Airbnb data pipeline using Apache Airflow

By Roger Yu 10906675

# Introduction

This document specifies the setup of a data pipeline using a DAG (Directed Acyclic Graph) using the Airflow framework. The tasks and the relationships with each other are also described. The goal of the data pipeline is to allow the data engineers/scientists to complete the following steps repeatably and idempotently:

1. Create a data warehouse
2. Populate a data mart that can be used to quickly answer regular business questions
3. Use the data warehouse and data mart to answer ad-hoc business questions

# Data summary

|  |  |
| --- | --- |
| Table | Description |
| 2016Census\_G01\_NSW\_LGA | Census data from ABS containing population data per LGA |
| 2016Census\_G02\_NSW\_LGA | Census data from ABS containing median monthly mortgages per LGA |
| LGA\_2016\_AUST | Australian shapefile for LGA |
| SSC\_2011\_AUST | Australian shapefile for SSC (suburb) |
| host\_neighbourhood\_mapping | Mapping for fixed spelling of misspelled and non-Australian `host\_neighbourhood` |
| airbnb\_listing\_yyyymm | The monthly Airbnb listing data |

Table 1:

## Schemas

|  |  |  |
| --- | --- | --- |
| Schema | Description | Tables |
| raw | The first landing stage for the Airbnb data. | * airbnb\_yyyymm from 2020-05-01 to 2021-04-01 * airbnb\_latest |
| star | Containts fact and dimension tables. | * 2016Census\_G01\_NSW\_LGA * 2016Census\_G02\_NSW\_LGA * LGA\_2016\_AUST * SSC\_2011\_AUST * dim\_host * dim\_property * fact\_airbnb * host\_neighbourhood\_mapping |
| meta | Contains metadata for which tables has been loaded | history |
| data\_mart | Calculated fields from the tables in the star schema | * table1 * table2 * table3 |

# Data pipeline

## Project set up

The environment is run from a [docker-compose.yaml](https://github.com/roger-yu-ds/airflow_airbnb/blob/main/docker-compose.yaml) and a [Dockerfile](https://github.com/roger-yu-ds/airflow_airbnb/blob/main/Dockerfile). Note the following points:

* The Airflow image name is apache/airflow:2.0.1-python3.8
* The following volume mappings:
  + - ./data:/opt/airflow/data
  + - ./references:/opt/airflow/references
  + - ./src:/opt/airflow/src
  + - ./dags:/opt/airflow/dags
  + - ./logs:/opt/airflow/logs
  + - ./sql:/opt/airflow/sql
* The database image is kartoza/postgis, which is a Postgres image with [Postgis](https://postgis.net/) installed. This is required due to the use of Spatial joins.

After running docker-compose up –build, the following packages need to be installed in the scheduler and worker containers:

* [geopandas](https://geopandas.org/)
* [python-dotenv](https://pypi.org/project/python-dotenv/)
* [GeoAlchemy2](https://geoalchemy-2.readthedocs.io/en/latest/)
* [rtree](https://rtree.readthedocs.io/en/latest/)

This can be done by first attaching the command prompt to the relevant containers by executing:

docker exec -ti airflow\_airbnb\_airflow-worker\_1 /bin/bash

then pip installing using the command:

pip install geopandas python-dotenv GeoAlchemy2 rtree

## Connection to the [Postgis](https://postgis.net/) database server

After building the containers, connect to the Airflow web UI in a browsers by navigating to <http://localhost:8080/>, then click on go to configurations, see Figure 1.

Graphical user interface, application

Description automatically generated

Figure 1: Click on Connections to set up the connection to the database server

The settings required to connect to the server is shown in Figure 2.

Graphical user interface, text, application, email

Description automatically generated

Figure 2: Settings to connect to the Postgis server

## Common fields

Not all the fields exist throughout the different monthly Airbnb extracts. The common fields throughout were calculated outside of the pipeline, see Appendix A: Common fields.

## Star Schema

The star schema, see Figure 3 contains the fact\_airbnb table with the following dimension tables:

* 2016Census\_G01\_NSW\_LGA
* 2016Census\_G02\_NSW\_LGA
* LGA\_2016\_AUST
* SSC\_2011\_AUST
* dim\_host
* dim\_property

The fact\_airbnb table is created from the raw.airbnb\_yyyymm tables omitting fields related to the hosts and property except for id and host\_id, which are the keys to join the tables.

