# Data pipeline with Airflow And DBT


***Student Name:*** *Yasaman Mohammadi*
**Student ID:** 24612626
***course Name****: Big Data Engineering*

# 1. Introduction

Airbnb is an online-based marketing company that connects people looking for accommodation (Airbnb guests) to people looking to rent their properties (Airbnb hosts) on a short-term or long-term basis. The rental properties include apartments (dominant), homes, boats, and a whole lot more. As of 2019, there are 150 million users of Airbnb services in 191 countries, making it a major disruptor of the traditional hospitality industry (this is akin to how Uber and other emerging transportation services have disrupted the traditional intra-city transportation services). As a rental ecosystem, Airbnb generates tons of data including but not limited to: density of rentals across regions (cities and neighborhoods), price variations across rentals, host-guest interactions in the form of reviews, and so forth.

We will focus on Sydney for this assignment, and the dataset used is from May 2020 to April 2021.

## *1.1. Role*

The goal of this assignment is to build production-ready data pipelines with Airflow. We worked with two input datasets that needed to be processed and cleaned before loading this insightful information separately into a data warehouse (using ELT pipelines) and a data mart for analytical purposes.

# 2. Setup

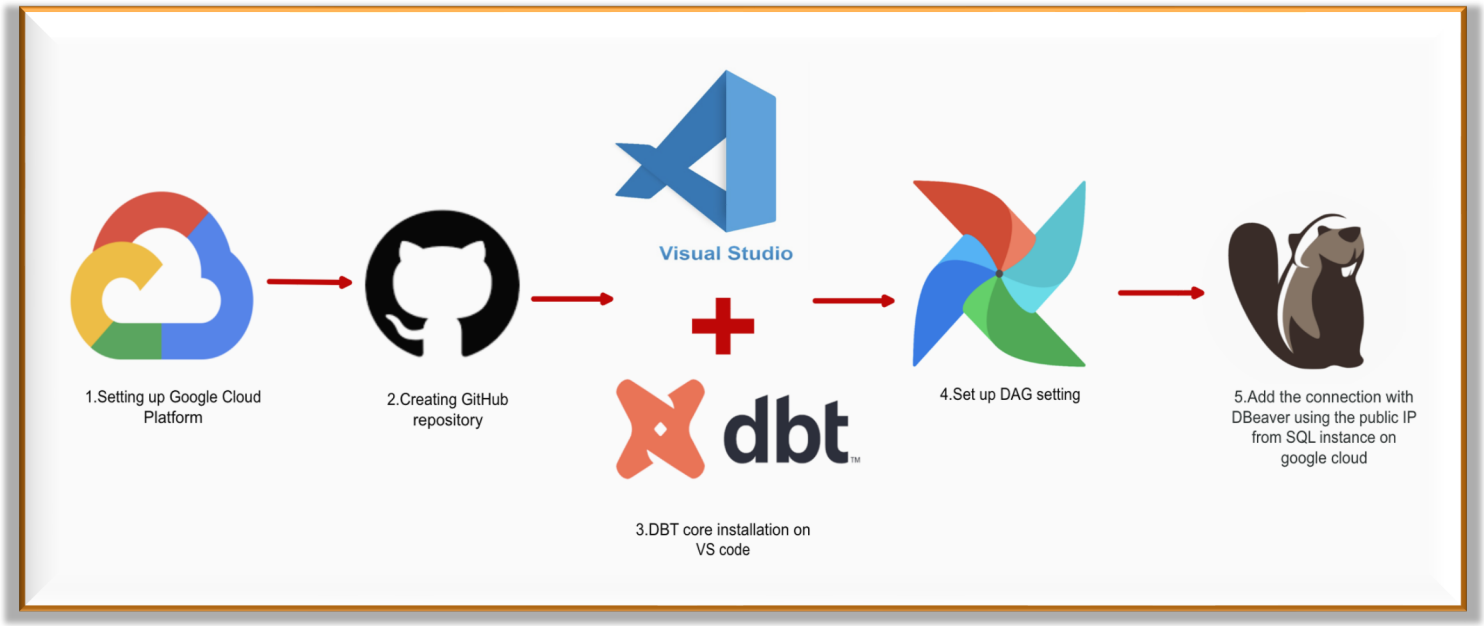



*Figure 1-workflow*

## *2.1 setting up Google Cloud Platform*

We initiated the process by creating an account on the Google Cloud Platform. Following this, we activated Cloud Composer and configured our environment by including essential packages such as "pandas" and "apache-airflow-providers-postgres." Next, we navigated to the "Airflow Configuration Overrides" section and enabled the option "enable_xcom_pickling True."

Subsequently, we set up a managed Postgres instance on GCP, and once it was successfully created, we retrieved both the public and IP addresses. To ensure secure access, we accessed the Connection tab on the left and then proceeded to the Networking tab, where we added our specific IP address (which can be found easily by searching “what is my IP” on Google). Finally, we confirmed the connectivity by testing the connection using DBeaver, utilizing the public IP and the previously defined password.

## *2.2 Creating GitHub Repository*

A GitHub repository named "dbt" was established within the "Jyasimo" account.

### *2.3 DBT Core Installation*

1. First, we installed Visual Studio Code (VS Code).
2. We installed Python version 3.
3. In VS Code, we added the following extensions: "Python Environment Manager" and "Python".
4. Next, we created a new folder in VS Code called "dbt-dev" and selected it.
5. Inside VS Code, we set up a new virtual Python environment.
6. In the terminal window, we executed the following commands:
    - pip install dbt-postgres
    - dbt init
1. During the initialization process, we named our project "bde."
2. When asked about the database choice, we selected option 1.
3. A new "profiles.yml" file was generated.
4. We opened the "profiles.yml" with a text editor or in VS Code to add a connection to the SQL instance created on Google Cloud Platform (GCP).

```
profiles.yml:
bde:
  outputs:
    dev:
      type: postgres
      threads: 4
      host: 35.244.112.19
      port: 5432
      user: postgres
      pass:
      dbname: postgres
      schema: raw
  target: dev
```

5. Back in the terminal window, we used the cd command to navigate into our new project.
6. We then executed “dbt debug” in the terminal to test the setup of dbt and the connection with Snowflake.

```
Done. PASS=17 WARN=0 ERROR=1 SKIP=0 TOTAL=18
yasamanmohammadi@Yasamans-MacBook-Pro dbt-dev % dbt debug
Running with dbt=1.6.6
dbt version: 1.6.6
python version: 3.10.13
python path: /Users/yasamanmohammadi/anaconda3/envs/dbt_env/bin/python
os info: macOS-13.0-arm64-arm-64bit
Using profiles dir at /Users/yasamanmohammadi/Documents/GitHub/dbt-dev
Using profiles.yml file at /Users/yasamanmohammadi/Documents/GitHub/dbt-dev/profiles.yml
Using dbt_project.yml file at /Users/yasamanmohammadi/Documents/GitHub/dbt-dev/dbt_project.yml
adapter type: postgres
adapter version: 1.6.6
Configuration:
  profiles.yml file [OK found and valid]
  dbt_project.yml file [OK found and valid]
Required dependencies:
 - git [OK found]

Connection:
  host: 35.244.112.19
  port: 5432
  user: postgres
  database: postgres
  schema: raw
  connect_timeout: 10
  role: None
  search_path: None
  keepalives_idle: 0
  sslmode: None
  sslcert: None
  sslkey: None
  sslrootcert: None
  application_name: dbt
  retries: 1
Registered adapter: postgres=1.6.6
  Connection test: [OK connection ok]

All checks passed!
```

7. In VS Code, we accessed the Source Control tab on the left, initialized the repository, and added a commit message.
8. Finally, we clicked "Publish Branch" and selected the previously created branch, where we were prompted to log in and permit GitHub.

### *2.4 DBeaver connection*

After obtaining the public IP address from Google Cloud Platform (GCP), we establish a connection in DBeaver.

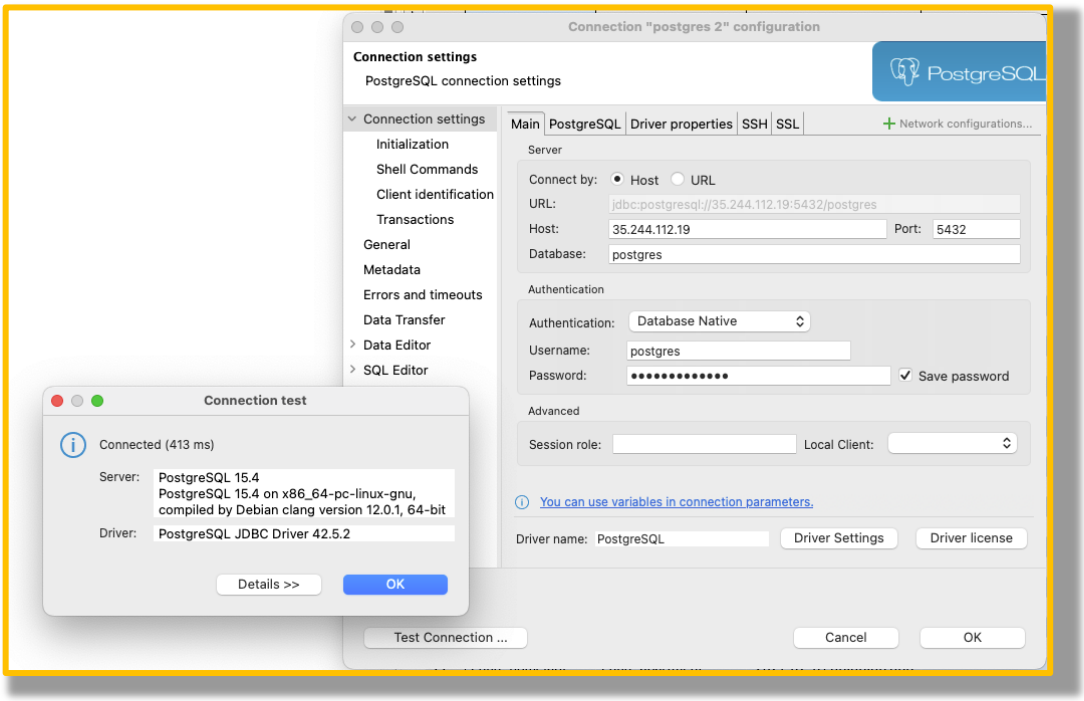



*Figure 2-Dbeaver connection*

After obtaining the public IP address from Google Cloud Platform (GCP), we established a connection in DBeaver. Subsequently, we tested the connection, and it was successfully connected.

```
--Create a raw schema on Postgres
CREATE SCHEMA IF NOT EXISTS raw;
```

