Analyses for Housing prices of Greater Melbourne

Project Proposition:

This ETL project relied on using different data sources to compare the Adult population people with the median house price depending on where they lived in Greater Melbourne. This analysis was completed at two levels. The first level was comparing the Adult population of people with the median house price of the suburb that they lived in within Greater Melbourne. The second level of analysis involved conducting a similar comparison, but instead compared the Adult Population of people within Greater Melbourne. The demographic data was sourced using Australian Bureau of Statistics dataset, while the house price median data was sourced using the Domain API. Jupyter notebooks and Python were used to conduct the analysis. The year 2016 was chosen for all analysis because this had the most demographic data available as that year was a Census year.

PgAdmin4 was the chosen forum used to store the concluding table at the end of the analysis as it interacts well with the .csv file which is the chosen output from all the notebooks. In addition, the combined database table, once all the cleaning was completed was not very complex and thus it was good to use the pgAdmin4 server.

Data extraction is done from following sources:

1) Domain API
2) https://en.wikipedia.org/wiki/Local_government_areas_of_Victoria
3) https://knowyourcouncil.vic.gov.au/councils
4) https://www.abs.gov.au/AUSSTATS/abs@.nsf/DetailsPage/1410.02014-19?OpenDocument

Step 1 and 2 discuss the web-scraping techniques used while Step 3 and 4 discusses the use of an API. Step 5 is the concluding steps.

Extracting Step 1: https://en.wikipedia.org/wiki/Local_government_areas_of_Victoria

The first step was to find a list of all the Local Government Areas in Greater Melbourne. This table was web-scraped from a Wikipedia page (see link above). Only the first table needed to be web-scraped for this analysis

Transforming Step 1.1: The web-scraped table was then converted to a Pandas Dataframe (called lga_df) so that it was easier to use within Jupyter Notebook.

Transforming Step 1.2: Furthermore, it was noticed that all the Local Government Areas scraped from the Wikipedia Page had the prefix of “Shire of...” or “City of...”. However, to interact with the Local Council Website, the prefix of “Shire of” and “City of” needed to be changed to a suffix of ‘Shire” and “City” respectively.

This string transformation was achieved by splitting the lga_df dataframe into two separate dataframes using the loc command. Local Government Areas that had a prefix of “City of” were stored in one dataframe (city_of), and Local Government Areas that had a prefix of “Shire of” were stored in another dataframe (shire_of). String transformations were then applied to both city_of and shire_of separately so that all the Local Government Areas had the format needed to interact with the Local Council Website. The shire_of and city_of dataframe were then once more combined using the concat command.

Transforming Step 1.3: The dataframe of Local Government areas was converted to a list so that it is easier to interact with the Council Website.

EDA Step 1.1: This initial step was relatively simple so not much EDA was needed. However, it was confirmed that all the LGA areas in Greater Melbourne were included by checking the number of unique rows in the dataset (31). This matched the amount of LGAs in Greater Melbourne according to the Local Council Website, so therefore we are confident that this list is accurate. Both the count() and unique() functions were used to ensure that there were no duplicates in this list.

Extracting Step 2: https://knowyourcouncil.vic.gov.au/councils (and links on this website)

Using the Local Council Website to extract a list of suburbs for each Local Government Area. This was achieved using web-scraping and iterating through the list of Local Government Areas (LGA) obtained in the previous step. The methodology involved using the browser button click methodology to click on each individual local Council website once during each iteration of the LGA list, and using the beautiful soup function to obtain a paragraph that had the list of suburbs in each LGA.

Note: The Wikipedia extraction was needed first because the Local Council Government Website provided information of every LGA in Victoria but did not specify which regions were from Greater Melbourne.

The following steps were completed for every iteration of the LGA list:
Transforming Step 2.1:
Converting the paragraph to a comma separated list firstly involved a number of text replacements (for full details please check the Jupyter Notebook).

Main Problems Encountered:

One thing to note is that the end of the suburb paragraph always ended with the last two suburbs being separated by the word and. When the ‘and’ word was simply converted to a comma, this impacted the suburbs that had the word ‘and’ in the middle. To solve this problem, all spaces were converted to underscores first and then the characters ‘_and” were converted to a comma.

Some suburbs had the words /xa0 at the end due to the format of the html document so this was also replaced.

