The Whimsical Wrigley House - Minimum 30-Day Stays), it is a good indication that this value was entered in incorrectly given the 30-Day Stays provided in the title.

**price** - should not have a value of 0. In addition, basic checks should be applied, given the example below.

Another example of an obvious data quality issue is when **minimum\_nights** is greater than the number of available nights found in the column: **availability\_365**

**availability\_365** should not have an availability of 0, unless it is an inactive listing. This should be identified and corrected.

To provide additional context, when looking at a host Ms. Vee, we can see that many of her properties have the same amenities, however due to inconsistent naming, they are difficult to group together.

We can see that the pricing on these listings does not reflect what is seen in the description. The price of these listings as shown in Figure 2, range from 20-25, but many of these include a price of \$350 in the name itself. With proper formatting, it should be clear to consumers of the data set what makes the listings unique. We can see that the listings are in fact unique given the latitude and longitude values are different for each of the listings. Review of the latitude and longitude values online does appear that they are in fact unique address locations as depicted in Figure 3 & Figure

4. when leveraging an online resource.<sup>2</sup>

Figure 2: Ms. Vee listings

id	name	host_id	host_name	neighbourhood_id	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	availability_365
14855818 (\$350 PER MO. PER PERSON -BUNK BED / FREE WIFI & )	44358452 Ms. Vee	South Chic	41.73813	-87.5642	Shared room	20	32	3	9/3/2017	0.12	8	364			
14856790 (BUNK BED, FREE WIFI, KITCHEN & More...)	44358452 Ms. Vee	South Chic	41.73964	-87.564	Shared room	20	32	6	8/11/2018	0.23	8	364			
14857073 (\$350 PER MON. PER PERSON -BUNK BED, FREE WIFI & )	44358452 Ms. Vee	South Chic	41.73943	-87.5643	Shared room	20	32	5	8/15/2018	0.19	8	364			
14857205 (BUNK BED, FREE WIFI, KITCHEN and more...)	44358452 Ms. Vee	South Chic	41.7396	-87.5643	Shared room	20	32	1	9/24/2016	0.04	8	364			
14857151 (\$350 PER MONTH, PER PERSON -BUNK BED, FREE WIFI & )	44358452 Ms. Vee	South Chic	41.73967	-87.5639	Shared room	20	5	3	8/11/2018	0.11	8	365			
14857431 (BUNK BED, KITCHEN, FREE WIFI & More...)	44358452 Ms. Vee	South Chic	41.73976	-87.5653	Shared room	20	32	1	10/4/2018	0.7	8	364			
14857129 (\$350 PER MONTH, PER PERSON -BUNK BED, FREE WIFI & )	44358452 Ms. Vee	South Chic	41.73936	-87.5653	Shared room	20	32	2	8/11/2018	0.08	8	365			
19100264 (Spacious Private Bedroom, FREE WIFI, & more...)	44358452 Ms. Vee	South Chic	41.73802	-87.5642	Private room	25	32	1	8/4/2018	0.29	8	365			

Figure 3: Latitude & Longitude Check

To get an address from latitude and longitude enter them in the fields and click find.

LATITUDE: 41.73812771 LONGITUDE: -87.56415128 **Find**

Output Location:  
3825 S. Essex Ave., South Chicago, Chicago, Cook County, Illinois, 60617, USA

Output Latitude, Longitude:  
41.738128° -87.564151°

Figure 4: Latitude & Longitude Check

To get an address from latitude and longitude enter them in the fields and click find.

LATITUDE: 41.7394294 LONGITUDE: -87.5643077 **Find**

Output Location:  
8549 S Phillips Ave. South Chicago, Chicago, Cook County, Illinois, 60617, USA

Output Latitude, Longitude:  
41.739429° -87.564308°

### 3. IMPLEMENTATION PLAN

Our implementation plan is as follows...

1. Data set evaluation and evidence gathering with Megan Masanz as primary. This step will provide evaluation for the need data cleansing as well as evidence.

2. Leverage OpenRefine for basic data cleansing, with Amrutha Gujjar as primary. The output of this step will be the json encapsulating the changes required.<sup>3</sup>

3. Leverage Python for additional data cleansing with Megan Masanz as primary. The output of this step will be the python script required to provide improved data quality.

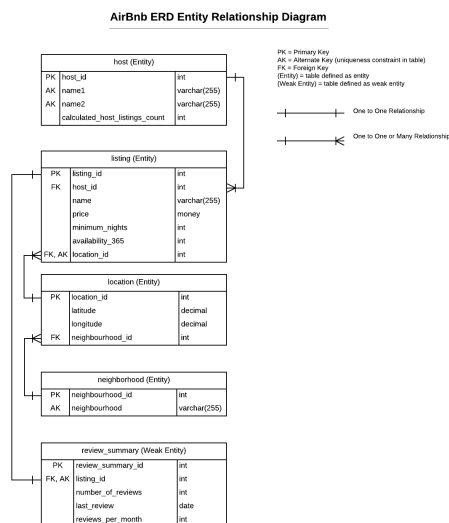
4. Leverage SQL for integrity constraints with Dixon Liang as primary. The output of this step will be delta SQL scripts showcasing the changes in the data set before and after data cleansing.

Figure 5. ERD SQL Table showcases how we plan to move from our entity model into a SQL implementation to ensure integrity constraints can be applied to the data set. This is an initial assessment and will be updated based on findings during our investigation.

5. Encapsulate work flow leveraging YesWorkflow with Dixon Liang as Primary.<sup>4</sup>

6. Document results with Megan Masanz as primary. The output of this step will be delta queries showcasing the before and after results.