Minor string processing were done on the following tables:

* 2016Census\_G01\_NSW\_LGA
* 2016Census\_G02\_NSW\_LGA
* LGA\_2016\_AUST
* SSC\_2011\_AUST

To make the joining a more convenient, the “LGA” from the LGA code was removed from these tables, e.g. from “LGA10500” to “10500”.

Diagram

Description automatically generated

Figure 3: Diagram of the star schema

# Airflow

## Folder structure

The DAG requires the following files:

* /dags/ calculate\_airbnb\_kpis\_2.py
* /data/raw/{yyyy-mm-dd.gz}; there should be 12 of these starting from 2020-05 to 2021-04
* /data/raw/shapefile/
  + LGA\_2016\_AUST.dbf
  + LGA\_2016\_AUST.prj
  + LGA\_2016\_AUST.shp
  + LGA\_2016\_AUST.shx
  + LGA\_2016\_AUST.xml
* /data/raw/shapefile\_ssc\_2011/
  + SSC\_2011\_AUST.cpg
  + SSC\_2011\_AUST.dbf
  + SSC\_2011\_AUST.prj
  + SSC\_2011\_AUST.shp
  + SSC\_2011\_AUST.shx
* /references/host\_neighbourhood\_mapping.csv

## DAG

The DAG is shown in Figure 6 starting from the left, finishing on the right with the creation of the data marts.

The tasks in the DAG are either [PythonOperators](https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/python.html) or [PostgresOperators](https://airflow.apache.org/docs/apache-airflow-providers-postgres/stable/operators/postgres_operator_howto_guide.html).

Text

Description automatically generated with medium confidence

Figure 4: Screenshot of the DAG diagram

### Schedule

The schedule is monthly (starting on the first day of the month) because the Airbnb data is updated monthly. Airflow expects the data to have the filename of the format yyyy-mm-dd.gz in the data/raw directory.

### Start date

The start date is set to 13 months before now, which makes Airflow first check for the file 2020-05-dd.gz. If older files are added then the start\_date should be modified accordingly, see Figure 5.

Figure 5: The DAG’s default arguments, note the start date, which is set to be several months before now.

dag\_default\_args = {

'owner': 'roger\_yu',

'start\_date': datetime.now() - relativedelta(months=13),

'email': [],

'email\_on\_failure': True,

'email\_on\_retry': False,

'retries': 1,

'retry\_delay': timedelta(minutes=60),

'depends\_on\_past': False,

'wait\_for\_downstream': False,

}

### Max DAG runs

The max DAG runs is set to 1, this is because the task extract\_airbnb\_into\_fact appends new data to the star.fact\_airbnb table. extract\_property\_into\_dim and extract\_host\_into\_dim relies on star.fact\_airbnb to be up to date.

## Meta

The tasks starting with create\_ are intended to run only once. This feature is implemented through a table meta.history (see ) that contains the table name that has been uploaded. The create\_ tasks first checks if the table name to be uploaded exists in meta.history and then proceeds if the name doesn’t exist, otherwise the loading is skipped. After loading the table is updated with the table name.

This makes the DAG run faster by avoiding unnecessary loading.

Table

Description automatically generated

Figure 6: Screenshot of the history table.

# Backfilling

If older files are to be added after the DAG has been switched on, this could be achieved through backfilling[[1]](#footnote-1).

1. Save the file in the /data/raw directory
2. In the Airflow scheduler container, execute the following command to add a 2020-04 file:  
   airflow dags backfill calculate\_airbnb\_kpis\_2 -s 2020-04-01

# Spatial joins

The neighbourhood\_cleansed field is difficult to match with the LGA dimension tables because some of the values are out of date, e.g. ‘Bondi’ has been updated to ‘Northern Beaches’. Therefore, the latitude and longitude values are used instead.

These values are joined onto the LGA files using a point-in-polygon method.

