94693 – Big Data Engineering

Assignment 3

Data pipelines with Airflow

Done by: Justin Qi (14337911)

Contents

1. Introduction 3

2. Data ingestion (Part 1) 4

3. Data mart usage 10

4. Automate ELT process using Airflow DAG 15

5. Business Questions 18

6. Conclusion 24

1. Introduction

1.1. Business understanding

This report aims to serve as a handover report to future users seeking a clear explanation on how to perform the following task scopes on Snowflake alongside Airflow:

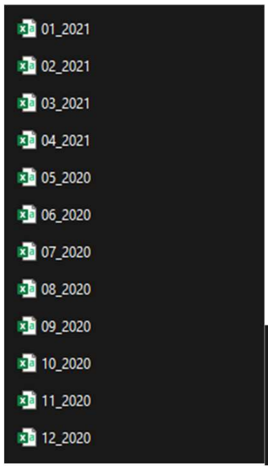
1. Data ingestion and warehouse design
2. Data mart usage
3. Automate ELT process using Airflow DAG
4. Business question analysis

Within this report I have documented the procedure, challenges, and thought process behind how

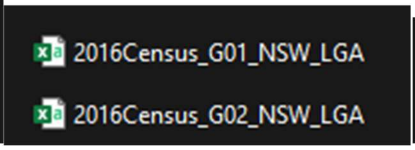
1.2. Data understanding

The datasets used for this assignment were the Airbnb listing data and LGA census data. The data is in the format of csv, separated into 3 files, one year worth of 'Airbnb listing' data, 2016 census at the LGA level, and a mapping dataset for LGAs.

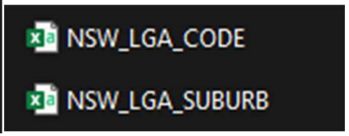
There were total of 12 listing data, 2 census data, and 2 mapping data



01_2021
02_2021
03_2021
04_2021
05_2020
06_2020
07_2020
08_2020
09_2020
10_2020
11_2020
12_2020



2016Census_G01_NSW_LGA
2016Census_G02_NSW_LGA

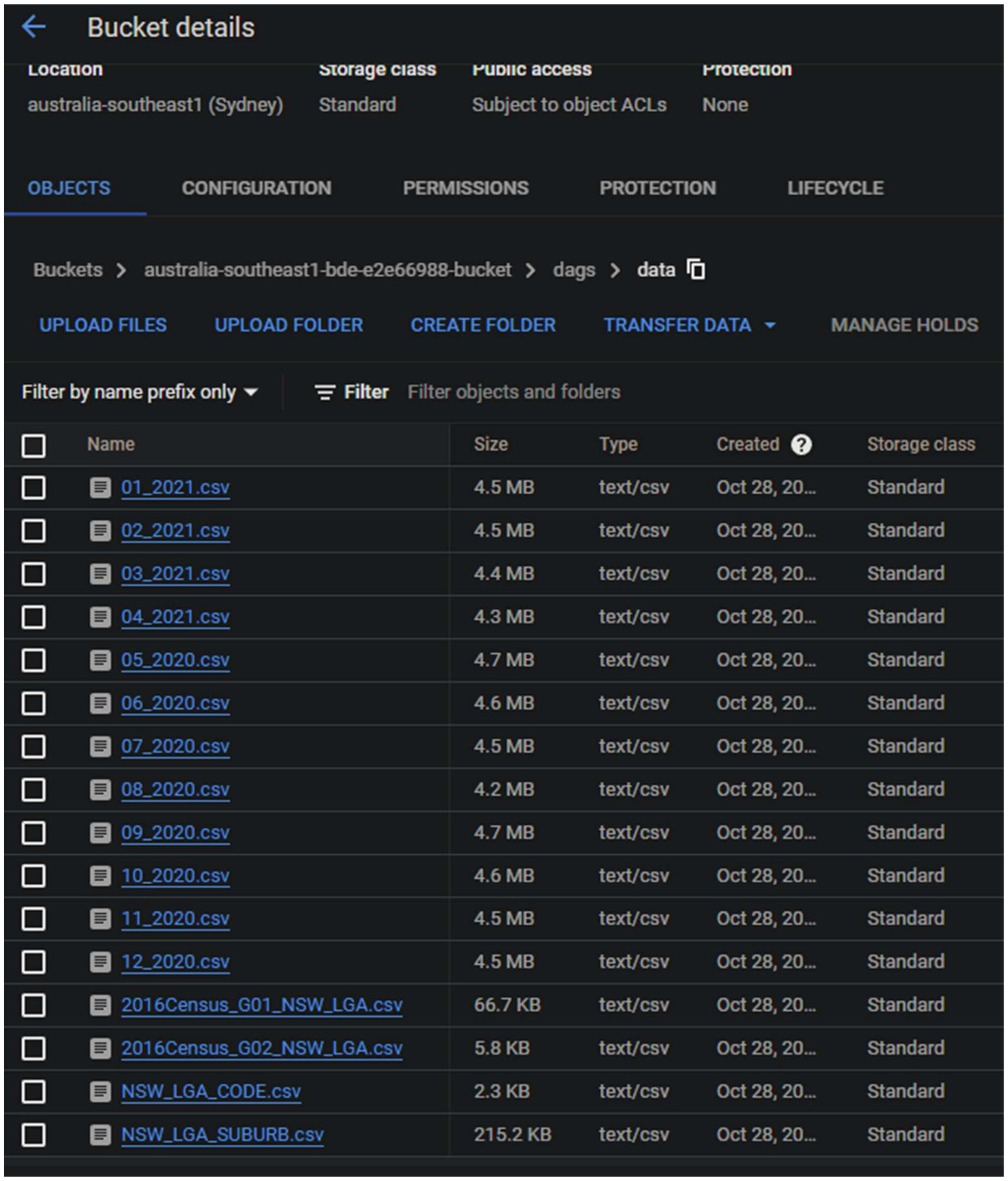


NSW_LGA_CODE
NSW_LGA_SUBURB

2. Data ingestion (Part 1)

2.1 Upload data into Google Cloud Storage

Visit your Cloud Storage bucket > Select 'data' folder > Upload the necessary files



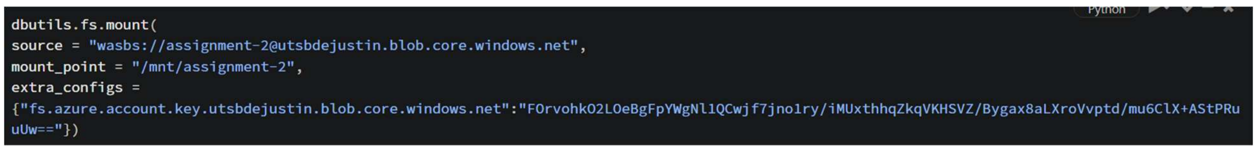
Bucket details

Location	Storage class	Public access	Protection
australia-southeast1 (Sydney)	Standard	Subject to object ACLs	None

Name	Size	Type	Created	Storage class
01_2021.csv	4.5 MB	text/csv	Oct 28, 20...	Standard
02_2021.csv	4.5 MB	text/csv	Oct 28, 20...	Standard
03_2021.csv	4.4 MB	text/csv	Oct 28, 20...	Standard
04_2021.csv	4.3 MB	text/csv	Oct 28, 20...	Standard
05_2020.csv	4.7 MB	text/csv	Oct 28, 20...	Standard
06_2020.csv	4.6 MB	text/csv	Oct 28, 20...	Standard
07_2020.csv	4.5 MB	text/csv	Oct 28, 20...	Standard
08_2020.csv	4.2 MB	text/csv	Oct 28, 20...	Standard
09_2020.csv	4.7 MB	text/csv	Oct 28, 20...	Standard
10_2020.csv	4.6 MB	text/csv	Oct 28, 20...	Standard
11_2020.csv	4.5 MB	text/csv	Oct 28, 20...	Standard
12_2020.csv	4.5 MB	text/csv	Oct 28, 20...	Standard
2016Census_G01_NSW_LGA.csv	66.7 KB	text/csv	Oct 28, 20...	Standard
2016Census_G02_NSW_LGA.csv	5.8 KB	text/csv	Oct 28, 20...	Standard
NSW_LGA_CODE.csv	2.3 KB	text/csv	Oct 28, 20...	Standard
NSW_LGA_SUBURB.csv	215.2 KB	text/csv	Oct 28, 20...	Standard

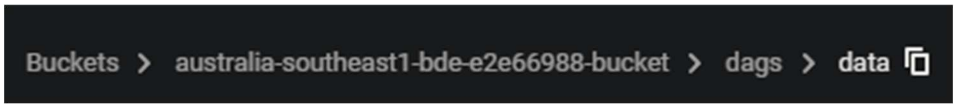
2.2 Set up Storage Integration between Azure and Databricks and copy it into DBFS

Mounting



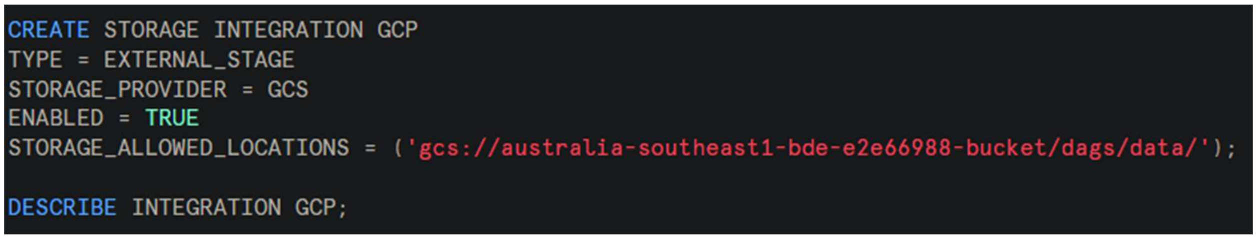
```
dbutils.fs.mount(
source = "wasbs://assignment-2@utsbdejustin.blob.core.windows.net",
mount_point = "/mnt/assignment-2",
extra_configs =
{"fs.azure.account.key.utsbdejustin.blob.core.windows.net":"FOrvohkO2LOeBgFpYWgNl1QCwjf7jno1ry/iMUxthhqZkqVKHSVZ/Bygax8aLXroVvptd/mu6ClX+AStPRu
uUw=="})
```

Copy path of data folder



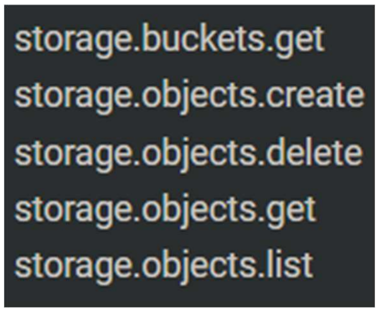
Buckets > australia-southeast1-bde-e2e66988-bucket > dags > data

