**Handover Report**

**Big Data Engineering – Assignment Task 3**

1. **Overview**

Founded by Brian Chesky, Nathan Blecharczyk, and Joe Gebbia in 2008, *Airbnb, Inc*., operates as an online marketplace focused on short-term homesteads and experiences. Airbnb, a shortened version of its original name, *AirBedandBreakfast*, revolutionised traveling by providing easy, accessible ways to customise itineraries without compromising on quality. As one of the world’s most liveable cities, Sydney is a popular attraction for various Airbnb’s today.

Our dataset deals with 1 year of Airbnb listings in the New South Wales region, from **May 2020 to January 2021**. Containing details such as the type of listing, its neighbourhood, and also about the host, this is an extensive dataset with over **400K** rows. In addition to this, we also have **Australia Census Data** from 2016 that provides details of the public demographics and their households such as various age groups, education levels, incomes etc. These are categorised based on “**LGA**” or local government area.

The aim of the assignment is to build data pipelines using the two different input datasets. These datasets will go through all the necessary steps before being ingested into a data warehouse and used in data marts for analytical purposes. Snowflake will be used for processing and analysing the data while Airflow will be used to build the pipelines and GCP to store the datasets.

1. **Dataset Exploration**

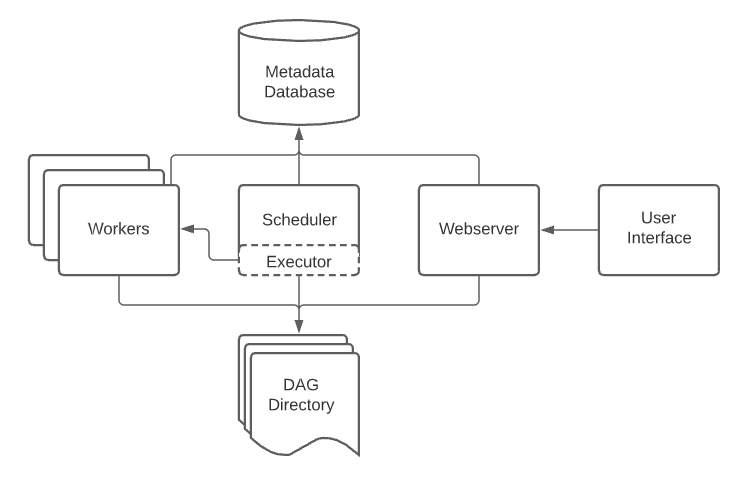
The **Airbnb** Listing dataset consists of 12 different files each divided by month and year. In total they have **22 columns and 412,122 rows**. There are **two census** tables available each identified as G01 and G02. The **G01** table consists of **109 columns and 132 rows** (each row dedicated to a single LGA). This table entails all information about the different age groups, their education levels and other split-ups of demographics. The **G02** table has a total of **9 columns and 132 rows** and various household details such as median income, household size and mortgage rates etc.

For the purpose of mapping the two datasets for analysis and exploration, **two additional datasets** have been provided. One contains the LGA Code along with the LGA Name, and the other consists of the LGA NAME and Suburb Name.

1. **System Architecture**

Airflow is a platform that lets you build and run workflows. A workflow is represented as a [DAG](https://airflow.apache.org/docs/apache-airflow/stable/concepts/dags.html) (a Directed Acyclic Graph), and contains individual pieces of work called [Tasks](https://airflow.apache.org/docs/apache-airflow/stable/concepts/tasks.html), arranged with dependencies and data flows taken into account. A DAG specifies the dependencies between Tasks, and the order in which to execute them and run retries; the Tasks themselves describe what to do, be it fetching data, running analysis, triggering other systems, or more.

Building a Data Pipeline can automate the ELT process, minimising errors and improving efficiency while doing so. Leveraging Airflow to build these data pipelines allows for easier data ingestion and data processing. The Airflow UI also lets you see what DAGs and their tasks are doing, view logs, and allow for debugging.



*Fig 1: Airflow Architecture*

1. **Data Ingestion (Part 0)**

*Note: proceed with this only after completing the following checklist -*

* + Creation and configuration of Snowflake account & GCP account
  + Setup of appropriate libraries and packages in Airflow environment (Appendix 2)
  + Creation of custom role for storage GCP service account in Snowflake on GCP (Appendix 3)
  + Creation of Warehouse with valid credentials on snowflake to link to Airflow (Appendix 4)

**Step 1**: Upload all the individual data files from local storage to GCP cloud storage and make note of the URL.

**Step 2**: In Snowflake environment, create a DATABASE for this project. Use that database and create a SCHEMA named RAW.

**Step 3**: Create a Storage Integration named GCP with the storage location set to the data URL retrieved earlier from GCP.

**Step 4:** Create a stage based of the storage integration created and the URL obtained.

**Step 5**: Verify that a connection has been established by trying the following command to see the list of files: list @stage\_gcp.

1. **Datawarehouse Design (Part 1)**
2. **Schema Raw**

The raw schema will contain all external tables of the available datasets, ingested in the appropriate file format.