Figure 5: ERD SQL Implementation



2. "Find Latitude and Longitude," 2021, <https://www.findlatitudeandlongitude.com/find-address-from-latitude-and-longitude/>.

3. David Huynh, "OpenRefine," 2021, <https://openrefine.org/>.

4. Timothy McPhillips Bertram Ludaescher Tyler Kolisnik, "https://github.com/yesworkflow-org/," 2021,

## 4. PHASE II - DATA CLEANING PERFORMED

In order to facilitate our use case, we found that leveraging OpenRefine, Python, and SQL Server technologies would best suite establishing a clean data set.

### 4.1 Step 1: OpenRefine Data Cleansing performed

Within the tool OpenRefine, the data was cleansed column by column. This was not required specifically to address the answer to our use case, but was required to support our efforts to establish a data set that would be suitable for ingesting into SQL Server where our integrity checks were to be established on the data set.

OpenRefine: Inconsistent naming, syntax, duplication, and typos were nearly all resolved with series of OpenRefine operations. The result was a nearly syntactically clean data set ready for schematic fixes.

With OpenRefine, we were able to swiftly improve the quality of the initial AirBNB data set.<sup>5</sup> The steps performed are outlined in the following subsection. For starters, we established text fields (name, neighborhood, etc), and numerical fields (lat, long, price, etc). Then we did a normalization of the cases into Title Casing. After that, we used text facets to cluster variations of spellings referring to the same neighborhood. We used clustering and n-gram specifications to find examples of these instances. For the host name, we wanted to normalize the case where either 1 or 2 people were associated with the host name. Rather than having multi-host situations appended together with keyword 'and', we used the split cell functionality within OpenRefine, along with regex-matching the word 'and' to split multi-host entries into different columns for names. The text facets were also useful for removing problematic syntax in the name field. Many entries had unusual use of parentheses and special characters that interfered with both readability and consistency of the data. By using GREL, we could replace/remove these unnecessary characters. As a final step, we also trimmed leading and ending whitespace, and removed consecutive whitespaces. For details on the steps performed see the following section.

These steps are also identified in YesWorkflow Figure 20. OpenRefine Workflow

## 4.2 OpenRefine Steps Performed

### 4.2.1 Steps on id column

Step 1. Convert id column into an integer, which was required to ensure future processing of the data would treat the column as a numeric value.

### 4.2.2 Steps on name column

Transformations on the name column would appear at first glance not be required to answer our use case. However, the transformations applied, enabled successful ingestion into SQL Server, and would support future use cases. With the data in its initial format, it is difficult to read. During our analysis this column was determined to be very problematic, so it was determined that we should apply best practices to this column.

Step 2. Transform the name column into a string, to support further data transformations as a required step.

5. Li, "CS513-Data Cleaning Reference Datasets."

Step 3. Trim leading and trailing spaces from column.  
Step 4. Remove consecutive spaces from column.  
Step 5. Removing outer parenthesis from the data, if a value started and ended with parenthesis, they were providing no value to the data itself, and thus were removed.  
Step 6. After removing the parenthesis, re-trimming.  
Step 7. Removing \* character from column.  
Step 8. Removing special characters from column.  
Step 9. Removing consecutive spaces from column.  
Step 10. Trimming spaces from column.  
Step 11. Convert to Title Case for consistency.  
Step 12. Removing '/' character and replacing with a space.  
Step 13. Removing consecutive spaces from column.  
Step 14. Trimming spaces from column.  
Step 15. Converting to Title Case. After removing a missed special character, and re-trimming the text, it was determined some values were inconsistent and the Title Case needed to be reapplied.

### 4.2.3 Steps on host\_id column

Converting the **host\_id** column was not required to answer the specific question for the use case, however to ensure proper ingestion into SQL Server, we needed to ensure this column was numeric as it would be a primary key for the host table as shown in the ERD Diagram Figure 6.

Step 16. Convert host\_id to numeric to support the requirement of it representing the primary key in the host table.

### 4.2.4 Steps on host\_name

Step 17. Remove consecutive spaces.

Step 18. Convert host\_name from one column into two columns. Given many listings had more than a single host, it was determined to make this clear we converted it from a single to two columns. This was not required to support the specific use case, but we took a holistic approach to cleansing the data so this extra step was completed.

### 4.2.5 Steps on neighbourhood\_group

Step 19. Removed this column. This column did not provide value in its current state so it was removed from the data set. This was not required to support our use case, but keeping consistent with providing a cleansed data set, this step was completed.

### 4.2.6 Steps on neighbourhood

Modifications made to this column were made to support the intended use case and are thus required. The primary objective with this column is to clean the values and provide a consistent value for each neighborhood to ensure as the data is loaded into SQL server would not be seen as distinct values and would thus share the same primary key in the neighbourhood table as shown in Figure 6.

Step 20. Consecutive spacing was removed from the column.  
Step 21. Spaces were trimmed from the column.

Step 22-27. Leveraging OpenRefine facets were created so neighbourhoods with varying names would have identical values. The value of L??p was replaced with Loop, the various values for O'hare were all given a value of O'hare so they would not be seen as individual neighborhoods. West Garfield Park values were also cleaned and given a consistent value.

Step 28. Consecutive spaces were removed from the column.  
Step 29. Spaces were trimmed from the column.

Step 30. Title case was applied to the various neighbourhood values. This step was not required to answer the specific use case, however it results in a consistent data set.