Establish Google Cloud Storage connection on Snowflake

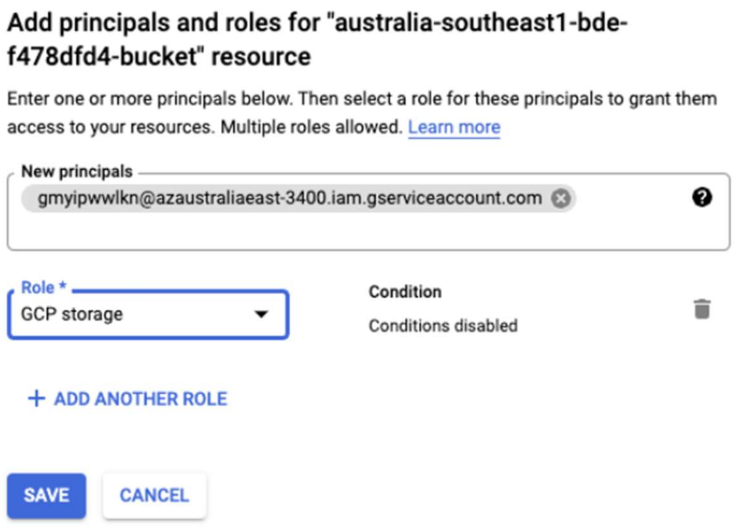


```
CREATE STORAGE INTEGRATION GCP
TYPE = EXTERNAL_STAGE
STORAGE_PROVIDER = GCS
ENABLED = TRUE
STORAGE_ALLOWED_LOCATIONS = ('gcs://australia-southeast1-bde-e2e66988-bucket/dags/data/');

DESCRIBE INTEGRATION GCP;
```

Create role, add permission in Google Cloud Storage



```
storage.buckets.get
storage.objects.create
storage.objects.delete
storage.objects.get
storage.objects.list
```



Add principals and roles for "australia-southeast1-bde-f478dfd4-bucket" resource

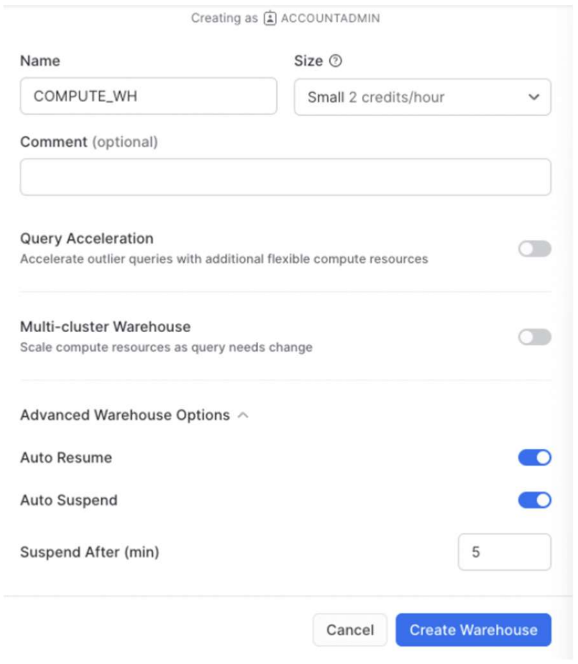
Enter one or more principals below. Then select a role for these principals to grant them access to your resources. Multiple roles allowed. Learn more

New principals: gmyipwwlkn@azaustraliaeast-3400.iam.gserviceaccount.com

Role *: GCP storage

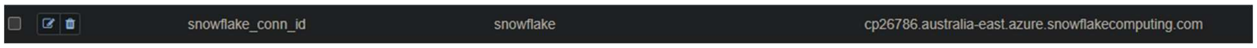
Condition: Conditions disabled

Create warehouse in Snowflake


Creating as ACCOUNTADMIN

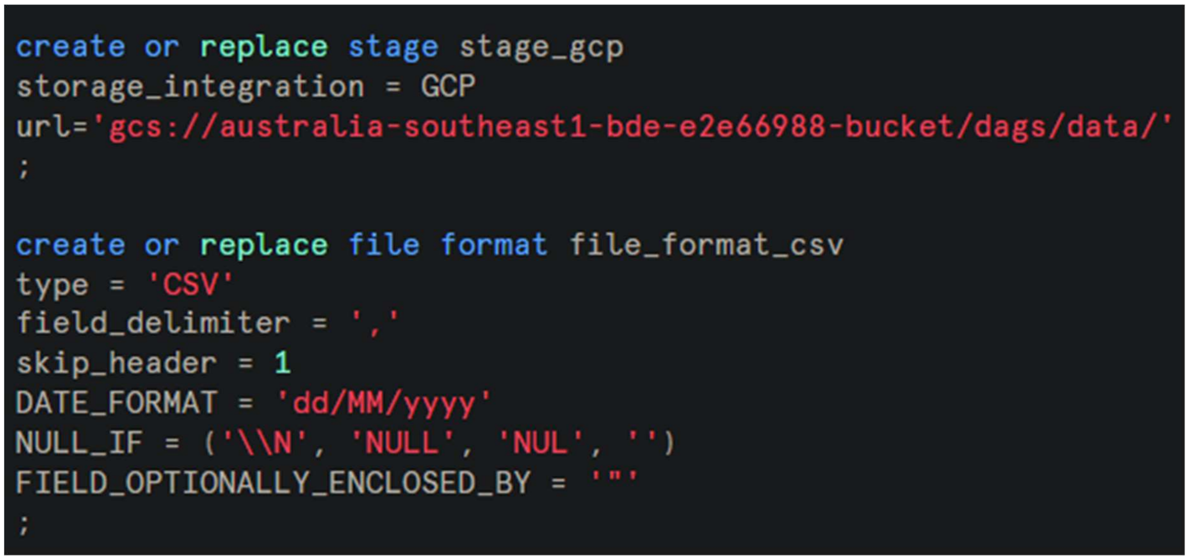
Name	COMPUTE_WH
Size	Small 2 credits/hour
Comment (optional)	
Query Acceleration — Accelerate outlier queries with additional flexible compute resources	Off
Multi-cluster Warehouse — Scale compute resources as query needs change	Off
Auto Resume	On
Auto Suspend	On
Suspend After (min)	5

Setup Snowflake connection in Airflow



snowflake_conn_id	snowflake	cp26786.australia-east.azure.snowflakecomputing.com

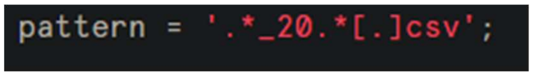
2.3 Begin extracting data from Google Cloud Storage into Snowflake



```
create or replace stage stage_gcp
storage_integration = GCP
url='gcs://australia-southeast1-bde-e2e66988-bucket/dags/data/'
;

create or replace file format file_format_csv
type = 'CSV'
field_delimiter = ','
skip_header = 1
DATE_FORMAT = 'dd/MM/yyyy'
NULL_IF = ('\\N', 'NULL', 'NUL', '')
FIELD_OPTIONALLY_ENCLOSED_BY = '"'
;
```

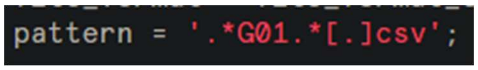
In this stage, we will create raw tables for listing, census, and LGA data.

We can identify them by setting pattern recognition of our data file name to as such:

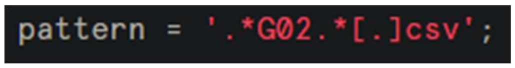


```
pattern = '.*_20.*[.]csv';
```

for listings

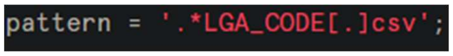


```
pattern = '.*G01.*[.]csv';
```

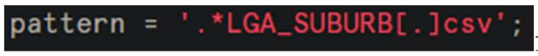


```
pattern = '.*G02.*[.]csv';
```

for census



```
pattern = '.*LGA_CODE[.]csv';
```



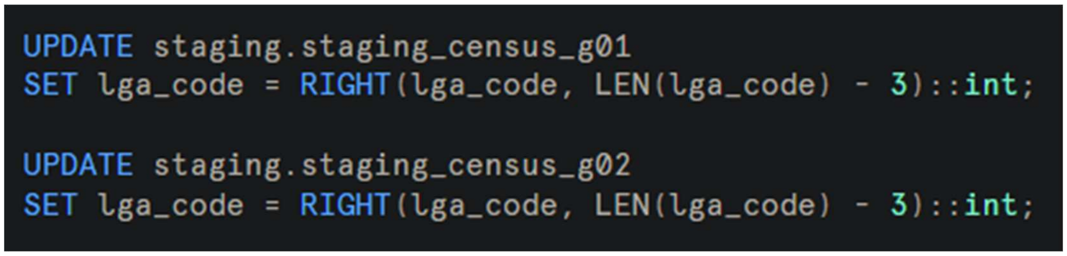
```
pattern = '.*LGA_SUBURB[.]csv';
```

for LGA

2.4 Create staging table to perform initial data cleaning on the tables

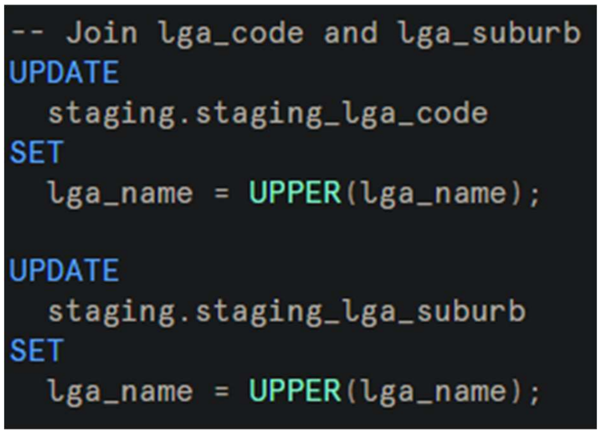
Data tables were created as per standard procedure, with the following cleaning procedures:

- As LGA_Code has 'LGA' prefix in front of the actual code we are interested in, we can remove them by using the below queries



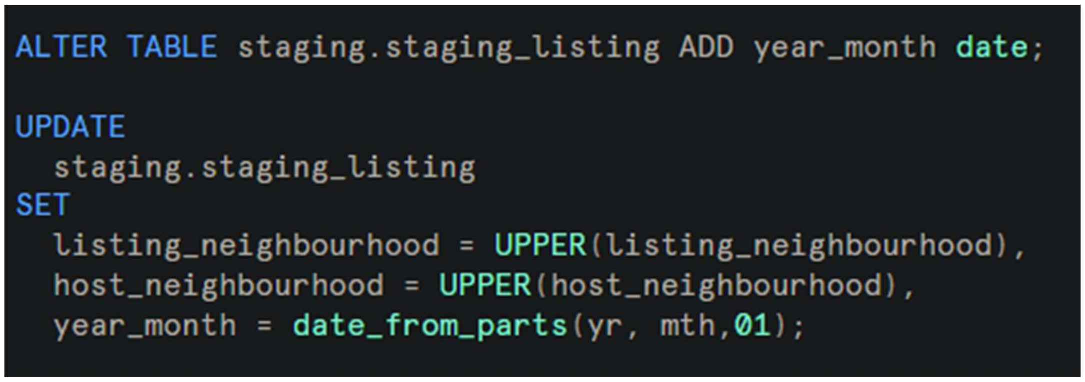
```
UPDATE staging.staging_census_g01
SET lga_code = RIGHT(lga_code, LEN(lga_code) - 3)::int;

UPDATE staging.staging_census_g02
SET lga_code = RIGHT(lga_code, LEN(lga_code) - 3)::int;
```