Initially, we generated a raw schema in DBeaver by composing an SQL query.

```
--CREATE TABLE raw.Census_G02_NSW_LGA_2016
CREATE TABLE raw.Census_G02_NSW_LGA_2016 (
    LGA_CODE_2016 VARCHAR(10) ,
    Median_age_persons INT,
    Median_mortgage_repay_monthly INT,
    Median_tot_prsnl_inc_weekly INT,
    Median_rent_weekly INT,
    Median_tot_fam_inc_weekly NUMERIC,
    Average_num_psns_per_bedroom NUMERIC,
    Median_tot_hhd_inc_weekly NUMERIC,
    Average_household_size NUMERIC
);
```

Then, within the raw schema, we created individual raw tables for each CSV file, facilitating data transformation using a DAG file.

### *2.4 Setup DAG settings on Airflow*

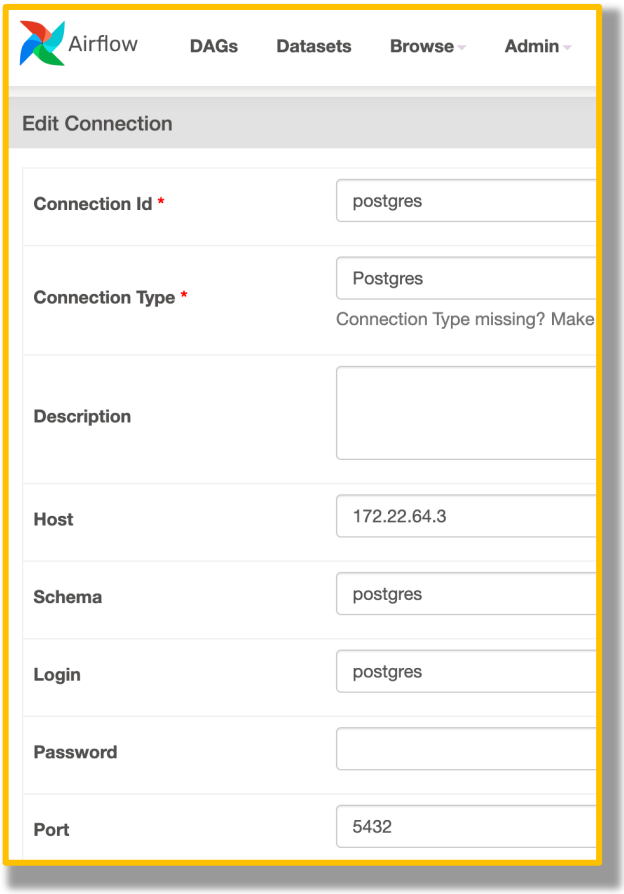



*Figure 3-Airflow Admin tab*

In the "Admin" tab, we established a new record, inputting the private IP address obtained from GCP as indicated.

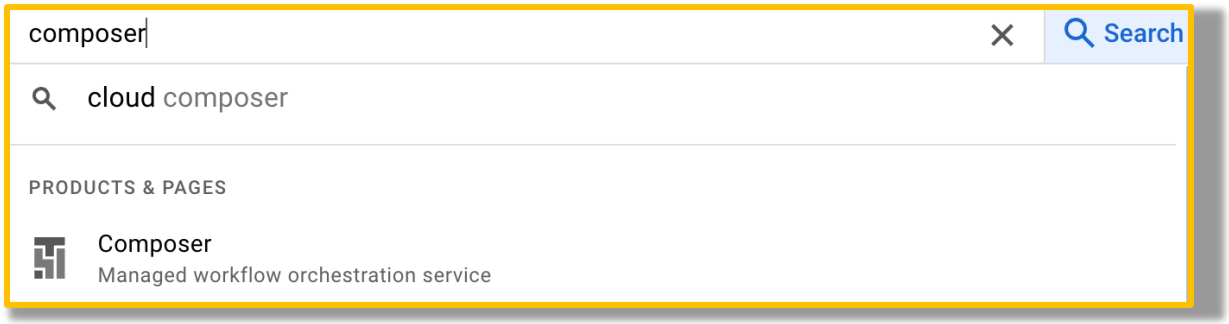



*Figure 4-Cloud Composer*

We accessed Google Cloud Composer.

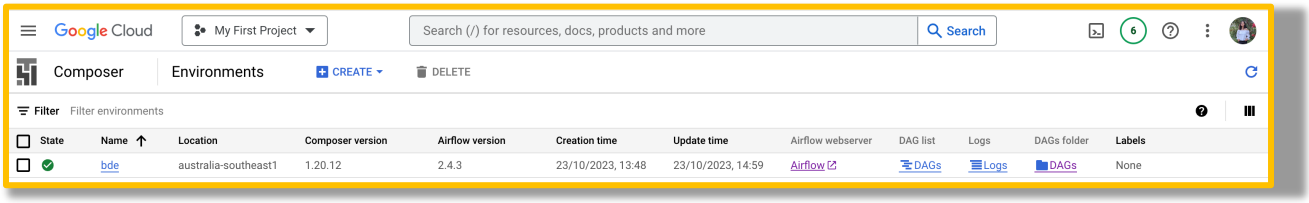



*Figure 5-Cloud Environment*

After identifying the instance, we had previously created, we navigated to the associated bucket.

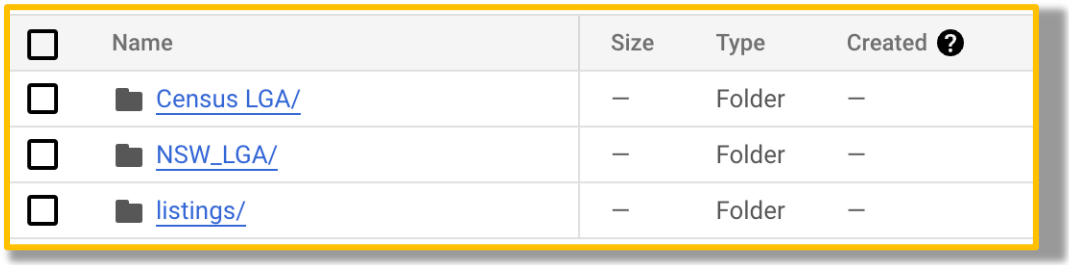



*Figure 6-Data folder*

There, we uploaded our data into the "data" folder.

```
#######################################################
#
#   DAG Settings
#
#######################################################


dag_default_args = {
    'owner': 'bde-at3',
    'start_date': datetime.now() - timedelta(days=2+4),
    'email': [],
    'email_on_failure': True,
    'email_on_retry': False,
    'retries': 2,
    'retry_delay': timedelta(minutes=5),
    'depends_on_past': False,
    'wait_for_downstream': False,
}


dag = DAG(
    dag_id='Airflow_DAG_PART_1',
    default_args=dag_default_args,
    schedule_interval=None,
    catchup=True,
    max_active_runs=1,
    concurrency=5
)
```

This code configures an Apache Airflow Directed Acyclic Graph (DAG) named 'Airflow_DAG_PART_1'. It defines default arguments like the owner, start date, email settings, and retry behaviour. The DAG is designed without a fixed schedule, enabling external triggering, can catch up on missed runs, allows only one active run at a time, and permits up to 5 concurrent task executions. This flexibility makes it adaptable to various workflow requirements.

```
#########################################################
#
#   Load Environment Variables
#
#########################################################
AIRFLOW_DATA = "/home/airflow/gcs/data"

Census_LGA = AIRFLOW_DATA+"/Census LGA/"
NSW_LGA = AIRFLOW_DATA+"/NSW_LGA/"
LISTINGS = AIRFLOW_DATA+"/listings/"
```