#### 4.2.7 Steps on Latitude

Step 31. This value was converted to a numeric value. This was not a required step to answer the questions posed in the use case, but for consistent data processing was applied.

#### 4.2.8 Steps on Longitude

Step 32. This value was converted to a numeric value. This was not a required step to answer the questions posed in the use case, but for consistent data processing was applied.

#### 4.2.9 Steps on room\_type

Step 33. Remove consecutive spaces. This step was not required given there were no changes to the data set, however, if there were consecutive spaces, this would have created duplicate room\_type values for a the same type, so thus was applied.

Step 34. Trim spaces. This step was not required given there were no changes to the data set, however, if there were consecutive spaces, this would have created duplicate room\_type values for a the same type, so thus was applied.

#### 4.2.10 Steps on price

Step 35. The value was converted to a numeric. This value was converted to a numeric value. This was not a required step to answer the questions posed in the use case given the analysis would be off of the data set from SQL Server, but for consistent data processing was applied.

#### 4.2.11 Steps on num\_nights

Step 36. The value was converted to a numeric. This value was converted to a numeric value. This was not a required step to answer the questions posed in the use case given the analysis would be off of the data set from SQL Server, but for consistent data processing was applied.

#### 4.2.12 Steps on number\_of\_reviews

Step 37. The value was converted to a numeric. This value was converted to a numeric value. This was not a required step to answer the questions posed in the use case given the analysis would be off of the data set from SQL Server, but for consistent data processing was applied.

#### 4.2.13 Steps on last\_review

The steps applied here were not required to support the use case specifically, but will provide usable date time format for querying to support answering additional questions. As the approach here was holistic, the step was included.

Step 38. Converted the date time column to leverage the format yyyy-MM-dd

#### 4.2.14 Steps on reviews\_per\_month

Step 39. The value was converted to a numeric. This value was converted to a numeric value. This was not a required step to answer the questions posed in the use case given the analysis would be off of the data set from SQL Server, but for consistent data processing was applied.

#### 4.2.15 Steps on calculated\_host\_listings

Step 40. The value was converted to a numeric. This value was converted to a numeric value. This was not a required step to answer the questions posed in the use case given the analysis would be off of the data set from SQL Server, but for consistent data processing was applied.

### 4.3 Step 2: Python Processing

OpenRefine resolved many of the data quality issues required to solve our use case, but given the poor data quality that remained in the data set, additional processing was done leveraging Python. This step established flags in the data set to enable the analysis of the data before and after processing occurred. This step enabled quantifying the data quality changes established in the processing of the data. This information is captured in Figure 21: Python Workflow.

T: 02.data-cleansing.ipynb which can be found in the github repo leveraged for this project.<sup>6</sup>. The workflow for this notebook is provided in Figure 21. The output of the OpenRefine step produced a file: AirBNBV2.csv. The file was loaded into a pandas dataframe. A step was provided to drop the column neighbourhood\_group if it exists in the data set. It has already been dropped in the previous step, so this is not required, but for completeness has been incorporated into the notebook. Listings where the minimum\_nights is greater than 300 are marked as true in the quality\_minimum\_nights\_long column. While this is not required to enable analysis, it will allow us to filter out possible base listings. Further, listings with no availability are identified and marked with quality\_availability\_365 set to true. This will also enable the analysis, if we want to look at active listings, we can filter the data set by this identifier.

The data set is further marked with price outliers. The mean and standard deviation of price are calculated, and an upper and lower bound of 3 standard deviations is established. Outliers are marked in the column quality\_price\_outlier are marked as true, this is required to filter out possible outlier listings from impacting the average price by location.

The dataset is further cleaned where the host name is marked as 'Unavailable in the case when the host name has the value of '(Email hidden by Airbnb', as this message provides no value to the consumers of the data. This step is not required, but ensures the wholistic approach to data set cleansing.

In the case when the name column includes a 30 Day stay, the data set is updated to reflect that. This is not required to support this use case, but ensures analysis can be done to support future use cases.

As the data set will be moved into SQL Server, it was desired to remove all non-ASCII characters. Regex was applied to the name column to support this effort. This is not a required step, but ensures future processing of the data by other systems can easily be processed.

An additional step was established to remove special characters that were not identified during the OpenRefine step. While in open refine - special characters were removed from the name column, (excluding an underscore), there were not performed on the host name, so this step ensured consistency across the columns. This step was not required to support our specific use case, but does ensure a quality data set is provided.

6. [https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data\\_Processing/02.data-cleansing.ipynb](https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data_Processing/02.data-cleansing.ipynb)

After cleansing out the non-ASCII characters, the data set was further cleaned to remove consecutive spaces, followed by trimming the data. This was again not required for our specific use case, but ensures a quality data set is provided through this processed.

Empty names are populated with the value of 'AirBNBListing'. This step is also not required, but ensures a quality data set can be evaluated to support future use cases.

After the data has been cleansed, we establish a column quality\_num\_missing that provides an indication for a given listing the fields that are not properly populated, this is not required, but again is helpful during the data set analysis. We can easily filter the data set based on the number of quality attributes that are missing.

The next step was to rename the columns host\_name 1 and host\_name 2. This was not a required step, however when dealing with sql server, it is best not to include spaces in column names, so they were removed.

The next step was to populate the column reviews\_per\_month with 0's were their were null values. This was not a required step, however a best practice to ensure quantity values could easily be inserted into the sql server database.

The final step in our python cleansing process was to write out the data set to a file. This was not technically a required step, but made separating out steps in the work flow easy to isolate. The file output is: 02.airbnb\_step2.complete.csv<sup>7</sup>.