- Solving case sensitivity issue by changing all strings into uppercase



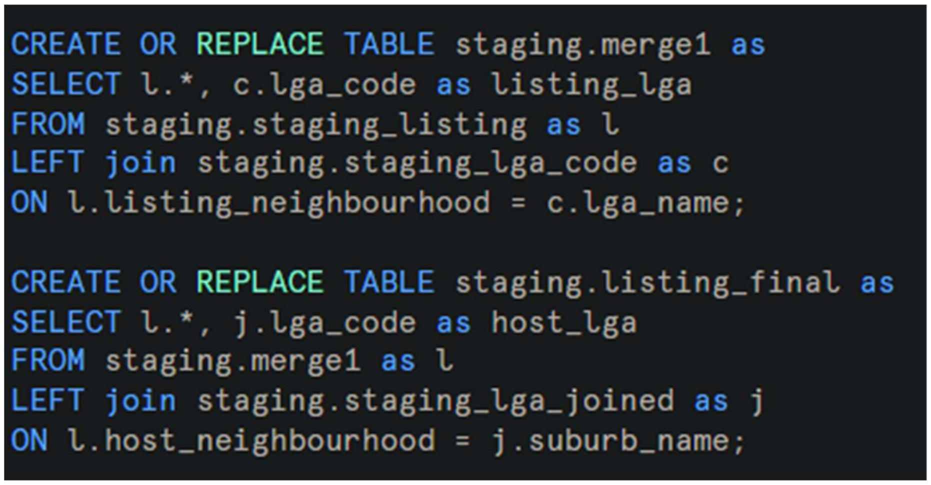
```
-- Join lga_code and lga_suburb
UPDATE
  staging.staging_lga_code
SET
  lga_name = UPPER(lga_name);

UPDATE
  staging.staging_lga_suburb
SET
  lga_name = UPPER(lga_name);
```

- In the listing table, we add a year_month column, by putting together date parts extracted from the file name itself.



```sql
ALTER TABLE staging.staging_listing ADD year_month date;

UPDATE
  staging.staging_listing
SET
  listing_neighbourhood = UPPER(listing_neighbourhood),
  host_neighbourhood = UPPER(host_neighbourhood),
  year_month = date_from_parts(yr, mth,01);
```

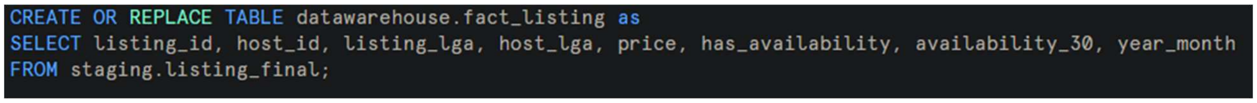
- Final listing table was done by adding listing_lga and host_lga, which are LGA codes that are going to be used as foreign keys in later steps.



```sql
CREATE OR REPLACE TABLE staging.merge1 as
SELECT l.*, c.lga_code as listing_lga
FROM staging.staging_listing as l
LEFT join staging.staging_lga_code as c
ON l.listing_neighbourhood = c.lga_name;

CREATE OR REPLACE TABLE staging.listing_final as
SELECT l.*, j.lga_code as host_lga
FROM staging.merge1 as l
LEFT join staging.staging_lga_joined as j
ON l.host_neighbourhood = j.suburb_name;
```

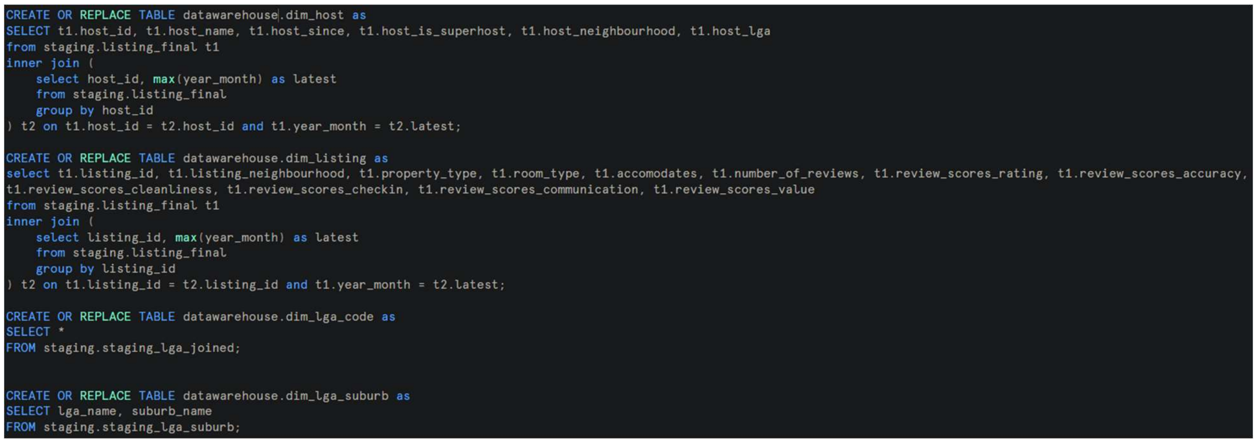
2.5 Design data warehouse by creating fact and dimension tables

As we are required to follow the star schema format, our fact_listing table will consist of all the foreign keys, which are listing_id, host_id, listing_lga, and host_lga, alongside other business metrics.



```sql
CREATE OR REPLACE TABLE datawarehouse.fact_listing as
SELECT listing_id, host_id, listing_lga, host_lga, price, has_availability, availability_30, year_month
FROM staging.listing_final;
```

For dimension tables on listing, host, LGA code, and LGA suburb, we include their respective IDs as primary key and relevant information as other attributes.

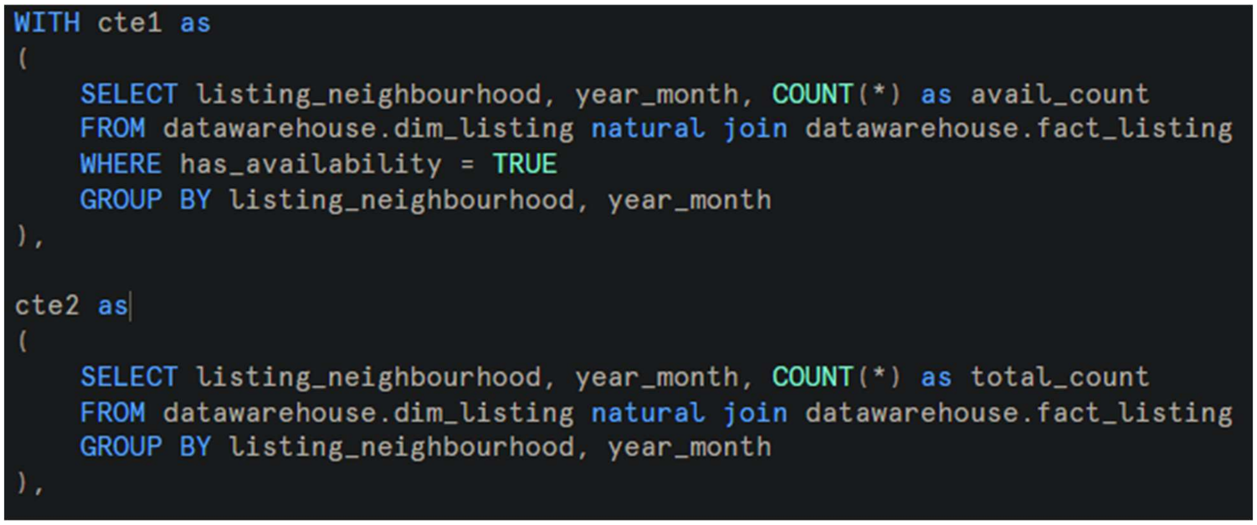

```
CREATE OR REPLACE TABLE datawarehouse.dim_host as
SELECT t1.host_id, t1.host_name, t1.host_since, t1.host_is_superhost, t1.host_neighbourhood, t1.host_lga
from staging.listing_final t1
inner join (
    select host_id, max(year_month) as latest
    from staging.listing_final
    group by host_id
) t2 on t1.host_id = t2.host_id and t1.year_month = t2.latest;

CREATE OR REPLACE TABLE datawarehouse.dim_listing as
select t1.listing_id, t1.listing_neighbourhood, t1.property_type, t1.room_type, t1.accomodates, t1.number_of_reviews, t1.review_scores_rating, t1.review_scores_accuracy,
t1.review_scores_cleanliness, t1.review_scores_checkin, t1.review_scores_communication, t1.review_scores_value
from staging.listing_final t1
inner join (
    select listing_id, max(year_month) as latest
    from staging.listing_final
    group by listing_id
) t2 on t1.listing_id = t2.listing_id and t1.year_month = t2.latest;

CREATE OR REPLACE TABLE datawarehouse.dim_lga_code as
SELECT *
FROM staging.staging_lga_joined;


CREATE OR REPLACE TABLE datawarehouse.dim_lga_suburb as
SELECT lga_name, suburb_name
FROM staging.staging_lga_suburb;
```

At this step, our star schema is designed with 1 fact table and 4 dimension tables, which we can use them to get necessary business insights by joining these tables in an easy-to-understand manner.

3. Data mart usage

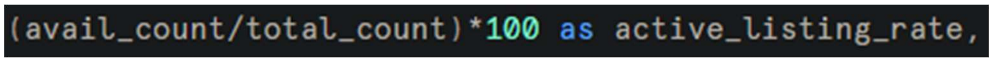
3.1 Create table dm_listing_neighbourhood based on the below requirements

- **Active listing rate**



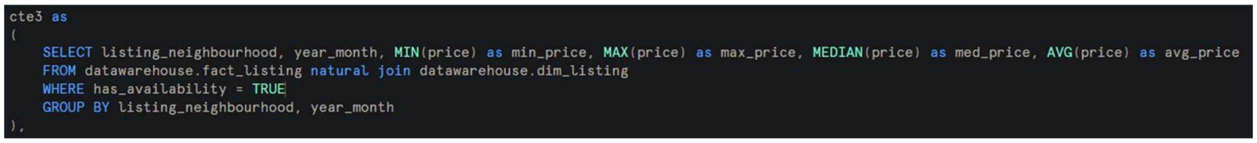
```
WITH cte1 as
(
    SELECT listing_neighbourhood, year_month, COUNT(*) as avail_count
    FROM datawarehouse.dim_listing natural join datawarehouse.fact_listing
    WHERE has_availability = TRUE
    GROUP BY listing_neighbourhood, year_month
),

cte2 as
(
    SELECT listing_neighbourhood, year_month, COUNT(*) as total_count
    FROM datawarehouse.dim_listing natural join datawarehouse.fact_listing
    GROUP BY listing_neighbourhood, year_month
),
```

By creating 2 CTEs, we get the active listing count and total listing counts respectively, then we can arrive at our business metrics by the following query



```
(avail_count/total_count)*100 as active_listing_rate,
```