In this code, we specify directory paths in the filesystem, making it easier to reference and work with specific data directories or locations within the context of the Airflow environment.

```
def import_census_lga_g02_func(**kwargs):

    #set up pg connection
    ps_pg_hook = PostgresHook(postgres_conn_id="postgres")
    conn_ps = ps_pg_hook.get_conn()


    #generate dataframe by combining files
    df = pd.read_csv(Census_LGA+'2016Census_G02_NSW_LGA.csv')

    if len(df) > 0:
        col_names = ['LGA_CODE_2016', 'Median_age_persons',
                     'Median_mortgage_repay_monthly',
                     'Median_tot_prsnl_inc_weekly', 'Median_rent_weekly',
                     'Median_tot_fam_inc_weekly', 'Average_num_psns_per_bedroom',
                     'Median_tot_hhd_inc_weekly', 'Average_household_size']



        values = df[col_names].to_dict('split')
        values = values['data']
        logging.info(values)

        insert_sql = """
                    INSERT INTO raw.Census_G02_NSW_LGA_2016(LGA_CODE_2016,
                    Median_age_persons, Median_mortgage_repay_monthly,
                    Median_tot_prsnl_inc_weekly, Median_rent_weekly,
                    Median_tot_fam_inc_weekly, Average_num_psns_per_bedroom,
                    Median_tot_hhd_inc_weekly, Average_household_size)

                    VALUES %s
                    """

        result = execute_values(conn_ps.cursor(), insert_sql, values, page_size=len(df))
        conn_ps.commit()
    else:
        None

    return None
```

This Python function is designed to import data from a CSV file named '2016Census_G02_NSW_LGA.csv' into a PostgreSQL database table on DBeaver called 'raw. Census_G02_NSW_LGA_2016.' It does so by reading the CSV file, selecting specific columns from the data, and then inserting these extracted values into the corresponding PostgreSQL table. The code follows a similar process for other tables with different CSV files, all aimed at maintaining structured and accessible databases containing various information for further analysis and reporting.

```
############################################################
#
#   DAG Operator Setup
#
############################################################

import_census_lga_g01_task = PythonOperator(
    task_id="import_census_lga_g01_id",
    python_callable=import_census_lga_g01_func,
    op_kwargs={},
    provide_context=True,
    dag=dag
)
```

This code snippet sets up a PythonOperator within an Apache Airflow DAG. It will execute the "import_census_lga_g01_func" Python function with a task ID of "import_census_lga_g01_id." The operator can pass additional keyword arguments, and it provides context information. It is an integral part of a larger workflow represented by the "dag."The code follows a similar process for other tables.

```
## tasks all run parallel
[import_census_lga_g01_task , import_census_lga_g02_task,import_load_listing_func_task ,import_NSW_LGA_CODE_task,import_NSW_LGA_SUBURB_task]
```

This ultimate code specifies tasks to be executed without any sequence, allowing them to run concurrently.

| Name | Size | Type | Created | Storage class |
|---|---|---|---|---|
| Airflow_DAG_PART_1.py | 15.3 KB | text/x-python-script | 23 Oct 2023, 15:23:27 | Standard |

*Figure 7-DAG folder*

Upon completing our DAG file, we proceeded to upload it to the "DAG" folder of the “bde” instance.


*Figure 8-Airflow*

We triggered the DAG in Airflow, and it seems that everything ran successfully.

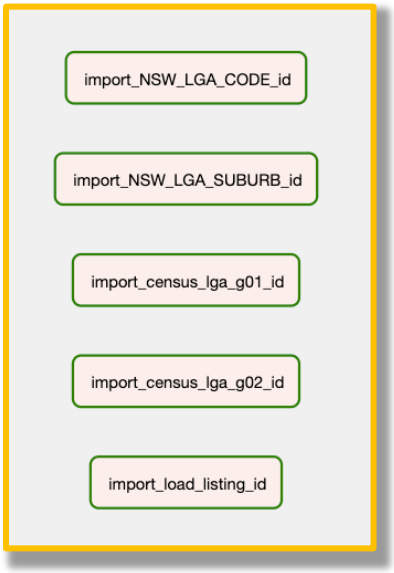



*Figure 9-DAG tasks*

In the Airflow workflow's graph section, we verified the successful execution of each task, and all tasks ran without any issues.

# 3. DBT Architecture

The DBT model for this project has four layers: raw, staging, warehouse, and data mart.

## *3.1 source.yml*

To establish a comprehensive data lineage from our source data to our data models, it is essential to include our source tables in a "source.yml" file. This file will define the tables we intend to utilize and their corresponding database/schema locations, and we can optionally provide brief descriptions for documentation purposes. We put this file in the model folder on VS Code.

```yaml
version: 2

sources:
  - name: raw
    description: Raw data from CSV files
    database: postgres
    schema: raw

    tables:
      - name: census_g01_nsw_lga_2016
        description: Description of census_g01_nsw_lga_2016
      - name: census_g02_nsw_lga_2016
        description: Description of census_g02_nsw_lga_2016
      - name: listing
        description: Description of listing
      - name: nsw_lga_code
        description: Description of nsw_lga_code
      - name: nsw_lga_suburb
        description: Description of nsw_lga_suburb
      - name: host_snapshot
        description: Description of host_snapshot
      - name: room_snapshot
        description: Description of room_snapshot
      - name: property_snapshot
        description: Description of property_snapshot
```

## *3.2 Snapshots*

For this project, we implemented timestamp-based snapshots. The timestamp strategy is employed when the source data incorporates a column that updates whenever a record changes. With this approach, DBT generates a fresh snapshot record each time the timestamp column surpasses the timestamp of the previous snapshot execution.
To organize these snapshots, we put them into a "snapshots" folder of VS Code.

```
{% snapshot property_snapshot %}

{{
  config(
    target_schema='raw',
    strategy='timestamp',
    unique_key='LISTING_ID',
    updated_at='SCRAPED_DATE'
  )
}}

select  SCRAPED_DATE,LISTING_ID,PROPERTY_TYPE,LISTING_NEIGHBOURHOOD
from {{ source('raw', 'listing') }}

{% endsnapshot %}
```

Next, we executed "dbt snapshot" in the VS Code terminal. After running this command, the snapshot table became accessible in DBeaver.

*Table 1-Property snapshot*

| | scraped_date | listing_id | property_type | listing_neighbourhood | dbt_scd_id | dbt_updated_at | dbt_valid_from | dbt_valid_to |
|---|---|---|---|---|---|---|---|---|
| 1 | 2020-05-10 00:00:00.000 | 65,126 | Apartment | Waverley | 7ae23cf720ffc990fb4b02bfa48be1d1 | 2020-05-10 00:00:00.000 | 2020-05-10 00:00:00.000 | 2020-07-15 00:00:00.000 |
| 2 | 2020-07-15 00:00:00.000 | 65,126 | Apartment | Waverley | 2352ada5c67f7e84d478afe6ca6d1aac | 2020-07-15 00:00:00.000 | 2020-07-15 00:00:00.000 | 2020-08-21 00:00:00.000 |
| 3 | 2020-08-21 00:00:00.000 | 65,126 | Entire apartment | Waverley | a2273493b77492c584a149e786037715 | 2020-08-21 00:00:00.000 | 2020-08-21 00:00:00.000 | 2020-09-11 00:00:00.000 |
| 4 | 2020-09-11 00:00:00.000 | 65,126 | Entire apartment | Waverley | 4571b8f7f6c172e11a19dcfb526c02a8 | 2020-09-11 00:00:00.000 | 2020-09-11 00:00:00.000 | 2020-11-05 00:00:00.000 |
| 5 | 2020-11-05 00:00:00.000 | 65,126 | Entire apartment | Woollahra | 9ee479a0f8600d32636289008ad257e1 | 2020-11-05 00:00:00.000 | 2020-11-05 00:00:00.000 | 2020-12-15 00:00:00.000 |
| 6 | 2020-12-15 00:00:00.000 | 65,126 | Entire apartment | Woollahra | b0e5d9218a12b9c0bf5db47a549f7a7c | 2020-12-15 00:00:00.000 | 2020-12-15 00:00:00.000 | 2021-01-11 00:00:00.000 |
| 7 | 2021-01-11 00:00:00.000 | 65,126 | Entire apartment | Woollahra | e3927fbbbe70603abae8e70431e8a147 | 2021-01-11 00:00:00.000 | 2021-01-11 00:00:00.000 | 2021-02-10 00:00:00.000 |
| 8 | 2021-02-10 00:00:00.000 | 65,126 | Entire apartment | Woollahra | 45001db588d43a32c939f3c638358041 | 2021-02-10 00:00:00.000 | 2021-02-10 00:00:00.000 | 2021-03-06 00:00:00.000 |
| 9 | 2021-03-06 00:00:00.000 | 65,126 | Entire apartment | Woollahra | f61a5af4581e0907fa0c385ede634af0 | 2021-03-06 00:00:00.000 | 2021-03-06 00:00:00.000 | 2021-04-11 00:00:00.000 |
| 10 | 2021-04-11 00:00:00.000 | 65,126 | Entire apartment | Waverley | 2016580efbfa70551c90a4091b5b25f1 | 2021-04-11 00:00:00.000 | 2021-04-11 00:00:00.000 | [NULL] |