## SQL

In SQL, the function that performs spatial joins is [ST\_CONTAINS](https://postgis.net/docs/ST_Contains.html), e.g.

on ST\_CONTAINS(st\_setsrid(t2.geometry, 4326), ST\_SetSRID(st\_point(t1.longitude, t1.latitude), 4326))

The SQL snippet above shows part of a join query; t1 is the fact\_airbnb table, t2 is the LGA\_2016\_AUST table, which contains the LGA shape information.

## Python

In Python, the fact\_airbnb dataframe has to be converted into a Geopandas dataframe and a GeometryArray created using [geopandas.points\_from\_xy](https://geopandas.readthedocs.io/en/latest/docs/reference/api/geopandas.points_from_xy.html). And then the join using [geopandas.sjoin](https://geopandas.org/docs/user_guide/mergingdata.html) method.

gpd.sjoin(left\_df=gdf.set\_crs('EPSG:4283'),

right\_df=df\_lga.set\_crs('EPSG:4283'),

op='intersects',

how='left')

gdf = gpd.GeoDataFrame(

df\_merged,

geometry=gpd.points\_from\_xy(df\_merged.longitude, df\_merged.latitude)

)

### Functions

* drop nulls (pandas)
* set column as primary key (postgres)
* PostgresOperator to add a primary key  
  ALTER TABLE table\_name ADD PRIMARY KEY (column\_1, column\_2);

# Challenges

## Adding key constraints in Pandas

The pandas method doesn’t have a parameter to specify the keys. Therefore, first used [pandas.DataFrame.to\_sql](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_sql.html) and [geopandas.GeoDataFrame.to\_postgis](https://geopandas.readthedocs.io/en/latest/docs/reference/api/geopandas.GeoDataFrame.to_postgis.html) instead. Adding the key constraints is done after uploading the table.

---------------------------------------------------------------------------

InFailedSqlTransaction Traceback (most recent call last)

<ipython-input-136-8d87cf2c9e47> in <module>

----> 1 execute\_values(cursor, insert\_query, values, page\_size=len(insert\_df))

/opt/conda/lib/python3.8/site-packages/psycopg2/extras.py in execute\_values(cur, sql, argslist, template, page\_size, fetch)

**1290** parts.append(b',')

**1291** parts[-1:] = post

-> 1292 cur.execute(b''.join(parts))

**1293** if fetch:

**1294** result.extend(cur.fetchall())

InFailedSqlTransaction: current transaction is aborted, commands ignored until end of transaction block

Figure 7

## Timeout period

Initial failures of this task was due to an insufficiently long timeout period. Changing this to 300 was sufficient. However, due to varying sizes of other data sets, the actual value used is 1000 seconds.

join\_postgis = PostgresOperator(

    task\_id='join\_postgis',

    postgres\_conn\_id='postgres\_airflow',

    sql=query,

    execution\_timeout=timedelta(seconds=300),

    dag=dag

)

## Creating tables from scratch

This was a challenge because the INSERT statement would throw errors due to data mismatch. Extra care must be taken in ensuring that the values inserted at in the same order as the columns in the table.

## Schema vs Database

## A DAG triggerring another DAG

The term that Airflow uses to refer to a database is “Schema”, which is not related to a typical meaning of the term, see Figure 8.

Graphical user interface, application

Description automatically generated

Figure 8: What “schema” means in the context of Airflow.

Not specifying the Schema correctly results in a connection error, see Figure 9.

Figure 9: Error output due to mismatched schema name.

[2021-05-20 06:03:04,514] {taskinstance.py:1455} ERROR - could not translate host name "airflow" to address: Temporary failure in name resolution

Traceback (most recent call last):

File "/home/airflow/.local/lib/python3.6/site-packages/airflow/models/taskinstance.py", line 1112, in \_run\_raw\_task

self.\_prepare\_and\_execute\_task\_with\_callbacks(context, task)

File "/home/airflow/.local/lib/python3.6/site-packages/airflow/models/taskinstance.py", line 1285, in \_prepare\_and\_execute\_task\_with\_callbacks

result = self.\_execute\_task(context, task\_copy)

File "/home/airflow/.local/lib/python3.6/site-packages/airflow/models/taskinstance.py", line 1315, in \_execute\_task

result = task\_copy.execute(context=context)