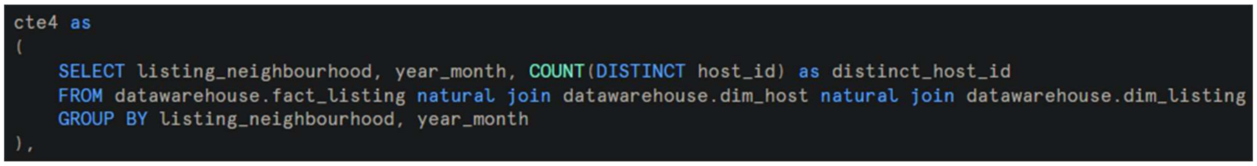
- **Minimum, maximum, median and average price for active listings**



```
cte3 as
(
    SELECT listing_neighbourhood, year_month, MIN(price) as min_price, MAX(price) as max_price, MEDIAN(price) as med_price, AVG(price) as avg_price
    FROM datawarehouse.fact_listing natural join datawarehouse.dim_listing
    WHERE has_availability = TRUE
    GROUP BY listing_neighbourhood, year_month
),
```

By creating a CTE and SQL aggregating functions, we extracs relevant information required

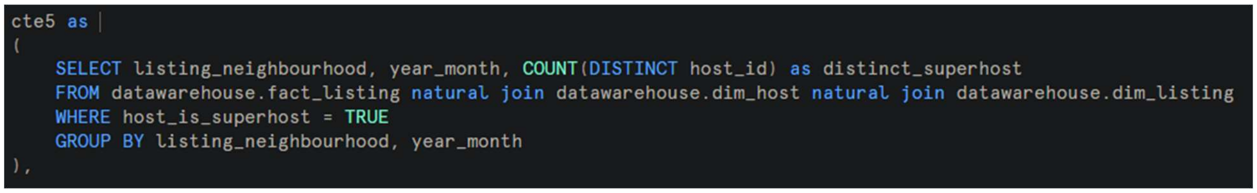
- **Number of distinct hosts**



```
cte4 as
(
    SELECT listing_neighbourhood, year_month, COUNT(DISTINCT host_id) as distinct_host_id
    FROM datawarehouse.fact_listing natural join datawarehouse.dim_host natural join datawarehouse.dim_listing
    GROUP BY listing_neighbourhood, year_month
),
```

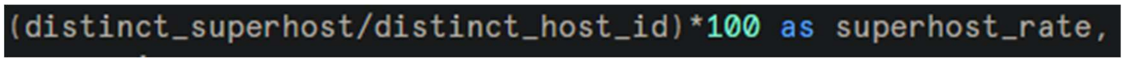
Simply count number of distinct host_id

- **Superhost rate**



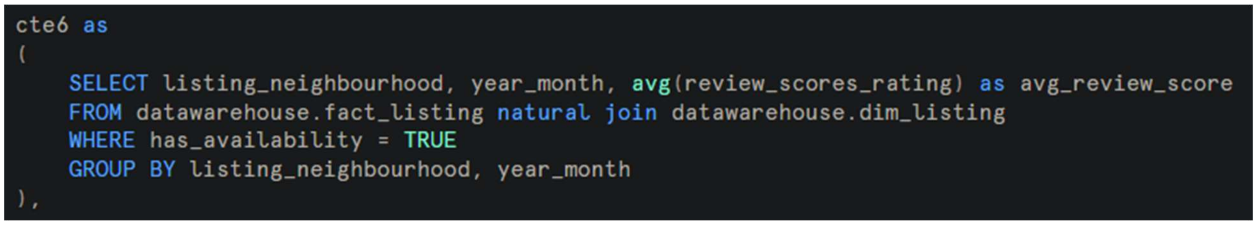
```
cte5 as
(
    SELECT listing_neighbourhood, year_month, COUNT(DISTINCT host_id) as distinct_superhost
    FROM datawarehouse.fact_listing natural join datawarehouse.dim_host natural join datawarehouse.dim_listing
    WHERE host_is_superhost = TRUE
    GROUP BY listing_neighbourhood, year_month
),
```

Simply count number of distinct host_id, setting condition to host_is_superhost to TRUE and apply the arithmetic function as below



```
(distinct_superhost/distinct_host_id)*100 as superhost_rate,
```

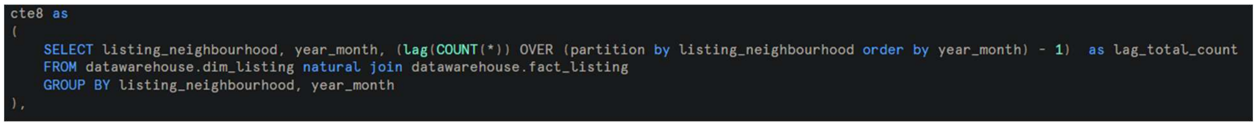
- **Average of review_scores_rating for active listings**



```
cte6 as
(
    SELECT listing_neighbourhood, year_month, avg(review_scores_rating) as avg_review_score
    FROM datawarehouse.fact_listing natural join datawarehouse.dim_listing
    WHERE has_availability = TRUE
    GROUP BY listing_neighbourhood, year_month
),
```

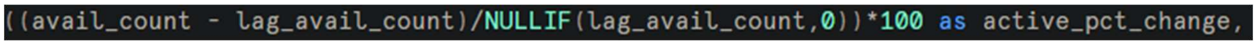
Aggregate review_scores_rating using AVG() function

- **Percentage change for active listings**



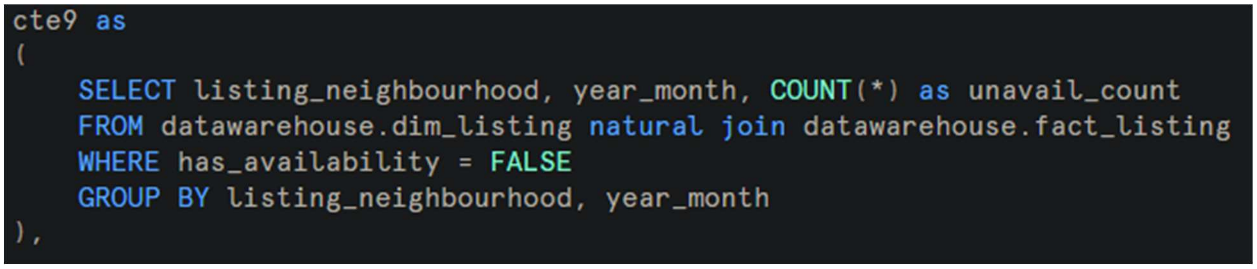
```
cte8 as
(
    SELECT listing_neighbourhood, year_month, (lag(COUNT(*)) OVER (partition by listing_neighbourhood order by year_month) - 1)  as lag_total_count
    FROM datawarehouse.dim_listing natural join datawarehouse.fact_listing
    GROUP BY listing_neighbourhood, year_month
),
```

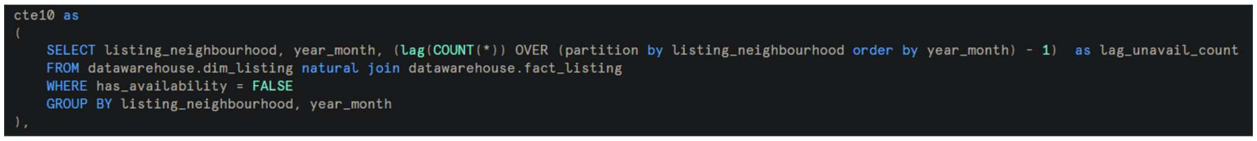
To get percentage change, we need to create another column that contains the number of active listing from the previous month, which can be done using the LAG() function. Avail_count was already derived earlier in cte1, hence we can compute the percentage change by the following query, added NULLIF() in case of division by zero error.



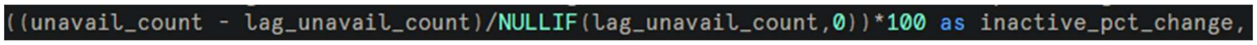
```
((avail_count - lag_avail_count)/NULLIF(lag_avail_count,0))*100 as active_pct_change,
```

- **Percentage change for inactive listings**

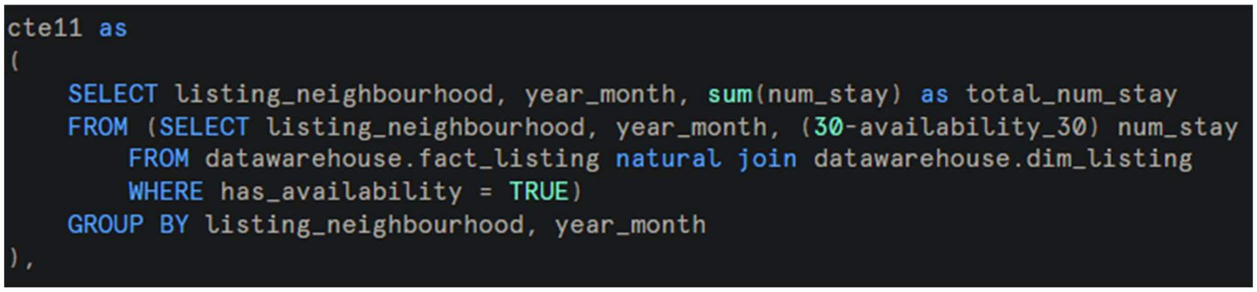

```
cte9 as
(
    SELECT listing_neighbourhood, year_month, COUNT(*) as unavail_count
    FROM datawarehouse.dim_listing natural join datawarehouse.fact_listing
    WHERE has_availability = FALSE
    GROUP BY listing_neighbourhood, year_month
),
```


```
cte10 as
(
    SELECT listing_neighbourhood, year_month, (lag(COUNT(*)) OVER (partition by listing_neighbourhood order by year_month) - 1)  as lag_unavail_count
    FROM datawarehouse.dim_listing natural join datawarehouse.fact_listing
    WHERE has_availability = FALSE
    GROUP BY listing_neighbourhood, year_month
),
```

Similar to previous metric, we require the current count and previous period count of unavailable listing units. We can then compute the percentage change by the following query, added NULLIF() in case of division by zero error.

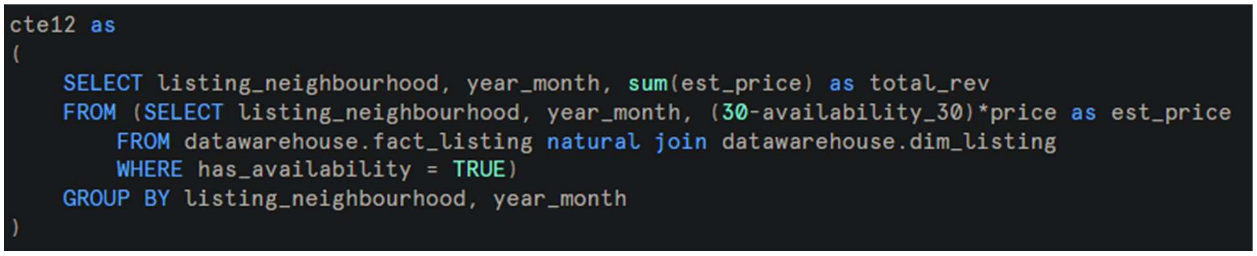

```
((unavail_count - lag_unavail_count)/NULLIF(lag_unavail_count,0))*100 as inactive_pct_change,
```

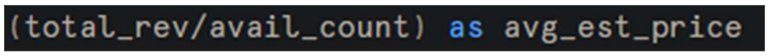
- **Total Number of stays**


```
cte11 as
(
    SELECT listing_neighbourhood, year_month, sum(num_stay) as total_num_stay
    FROM (SELECT listing_neighbourhood, year_month, (30-availability_30) num_stay
        FROM datawarehouse.fact_listing natural join datawarehouse.dim_listing
        WHERE has_availability = TRUE)
    GROUP BY listing_neighbourhood, year_month
),
```

First we find the number of stay for each listing, by taking 30 – available days left. Then we aggregate by summing with group_by function.