## 4.4 SQL Server Python Notebook

The Python notebook in the previous section resolved the data quality issues and prepared the data for entry into our SQL Server database. Leveraging primary keys, foreign keys and uniqueness constraints the database was loaded with quality information with key indicators identifying possible data quality issues leveraging the notebook from a base table of the cleansed data. The ERD Diagram provided in Figure 6. shows the design leveraged to enable integrity constraints on the data set.

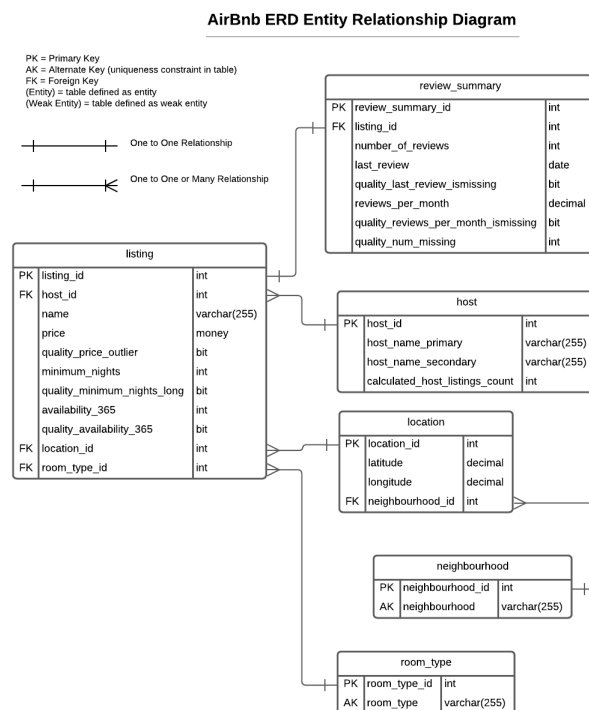
03.TableGenerationAndPopulation.ipynb. Instead of taking the approach of deleting rows that would impact the results required to answer our key question represented in our use case, we flagged the data with quality markers. This resulted in a querable dataset that would allow analysis of a data set before and after removing results based on possible data errors. The database design is include in Figure 6 identifying uniqueness constraints, as well as primary and foreign keys. The SQL queries leveraged within the notebook have been provided in the Queries.sql file, which can also be found in the github repository.<sup>8</sup>

The database was designed to impose integrity constraints to ensure data was represented appropriately to consumers of the data set. SQL server has a limitation on seed scripts to limit ingestion to 1,000 records at a time. We could have broken up our seed scripts and loaded the data manually, but instead leverage the python package PyOdbc to move the data. Before loading any data, we ran a SQL statement to drop existing tables. While our leveraging of primary and foreign keys would result in quality data being inserted

7. [https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data\\_through\\_Process\\_as\\_csv/airbnb\\_step2\\_complete.csv](https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data_through_Process_as_csv/airbnb_step2_complete.csv)

8. <https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/etc/Table.Populate/Queries.sql>

Figure 6: ERD SQL Implementation



into the tables, in general it is a best practice if you are re-inserting the same data over and over to include dropping of tables as a quality check. The workflow of this step is identified in Figure 22.

The data set as a file was loaded into a pandas data frame and was inserted into a staging table call airbnb\_base row by row. The staging table had no primary key, no uniqueness constraints. This meant the data could easily be loaded. We did explicitly covert longitude, latitude, and reviews\_per\_month values to decimals. This was important to ensure proper loading into future tables, as SQL server would have an issue converting an nvarchar value into a decimal. Once the data was loaded into the staging table, we were able to access the data set for loading into tables that had integrity constraints applied. The following steps were required to enable our use case.

The first table that was populated was the host table. It relied on no other data as foreign keys so loading this table was a logical first step. We were able to select distinct values from the staging table and insert those into the staging table. A primary key of host\_id was applied. This ensured if there were duplicate host ids, the insertion into this table would fail. Given the select statement included selecting distinct values, the insert was completed successfully.

The second table populated was the neighbourhood table. What makes the neighbourhood entity unique is a string value, so a primary key was created for each unique value of neighbourhood from the staging table. A uniqueness constraint was placed on the column neighbourhood to ensure if data was inserted that was not unique, the insert would fail. Given we selected distinct neighbourhood values from

the staging table, the insert succeeded.

The next table populated was the location table. There was no natural primary key, so one was created, and the id column from the neighbourhood table was leveraged as a foreign key. Leveraging this value as a foreign key meant that unless that key was in the neighbourhood table, it would not be inserting into the table. During the insertion, a left join between the base table and the neighborhood table meant that for each row in the base table, we were able to retrieve the neighborhood id for inserting into the location table.

The next step that was applied was creation of the room\_type table. A uniqueness constraint was applied to the room\_type. Selecting unique room\_type values from the base table for insertion into the room\_type ensured distinct values for room\_type were inserted into the table.

The next step that was created was the listing table. The id from the base table was leveraged as the primary key for the table. The location\_id was leveraged as a foreign key to ensure integrity constraints were applied to the location attributes of the listing. Further, a foreign key of room\_type\_id was applied to tie the listing to a specific room type.

During the insertion of the data into the table, a left join on the base table information, joining on the location table base on the longitude and latitude values. Given the id was leveraged as primary key - if duplicate values were found, the insertion would have failed due to the integrity constraints, and thus ensuring appropriate data was inserted. In addition an inner join based on room\_type was applied to tie a listing to a specified room type.