Create a file format called *file\_format\_csv* with the following arguments so that the CSV files are read properly.

Type as CSV

Field Delimiter as ‘, ‘

Skip Header set to 1

NULL\_IF = ('\\N', 'NULL', 'NUL', ‘')

FIELD\_OPTIONALLY\_ENCLOSED\_BY = ‘“‘

Using the above file mart created, begin creating 5 separate external tables for each data base with the following arguments.

1. raw\_airbnb with the pattern = ‘.\*[0-9]\_[0-9]\*.csv’
2. raw\_g01census with the pattern = ‘.\*2016Census\_G01\_NSW\_LGA.csv'
3. raw\_g02census with the pattern = '.\*2016Census\_G02\_NSW\_LGA.csv'
4. raw\_suburb with the pattern ‘.\*NSW\_LGA\_SUBURB.csv’
5. raw\_code with the pattern = ‘.\*NSW\_LGA\_CODE.csv’

For verifying whether the data has been processed correctly, display only the first row of each table to see the nested column values.

1. **Schema Staging**

This step plays an important role as it involves creating a data table by flattening and parsing the existing fields from the raw tables using appropriate column types.

Each raw table must be staged so to be used as dimension tables in the next phase of building a data warehouse.

Example: Create table staging\_listing by parsing the first row from raw\_airbnb as follows:

|  |  |  |
| --- | --- | --- |
| **Existing Field** | **Data Type/Parsing Method** | **New Column Name** |
| C1 | VARCHAR | listing\_id |
| C4 | VARCHAR | host\_id |
| C5 | VARCHAR | host\_name |
| C6 | VARCHAR | host\_since |
| C7 | CHAR | host\_superhost |
| C8 | VARCHAR | host\_neighbourhood |
| C9 | VARCHAR | listing\_neighbourhood |
| C10 | VARCHAR | propety\_type |
| C11 | VARCHAR | room\_type |
| C12 | INT | accommodates |
| C13 | INT | price |
| C14 | CHAR | has\_availability |
| C15 | INT | availability\_30 |
| C16 | INT | number\_of\_reviews |
| C17 | INT | review\_scores\_rating |
| C18 | INT | review\_scores\_accuracy |
| C19 | INT | review\_scores\_cleanliness |
| C20 | INT | review\_scores\_checkin |
| C21 | INT | review\_scores\_communication |
| C22 | INT | review\_scores\_value |
| metadata$filename | SUBSTR(metadata$filename, 6, 7) | month\_year |

Repeat the process for 4 other tables using the names:

* staging\_code from raw\_code
* staging\_suburb from raw\_suburb
* staging\_g01 from raw\_g01census
* staging\_g02 from raw\_g02census

**Note**: While the general rule for building data warehouse is to parse every column in a raw table, for the purpose of this assignment I have taken columns C1 to C37 for the G01 Census table as those are of interest.

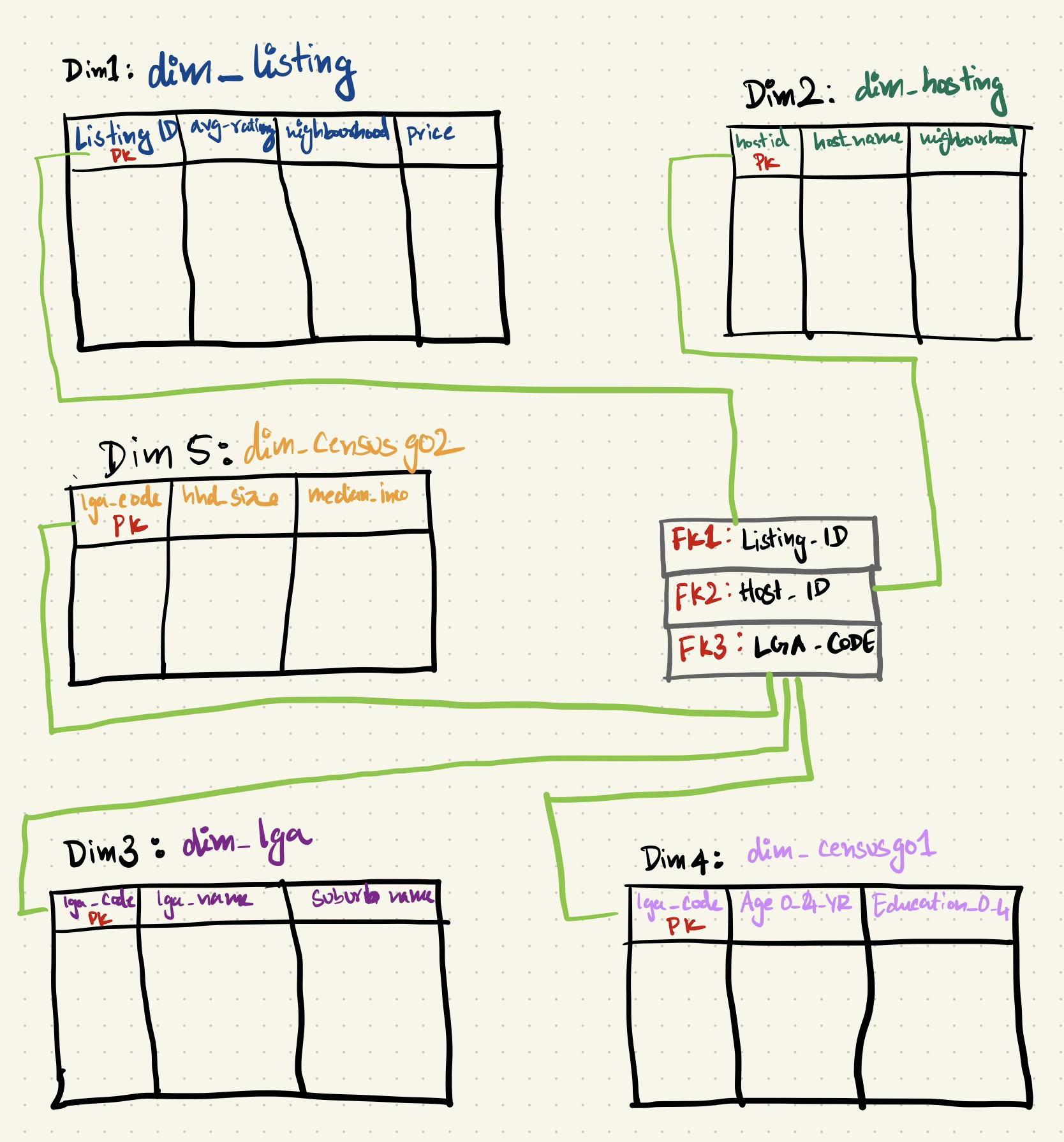
1. **Dimension and Fact Table Creation**