File "/home/airflow/.local/lib/python3.6/site-packages/airflow/operators/python.py", line 117, in execute

return\_value = self.execute\_callable()

File "/home/airflow/.local/lib/python3.6/site-packages/airflow/operators/python.py", line 128, in execute\_callable

return self.python\_callable(\*self.op\_args, \*\*self.op\_kwargs)

File "/opt/airflow/dags/dag\_test\_load\_multiple\_listing\_tables.py", line 310, in \_upload\_listing\_to\_postgres

conn\_pg\_hook = pg\_hook.get\_conn()

File "/home/airflow/.local/lib/python3.6/site-packages/airflow/providers/postgres/hooks/postgres.py", line 108, in get\_conn

self.conn = psycopg2.connect(\*\*conn\_args)

File "/home/airflow/.local/lib/python3.6/site-packages/psycopg2/\_\_init\_\_.py", line 127, in connect

conn = \_connect(dsn, connection\_factory=connection\_factory, \*\*kwasync)

psycopg2.OperationalError: could not translate host name "airflow" to address: Temporary failure in name resolution

While the target DAG was successfully triggered, the tasks did not run for some reason.

## Each file having different columns

This was a source of difficulty at first because some DAG runs would be successful, but others would fail. This was due to missing columns in different files. The solution was to glance at all the files and get a list of all the common fields in all the available files. The loading task then subseted the fields to just these common fields before uploading them to the database.

Note that this method is not robust because newer files might have other missing columns. This is an opportunity to improve the design of the DAG.

## A stuck task

During testing, the task load\_from\_file\_upload\_to\_postgress\_2016Census\_G01\_NSW\_LGA (highlighted in red in Figure 2) was stuck in the “running” status despite rerunning the DAG several times. There didn’t seem to be anything wrong with the task itself. Furthermore, its sister task load\_from\_file\_upload\_to\_postgress\_2016Census\_G02\_NSW\_LGA continued to reliably complete. The solution was to restart the Docker containers.

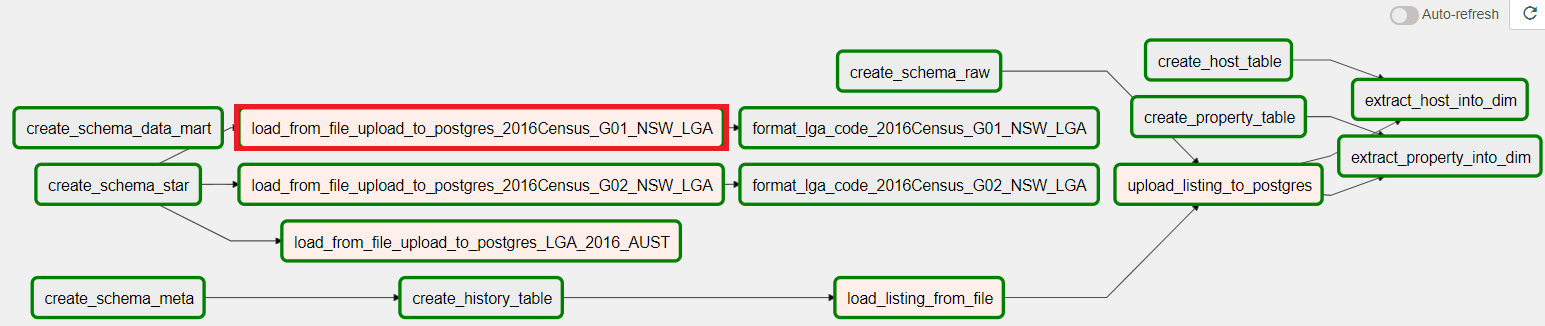


Figure 10: DAG graph view with the red highlighted task stuck in the “running” status.

The cause of this issue is still unknown

## Time spent matching column types and expression

Using SQL directly to specify table creation and value insertion was quite tedious. The ordinal position of column names has to match exactly the fields being inserted. Mismatches will produce nonsensical error messages.

psycopg2.errors.DatatypeMismatch: column "longitude" is of type double precision but expression is of type text

LINE 13: property\_type::TEXT,

## Dropping tables while developing

The table creation queries were built using

CREATE TABLE IF NOT EXISTS

However, during development, it would be better to drop the table and recreate it, which would reflect changes in the query. Having the “IF NOT EXISTS” clause means that the table would not be recreated even if the query has changed.