Transforming Step 2.2 The modified text paragraph was then converted to a list of suburbs, by splitting the paragraph of text using commas and converting to a list. This meand the paragraph text could now be used in python functions

Transforming Step 2.3: The list of suburbs was then passed through a dictionary where the key value was a local Government area and the value was the list of suburbs that corresponded to that particular Local Government Area

Transforming Step 2.4 The dictionary was then appended to a new list called suburb_sample. At the end of the iteration process, the suburb_sample list contained a dictionary with every single LGA within Greater Melbourne as the key and a list of suburbs for each LGA was the values.

Transforming Step 2.5 The suburb_sample list was converted to a pandas dataframe where the LGA was in one column and the list of suburbs for that LGA was in another value. The explode function was then used on this pandas dataframe so that each suburb and the corresponding LGA was in a separate row of the dataframe. This helped ensure the dataset could interact with the Domain API.

Transforming Step 2.6 Removing the leading space before suburbs: It was noticed that some (but not all) suburbs had a leading space. In order to later interact with the Domain API, the leading space before all of the suburbs needed to be stripped. This was achieved by converting the suburb column in the dataframe to a list, iterating through this list to strip the leading space where it existed and then re-appending the Suburb Names without the leading space to the dataset.

Transforming Step 2.7 The column that had the suburb name and the leading space was removed from the database. The variable in the dataframe was then renamed to “Local Government Area”. The index was also reset so that each row in the dataframe had a unique index. This was so it was easier to iterate through the dataset when making Domain API calls.

The transformation steps meant that the data was converted from a messy web-scraped data to cleanly formatted dataset that could be used to interact with the Domain API.

EDA Step 2.1: Dealing with null values All null values were dropped from the database once the database was exploded. This could be done with confidence as after exploration of the dataset it was shown that a large amount of null rows when the database was exploded was just because some LGAs had a larger amount of suburbs than other LGAS. Thus, when the explode function was used all LGAs filled up the same amount of rows (the amount of suburbs of the LGA with the highest amount of suburbs). Thus LGAs with a smaller number of suburbs had a large number of null values. Thus, all rows with any null values were dropped with confidence to make the database cleaner

EDA Step 2.2 Dealing with duplicate values : Using the count and nunique functions in the dataset, it was observed that both the LGA and Suburb Name values had duplicates. This was expected in the LGA column as multiple suburbs would belong to one LGA. However, it was more of a concern that the suburbs had duplicate values. After further inspection of the dataset and data it was revealed the reason for this duplication was because some suburbs also could belong to more than one LGA. This was found using the duplicate command and then sorting the duplicate values by alphabetical order.

It was decided that obviously as these LGAs overlap one another slightly, it was not of significant importance of which LGA duplicate suburbs were assigned to. However, it was decided that only one row per suburb should be a part of the data to avoid double-counting suburbs in the analysis especially when aggregating the suburbs to LGAs. Thus, the decision was made to keep the first suburb value and corresponding LGA to avoid duplicates within the suburb name (thus only counting each suburb once).

The count and nunique suburb were once again used to ensure that each suburb name was unique and also showed that there were no null values in the dataset.

Extraction Step 3 : Using Domain API 1:

The next step was to interact the web-scraped dataset with the Domain API to obtain the median price. This involved two separate API calls. The first API call involved using the suburb name to obtain a Suburb ID. The suburb ID was then used to interact with a separate Domain API call to obtain the median price in 2016.

The first step involved iterating through the dataframe produced by the web-scraping extraction and transformation. The Domain API call looped through the dataset and then produced the ID for each suburb. A try, except approach was used so that if a suburb did not have an ID the API call could still collect the data that was needed .

Transformation Step 3.3 ID and Postcode columns were added to the web-scraped dataframe so that a columns were ready for the output of the Domain API call.

Transformation Step 3.2 The looped Domain API call and response was then put into a pandas dataframe so that it could be used to interact with the next API call.

EDA Step 3.1 Dealing with the "No ID" entry in the ID column
The way that the code was set up, when the API call to obtain a suburb ID was unsuccessful, the dataframe entry in the ID column was "No ID". Loc commands on this dataframe was then used to identify the suburbs where this was the case. After looking and researching these 29 suburbs, it was realised that within the Domain API, these suburbs were included as part of another suburb within Domain. Thus, all the rows with "No ID" were dropped from the dataset.

EDA Step 3.2 Duplicate and Null Values Once the "No ID" column was removed, once again the values in the dataframe were counted using both count() and unique(). It was observed that both the Suburb Name and ID were all unique values which was great. Postcodes and LGAs had multiple duplicate values but this was expected as multiple suburbs can share postcodes and LGAs. There were also no null values in the dataset.