For the purpose of creating this warehouse, it will be organised using a star schema. A star schema uses a single large fact table to store transactional data or measured data, and around 4 smaller dimensional tables that store attributes of the data, forming the points of a star.

For this project, there are five dimension tables:

1. **Dimension Listing:** With primary key as listing\_id, containing all details about a particular listing.
2. **Dimension Hosting**: With primary key as host\_id, containing all details about the hosts.
3. **Dimension LGA:** With primary key as lga\_code, containing a joined table with lga\_suburb and lga\_code.
4. **Dimension CensusG01:** With primary key as lga\_code and all other columns from the censusG01 table.
5. **Dimension CensusG02:** With primary key as lga\_code and all other columns from the censusG02 table.

The one fact table contains foreign-key references to all the primary keys from the dimension tables. A rough diagrammatic understanding of the schematic relationship between these tables is described as below. The schema for the dimension tables can be found in (Appendix).



*Fig 2: Star Schema Structure for Data Warehouse*

1. **Data Mart Creation**

The three data marts created, dm\_listing\_neighbourhood, dm\_host\_neighbourhood, and dm\_property\_type provide a summarized breakdown of the Airbnb data which can be used to attempt the adhoc questions.

These datamarts were created using a combination of the existing dimension tables and a derivation of the dimensions as smaller datamarts to calculate percentage change in active and inactive listings, total number of stays and estimated revenue, as well as superhost rates.

Breaking down of the entire query creation into smaller tables and joining them to form the datamart allowed for accurate results.

These datamarts then need to be ordered as per instructions given an exported into individual *‘.csv’* files.

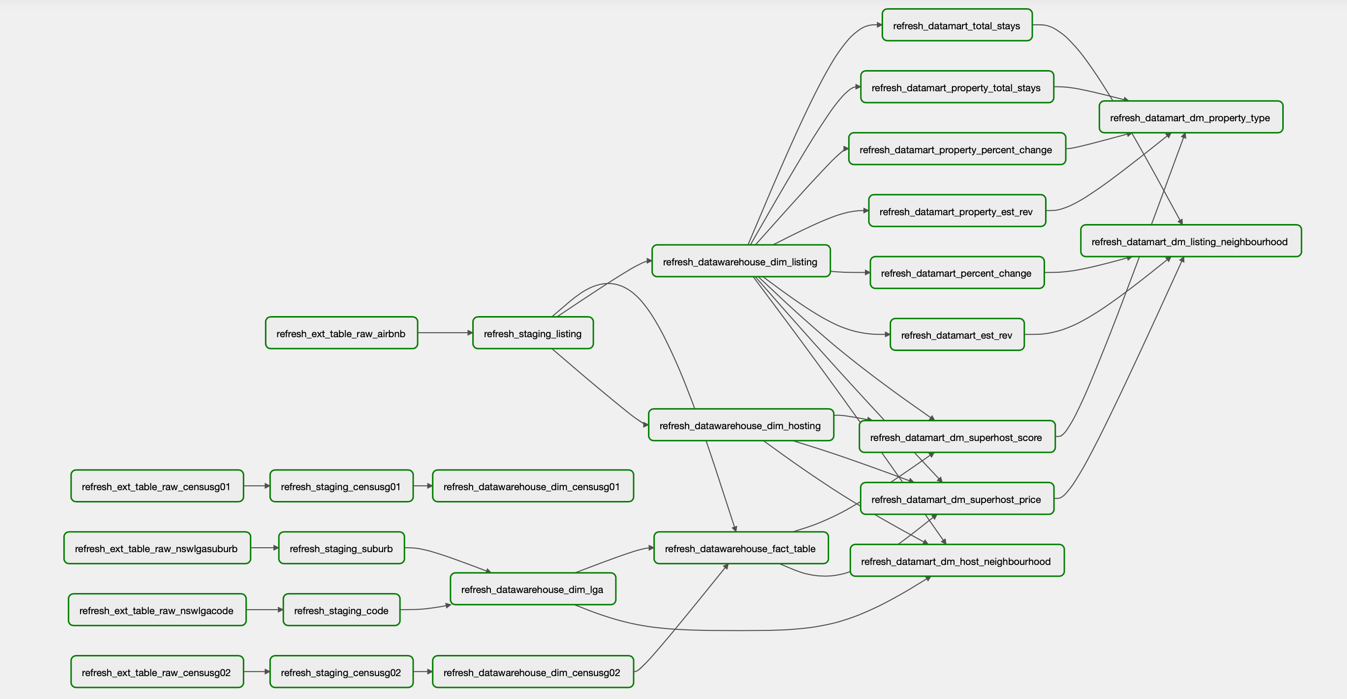
1. **ELT Pattern with Airflow (Part 2)**