- **Average Estimated revenue per active listings**


```
cte12 as
(
    SELECT listing_neighbourhood, year_month, sum(est_price) as total_rev
    FROM (SELECT listing_neighbourhood, year_month, (30-availability_30)*price as est_price
        FROM datawarehouse.fact_listing natural join datawarehouse.dim_listing
        WHERE has_availability = TRUE)
    GROUP BY listing_neighbourhood, year_month
)
```


```
(total_rev/avail_count) as avg_est_price
```

Similar to previous metric, except we multiply days of stay with the daily price of listing, summing them up with group_by function to get total revenue. Finally, attain average price by taking total revenue divided by available listing count, which was derived in cte1.

With that, dm_listing_neighbourhood table was completed, for each listing neighbourhood and month.

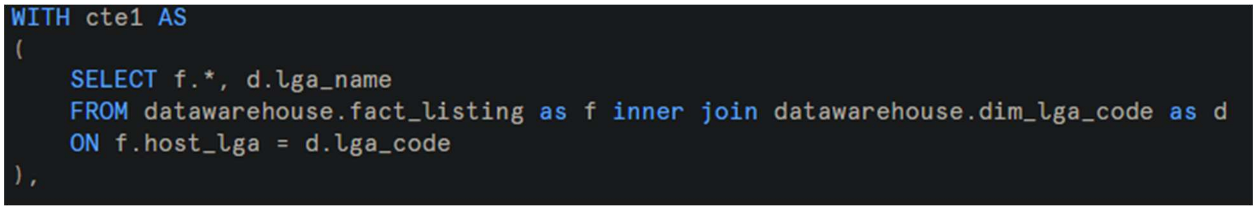
3.2 Create table dm_property_type based on the below requirements

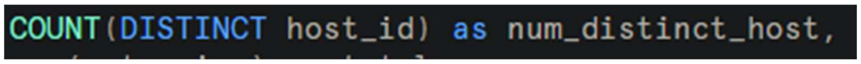
- **Active listings rate**
- **Minimum, maximum, median and average price for active listings**
- **Number of distinct hosts**
- **Superhost rate**
- **Average of review_scores_rating for active listings**
- **Percentage change for active listings**
- **Percentage change for inactive listings**
- **Total Number of stays**
- **Average Estimated revenue per active listings**

As the requirements are the same as what was required for dm_listing_neighbourhood, we can simply use the same query, changing listing_neighbourhood to property_type, room_type, accommodates

3.3 Create table dm_host_neighbourhood based on the below requirements

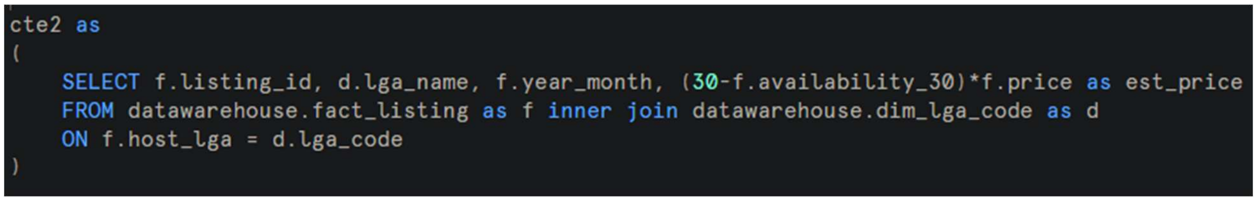
- **Number of distinct host**

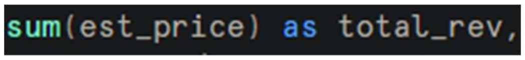

```
WITH cte1 AS
(
    SELECT f.*, d.lga_name
    FROM datawarehouse.fact_listing as f inner join datawarehouse.dim_lga_code as d
    ON f.host_lga = d.lga_code
),
```


```
COUNT(DISTINCT host_id) as num_distinct_host,
```

We can simply use COUNT() on distinct host_id

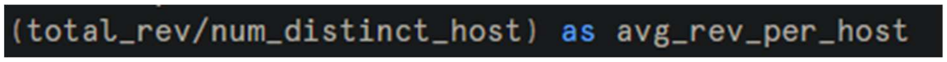
- **Estimated Revenue**


```
cte2 as
(
    SELECT f.listing_id, d.lga_name, f.year_month, (30-f.availability_30)*f.price as est_price
    FROM datawarehouse.fact_listing as f inner join datawarehouse.dim_lga_code as d
    ON f.host_lga = d.lga_code
)
```


```
sum(est_price) as total_rev,
```

We multiply days of stay with the daily price of listing, summing them up with group_by function to get total revenue of each LGA area.

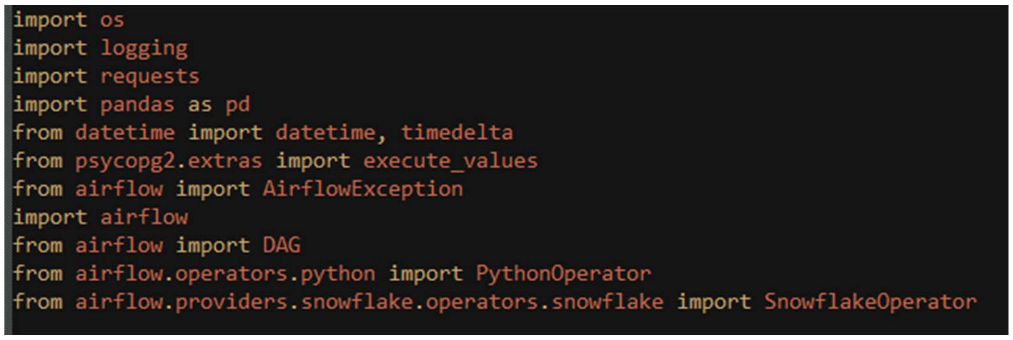
- **Estimated Revenue per host (distinct)**


```
(total_rev/num_distinct_host) as avg_rev_per_host
```

By using the number of distinct hosts derived in cte1, and total revenue derived in cte2, we can get average revenue per host by dividing the two metrics.

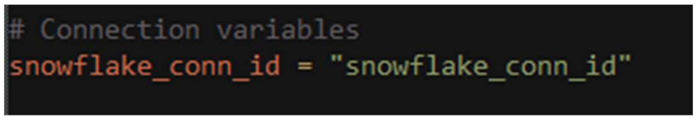
4. Automate ELT process using Airflow DAG

4.1 Packages used



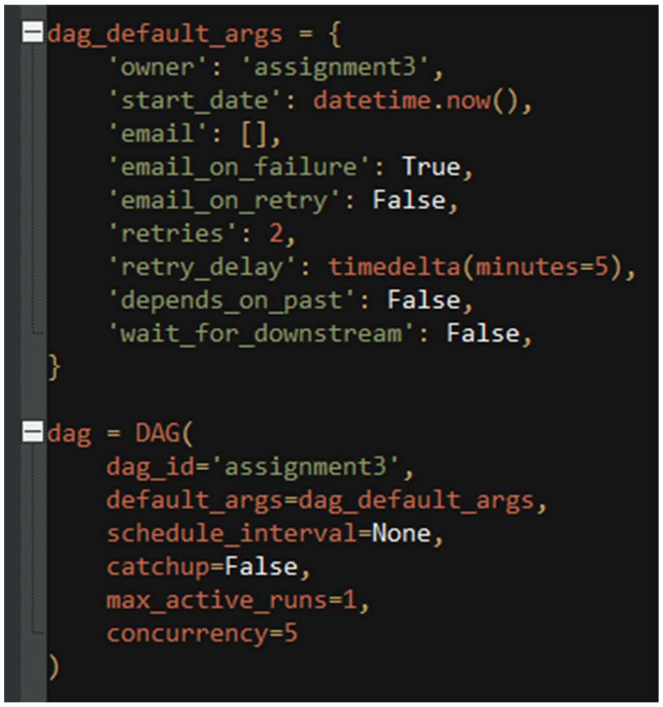
```
import os
import logging
import requests
import pandas as pd
from datetime import datetime, timedelta
from psycopg2.extras import execute_values
from airflow import AirflowException
import airflow
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.providers.snowflake.operators.snowflake import SnowflakeOperator
```

4.2 Establish snowflake connection parameter



```
# Connection variables
snowflake_conn_id = "snowflake_conn_id"
```

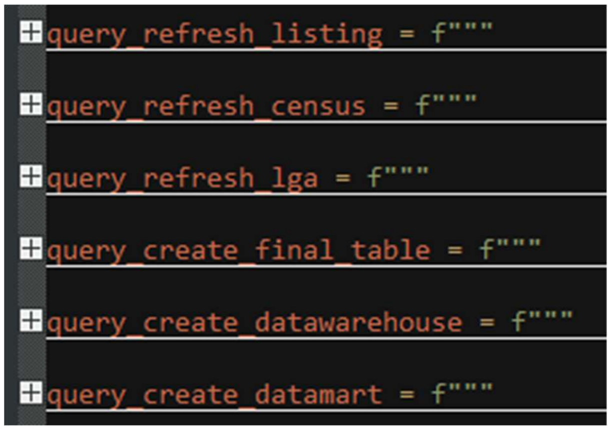
4.3 DAG Settings



```
dag_default_args = {
    'owner': 'assignment3',
    'start_date': datetime.now(),
    '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='assignment3',
    default_args=dag_default_args,
    schedule_interval=None,
    catchup=False,
    max_active_runs=1,
    concurrency=5
)
```

4.4 Custom logics for operator

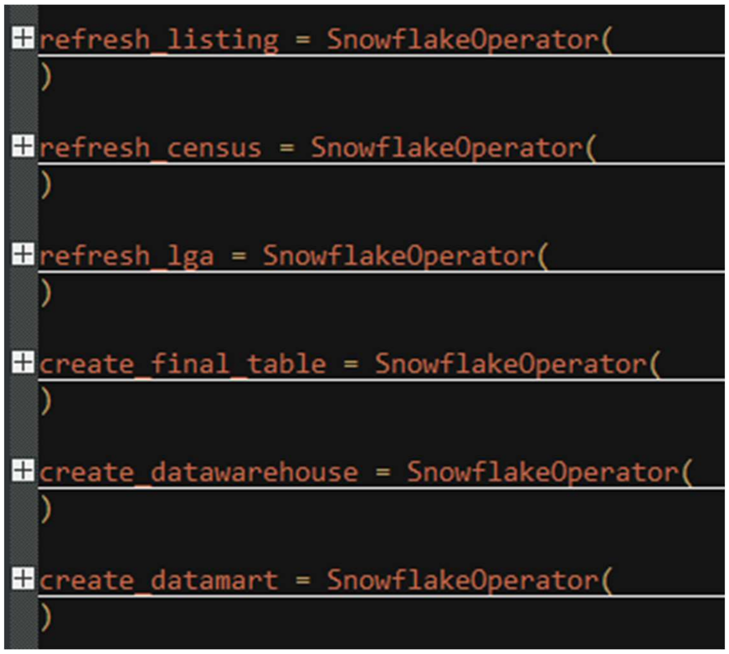
SQL queries performed in Snowflake earlier were grouped into 5 steps, which we can automate the process from pulling data from Google Cloud Storage, to creating business table in data mart.