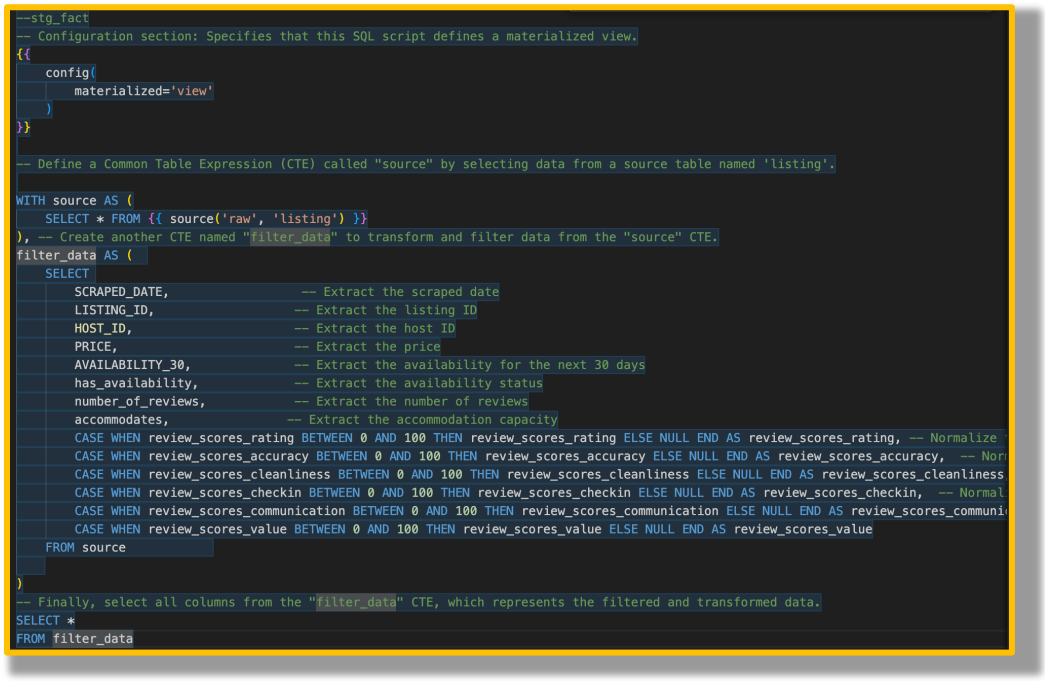
Additionally, we generated snapshots for both host-related and room-related columns.

## *3.3 Staging*

The staging models served as the foundation of our project, where we incorporated all the individual components required to construct our more intricate and valuable models. In this layer, we performed necessary cleaning, renaming, and transformations for each table.

- ***Stg_fact***

We generated a fact table from the listing table and conducted data normalization on the necessary columns as follows:



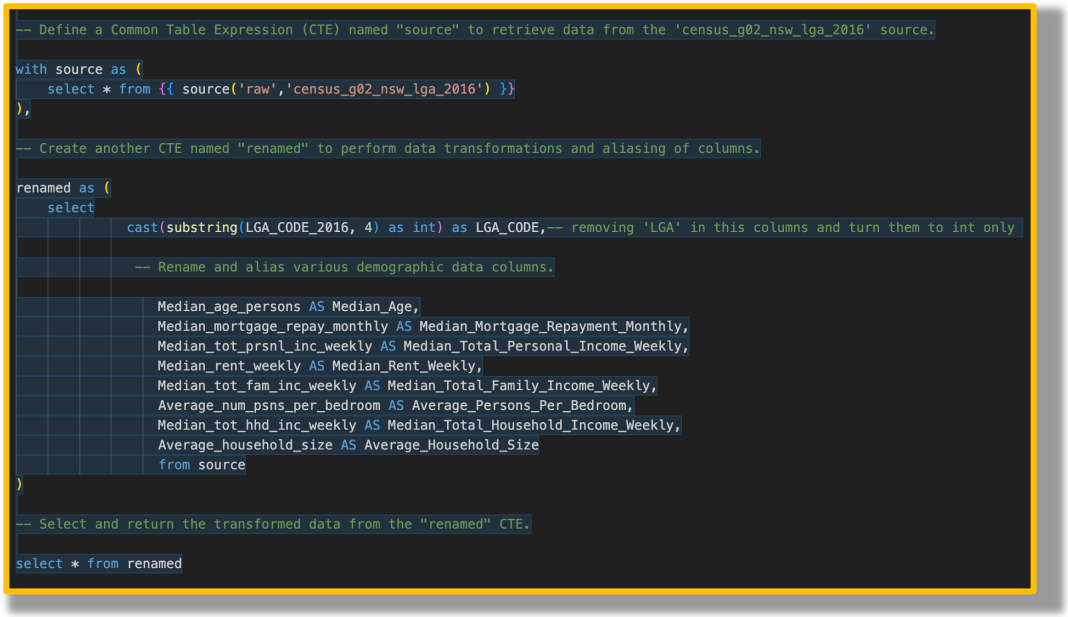
```sql
--stg_fact
-- Configuration section: Specifies that this SQL script defines a materialized view.
{{
    config(
        materialized='view'
    )
}}

-- Define a Common Table Expression (CTE) called "source" by selecting data from a source table named 'listing'.

WITH source AS (
    SELECT * FROM {{ source('raw', 'listing') }}
), -- Create another CTE named "filter_data" to transform and filter data from the "source" CTE.
filter_data AS (
    SELECT
        SCRAPED_DATE,                 -- Extract the scraped date
        LISTING_ID,                   -- Extract the listing ID
        HOST_ID,                      -- Extract the host ID
        PRICE,                        -- Extract the price
        AVAILABILITY_30,              -- Extract the availability for the next 30 days
        has_availability,             -- Extract the availability status
        number_of_reviews,            -- Extract the number of reviews
        accommodates,                -- Extract the accommodation capacity
        CASE WHEN review_scores_rating BETWEEN 0 AND 100 THEN review_scores_rating ELSE NULL END AS review_scores_rating, -- Normalize
        CASE WHEN review_scores_accuracy BETWEEN 0 AND 100 THEN review_scores_accuracy ELSE NULL END AS review_scores_accuracy,  -- Nor
        CASE WHEN review_scores_cleanliness BETWEEN 0 AND 100 THEN review_scores_cleanliness ELSE NULL END AS review_scores_cleanliness
        CASE WHEN review_scores_checkin BETWEEN 0 AND 100 THEN review_scores_checkin ELSE NULL END AS review_scores_checkin,  -- Normal
        CASE WHEN review_scores_communication BETWEEN 0 AND 100 THEN review_scores_communication ELSE NULL END AS review_scores_communi
        CASE WHEN review_scores_value BETWEEN 0 AND 100 THEN review_scores_value ELSE NULL END AS review_scores_value
    FROM source

)
-- Finally, select all columns from the "filter_data" CTE, which represents the filtered and transformed data.
SELECT *
FROM filter_data
```

- ***Stg_g02***



```sql
-- Define a Common Table Expression (CTE) named "source" to retrieve data from the 'census_g02_nsw_lga_2016' source.

with source as (
    select * from {{ source('raw','census_g02_nsw_lga_2016') }}
),

-- Create another CTE named "renamed" to perform data transformations and aliasing of columns.

renamed as (
    select
            cast(substring(LGA_CODE_2016, 4) as int) as LGA_CODE,-- removing 'LGA' in this columns and turn them to int only

             -- Rename and alias various demographic data columns.

                Median_age_persons AS Median_Age,
                Median_mortgage_repay_monthly AS Median_Mortgage_Repayment_Monthly,
                Median_tot_prsnl_inc_weekly AS Median_Total_Personal_Income_Weekly,
                Median_rent_weekly AS Median_Rent_Weekly,
                Median_tot_fam_inc_weekly AS Median_Total_Family_Income_Weekly,
                Average_num_psns_per_bedroom AS Average_Persons_Per_Bedroom,
                Median_tot_hhd_inc_weekly AS Median_Total_Household_Income_Weekly,
                Average_household_size AS Average_Household_Size
                from source
)

-- Select and return the transformed data from the "renamed" CTE.

select * from renamed
```

In the staging layer for the 'census_g02_nsw_lga_2016' table, we appropriately renamed columns as required. We also conducted column renaming in the 'census_g01_nsw_lga_2016' table, excluding unnecessary columns and focusing on specific columns related to age, population, birthplace, and citizenship. We have performed these operations in the 'stg_G01' table.

- ***Stg_host***

```sql
    select
        HOST_ID,                -- Extract host ID
        scraped_date,           -- Extract scraped date
        HOST_NAME,              -- Extract host name

        -- Convert 'host_since' column to DATE format if it matches the expected pattern (DD/MM/YYYY), otherwise set to NULL.

    CASE
        WHEN host_since ~ '^\d{1,2}/\d{1,2}/\d{4}$' THEN TO_DATE(host_since, 'DD/MM/YYYY') --format host_since column to DATE
        ELSE NULL
    END AS host_since_date,

    host_is_superhost, -- Extract superhost status

    -- Convert 'HOST_NEIGHBOURHOOD' to uppercase and set to NULL if it's 'NAN' ( for case-insensitive matching with LGA).

    CASE
        WHEN UPPER(HOST_NEIGHBOURHOOD) = 'NAN' THEN NULL
        ELSE UPPER(HOST_NEIGHBOURHOOD) --MAKE IT UPPER CASE TO MATCH LGA
    END as HOST_NEIGHBOURHOOD,
    dbt_valid_to,  -- Extract 'dbt_valid_to'
    dbt_valid_from as last_updated_at -- Extract 'dbt_valid_from' as 'last_updated_at'
    from source
    where dbt_valid_to IS NULL -- Filter for current records
)

    -- Finally, select and return the cleaned host data for the dimension table.

select
    HOST_ID,
    scraped_date,
    HOST_NAME,
    host_since_date,
    host_is_superhost,
    HOST_NEIGHBOURHOOD,
    last_updated_at

from cleaned_host
```

In this layer, we formatted the 'host_since' column, converted 'host_neighbourhood' to uppercase to match LGA columns, and applied data filtering for the current records.

