Extraction, Transformation, and Load (ETL) Specification

Airbnb vs. Hotels.com

Price Comparison

Table of Contents

[Revision History 3](#_Toc6423811)

[1. Introduction 4](#_Toc6423812)

[1.1 Summary 4](#_Toc6423813)

[1.2 Data Sources 4](#_Toc6423814)

[2. Transformation Component Documentation 5](#_Toc6423815)

[2.1 Data Import/Extract 5](#_Toc6423816)

[2.2 Data Transformation 5](#_Toc6423817)

[2.3 Data Load 6](#_Toc6423818)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mm/yy> | <x.x> | <details> | <name> |
| 17/04/2019 | 1.0 | Initial document outlining ETL process | Valerie, Alicia and Suman |

# Introduction

## 

## Summary

The purpose of this project is to find the best discount price for accommodations in Toronto. To do this, we are comparing prices from Airbnb to prices on Hotels.com within the same Forward Sortation Areas (FSA). The data is obtained from two separate places, cleaned and then loaded into a MySQL database to be joined and become ready for queries.

## Data Sources

Hotel prices: [https://ca.hotels.com](https://ca.hotels.com/)

Airbnb prices: Toronto locations CSV obtained from <http://insideairbnb.com/get-the-data.html>

Github repository: <https://github.com/valltomm/ETL-Project.git>

# Transformation Component Documentation

## Data Import/Extract

* Airbnb import table: listings-3.csv
* Airbnb data exploration and clean up: Airbnb.ipynb
* Airbnb export table: airbnb\_table.csv
* Exported table of Hotel locations for web scraping: hotel\_list.csv
* Hotel website scrape file: Hotel\_db\_code.ipynb
* List of Hotel URLs to scrape: Book2.txt
* Table created by MongoDB: Hotel\_list\_raw.csv
* Hotel export table: Hotel\_list.csv

## Data Transformation

MySQL database setup for loading final data into:

* Schema: schema.sql
* Database: Hotels\_db
* Joining file: query.sql
* Create table for Airbnb with:
  + id INT PRIMARY KEY,
  + FSA TEXT,
  + postal\_code TEXT,
  + property\_type TEXT,
  + room\_type TEXT,
  + price integer,
  + source\_loaded text
* Create table for Hotels with:
  + id INT PRIMARY KEY,
  + postal\_code TEXT,
  + price integer,
  + source\_loaded text

Airbnb cleaning and transformation method: Airbnb.ipynb

* With Pandas, read listings-3.csv file to create DataFrame named “airbnb\_df”
* Create a copy of “airbnb\_df” with required columns 'zipcode', 'property\_type', 'room\_type', 'price' as “airbnb\_data\_df”
  + Check with airbnb\_df.info for missing values - These columns are non-null and therefore do not require further cleaning
* Rename ‘zipcode’ column to ‘postal\_code’
* Add column “source\_loaded” filling all rows with “Airbnb” – this will be useful when combining hotel and Airbnb data later to know where a price came from
* Pass a new column “FSA” to the “airbnb\_data\_df”, by grabbing the first 3 digits from the postal code column
* We removed any incomplete postal codes – however since we ended up using FSA, this step would not be required
* We then created a list from Airbnb data locations to run the Hotel website scraping against – this ensures we will have common FSAs for price comparison
  + Export this to hotel\_list.csv
  + Because the postal code will need to be hard coded into the Hotels.com URL for scraping, we manually did this in the Book2.txt file, using the postal codes from hotel\_list.csv
* Set Airbnb DataFrame index to FSA column and change price column data type from INT to float
* Export this final Airbnb\_transformed DataFrame to CSV as Airbnb\_table.csv– which will be used to connect to MySQL

Hotel data scrape, cleaning and transformation method:

* Using BeautifulSoup, run through a list of URLs from Book2.txt to grab hotel data
* To scrape the hotel postal code and price, results are found within a section with class “hotel-wrap"
  + Run a for loop within this and try for error handling
  + Postal code is found under: address = result.find('address', class\_="contact").text
  + Price is found under: price = result.find('a', class\_="price-link").text
  + Use if (address, price) to capture only results with both
  + Because the address contains several elements: street number and street name, neighbourhood (sometimes neighbourhood is included and sometimes it is not), city, province, postal code, country and phone number – we need to split this and grab only the postal code
    - For addresses with all 7 elements, split and grab the 4th element
    - For addresses with 6 elements, split and grab the 3rd element
  + Because some prices return 2 values and others return only the discount price, we need to make sure we grab only the discount price:
    - For prices with len >2, grab index value [1]
    - Else (for prices with len 1), grab index value [0]
* When postal code and price results are printed, display in a MongoDB collection “listings” and from that a pandas DataFrame “new\_customer\_location\_df”
* Pass a new column “FSA” to the “new\_customer\_location\_df”, by grabbing the first 3 digits from the postal code column
* Remove the dollar sign from price column with new\_customer\_location\_df["price"] = new\_customer\_location\_df["price"].apply(lambda x : x.replace('$','')) and change to type float
* Set FSA column as index and export the transformed “new\_customer\_location\_df” to csv as “Hotel\_list.csv”

## Data Load

MySQL connection file: hotels\_abnb\_etl.ipnyb

Schema: schema.sql

Final database: Hotels\_db

SQL joining/query file: query.sql

Loading and joining method:

* As cleaning hotel and Airbnb data was completed in separate notebooks, we connected to the csv’s in a new notebook and then connected to the Hotels\_db in MySQL
* Using the tables we had set up in the schema file (Listing\_abnb and listing\_hotels) we joined the two data sets with a sub-query
  + Select the columns: FSA from Airbnb as a.FSA, FSA from Hotels as b.FSA, Airbnb price and Hotel price
  + As we want to compare the best/lowest prices we then select the FSA’s MIN(price) from Airbnb table (Listing\_abnb) using GROUP BY FSA
  + Join with selection of Hotel MIN(price) grouped by FSA on FSA columns
    - We used left and right trim (LTRIM, RTRIM) and UPPER to ensure proper formatting when joining
  + To return a results we used DISTINCT to remove any duplicate FSAs and returned a search for one FSA (e.g. M4M) to obtain the lowest price in that FSA