Once the data warehouse is setup and designed on Snowflake, Airflow can be used to automate the build of a data pipeline. A general DAG script consists of the queries used, classified as tasks, their dependencies and the flows between each task. It is crucial to order the tasks correctly as it can hinder the creation of the warehouse by running the SQL queries incorrectly.

Data Flows in a DAG must only flow in a single direction and avoid any loops, i.e. a later task is dependent on a previous task, creating a deadlock situation.

After importing the necessary modules into the DAG script and configuring the DAG settings as per system requirements, all the SQL queries of designing a data warehouse must be copied in the right order. Each query has its own task\_id which will then be used to dictate the order of flow. In the final part of the script, the order of operations are declared to inform Airflow on the execution process.

In case of errors within the DAG script, the logs are available in the Airflow UI.



*Fig 3: Successful running of the DAG*

1. **Ad-hoc Analysis (Part 3)**
2. We first retrieve the best and worst performing neighbourhood in terms of estimated revenue using the previously created data mart. The estimated revenue over the last 12 months can be calculated using a rolling summation where only the final value from April 2021 will be considered.

This yields the result that Mosman does best with an estimated annual revenue of *AUD$111,922* and Fairfield is worst performing with approximately *AUD$15713.*



*Fig 4: Worst Performing Listing Neighbourhood*

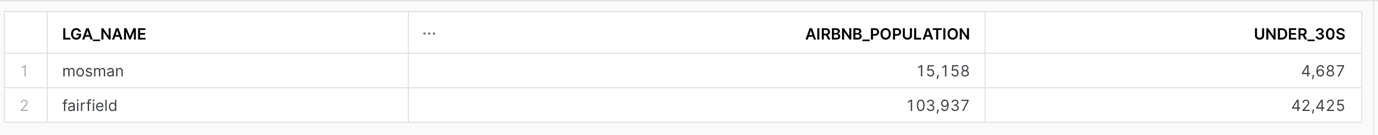
*Table

Description automatically generated*

*Fig 5: Best Performing Listing Neighbourhood*

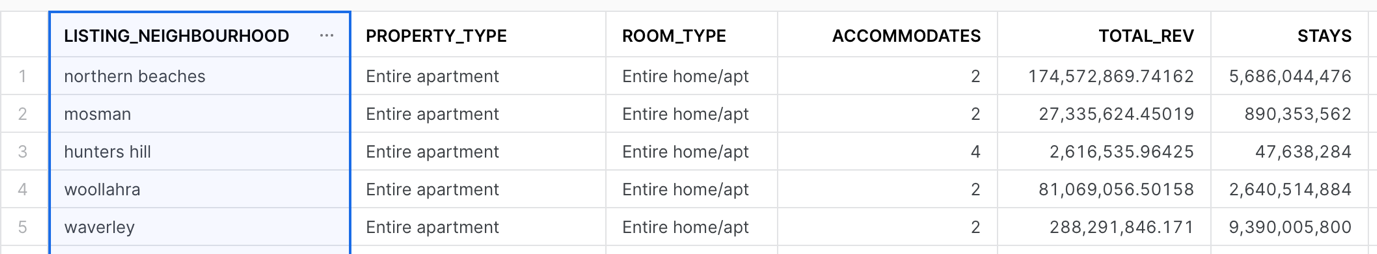
Assuming that the general population that rent out or prevail the services of Airbnb fall within the age categories of 20 years to 64 years it is interesting to note that Mosman has a significantly lower population than Fairfield.