The final table populated was the review\_summary table. A primary key was created, and a foreign key constraint was created leveraging the textlisting\_id. While inserting into the review\_summary table distinct values were pulled from the staging table to be inserted into the review\_summary table.

SQL Tables were leveraged to ensure the appropriate integrity constraints were applied to the data set, but consumption from raw tables can be a challenge to consumed. To enable analysis of the data set, a view was finally created to pull the data together. The view joined together the tables, and pyodbc was leveraged to pull the information out of the resulting table to place into a pandas data frame to output into a csv file. Typically reporting would be done leveraging the data within SQL Server, but for completeness a csv file was generated: finaldataset.csv to demonstrate the valuable dataset generated through this process.

## 5. PHASE II - DOCUMENT DATA QUALITY CHANGES

### 5.1 Quantify the results of your efforts

The table below highlights the changes made for each column. As shown in Table 1, the column host\_name was broken into 2 columns resulting in a change for every row. The column neighbourhood\_group was removed, resulting in a change for every row. The column neighbourhood was transformed and cleansed resulting in significant changes to standardize. The columns latitude, longitude, room\_type and price had no changes. The column minimum\_nights had a 1 cell change. There was a single row that mentioned a 30 day stay where the minimum\_nights was updated to indicate 30 based on the description. The column

number\_reviews had no changes. The column last\_review had significant changes due to date time formatting. For the column reviews\_per\_month, null values were converted to 0 given a null value would indicate a 0, thus resulting in the changes.

Table 1: Data Set Changes of 7594 Rows

Column Name	Cell Change Count
id	0
host_id	0
host_name	7594
neighbourhood_group	7594
neighbourhood	1224
latitude	0
longitude	0
room_type	0
price	0
minimum_nights	1
number_of_reviews	0
last_review	6699
reviews_per_month	896
calculated_host_listings_count	0
availability_365	0

While values were changed, rows were not dropped. This ensures lineage to the initial data set. Rows were flagged based on quality concerns found within the data set as shown in Table 2. When a listing required more than 300 nights, the flag

quality\_minimum\_nights\_long was set to true. This was an arbitrary number and can be refined based on customer needs. Rows were also flagged if the standard deviation went out by 3 times the standard deviation in the column quality\_price\_outlier. Listings that did not have a last review data were also flagged with quality\_last\_review\_ismissing. Rows missing a

quality\_reviews\_per\_month\_ismissing were also flagged. This allows for analysis to be based on the either the raw data, or filtered based on the selected criteria giving fine grained control over answering the use case with quality data based on the customer's needs.

Table 2: Quality Flagged Row Counts

Column Name	Cell Change Count
quality_minimum_nights_long	10
quality_price_outlier	70
quality_last_review_ismissing	895
quality_reviews_per_month_ismissing	896

### 5.2 Demonstrate Data Quality Improvements

To illustrate data quality issues, the raw data set was loaded into a SQL table airbnb\_raw to demonstrate the valuable information handled before and after data quality improvements were made. The sql provided in this section has also been included in the Queries.sql file, which can also be found in the github repository.<sup>9</sup>

9. <https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/etc/Table.Populate/Queries.sql>

Figure 7 shows for a given host\_id, 997 records had a single host\_id associated with multiple records with unique attributes from the raw table. After placing the data into a host table and using the host\_id as a foreign key in the listing table, no records were found. This is due to the fact that in the host table, the host\_id is only found 1 time per host, as it is primary key for that table.

Figure 7: IC Violations on raw table for host



Figure 8: IC No Violations on listing table for host\_id table

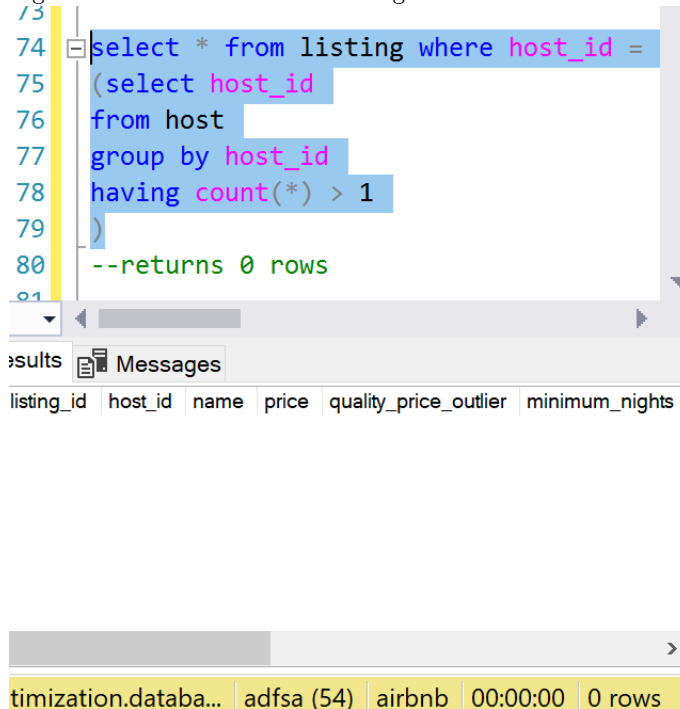


Figure 9 shows the IC violation for the column neighbourhood, showing for a given neighbourhood value, unique attributes were associated with it in the raw table. while Figure 10 demonstrates how the neighbourhood table only has one record per neighbourhood