- ***Stg_LGA***

```sql
-- stg_LGA.sql

-- Configuration section: Set the unique key to 'LGA_CODE' for this data and this SQL script defines a materialized view.

{{
    config(
        unique_key='LGA_CODE',
        materialized='view'
    )
}}

-- Define a Common Table Expression (CTE) named "source" to retrieve data from the 'nsw_lga_code' raw source.

with source as (
    select * from {{ source('raw','nsw_lga_code') }}
),

-- Create another CTE named "renamed" to rename and transform columns for clarity.


renamed as (
    select
        -- Rename the 'LGA_CODE' column as 'Local_Government_Area_Code'.
        LGA_CODE as Local_Government_Area_Code,
        -- Convert 'LGA_NAME' to uppercase for consistency,  to match it with other data.
        UPPER(LGA_NAME) as Local_Government_Area
    from source
)

-- Select and return the transformed data from the "renamed" CTE.

select * from renamed
```

In this layer, we transformed the 'lga_name' column to uppercase to ensure compatibility with the 'lga_suburb' table.

- ***Stg_room***

```sql
- Define a Common Table Expression (CTE) named "source" to retrieve data from the 'room_snapshot' reference.

ith source  as (

   select * from {{ ref('room_snapshot') }}

,

- Create another CTE named "current_record" to filter and select relevant columns from the data.

urrent_record as (
   select
       SCRAPED_DATE,                    -- Extract the scraped date
       LISTING_ID,                      -- Extract the listing ID
       ROOM_TYPE,                       -- Extract the room type
       dbt_valid_to,                    -- Extract 'dbt_valid_to'
       dbt_valid_from as last_updated_at  -- Extract 'dbt_valid_from' as 'last_updated_at'
   from source
   where dbt_valid_to IS NULL --when dbt_valid_to is equal to null it means data is still valid


- Select and return specific columns from the "current_record" CTE.

elect
       last_updated_at,
       LISTING_ID,
       ROOM_TYPE,
       SCRAPED_DATE

rom current_record
```

In this layer, we constructed a dimension table derived from the listing table, which also incorporated the 'room_type' attribute.

- ***Stg_property***

```sql
-- This SQL script appears to define a materialized view configuration and extracts data from a 'property_snapshot' reference.

-- Configuration section: Specifies that this SQL script defines a materialized view.
{{
    config(

        materialized='view'
    )
}}

-- Define a Common Table Expression (CTE) named "source" to retrieve data from the 'property_snapshot' reference.

WITH source AS (
    SELECT * FROM {{ ref('property_snapshot') }}
),
-- Create another CTE named "current_data" to filter and transform the data.
current_data AS (
    SELECT
        SCRAPED_DATE,                    -- Extract the scraped date
        LISTING_ID,                      -- Extract the listing ID
        PROPERTY_TYPE,                   -- Extract the property type
        UPPER(LISTING_NEIGHBOURHOOD) AS LISTING_NEIGHBOURHOOD,  -- Convert the neighborhood name to uppercase
        dbt_valid_to,                    -- Extract 'dbt_valid_to'
        dbt_valid_from as last_updated_at  -- Extract 'dbt_valid_from' as 'last_updated_at'
    FROM source
    WHERE dbt_valid_to IS NULL --current data
)
-- Select and return specific columns from the "current_data" CTE.
SELECT
    SCRAPED_DATE,
    LISTING_ID,
    PROPERTY_TYPE,
    last_updated_at,
    LISTING_NEIGHBOURHOOD
FROM current_data
```

In this layer, we constructed a dimension table derived from the listing table, which also incorporated the 'property_type' and ‘listing_neighbourhood” attributes.

- ***Stg_suburb***

```sql
-- stg_suburb.sql

-- This SQL script 'stg_suburb.sql' is to focus on transforming data related to suburbs within local government areas (LGAs) in New South Wale

-- Define a Common Table Expression (CTE) named "source" to retrieve data from the 'nsw_lga_suburb' raw source.


with source as (

    select * from {{ source('raw','nsw_lga_suburb') }}

),

-- Create another CTE named "renamed" to rename and transform columns for clarity.

renamed as (

    -- Rename the 'LGA_NAME' column as 'Local_Government_Area' for clearer understanding.
    -- Rename the 'SUBURB_NAME' column as 'Local_Government_Area_SUBURB' for consistency and clarity.

    select

    LGA_NAME as Local_Government_Area , --renaming column

    SUBURB_NAME as Local_Government_Area_SUBURB --renaming column

    from source

)
-- Select and return the transformed data from the "renamed" CTE.

select * from renamed
```

In this layer, we applied necessary column renaming.

## *3.3 Warehouse*

A star schema in the DBT warehouse layer organizes data into a central fact table and dimension tables for efficient querying and reporting.

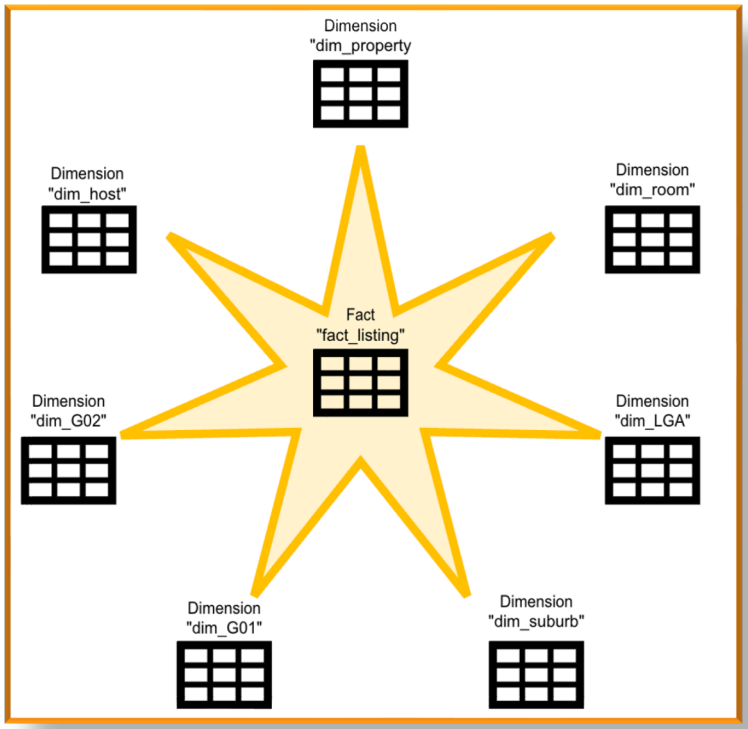



*Figure 10-Star Schema*

- **“dim_host”**

```sql
-- dim_host table
{{
    config(
        materialized = 'table',
        unique_key = 'surrogate_key_host'
    )
}}

WITH current_data AS (
    SELECT *

    FROM {{ ref('stg_host') }}
)

SELECT
    CONCAT(CAST(host_id AS VARCHAR), '_', CAST(scraped_date AS VARCHAR)) as surrogate_key_host,
    host_id,
    scraped_date,
    last_updated_at,
    host_name,
    host_since_date,
    host_is_superhost,
    host_neighbourhood

FROM current_data
```

This code establishes the data model for the 'dim_host' table within the data warehouse. The 'surrogate_key_host,' generated by combining 'host_id' and 'scraped_date,' is a distinct identifier for each row in the 'dim_host' table. The “surrogate_key_host” helps maintain uniqueness and aids in joining this dimension table with fact tables in a star schema or similar data modelling structures within the data warehouse.

- **“dim_property”**

```sql
-- dim_property table
{{
    config(
        materialized = 'table',
        unique_key = 'surrogate_key_property'
    )
}}

WITH current_data AS (
    SELECT *

    FROM {{ ref('stg_property') }}
)

SELECT
    CONCAT(CAST(listing_id AS VARCHAR), '_', CAST(scraped_date AS VARCHAR)) as surrogate_key_property,
    LISTING_ID,
    PROPERTY_TYPE,
    last_updated_at,
    LISTING_NEIGHBOURHOOD

FROM current_data
```

The 'dim_property' table employs a surrogate key that combines the 'listing_id' and 'scraped_date' attributes

- **“dim_room”**

```sql
-- dim_host table
{{
    config(
        materialized = 'table',
        unique_key = 'surrogate_key_room'
    )
}}

WITH current_data AS (
    SELECT *

    FROM {{ ref('stg_room') }}
)

SELECT
    CONCAT(CAST(listing_id AS VARCHAR), '_', CAST(scraped_date AS VARCHAR)) as surrogate_key_room,
        last_updated_at,
        ROOM_TYPE

FROM current_data
```

The 'dim_room' table employs a surrogate key that combines the 'listing_id' and 'scraped_date' attributes.

- **“dim_G01”**

```sql
--dim_G01.sql
{{
    config(
        materialized = 'table',
        unique_key='LGA_CODE_2016'
    )
}}

SELECT
    g1.*,
    lga.Local_Government_Area
FROM {{ ref('stg_G01') }} g1
JOIN {{ ref('stg_LGA') }} lga ON g1.LGA_CODE = lga.Local_Government_Area_Code
```

This code represents a SQL JOIN operation between two tables: “stg_LGA” and “stg_G01”. It's used to combine data from these two tables based on common columns.