Closely inspecting the “Under 30s” age, we can notice that Fairfield has a population nearly 10 times more than Mosman. The difference in revenue could be attributed to various external factors such as facilities and accessibility within a neighbourhood. Fairfield is almost an hour away from the city while Mosman is just a quick 15-minute taxi away. These factors play a huge role on the popularity of Airbnbs.



*Fig 6: Population Based View*

1. We first retrieve the top 5 listing neighbourhoods based on their total estimated revenue. Saving that as a temporary view, we can then rank these neighbourhoods for the type of property, type of room, and accommodation size ordered by their total number of stays.



*Fig 7: Top 5 listing neighbourhoods*

It is evident that the best kind of property type or room type for all the top neighbourhoods is an entire apartment or home that can accommodate between 2 to 4 people.

1. Upon initial analysis to get an understanding of the number of listings for each host, we can see that there are around 5,095 rows with hosts as having as less as 2 listings and as many as 496.

Table

Description automatically generated

*Fig 8: Hosts with Multiple Listings*

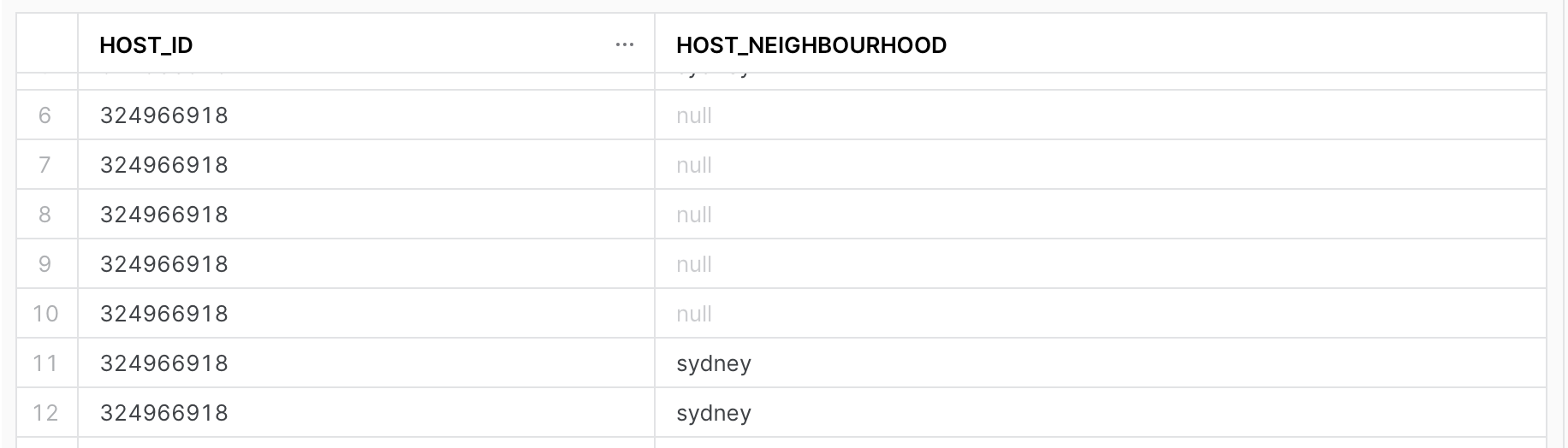
For those hosts with multiple listings, there is a general pattern noticed where most of the hosts do have their listing in the same LGA as their own neighbourhood.

Table

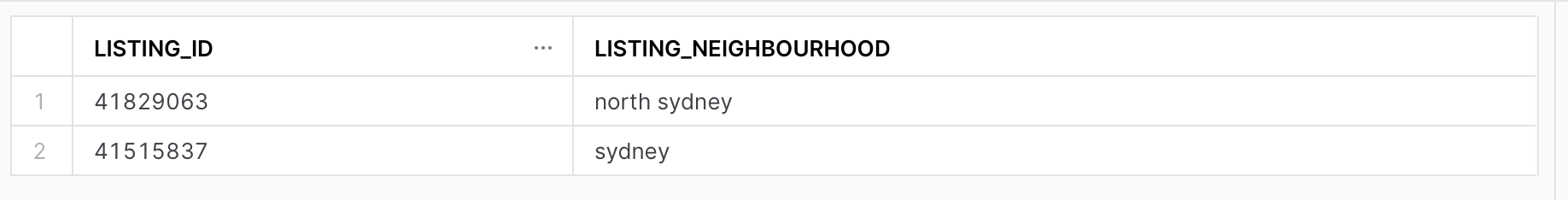
Description automatically generated

*Fig 9: Hosts living in same areas as listing*

With further inspection on certain hosts with multiple listings they have at-least 1 listing in their neighbourhood LGA. This could be attributed to the fact of convenience and ease of maintenance. For example, the following host has only 2 listings and live in Sydney but has 1 listing based in Sydney and the other in North Sydney.



*Fig 10: Host’s neighbourhood*

**

*Fig 11: Host’s listing’s neighbourhood*

Similarly, this other host with around 20 listing lives in the Inner West LGA and has a few of their listings in the same area, while a majority is spread around Sydney.