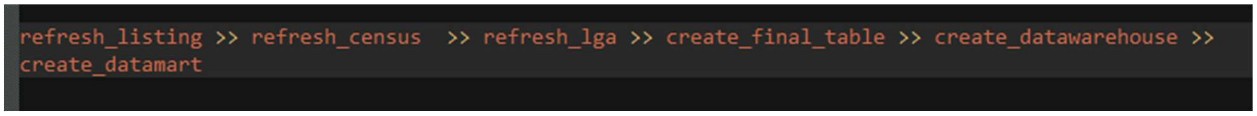
```
query_refresh_listing = f"""
query_refresh_census = f"""
query_refresh_lga = f"""
query_create_final_table = f"""
query_create_datawarehouse = f"""
query_create_datamart = f"""
```

4.5 Set up DAG operators

Create following Snowflake operator task, which are functions telling Airflow which are the steps to be performed, and in what sequence.

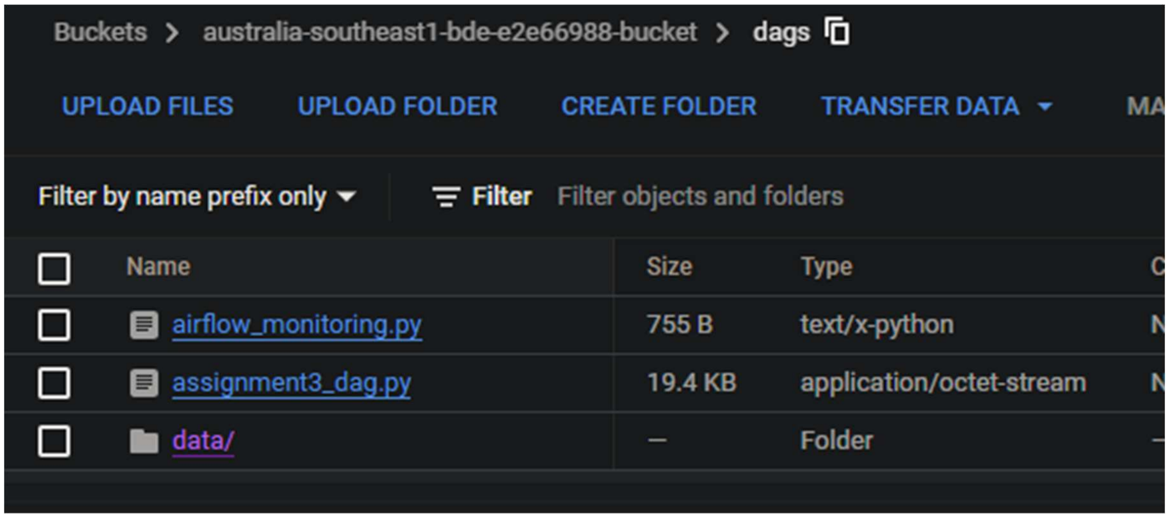


```
refresh_listing = SnowflakeOperator(
)
refresh_census = SnowflakeOperator(
)
refresh_lga = SnowflakeOperator(
)
create_final_table = SnowflakeOperator(
)
create_datawarehouse = SnowflakeOperator(
)
create_datamart = SnowflakeOperator(
)
```



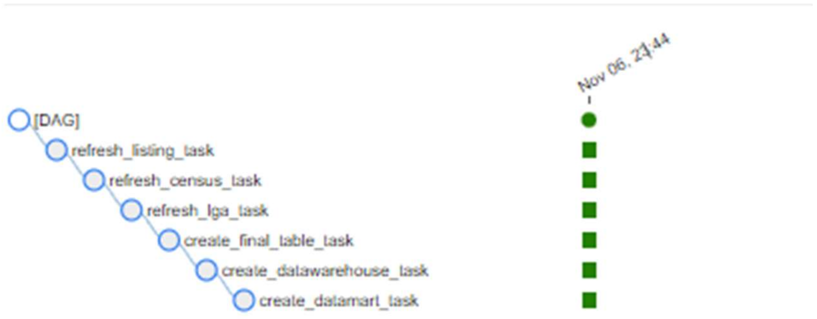
```
refresh_listing >> refresh_census  >> refresh_lga >> create_final_table >> create_datawarehouse >>
create_datamart
```

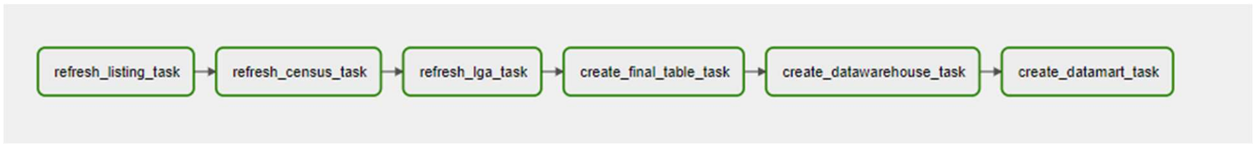
4.5 Airflow DAG triggering


Buckets > australia-southeast1-bde-e2e66988-bucket > dags

Name	Size	Type	C
airflow_monitoring.py	755 B	text/x-python	N
assignment3_dag.py	19.4 KB	application/octet-stream	N
data/	—	Folder	–

Upload DAG file into Google Cloud Storage, which will then be synced into Airflow automatically.

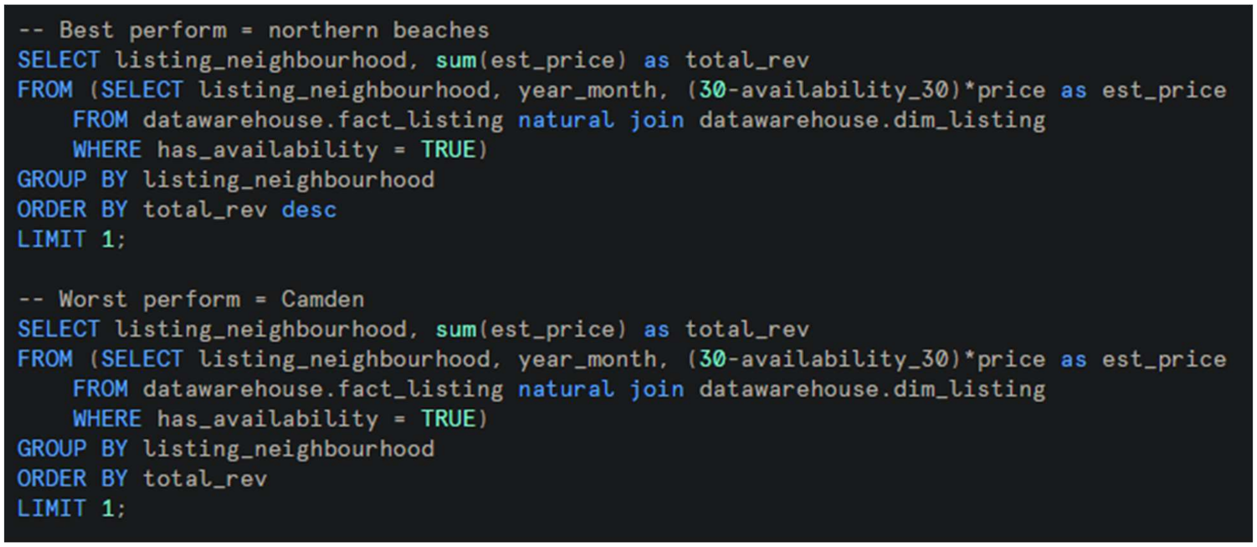




Finally we run the DAG file on Airflow and confirm that each step runs without any issue.

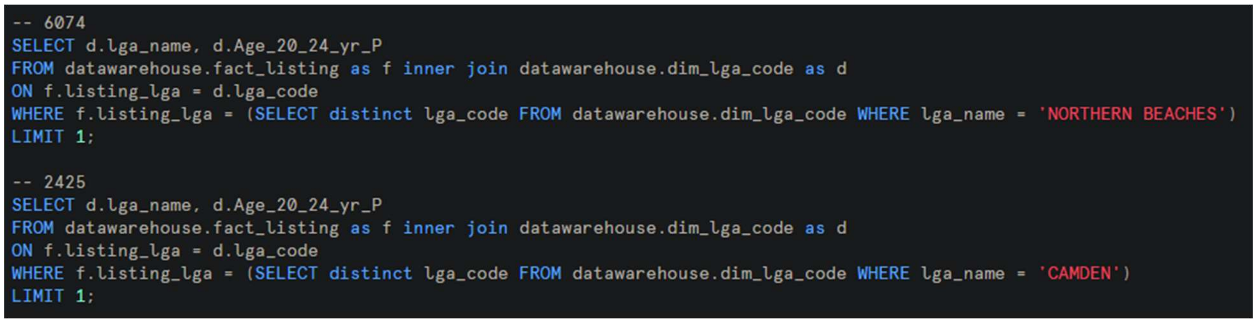
5. Business Questions

5.1 What are the main differences from a population point of view (i.g. higher population of under 30s) between the best performing “listing_neighbourhood” and the worst (in terms of estimated revenue per active listings) over the last 12 months?



```
-- Best perform = northern beaches
SELECT listing_neighbourhood, sum(est_price) as total_rev
FROM (SELECT listing_neighbourhood, year_month, (30-availability_30)*price as est_price
    FROM datawarehouse.fact_listing natural join datawarehouse.dim_listing
    WHERE has_availability = TRUE)
GROUP BY listing_neighbourhood
ORDER BY total_rev desc
LIMIT 1;

-- Worst perform = Camden
SELECT listing_neighbourhood, sum(est_price) as total_rev
FROM (SELECT listing_neighbourhood, year_month, (30-availability_30)*price as est_price
    FROM datawarehouse.fact_listing natural join datawarehouse.dim_listing
    WHERE has_availability = TRUE)
GROUP BY listing_neighbourhood
ORDER BY total_rev
LIMIT 1;
```