Figure 9: IC Violations on raw table for neighbourhood

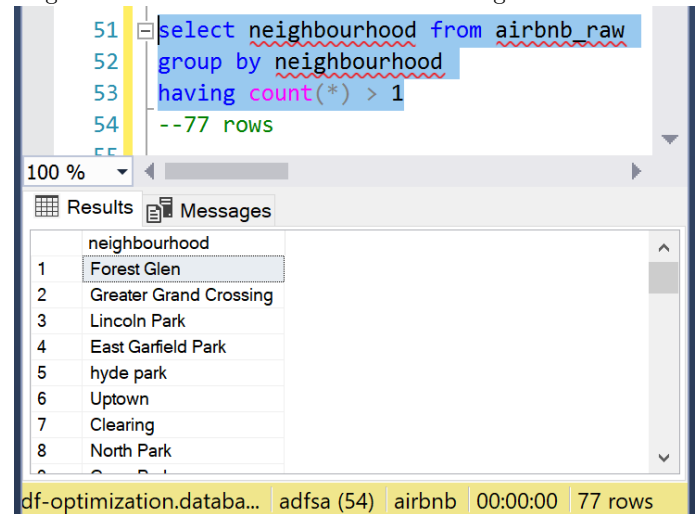


Figure 10: IC No Violations on neighbourhood table

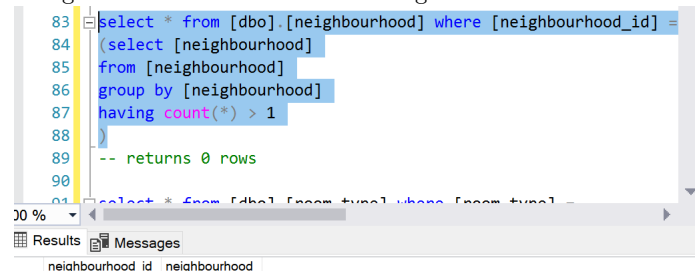


Figure 11: IC Violations on raw table for room\_type

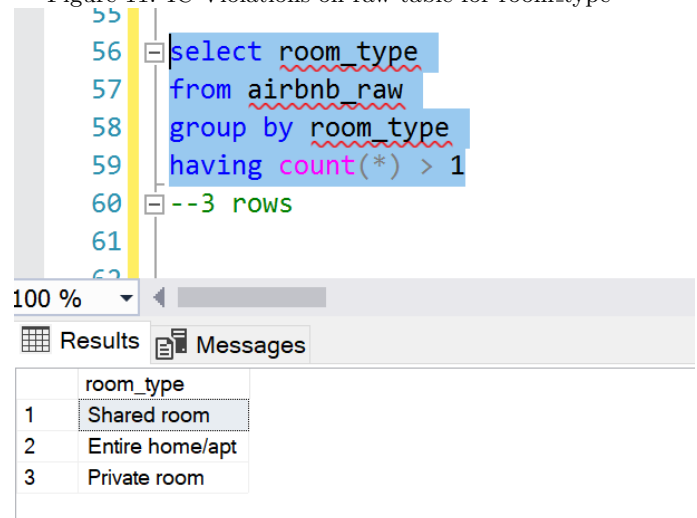


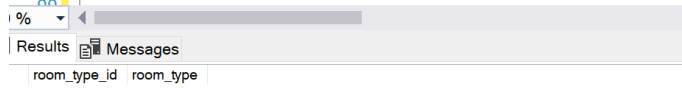


Figure 12: IC No Violations on room\_type table

```

91 select * from [dbo].[room_type] where [room_type] =
92 (
93 select room_type
94 from room_type
95 group by room_type
96 having count(*) > 1
97 )
98 --returns 0 rows

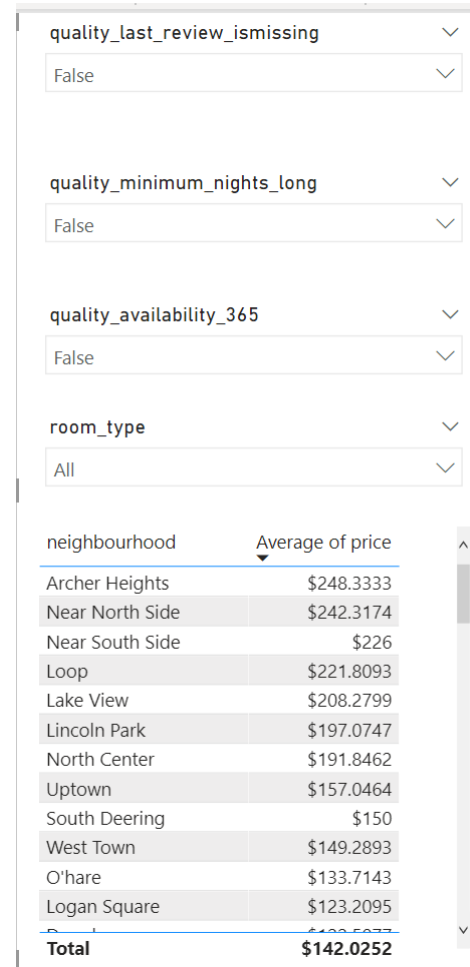
```



In addition for the room type, Figure 11 show cases the IC violation found in the raw data set, while Figure 12 show-cases the cleansed data by following database design strate-gies including a uniqueness constraint on the room\_type as well as generating a primary key room\_id to be leveraged in the listing table as showcased in the ERD diagram Figure 6.