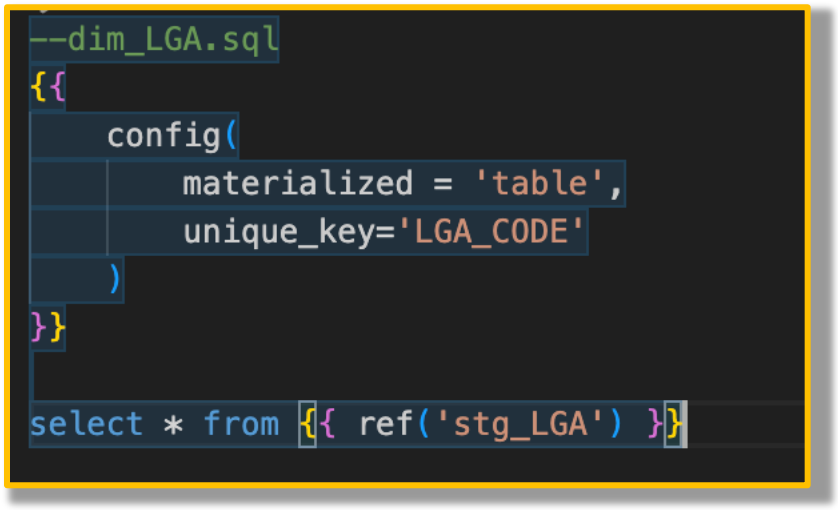
- **“dim_G02”**

```sql
--dim_G02.sql
{{
    config(
        materialized = 'table',
        unique_key='LGA_CODE'
    )
}}

SELECT
    g2.*,
    lga.Local_Government_Area
FROM {{ ref('stg_G02') }} g2
JOIN {{ ref('stg_LGA') }} lga ON g2.LGA_CODE = lga.Local_Government_Area_Code
```

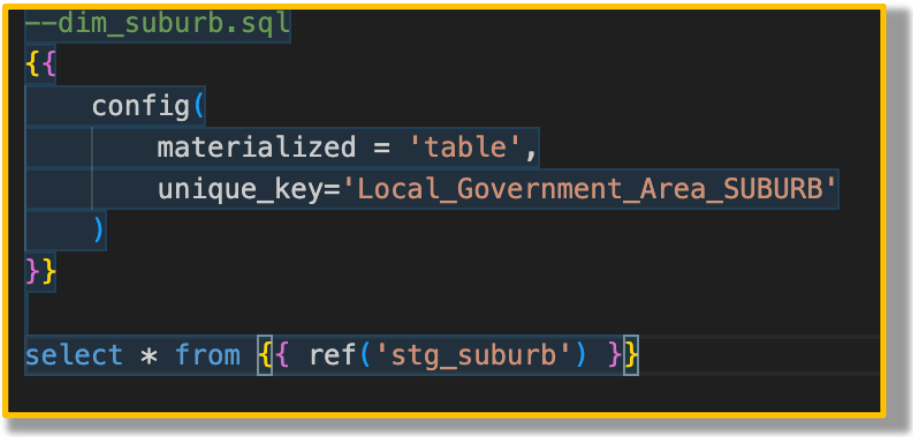
This code represents a SQL JOIN operation between two tables: “stg_LGA” and “stg_G02”. It's used to combine data from these two tables based on common columns.

- **“dim_LGA”**




This code configures and populates the "dim_LGA" table by selecting all data from the "stg_LGA" source table.

- **“dim_suburb”**




This code configures and populates the "dim_suburb" table by selecting all data from the "stg_suburb" source table.

### *3.4 Data Mart*

In this layer, we provided answers to the following questions.

- **“dm_listing_neighbourhood” view**

  a. Per “**listing_neighbourhood**” and “**month/year**”:
  - Active listings rate
  - Minimum, maximum, median and average price for active listings
  - Number of distinct hosts
  - Superhost rate
  - Average of review_scores_rating for active listings
  - Percentage change for active listings
  - Percentage change for inactive listings
  - Total Number of stays

- Average Estimated revenue per active listings

```sql
-- Create a common table expression (CTE) named sorted_listing
WITH sorted_listing AS (
    -- Select relevant columns from fact_listing and join with dimension tables
    SELECT
        prop.listing_neighbourhood,
        TO_CHAR(list.scraped_date, 'MM/YYYY') AS month_year,
        list.has_availability, -- Including the has_availability column
        list.listing_id, -- Including the listing_id column
        CASE
            WHEN list.has_availability = 't' THEN list.price
            ELSE NULL
        END AS price,
        host.host_id,
        host.host_is_superhost,
        review_scores_rating,
        availability_30
    FROM warehouse.fact_listing list
    LEFT JOIN warehouse.dim_property prop ON list.surrogate_key_property = prop.surrogate_key_property -- Left join for dim_property
    LEFT JOIN warehouse.dim_host host ON list.surrogate_key_host = host.surrogate_key_host -- Left join for dim_host
)
```

In this code, we established a Common Table Expression (CTE) named 'sorted_listing.' We then selected pertinent columns from the 'fact_listing' table and joined them with dimension tables.

```sql
SELECT
    listing_neighbourhood,
    month_year,
    ROUND(
        (CASE
            WHEN COUNT(DISTINCT CASE WHEN has_availability = 't' THEN listing_id END) > 0 THEN (COUNT(DISTINCT CASE WHEN has_availability = 't' T
            ELSE 0
        END), 2
    ) AS active_listing_rate,
    MIN(CASE WHEN has_availability = 't' THEN price END) AS min_price_active_listing,
    MAX(CASE WHEN has_availability = 't' THEN price END) AS max_price_active_listing,
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY price) AS median_price_active_listing,
    ROUND(
        AVG(CASE WHEN has_availability = 't' THEN price END), 2
    ) AS avg_price_active_listing,
    COUNT(DISTINCT host_id) AS number_of_distinct_hosts,
    ROUND(
        (CASE
            WHEN COUNT(DISTINCT CASE WHEN host_is_superhost = '1' THEN host_id END) > 0 THEN (COUNT(DISTINCT CASE WHEN host_is_superhost = '1' TH
            ELSE 0
        END), 2
    ) AS superhost_rate,
    ROUND(
        AVG(CASE WHEN has_availability = 't' THEN review_scores_rating END), 2
    ) AS avg_review_scores_rating_active,
```

```sql
    -- Calculate percentage change for active listings
    ROUND(
        ((COUNT(DISTINCT CASE WHEN has_availability = 't' THEN listing_id END) - LAG(COUNT(DISTINCT CASE WHEN has_availability = 't' THEN listing
    ) AS active_listings_percentage_change,
    -- Calculate percentage change for inactive listings
    ROUND(
        (CASE
            WHEN COUNT(DISTINCT CASE WHEN has_availability = 'f' THEN listing_id END) > 0 THEN (((COUNT(DISTINCT CASE WHEN has_availability = 'f'
            ELSE 0
        END), 2
    ) AS inactive_listings_percentage_change,
    -- Calculate the total number of stays
    SUM(CASE WHEN has_availability = 't' THEN (30 - availability_30) END) AS number_of_stays,
    -- Calculate the average estimated revenue per active listing
    ROUND(
        (CASE
            WHEN COUNT(DISTINCT CASE WHEN has_availability = 't' THEN listing_id END) > 0 THEN (SUM(CASE WHEN has_availability = 't' THEN (30 - a
            ELSE 0
        END), 2
    ) AS avg_estimated_revenue_per_active_listing
FROM sorted_listing
-- Group the results by listing_neighbourhood and month_year
GROUP BY listing_neighbourhood, month_year
```

Subsequently, we performed the required calculations using the 'sorted_listing' table to address the abovementioned questions.

Result:

| | listing_neighbourhood | month_year | active_listing_rate | min_price_active_listing | max_price_active_listing | median_price_active_listing |
|---|---|---|---|---|---|---|
| 2 | BAYSIDE | 02/2021 | 100 | 30 | 227 | 85 |
| 3 | BAYSIDE | 03/2021 | 100 | 25 | 2,833 | 114 |
| 4 | BAYSIDE | 04/2021 | 99.16 | 15 | 2,904 | 80 |
| 5 | BAYSIDE | 05/2020 | 100 | 18 | 2,544 | 119.5 |
| 6 | BAYSIDE | 07/2020 | 100 | 26 | 2,426 | 158.5 |
| 7 | BAYSIDE | 08/2020 | 100 | 19 | 158 | 69 |
| 8 | BAYSIDE | 09/2020 | 100 | 25 | 590 | 85 |
| 9 | BAYSIDE | 11/2020 | 100 | 15 | 500 | 80 |
| 10 | BAYSIDE | 12/2020 | 100 | 20 | 575 | 80 |

| | avg_price_active_listing | number_of_distinct_hosts | superhost_rate | avg_review_scores_rating_active | active_listings_percentage_change |
|---|---|---|---|---|---|
| 3 | 254.03 | 29 | 0 | 88.27 | 36.17 |
| 4 | 120.05 | 944 | 0 | 92.25 | 2,125 |
| 5 | 172.15 | 53 | 0 | 90.6 | -90.31 |
| 6 | 793.39 | 12 | 0 | 89.83 | -47.83 |
| 7 | 75.59 | 15 | 0 | 93.75 | -65.28 |
| 8 | 98.34 | 34 | 0 | 88.12 | 320 |
| 9 | 108.62 | 25 | 0 | 88.98 | -47.62 |
| 10 | 108.85 | 16 | 0 | 90.64 | -25.45 |

| inactive_listings_percentage_change | number_of_stays | avg_estimated_revenue_per_active_listing |
|---|---|---|
| 0 | 464 | 2,488.74 |
| 0 | 691 | 1,343.81 |
| 0 | 1,074 | 5,283.78 |
| [NULL] | 28,947 | 2,492.33 |
| 0 | 2,284 | 2,950.17 |
| 0 | 956 | 8,546.88 |
| 0 | 341 | 1,080.73 |
| 0 | 1,728 | 1,500.53 |
| 0 | 908 | 1,764.25 |
| 0 | 741 | 1,521.93 |