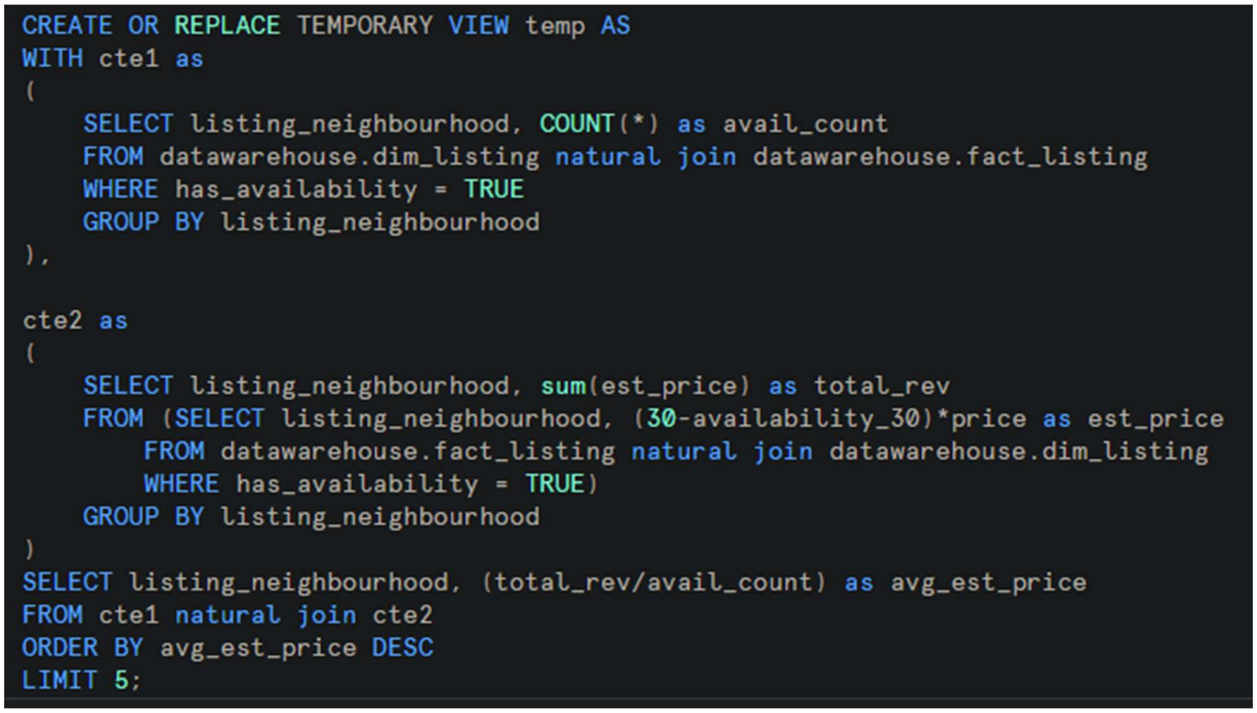
First we find the total revenue of best performing and worst performing neighbourhood, which were Northern Beaches and Camden respectively.



```
-- 6074
SELECT d.lga_name, d.Age_20_24_yr_P
FROM datawarehouse.fact_listing as f inner join datawarehouse.dim_lga_code as d
ON f.listing_lga = d.lga_code
WHERE f.listing_lga = (SELECT distinct lga_code FROM datawarehouse.dim_lga_code WHERE lga_name = 'NORTHERN BEACHES')
LIMIT 1;

-- 2425
SELECT d.lga_name, d.Age_20_24_yr_P
FROM datawarehouse.fact_listing as f inner join datawarehouse.dim_lga_code as d
ON f.listing_lga = d.lga_code
WHERE f.listing_lga = (SELECT distinct lga_code FROM datawarehouse.dim_lga_code WHERE lga_name = 'CAMDEN')
LIMIT 1;
```

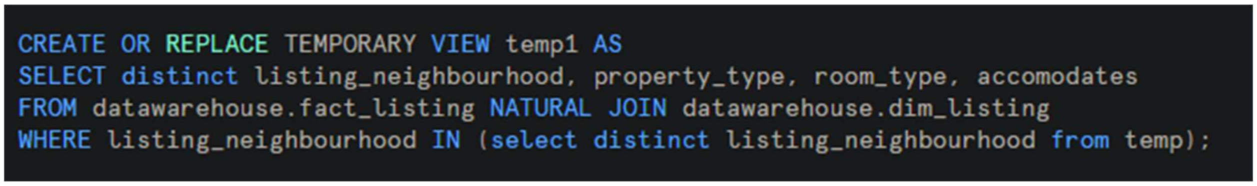
Then we find the population difference between under 30 which was 3649.

5.2 What will be the best type of listing (property type, room type and accommodates for) for the top 5 "listing_neighbourhood" (in terms of estimated revenue per active listing) to have the highest number of stays?



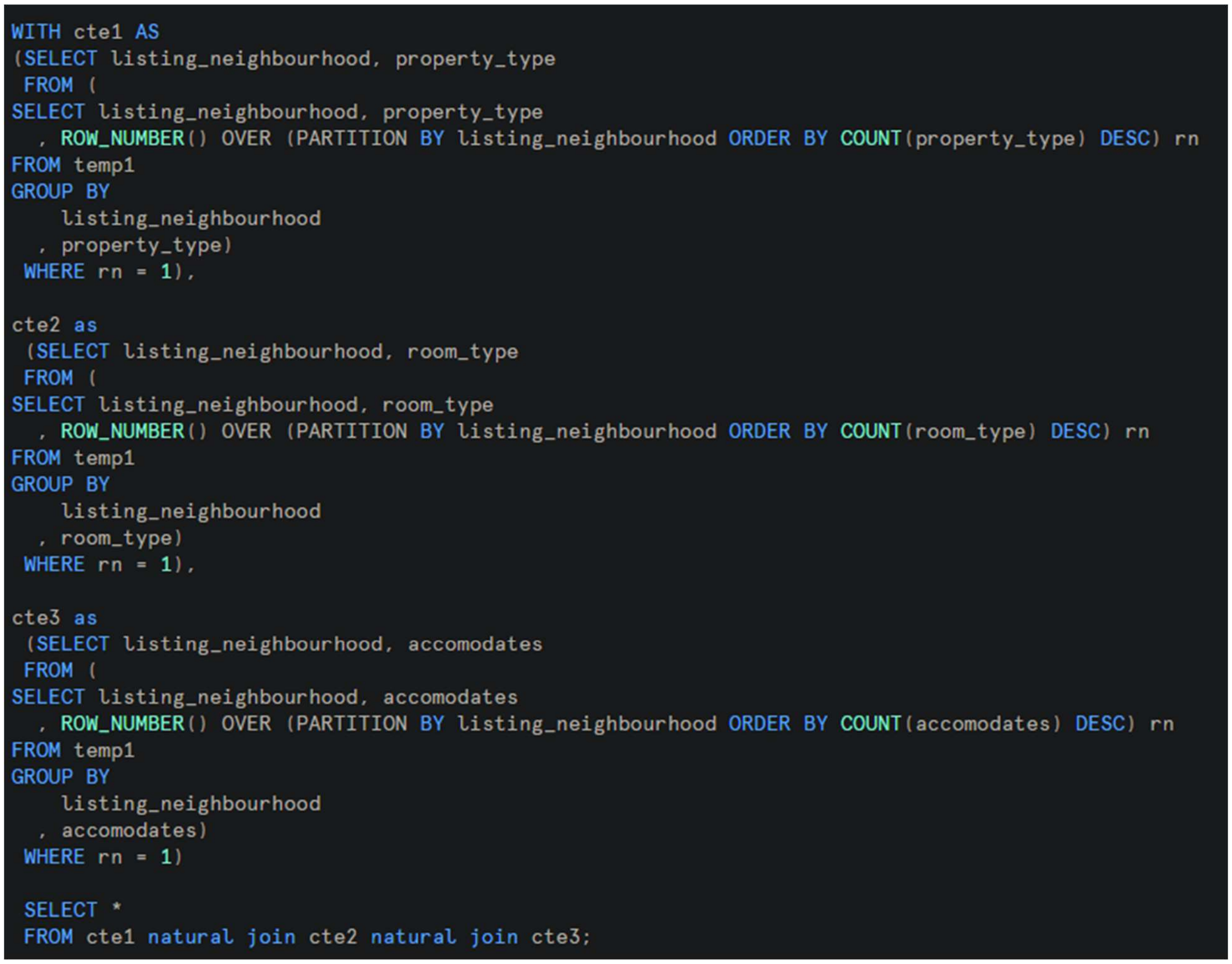
```
CREATE OR REPLACE TEMPORARY VIEW temp AS
WITH cte1 as
(
    SELECT listing_neighbourhood, COUNT(*) as avail_count
    FROM datawarehouse.dim_listing natural join datawarehouse.fact_listing
    WHERE has_availability = TRUE
    GROUP BY listing_neighbourhood
),

cte2 as
(
    SELECT listing_neighbourhood, sum(est_price) as total_rev
    FROM (SELECT listing_neighbourhood, (30-availability_30)*price as est_price
        FROM datawarehouse.fact_listing natural join datawarehouse.dim_listing
        WHERE has_availability = TRUE)
    GROUP BY listing_neighbourhood
)
SELECT listing_neighbourhood, (total_rev/avail_count) as avg_est_price
FROM cte1 natural join cte2
ORDER BY avg_est_price DESC
LIMIT 5;
```

First we create a temp view consisting of each neighbourhood and its respective average estimated earnings. Find the top 5 neighbourhoods using ORDER BY and LIMIT to 5.



```
CREATE OR REPLACE TEMPORARY VIEW temp1 AS
SELECT distinct listing_neighbourhood, property_type, room_type, accomodates
FROM datawarehouse.fact_listing NATURAL JOIN datawarehouse.dim_listing
WHERE listing_neighbourhood IN (select distinct listing_neighbourhood from temp);
```

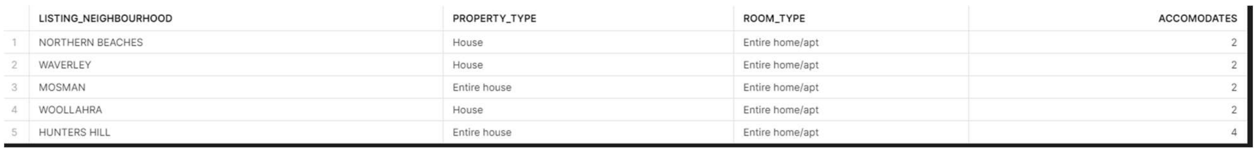
Create another table consisting of relevant information we need, filtering out the neighbourhoods we are interested in.



```
WITH cte1 AS
(SELECT listing_neighbourhood, property_type
 FROM (
SELECT listing_neighbourhood, property_type
  , ROW_NUMBER() OVER (PARTITION BY listing_neighbourhood ORDER BY COUNT(property_type) DESC) rn
FROM temp1
GROUP BY
    listing_neighbourhood
  , property_type)
 WHERE rn = 1),

cte2 as
 (SELECT listing_neighbourhood, room_type
 FROM (
SELECT listing_neighbourhood, room_type
  , ROW_NUMBER() OVER (PARTITION BY listing_neighbourhood ORDER BY COUNT(room_type) DESC) rn
FROM temp1
GROUP BY
    listing_neighbourhood
  , room_type)
 WHERE rn = 1),

cte3 as
 (SELECT listing_neighbourhood, accomodates
 FROM (
SELECT listing_neighbourhood, accomodates
  , ROW_NUMBER() OVER (PARTITION BY listing_neighbourhood ORDER BY COUNT(accomodates) DESC) rn
FROM temp1
GROUP BY
    listing_neighbourhood
  , accomodates)
 WHERE rn = 1)

 SELECT *
 FROM cte1 natural join cte2 natural join cte3;
```