To illustrate the data quality issues identified as part of the project, a power bi dashboard was included to demonstrate how the use case results were directly impacted by the data quality issues identified during the data cleansing process. The power bi dashboard is has been included in the supple-mental materials, and can be referenced in the github repo associated with this project.<sup>10</sup>. The objective of this power bi dashboard is to provide the client with the ability to filter based on the quality attributes captured, when a quality in-dicator is set to true, the power bi dashboard will filter and show attributes were a bad quality was found for a given in-dicator. Users are able to filter the data set based on quality indicators, as well as review the information based on room type.

Figure 13: Power BI Dashboard Filters - cleansed data



quality\_last\_review\_ismissing

quality\_minimum\_nights\_long

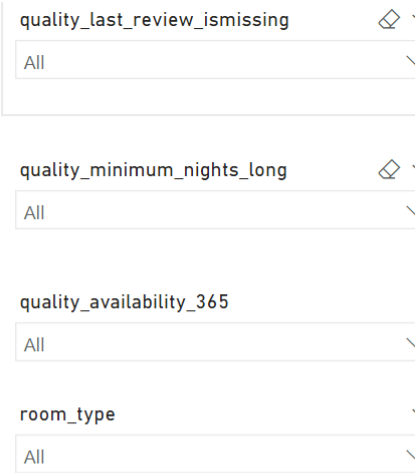
quality\_availability\_365

room\_type

neighbourhood	Average of price
Archer Heights	\$248.3333
Near North Side	\$242.3174
Near South Side	\$226
Loop	\$221.8093
Lake View	\$208.2799
Lincoln Park	\$197.0747
North Center	\$191.8462
Uptown	\$157.0464
South Deering	\$150
West Town	\$149.2893
O'hare	\$133.7143
Logan Square	\$123.2095
<b>Total</b>	<b>\$142.0252</b>

10. [https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data\\_Analysis/powerpoint.pbix](https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data_Analysis/powerpoint.pbix)

Figure 14: Power BI Dashboard Filters - all data



neighbourhood	Average of price
Near North Side	\$275.7951
Archer Heights	\$248.3333
Near South Side	\$222.8169
Loop	\$217.8474
Lincoln Park	\$196.3184
Lake View	\$194.9946
North Center	\$180.8269
Washington Park	\$169.619
West Town	\$162.8977
Uptown	\$150.7705
South Deering	\$150
Forest Glen	\$125.5294
Total	\$147.7406

Figure 15: Average Prices per neighbourhood with data quality issues

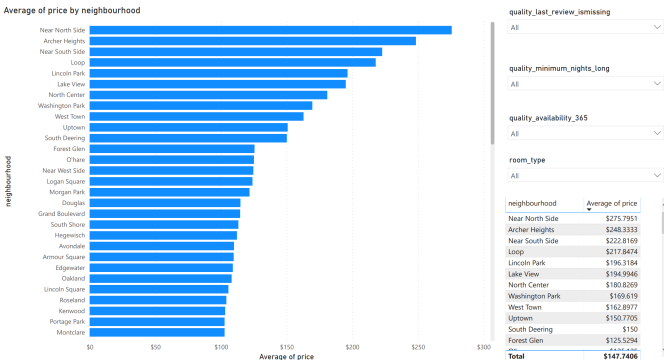
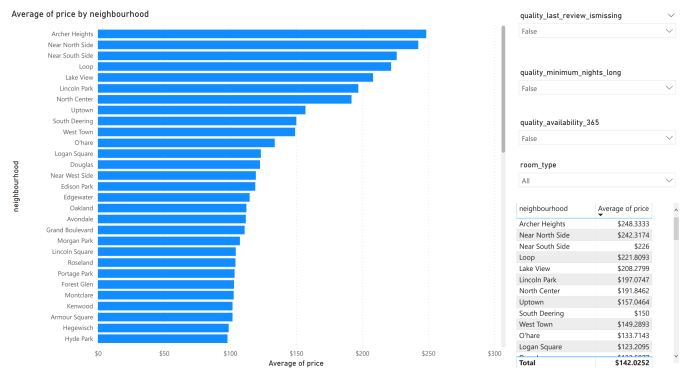


Figure 16: Average Prices per neighbourhood without data quality issues



In addition, a python notebook 04.Dataset\_AnalysisNotebook.ipynb was leveraged to illustrate the key differences found in the data set before and after filtering was applied on the data set based on the quality issues found through the cleansing process.<sup>11</sup> While Figure 13 and 14 are from the Power BI report, Figures 17 and 18 illustrate the differences found in the top 10 most expensive neighbourhoods on average. Both the Power BI dashboard<sup>12</sup>, as well as the python notebook<sup>13</sup> demonstrate that with the data loaded into a SQL Database, we can easily access the information and filter based on the quality columns now associated with the data set.

Figure 17: Top 10 Neighbourhoods by Price - Uncleaned Data

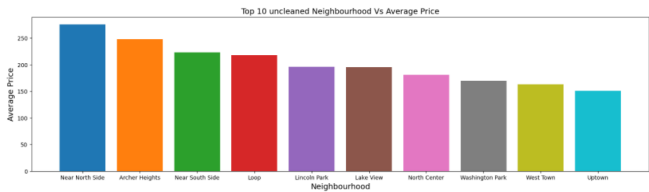
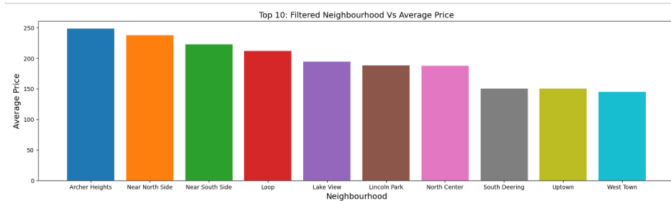