## Error when partitioned by execution\_date

Partitioning was attempted when creating the fact\_airbnb table, however, while the creation was successful, the insertion of data failed:

ERROR - no partition of relation "fact\_airbnb" found for row

Currently, no fix has been found.

## Skipping DAG runs

Airflow skipped a DAG run for no apparent reason, see Figure 11.

A picture containing table

Description automatically generated

Figure 11: Tree view showing that June was skipped

The cause was another DAG file in a subfolder of /dags with the same dag\_id, which conflicts with the actively developed DAG. Airflow looks at all the files in the /dags folder regardless of subfolders.

## UI slow to update

The UI is slow to load the DAG. During development, the DAGs are frequently deleted. Waiting 30 seconds for the new DAG file to be loaded slowed down development considerably. This is an opportunity to improve the process by understanding and setting the correct Airflow config settings.

## Engine objects are not pickleable

When triggering backfill, the following error is thown:

TypeError: cannot pickle '\_thread.\_local' object

This is due passing a [sqlalchemy.engine.base.Engine](https://docs.sqlalchemy.org/en/13/core/connections.html#sqlalchemy.engine.Engine) object into functions (which is against best practices). This is object is not pickleable.

## Task failures

Figure 12 shows a failed task in July. The solution was to clear that task. Subsequent rerun was successful. It is still not known why the task failed.

A picture containing text

Description automatically generated A picture containing diagram

Description automatically generated

Figure 12: Screenshot of the DAG tree and graph showing a failed task in the July DAG run.

## Reperforming once-only tasks

Some tasks, such as uploading mapping tables are to be run only once, however, the current DAG currently runs such tasks in each DAG run, see Figure 13. This is despite implementing the is\_loaded function to check if a table has already been loaded.

Graphical user interface

Description automatically generated

Figure 13: Screenshot of meta.history showing multiple loadings of the census table.

## DAG run dependence

Currently, the max\_active\_runs is set to 1. Otherwise, multiple DAG runs would be populating the raw.airbnb\_latest table, which is a requirement for downstream tasks. This table is intended to contain the data of the latest month, it would be logically erroneous for multiple DAG runs to update this table.

# Improvements

* Optimisations in the database datatypes, e.g.
  + BIGINT for all int columns is not necessary as some of the values are small.
  + TEXT for all string columns, where a smaller number of characters would suffice
* Perform more robust data validation, e.g. [Great Expectations](https://greatexpectations.io/), to address Each file having different columns
* Implement a [run-once operator](https://medium.com/@plieningerweb/use-apache-airflow-to-run-task-exactly-once-6fb70ca5e7ec) for setup tasks that are intended to be run only once. This would address Reperforming once-only tasks.
* Save the SQL scripts in the SQL directory instead of hardcoding it in the DAG script. This would make the DAG script shorter and easier to read, while also improving the version control of the individual SQL scripts.
* The current DAG requires that the new file be added for a particular interval before that DAG run is scheduled. Since the date (within the month) of file arrival is unknown, the DAG has to run at the end of the month to minimize the chance of a failure. The use of a [file sensor](https://airflow.apache.org/docs/apache-airflow/1.10.11/_modules/airflow/contrib/sensors/file_sensor.html) would allow the start date to be at the beginning of each month; the file sensor would trigger the DAG run as soon as it detect the appropriate file.
* Handle backfilling to process files with past dates. Note that one of the KPIs is to calculate the percentage change month to month. If an earlier file (e.g. 2020-04) is to be added, even if the backfilling is successful, the KPI calculations for 2020-05 should be retriggered.
* Make DAG runs independently of each other, so that multiple DAGs could be run simultaneously, currently there is a dependence on the raw.airbnb\_latest table. This improvement addresses DAG run dependence.
* A more robust way to handle missing columns in data; this addresses Each file having different columns.
* A more robust way of uploading to the database that is not dependent on the order of the fields in an existing table. Perhaps using [pandas.DataFrame.to\_sql](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_sql.html) exclusively because it takes care of the column order.
* Pass connection strings instead of [sqlalchemy.engine.base.Engine](https://docs.sqlalchemy.org/en/13/core/connections.html#sqlalchemy.engine.Engine) objects. This would allow backfill and address Engine objects are not pickleable
* Handle backfilling to process files with past dates. Note that one of the KPIs is to calculate the percentage change month to month. If an earlier file (e.g. 2020-04) is to be added, even if the backfilling is successful, the KPI calculations for 2020-05 should be retriggered.
* Find a way to insert values into a partitioned database. This would allow data mart and subsequent ad-hoc queries to run faster, especially when the data grows.