- **“dm_property_type”**

b. Per “**property_type**”, “**room_type**” ,“**accommodates**” and “**month/year**”:
   - Active listings rate
   - Minimum, maximum, median and average price for active listings
   - Number of distinct hosts
   - Superhost rate
   - Average of review_scores_rating for active listings
   - Percentage change for active listings
   - Percentage change for inactive listings
   - Total Number of stays
   - Average Estimated revenue per active listings

```sql
-- Create a Common Table Expression (CTE) called sorted_listing
-- This CTE retrieves data from multiple tables and calculates various metrics
-- It includes property_type, room_type, accommodates, and more.


WITH sorted_listing AS (
    SELECT
        prop.property_type,-- Property type
        room.room_type,-- Room type
        list.accommodates,-- Accommodates
        TO_CHAR(list.scraped_date, 'MM/YYYY') AS month_year, -- Truncate the date to the month
        list.has_availability, -- Including the has_availability column
        list.listing_id, -- Including the listing_id column
        CASE
            WHEN list.has_availability = 't' THEN list.price
            ELSE NULL
        END AS price,
        host.host_id, -- Host ID
        host.host_is_superhost,-- Host is superhost flag
        review_scores_rating,-- Review scores rating
        availability_30-- Availability for the next 30 days
    FROM warehouse.fact_listing list
    LEFT JOIN warehouse.dim_property prop ON list.surrogate_key_property = prop.surrogate_key_property -- Left join for dim_property
    LEFT JOIN warehouse.dim_host host ON list.surrogate_key_host = host.surrogate_key_host -- Left join for dim_host
    LEFT JOIN warehouse.dim_room room ON list.surrogate_key_room = room.surrogate_key_room -- Left join for dim_room
)
```

In this code, we established a Common Table Expression (CTE) named 'sorted_listing.' We then selected pertinent columns from the 'fact_listing' table and joined them with dimension tables.

```sql
-- Select the aggregated metrics for the property_type, room_type, accommodates, and month_year
SELECT
    property_type,-- Property type
    room_type,-- Room type
    accommodates,-- Number of accommodates
    month_year,-- Truncated month and year
    ROUND(
        (CASE
            WHEN COUNT(DISTINCT CASE WHEN has_availability = 't' THEN listing_id END) > 0 THEN (COUNT(DISTINCT CASE WHEN has_availability = 't' T
            ELSE 0
        END), 2
    ) AS active_listing_rate,
    MIN(CASE WHEN has_availability = 't' THEN price END) AS min_price_active_listing,-- Calculate the minimum price for active listings
    MAX(CASE WHEN has_availability = 't' THEN price END) AS max_price_active_listing,-- Calculate the maximum price for active listings
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY price) AS median_price_active_listing, -- Calculate the median price for active listings
    ROUND(
        AVG(CASE WHEN has_availability = 't' THEN price END), 2
    ) AS avg_price_active_listing,
    COUNT(DISTINCT host_id) AS number_of_distinct_hosts, -- Count the number of distinct hosts
     -- Calculate the superhost rate
    ROUND(
        (CASE
            WHEN COUNT(DISTINCT CASE WHEN host_is_superhost = '1' THEN host_id END) > 0 THEN (COUNT(DISTINCT CASE WHEN host_is_superhost = '1' TH
            ELSE 0
        END), 2
    ) AS superhost_rate,
     -- Calculate the average review scores rating for active listings
    ROUND(
        AVG(CASE WHEN has_availability = 't' THEN review_scores_rating END), 2
    ) AS avg_review_scores_rating_active,
```

```sql
     -- Calculate the percentage change for active listing
    ROUND(
        ((COUNT(DISTINCT CASE WHEN has_availability = 't' THEN listing_id END) - LAG(COUNT(DISTINCT CASE WHEN has_availability = 't' THEN listing
    ) AS active_listings_percentage_change,
    -- Calculate percentage change for inactive listing
    ROUND(
        (CASE
            WHEN COUNT(DISTINCT CASE WHEN has_availability = 'f' THEN listing_id END) > 0 THEN (((COUNT(DISTINCT CASE WHEN has_availability = 'f'
            ELSE 0
        END), 2
    ) AS inactive_listings_percentage_change,
    -- Calculate the total number of stays
    SUM(CASE WHEN has_availability = 't' THEN (30 - availability_30) END) AS number_of_stays,
    -- Calculate the average estimated revenue per active listing

    ROUND(
        (CASE
            WHEN COUNT(DISTINCT CASE WHEN has_availability = 't' THEN listing_id END) > 0 THEN (SUM(CASE WHEN has_availability = 't' THEN (30 - a
            ELSE 0
        END), 2
    ) AS avg_estimated_revenue_per_active_listing
FROM sorted_listing
GROUP BY property_type, room_type, accommodates, month_year
```

Subsequently, we performed the required calculations using the 'sorted_listing' table to address the abovementioned questions.

Result:

| | property_type | room_type | accommodates | month_year | active_listing_rate | min_price_active_listing |
|---|---|---|---|---|---|---|
| 2 | Aparthotel | Hotel room | 2 | 05/2020 | 100 | 47 |
| 3 | Apartment | Entire home/apt | 1 | 05/2020 | 100 | 55 |
| 4 | Apartment | Entire home/apt | 1 | 07/2020 | 100 | 95 |
| 5 | Apartment | Entire home/apt | 2 | 05/2020 | 100 | 20 |
| 6 | Apartment | Entire home/apt | 2 | 07/2020 | 100 | 36 |
| 7 | Apartment | Entire home/apt | 3 | 05/2020 | 100 | 69 |
| 8 | Apartment | Entire home/apt | 3 | 07/2020 | 100 | 14 |
| 9 | Apartment | Entire home/apt | 4 | 05/2020 | 100 | 60 |
| 10 | Apartment | Entire home/apt | 4 | 07/2020 | 100 | 55 |

| | max_price_active_listing | median_price_active_listing | avg_price_active_listing | number_of_distinct_hosts | superhost_rate |
|---|---|---|---|---|---|
| 1 | 108 | 103 | 103 | 0 | 0 |
| 2 | 47 | 47 | 47 | 0 | 0 |
| 3 | 150 | 77 | 98.6 | 3 | 0 |
| 4 | 2,157 | 101 | 751.57 | 2 | 0 |
| 5 | 2,546 | 130 | 349.89 | 213 | 0 |
| 6 | 2,711 | 2,157 | 1,417.57 | 87 | 0 |
| 7 | 2,544 | 139 | 278.59 | 46 | 0 |
| 8 | 2,164 | 116 | 216.77 | 20 | 0 |
| 9 | 10,000 | 165 | 246.92 | 184 | 0 |
| 10 | 2,164 | 159 | 386.53 | 82 | 0 |

| | rate | avg_review_scores_rating_active | active_listings_percentage_change | inactive_listings_percentage_change | number_of_stays | avg_estimated_revenue_per_active_listing |
|---|---|---|---|---|---|---|
| 2 | 0 | 78 | [NULL] | 0 | 1 | 47 |
| 3 | 0 | 97 | [NULL] | 0 | 111 | 1,959 |
| 4 | 0 | 76 | [NULL] | 0 | 123 | 19,500.14 |
| 5 | 0 | 92.2 | [NULL] | 0 | 6,834 | 5,091.66 |
| 6 | 0 | 91.18 | [NULL] | 0 | 5,708 | 11,342.61 |
| 7 | 0 | 89.35 | [NULL] | 0 | 1,866 | 3,365.75 |
| 8 | 0 | 88.95 | [NULL] | 0 | 974 | 4,706.62 |
| 9 | 0 | 91.71 | [NULL] | 0 | 7,434 | 3,210.56 |
| 10 | 0 | 87.25 | [NULL] | 0 | 4,353 | 4,297.25 |

- **“dm_host_neighbourhood”**

c. Per “**host_neighbourhood_lga**” which is “**host_neighbourhood” transformed to an LGA** (e.g host_neighbourhood = 'Bondi' then you need to create host_neighbourhood_lga = 'Waverley') and “**month/year**”:
   - Number of distinct host
   - Estimated Revenue
   - Estimated Revenue per host (distinct)

```
-- Create a Common Table Expression (CTE) called host_neighbourhood_lga_transform
-- This CTE retrieves data from the dim_host and dim_suburb tables and transforms it.
-- It includes host data, host neighborhood's local government area, and month_year.

WITH host_neighbourhood_lga_transform AS (
    SELECT
        host.*,
        suburb.Local_Government_Area AS host_neighbourhood_lga,
        TO_CHAR(host.scraped_date, 'MM/YYYY') AS month_year
    FROM warehouse.dim_host host
    LEFT JOIN warehouse.dim_suburb suburb ON host.host_neighbourhood = suburb.Local_Government_Area_SUBURB
),
-- Create a CTE called distinct_host_count
-- This CTE calculates the number of distinct hosts for each host_neighbourhood_lga and month_year.
distinct_host_count AS (
    SELECT
        TO_CHAR(ht.scraped_date, 'MM/YYYY') AS month_year,
        ht.host_neighbourhood_lga,
        COUNT(DISTINCT ht.host_id) AS num_distinct_hosts
    FROM host_neighbourhood_lga_transform ht
    GROUP BY TO_CHAR(ht.scraped_date, 'MM/YYYY'),ht.host_neighbourhood_lga
),
```

First CTE retrieves data from the dim_host and dim_suburb tables and transforms it.