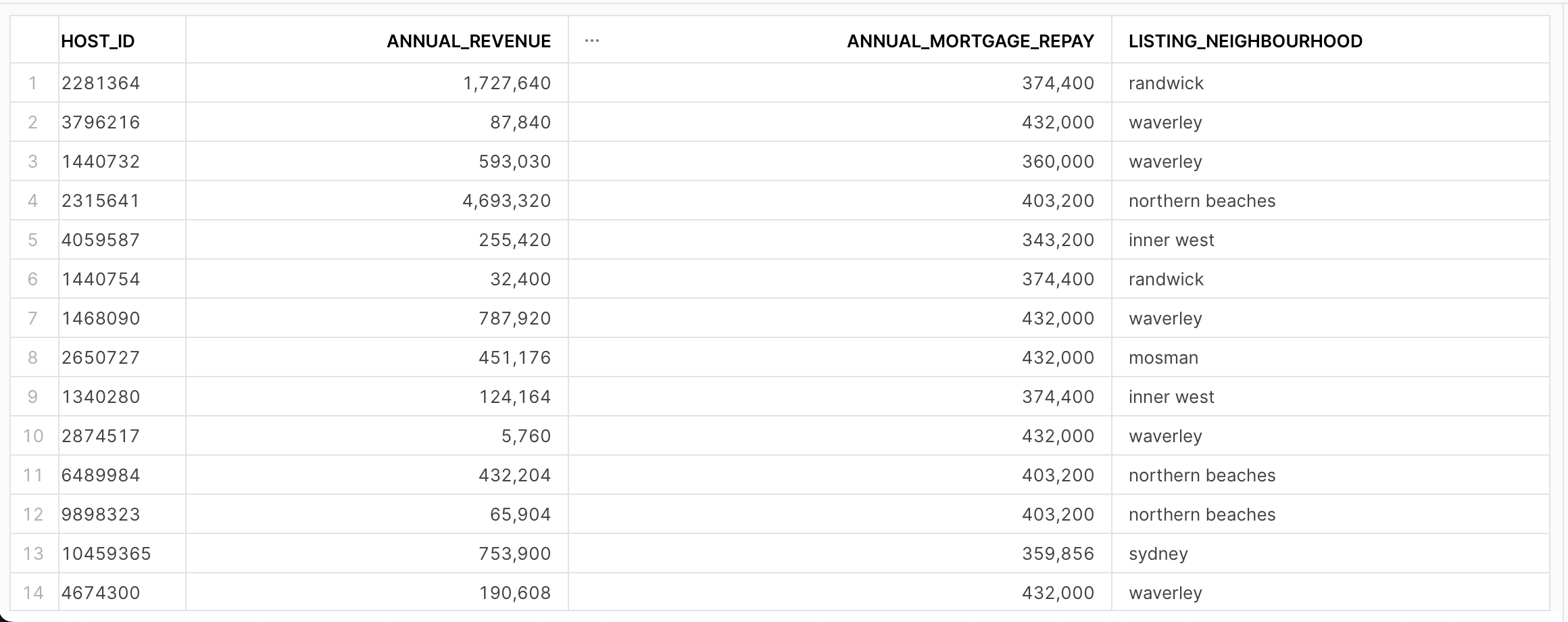
Table

Description automatically generated

*Fig 12: Host’s listing’s neighbourhood*

The need for expansion as the number of listings grow is understandable from a profit point of perspective. Growing competition in certain areas can also be a reason for moving out of your own LGA and into un-ventured zones.

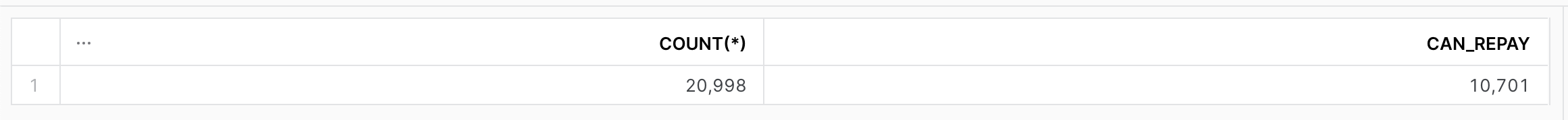
1. To approach this analysis, two views can be created to analyse the annual revenue and mortgage repayment rate for hosts and their listings in a particular neighbourhood. Hosts with unique listings alone yield around 20K rows.



*Fig 13: Host’s annual revenue and mortgage*

The annual revenue is calculated by taking a rolling summation over the 12-month period, and we only look at the final value from April 2021.

Our of the 20,998 hosts around 10,701 or 51% of them can cover the annualised median mortgage repayment rate.



*Fig 14: Hosts that can cover*

However, we must also consider that the census data reflects on 2016 median mortgage repayment values, and factors such as inflation could’ve contributed towards raising it. The listing data is also from the time of pandemic where travel was at an all-time low, and companies like Airbnb, faced a severe hit.