As this was a clean dataset, the next step was the last stage of the extraction process.

Extraction Step 4: The second Domain API call

This domain API call was more complex as there was a limit of 500 calls per day on each API key. Thus, transformation on the dataset had to be applied before the data could be extracted.

Transformation Step 4.1 The iloc command was used to split the dataset into two groups. One dataset had the first 300 rows and the other dataset had the last 228 rows. This was so two separate API calls using two separate API keys could be made on the data.

Transformation Step 4.2 The column "2016 Median Price" was then added as a column to both of the two dataframes

Extraction Step 4.1 Two separate API calls were made on the two datasets using a very similar process as before. Each API call iterated through the rows in the Pandas Dataframe, obtained the 2016 median price and recorded it in the 2016 Median Price row for the suburb

Transformation Step 4.3 Once the API calls were completed, the two dataframes were once again combined using the concat command

EDA Step 4.1 Dealing with Null values. The count command was used to identify null values. This showed that there were 94 suburbs with null values After inspection of the API calls, for these suburbs null values 'none' existed because these suburbs had not sold any houses within 2016 and thus were dropped from the dataset.

EDA Step 4.2 Dealing with 'None' values. The None values existed when the API call did not work in the dataset and were identified using the loc command. Only five suburbs had this particular problem as the houses sold in these suburbs were included as part of the sales in other suburbs and thus the API call was invalid. These values were also dropped from the dataset.

Step 5 Further Transformation and Exporting

Transformation Step 5.1 In order to be successfully merged with the ABS dataset, the suffix of 'City' and 'Shire' need to be removed from the Local Government Council Data column for all values. This was completed using a similar methodology as completed in the Transformation Step 1. The dataset was divided into two groups depending on whether the LGA had a 'City" suffix or "Shire" suffix and string transformations were applied to remove this suffix on both datasets. The two datasets were once again combined.

Transformation Step 5.2 The dataset Median price value had a data type of object and this was mapped to a float value so that it could be used in further numerical analysis

Transformation Step 5.3 Once this string transformation was completed this file with median price analysis at the Suburb level exported as a .csv file. This will now be used to interact with the suburb ABS data.

Transformation Step 5.4 The suburb data was when aggregated at the Local Government Area level using the groupby function. The values that were calculated were both the average of the median price for all LGAs and the median of the median price for all LGAs.

Transformation Step 5.5 The Average of Median price values and the Median of the Median price values were then passed through a dictionary to produce a Local Government Area dataframe.

Transformation Step 5.6 This Grouped dataset by LGA was also exported to a .csv to be compared to the ABS demographic data for the local Victorian Government.

Transformation of ABS data and merging with Suburb data:

Step 1: Extracted ABS data from website
https://www.abs.gov.au/AUSSTATS/abs@.nsf/DetailsPage/1410.02014-19?OpenDocument

Step 2 : Import of ABS site data CSV into data frame (This was done for both demographic data at the suburb level in Greater Melbourne and the LGA level). This was done using two separate Jupyter Notebooks (Notebook 2 and Notebook 3) and a similar process was applied to both. The demographic data that was used was the Adult Population Data.

Step 3: Import of Suburb List CSV into Jupyter notebook exported from above process (suburbs_median.csv) and (LGA_final_summary.csv)

Step 4: Filtering year 2016 data. As analysed in various years only 2016 data has more information and accuracy.

Step 5: Cleaning/renaming and Transforming both ABS data and Suburb data. The LGA data of the ABS datasets had a (C) suffix at the end and this was removed with the re command. This ensured the datasets from Notebook 1 and the ABS data could interact.

Step 6: Merging both ABS and Suburb data on “Suburb Name”

Step 7: Exporting final merged data into CSV

Transformations and EDA applied: Throughout the whole process of Notebook 2 and 3, null values were constantly dropped and this was checked using the count function. Furthermore, when the median price table was imported from the notebook file, there was an unnecessary ID column. This was also dropped in order to make the data cleaner when it was exported to a .csv format.

Overall, not much transformation and EDA was necessary in this step as ABS data was relatively clean due to the year that was chosen (2016) so there were very few null values.

Creating Database on PGAdmin and Loading data in SQL DB

Step 1 : Created DB in PGAdmin with name Housing database

Step 2: Created Table with columns matching to the final merged data exported from above Jupyter Notebook

Step 3: Imported CSV file into SQL DB