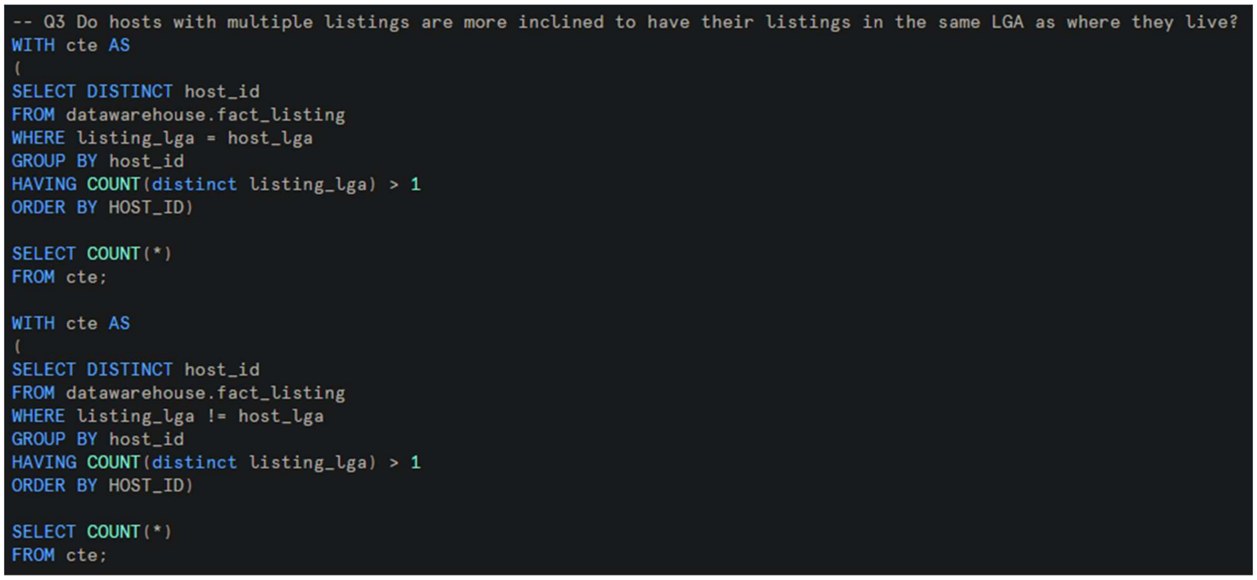
Lastly, we find the best type of each category we are interested in, using ROW_NUMBER and filtering to 1.

Results are as follow:



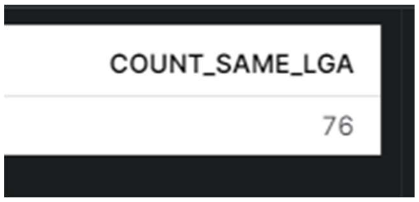
	LISTING_NEIGHBOURHOOD	PROPERTY_TYPE	ROOM_TYPE	ACCOMODATES
1	NORTHERN BEACHES	House	Entire home/apt	2
2	WAVERLEY	House	Entire home/apt	2
3	MOSMAN	Entire house	Entire home/apt	2
4	WOOLLAHRA	House	Entire home/apt	2
5	HUNTERS HILL	Entire house	Entire home/apt	4

5.3 Do hosts with multiple listings are more inclined to have their listings in the same LGA as where they live?

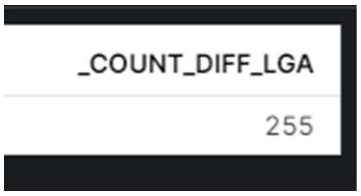


```sql
-- Q3 Do hosts with multiple listings are more inclined to have their listings in the same LGA as where they live?
WITH cte AS
(
SELECT DISTINCT host_id
FROM datawarehouse.fact_listing
WHERE listing_lga = host_lga
GROUP BY host_id
HAVING COUNT(distinct listing_lga) > 1
ORDER BY HOST_ID)

SELECT COUNT(*)
FROM cte;

WITH cte AS
(
SELECT DISTINCT host_id
FROM datawarehouse.fact_listing
WHERE listing_lga != host_lga
GROUP BY host_id
HAVING COUNT(distinct listing_lga) > 1
ORDER BY HOST_ID)

SELECT COUNT(*)
FROM cte;
```

We search for hosts with more than 1 distinct listing LGA, and count the number of rows respectively



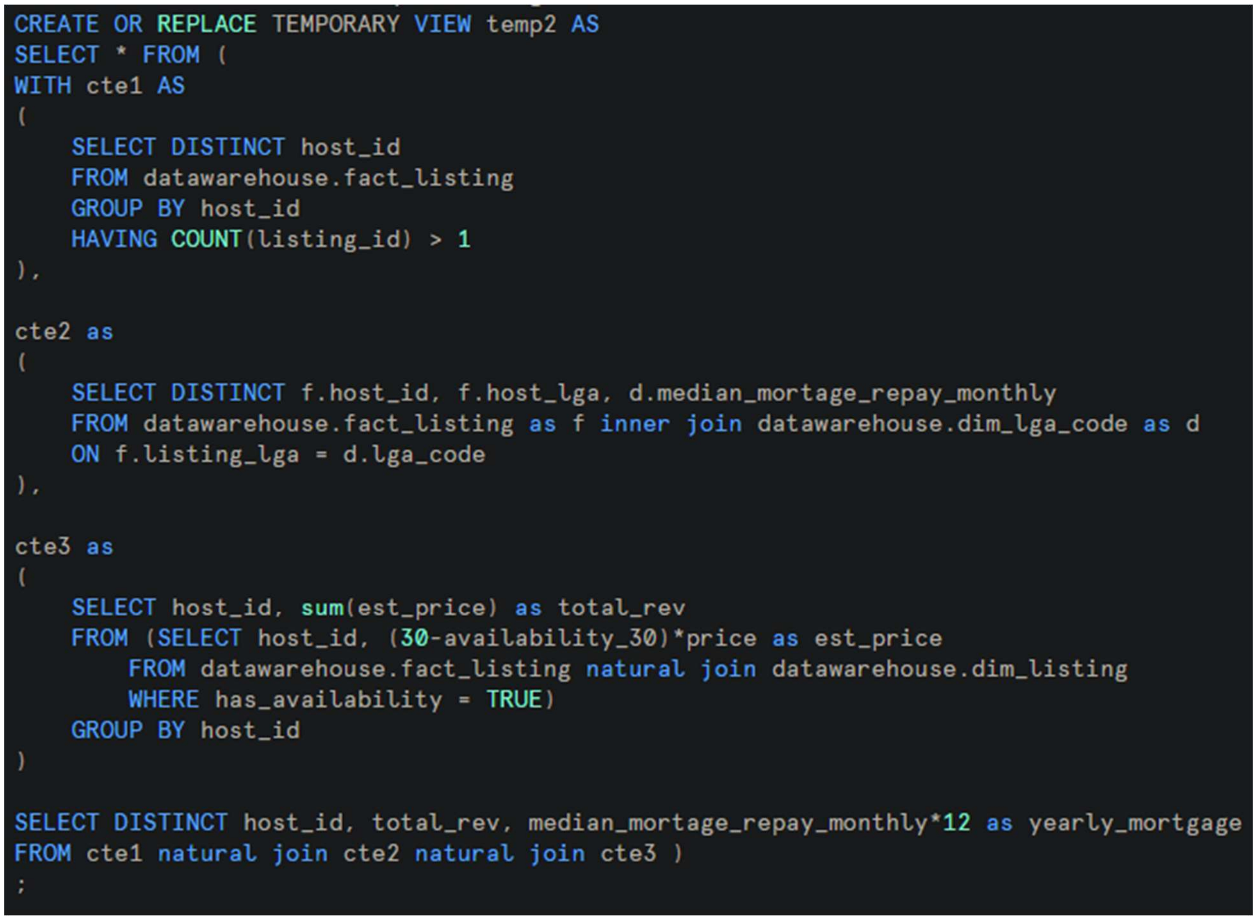
```
COUNT_SAME_LGA
            76
```



```
_COUNT_DIFF_LGA
            255
```

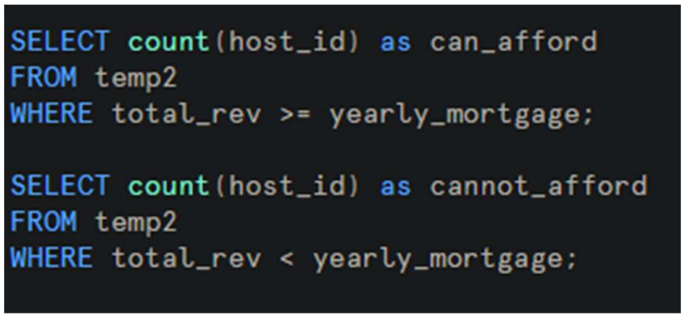
Which we can see that hosts are more inclined to having listings in different LGAs.

5.2 For hosts with a unique listing, does their estimated revenue over the last 12 months can cover the annualised median mortgage repayment of their listing's "listing_neighbourhood"?



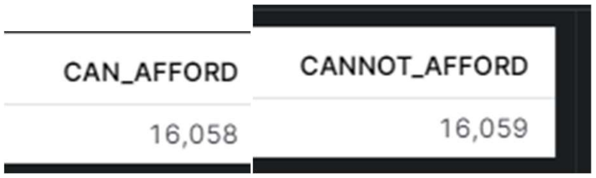
```
CREATE OR REPLACE TEMPORARY VIEW temp2 AS
SELECT * FROM (
WITH cte1 AS
(
    SELECT DISTINCT host_id
    FROM datawarehouse.fact_listing
    GROUP BY host_id
    HAVING COUNT(listing_id) > 1
),

cte2 as
(
    SELECT DISTINCT f.host_id, f.host_lga, d.median_mortage_repay_monthly
    FROM datawarehouse.fact_listing as f inner join datawarehouse.dim_lga_code as d
    ON f.listing_lga = d.lga_code
),

cte3 as
(
    SELECT host_id, sum(est_price) as total_rev
    FROM (SELECT host_id, (30-availability_30)*price as est_price
        FROM datawarehouse.fact_listing natural join datawarehouse.dim_listing
        WHERE has_availability = TRUE)
    GROUP BY host_id
)

SELECT DISTINCT host_id, total_rev, median_mortage_repay_monthly*12 as yearly_mortgage
FROM cte1 natural join cte2 natural join cte3 )
;
```

Create temp view of hosts with more than 1 listings, their total revenue, and estimated yearly mortgage repayment.



```
SELECT count(host_id) as can_afford
FROM temp2
WHERE total_rev >= yearly_mortgage;

SELECT count(host_id) as cannot_afford
FROM temp2
WHERE total_rev < yearly_mortgage;
```

Compare between total revenue and yearly mortgage, count number of rows between those who can afford those who cannot afford.



CAN_AFFORD	CANNOT_AFFORD
16,058	16,059

The result was almost exactly evenly matched between both categories, with hosts that cannot afford edged out by 1 count.

6. Conclusion

Overall, this report aims to provide readers with the understanding of how to operate with csv files in Snowflake environment using SQL, as well as setting up linkage between Google Cloud Storage with Snowflake and Airflow respectively.

Creating data pipeline that's used for automation purposes. As shown in this report, we can retrieve necessary business information just with a single click on Airflow, and the table is generated for us inside Snowflake.

The toughest issue I faced with this assignment was with the business question specifications as there are several computations which I wasn't sure have correctly matched the business requirement.