1. **Possible Issues & Bugs**

* While creating the dimension for hosting, the original dataset has a STRING datatype for the column “host\_since” which must be converted and parsed into the appropriate date format so that it can be used for calculations.
  + This can be achieved by using the *TO\_DATE* function.
* The census tables contain the LGA code with a prefix LGA-, which prohibits joins on other tables if necessary. Thus, a new column can be created in the dimension which extracts only the numeric code.
  + This can be achieved by using the *SUBSTR* function.
* Prior to running the DAG script, it is generally recommended to truncate all existing tables to verify the proper working of the code.

1. **Conclusion**

Designing a data pipeline with Airflow has numerous advantages. The potential of implementing these pipelines with Airflow using Python scripts enables one to build arbitrarily complex pipelines that can carry one’s desired tasks seamlessly. However, these pipelines can only perform as good as the underlying infrastructure supporting it, which is why it is crucial to get the order of dependencies correct. Analysing and attempting to answer the business questions for the given datasets yielded interesting results. An updated census dataset along with latest Airbnb listing details can provide more up-to-date information on the current status of the company’s performance in the Bed & Breakfast industry.

1. **Appendix**

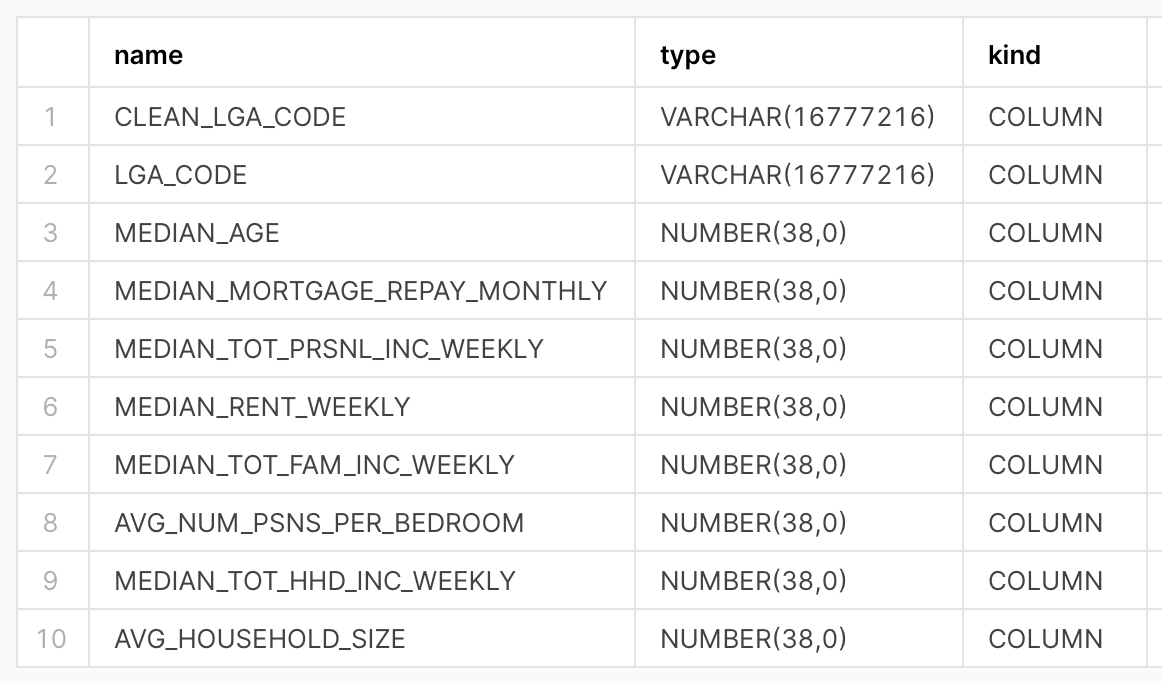
**Appendix 1: Schema of Dimension Tables**