The second CTE calculates the number of distinct hosts for each host_neighbourhood_lga and month_year.

```sql
-- Create a CTE called estimated_revenue
-- This CTE calculates the total estimated revenue for each host_neighbourhood_lga and month_year.
estimated_revenue AS (
    SELECT
        TO_CHAR(list.scraped_date, 'MM/YYYY') AS month_year,
        ht.host_neighbourhood_lga,
        SUM((30 - list.availability_30) * list.price) AS total_estimated_revenue
    FROM warehouse.fact_listing list
    LEFT JOIN host_neighbourhood_lga_transform ht ON list.surrogate_key_host = ht.surrogate_key_host
    WHERE list.has_availability = 't'
    GROUP BY
        TO_CHAR(list.scraped_date, 'MM/YYYY'),
        ht.host_neighbourhood_lga
)
```

This CTE calculates the total estimated revenue for each host_neighbourhood_lga and month_year.

```sql
-- Select the aggregated metrics including month_year, host_neighbourhood_lga, number of distinct hosts, total estimated revenue, and estimated r
SELECT
    dh.host_neighbourhood_lga,
    dh.month_year,
    dh.num_distinct_hosts,
    er.total_estimated_revenue,
    CASE
        WHEN dh.num_distinct_hosts > 0 THEN ROUND(er.total_estimated_revenue / dh.num_distinct_hosts, 2)
        ELSE 0
    END AS estimated_revenue_per_host
FROM distinct_host_count dh
LEFT JOIN estimated_revenue er ON dh.month_year = er.month_year AND dh.host_neighbourhood_lga = er.host_neighbourhood_lga
ORDER BY dh.host_neighbourhood_lga, dh.month_year
```

This code selects the aggregated metrics, including month_year, host_neighbourhood_lga, number of distinct hosts, total estimated revenue, and estimated revenue per host.

Result:

| | host_neighbourhood_lga | month_year | num_distinct_hosts | total_estimated_revenue | estimated_revenue_per_host |
|---|---|---|---|---|---|
| 1 | ARMIDALE REGIONAL | 04/2021 | 2 | 4,800 | 2,400 |
| 2 | BATHURST REGIONAL | 04/2021 | 7 | 166,946 | 23,849.43 |
| 3 | BATHURST REGIONAL | 05/2020 | 1 | 9,000 | 9,000 |
| 4 | BAYSIDE | 01/2021 | 9 | 35,827 | 3,980.78 |
| 5 | BAYSIDE | 02/2021 | 7 | 9,625 | 1,375 |
| 6 | BAYSIDE | 03/2021 | 20 | 44,596 | 2,229.8 |
| 7 | BAYSIDE | 04/2021 | 647 | 1,551,276 | 2,397.64 |
| 8 | BAYSIDE | 05/2020 | 41 | 81,709 | 1,992.9 |
| 9 | BAYSIDE | 07/2020 | 16 | 20,823 | 1,301.44 |
| 10 | BAYSIDE | 08/2020 | 7 | 7,169.3 | 1,024.19 |

# 5.Ad-hoc Analysis

***a. Distinguishing Population Characteristics: A Comparison of the Top-Performing and Lowest-Performing 'Listing_neighborhoods' in the Past 12 Months***

| | listing_neighbourhood | avg_estimated_revenue_per_active_listing |
|---|---|---|
| 1 | LIVERPOOL | 25,639.58 |
| 2 | CAMDEN | 10 |

Liverpool stands out as the top-performing listing neighbourhood, while Camden is positioned at the bottom regarding the average estimated revenue per active listing.

| | percentage_population__under_20 | percentage_population_between_20_and_55 | percentage_population__above_55 | percentage_population_above_75 |
|---|---|---|---|---|
| 1 | 29.9462623455 | 49.4978612609 | 20.5578340495 | 4.2657322122 |
| 2 | 30.7601830781 | 49.2751029175 | 19.9774987854 | 4.1831803421 |

The worst-performing and best-performing neighborhoods show little variation in their age-related demographics.

| | precentage_total_birthplace_elsewhere_pop | precentage_total_birthplace_australia_popul | precentage_total_male_population | precentage_total_female_population |
|---|---|---|---|---|
| 1 | 40 | 51 | 49 | 50 |
| 2 | 17 | 77 | 49 | 50 |

It appears that the worst-performing neighbourhood has a larger population of individuals born in Australia, while the best-performing neighbourhood has a higher proportion of individuals born elsewhere. However, in terms of male and female populations, both neighbourhoods exhibit similar distributions.

### *b. Optimal Listing Types for Highest Stay Numbers in Top 5 Listing Neighbourhoods*

| | listing_neighbourhood | room_type | property_type | accommodates | total_revenue | total_number_of_stays |
|---|---|---|---|---|---|---|
| 1 | NORTHERN BEACHES | Entire home/apt | Entire house | 8 | 9,393,442 | 519 |
| 2 | SYDNEY | Entire home/apt | Entire apartment | 2 | 7,575,744 | 2,201 |
| 3 | WAVERLEY | Entire home/apt | Entire apartment | 4 | 6,205,966.9 | 1,052 |
| 4 | RANDWICK | Entire home/apt | Entire apartment | 4 | 2,601,878.9 | 569 |
| 5 | WOOLLAHRA | Entire home/apt | Entire apartment | 4 | 1,477,871 | 246 |

Here are the top five combinations of listing neighbourhood, room type, property type, and accommodations that yield the highest revenue.

### *c. Hosts with Multiple Listings: Do They Prefer the Same LGA for Their Properties?*

| | host_id | host_neighborhood | listing_neighborhood | num_listings | same_lga |
|---|---|---|---|---|---|
| 1 | 16,474 | Ultimo | Sydney | 2 | f |
| 2 | 17,061 | Pyrmont | Sydney | 19 | f |
| 3 | 17,331 | North Bondi | Waverley | 20 | f |
| 4 | 18,459 | Darlinghurst | Sydney | 10 | f |
| 5 | 20,258 | Redfern | Sydney | 5 | f |
| 6 | 27,184 | Ultimo | Sydney | 4 | f |
| 7 | 33,294 | Greenwich | Lane Cove | 5 | f |

a significant majority of hosts with multiple listings, 16,877 out of 18,416 cases, tend to list properties in different LGAs compared to where they reside, with only 1,539 cases indicating the same LGA for both their residence and property listings.

### *d. Assessing Unique Listing Hosts: Can Estimated Revenue Cover Annualized Median Mortgage Repayment?*

| | host_id | month_year | listing_neighbourhood | local_government_area_code | num_listings | revenue |
|---|---|---|---|---|---|---|
| 1 | 10,857 | 2021-04-12 00:00:00.000 | SUTHERLAND SHIRE | 17,150 | 1 | 3,402 |
| 2 | 16,474 | 2020-08-21 00:00:00.000 | SYDNEY | 17,200 | 1 | 390 |
| 3 | 17,331 | 2021-04-13 00:00:00.000 | WAVERLEY | 18,050 | 1 | 10,500 |
| 4 | 18,459 | 2021-04-12 00:00:00.000 | SYDNEY | 17,200 | 1 | 681 |
| 5 | 20,258 | 2020-09-11 00:00:00.000 | SYDNEY | 17,200 | 1 | 0 |
| 6 | 21,741 | 2021-04-11 00:00:00.000 | NORTHERN BEACHES | 15,990 | 1 | 3,750 |
| 7 | 27,184 | 2020-09-11 00:00:00.000 | SYDNEY | 17,200 | 1 | 8,700 |

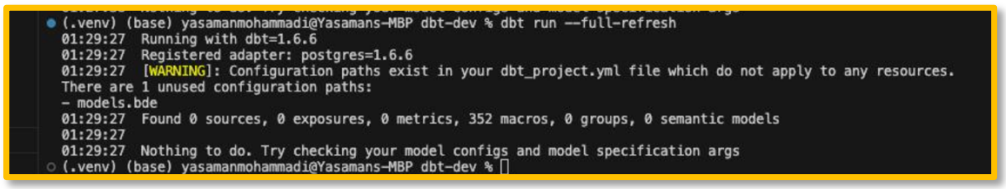
| median_mortgage_repayment_monthly | is_revenue_greater_than_mortgage |
|---|---|
| 2,427 | t |
| 2,499 | f |
| 3,000 | t |
| 2,499 | f |
| 2,499 | f |
| 2,800 | t |
| 2,499 | t |

the analysis of hosts with a unique listing shows that a substantial portion, 14,147 out of 28,535 cases, can cover the annualized median mortgage repayment of their listing's "listing_neighbourhood." However, 14,388 hosts in this category cannot cover this expense, indicating a relatively even split.

## 6.Issues:

⚠ First issue Resolution:

Erorr: Nothing to do.




This was solved by arrange folder correctly.

⚠Second Issue Resolution:

The error encountered, "Error loading configuration: could not open dbt_cloud.yml," was successfully resolved. The problem arose from working within the base conda environment, which lacked access to the required conda environment. To address this issue, a new conda environment was created, and both "dbt" and "dbt_postgres" were installed within the new environment.

# 6. Appendix

GitHub repository: https://github.com/JYasimo/dbt

# 7. References


1. dbt. (n.d.). Staging models. dbt Documentation. https://docs.getdbt.com/guides/dbt-ecosystem/dbt-python-snowpark/8-sources-and-staging#:~:text=Staging%20models%20are%20the%20base,useful%20models%20into%20the%20project.