Figure 18: Top 10 Neighbourhoods by Price - Cleaned Data



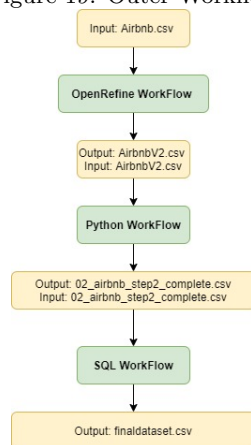
11. [https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data\\_Analysis/04.Dataset\\_AnalysisNotebook.ipynb](https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data_Analysis/04.Dataset_AnalysisNotebook.ipynb)

12. [https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data\\_Analysis/powerpoint.pptx](https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data_Analysis/powerpoint.pptx)

13. [https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data\\_Analysis/04.Dataset\\_AnalysisNotebook.ipynb](https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/Data_Analysis/04.Dataset_AnalysisNotebook.ipynb)

## 6. PHASE II - WORKFLOW

Figure 19: Outer Workflow



We incorporated three main steps to our workflow which resembled a pipeline. From the simple outer workflow, each of these three steps has been further broken down into inner workflows using YesWorkflow.<sup>14</sup> Starting from the raw data file from AirBnb, we began the process through OpenRefine<sup>15</sup>. With the file generated from OpenRefine, we then used Python to conduct further cleaning<sup>16</sup>. After cleaning with Python, the output file was then processed using SQL<sup>17</sup> to process IC violation checks and distinctness. The outer workflow<sup>18</sup> was generated using Draw.io, and each detailed workflow was generated using YesWorkflow to capture each coding step. It was most sensible for an outer workflow to be generated manually given how big each of the three individual components in YesWorkflow was. Each detailed inner workflow can be found in the figures below. For enlarged versions, please see our additional files or the files directly in our github repository.

14. Bertram Ludaescher, “<https://github.com/yesworkflow-org/>.”

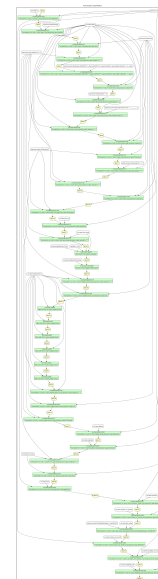
15. <https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/work-flow-digrams/YesWorkflow-Step1.pdf>

16. <https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/work-flow-digrams/YesWorkflow-Step2.pdf>

17. <https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/work-flow-digrams/YesWorkflow-Step3.pdf>

18. <https://github.com/megado123/CS513-DataCleansing-FinalProject/blob/main/work-flow-digrams/OuterWorkflow.pdf>

Figure 20: OpenRefine Workflow



As we can see from the figure above, the OpenRefine workflow is the most lengthy of three. This was primarily the case because the method itself went through column by column, requiring some type of cleaning to be done in each column.

Figure 21: Python Workflow

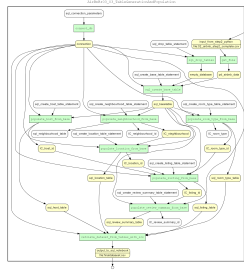


The Python workflow was bit more selective than the OpenRefine workflow as it was primarily focused on specific columns that needed identifying of “special” details such as outliers. Finally, the SQL data was the most interdependent as the creation of the final tables were all dependent on the beginning data set. Each individual table had the necessary constraints and checks, which were they populated by a script.

## 7. PHASE II - CONCLUSIONS SUMMARY

Our view of the data was that a significant amount of cleaning was necessary in order for the data to be feasible in total analysis. Although some basic analysis could be done as is, more in depth analysis required cleaning from consistency

Figure 22: SQL Workflow



and constraints among each column. The removal or marking of outliers or other significant data points was necessary too. We found that the best approach was to use a combination of tools, as we used OpenRefine, Python, and SQL.

OpenRefine provided a baseline of cleaning that was necessary. The approach of cleaning column by column laid the foundation for our further processing with SQL. Although we quickly realized that OpenRefine alone was not enough and we had to go through a few more steps using Python.

We found that Python was helpful in identifying specific outliers within columns or marking specific characteristics of entries such as more than 300 minimum nights. Then with the use of queries, SQL was then ideal to identify any constraints within the data or make sure each entry was distinct.

In conclusion, through this project, we have learned that the best way to approach cleaning is through the use of several tools. Cleaning leveraging multiple tools enabled each workflow to tackle a certain problem to ensure the data is suitable for any analysis use. The original data set is provided in a box location identified in the DataLinks.txt file.<sup>19</sup> which can also be found directly in the github repository associated with this project<sup>20</sup> or directly through the url<sup>21</sup>.

## References

- Bertram Ludaescher, Timothy McPhillips, Tyler Kolisnik. "https://github.com/yesworkflow-org/," 2021.
- "Find Latitude and Longitude," 2021. <https://www.findlatitudeandlongitude.com/find-address-from-latitude-and-longitude/>.
- Huynh, David. "OpenRefine," 2021. <https://openrefine.org/>.
- Li, Lan. "CS513-Data Cleaning Reference Datasets," 2021. <https://uofi.app.box.com/s/y7123ku1dpzd4m3126sqvpa2agvv08hv/file/82526727413>.

19. <https://uofi.app.box.com/folder/142035412824?s=vcj7qx52oiu93b1ix8prcgxbxupoprjo>

20. <https://github.com/megado123/CS513-DataCleaning-FinalProject/blob/main/DataLinks.txt>

21. <https://uofi.app.box.com/s/vcj7qx52oiu93b1ix8prcgxbxupoprjo>