**Dimension LGA**

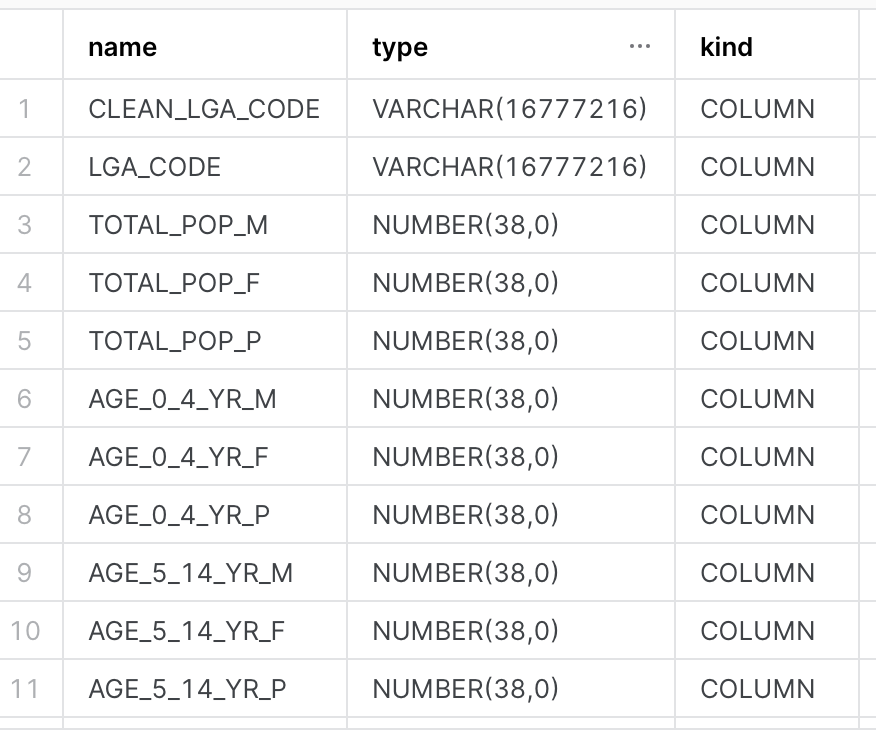
**Table

Description automatically generated**

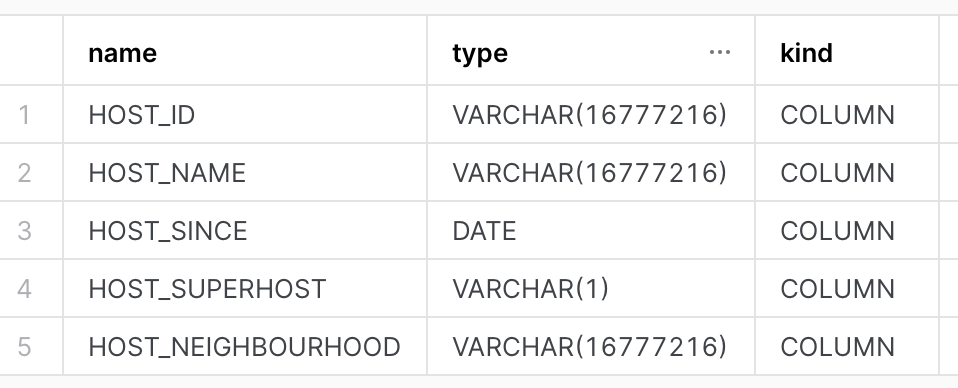
**Dimension CensusG02**

****

**Dimension CensusG01**

****

**Dimension Hosting**

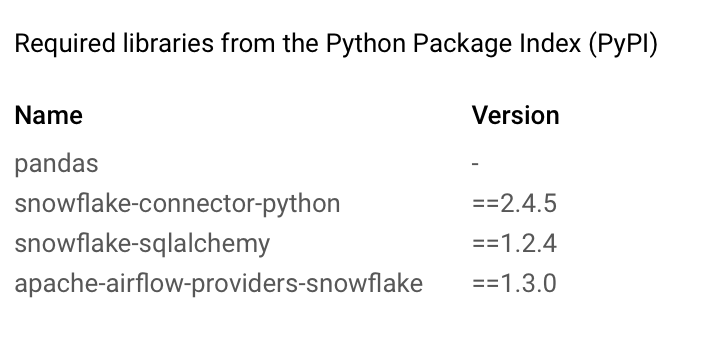
****

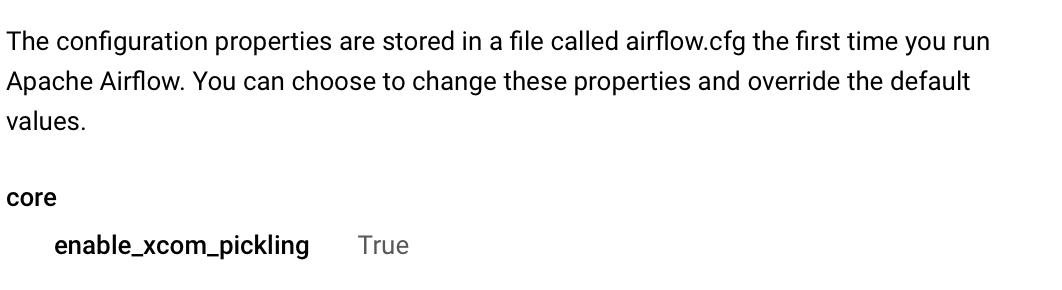
**Dimension Listing**

**Table

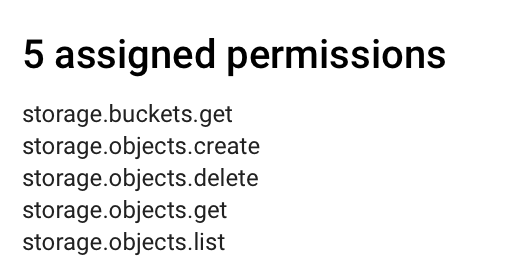
Description automatically generated**

**Appendix 2: Pypi Packages and Airflow Configurations**

****

****

**Appendix 3: Custom GCP role and Permissions**

****

**Appendix 4: Data Warehouse on Snowflake**