# Another thing to look out for

Terminated tasks for seemingly no good reason. Clearing that task (which Airflow would automatically rerun) solved the problem.

Figure 14: Error excerpt of SIGTERM signal

[2021-05-22 23:03:43,637] {local\_task\_job.py:188} WARNING - State of this instance has been externally set to None. Terminating instance.

[2021-05-22 23:03:43,638] {process\_utils.py:100} INFO - Sending Signals.SIGTERM to GPID 3146

[2021-05-22 23:03:43,639] {taskinstance.py:1239} ERROR - Received SIGTERM. Terminating subprocesses.

[2021-05-22 23:03:43,654] {taskinstance.py:1455} ERROR - Task received SIGTERM signal

# Business questions

## Q1

Note that for all the questions below, “neighbourhood\_cleansed” is replaced with the more accurate LGA code.

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

Table 2 shows the distribution of the population in the LGAs of interest. Mosman contains a higher proportion of older population, the cross over point being those aged 45-54. Another big difference is that of the total population size, with Mosman having more than 10 times fewer people.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Population count | | Population proportion | |
| LGA\_CODE16 | 10750 | 15350 | 10750 | 15350 |
| LGA\_NAME16 | Blacktown (C) | Mosman (A) | Blacktown (C) | Mosman (A) |
| Age\_0\_4\_yr\_P | 26,928 | 1,493 | 8% | 5% |
| Age\_5\_14\_yr\_P | 49,739 | 3,521 | 15% | 12% |
| Age\_15\_19\_yr\_P | 23,289 | 1,575 | 7% | 6% |
| Age\_20\_24\_yr\_P | 22,830 | 1,280 | 7% | 4% |
| Age\_25\_34\_yr\_P | 52,249 | 3,407 | 16% | 12% |
| Age\_35\_44\_yr\_P | 51,402 | 4,088 | 15% | 14% |
| Age\_45\_54\_yr\_P | 42,063 | 4,273 | 12% | 15% |
| Age\_55\_64\_yr\_P | 33,702 | 3,390 | 10% | 12% |
| Age\_65\_74\_yr\_P | 21,536 | 3,000 | 6% | 11% |
| Age\_75\_84\_yr\_P | 9,674 | 1,542 | 3% | 5% |
| Age\_85ov\_P | 3,553 | 910 | 1% | 3% |
| total\_population | 336,965 | 28,479 |  |  |

Table 2

## Q2

What will be the best type of listing (property type, room type and accommodates for) for the top 5 “neighbourhood\_cleansed” (in terms of estimated revenue per active listing) to have the highest number of stays?

The best type of listing is a villa, entire home/apt that accommodates 11, with the highest n\_stays of 4,488.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| property\_type | room\_type | accommodates | max\_revenue | total\_n\_stays |
| Boat | Entire home/apt | 10 | 67,500,000 | 433 |
| Entire home/apt | Entire home/apt | 9 | 23,547,000 | 127 |
| Villa | Entire home/apt | 11 | 12,654,000 | 4,488 |
| Villa | Entire home/apt | 9 | 12,177,000 | 4,095 |
| Cabin | Private room | 6 | 10,359,000 | 2,700 |

Table 3

## Q3

Are hosts with multiple listings more inclined to have their listings in the same “neighbourhood” as where they live?

Yes, the weighted proportion of hosts (who have more than one properties and whose host\_neighbourhood is known) to have properties in the same host\_neighbourhood is 61.4%.

## Q4

For hosts with a unique listing, does their estimated revenue over the last 12 months cover the annualised median mortgage repayment of their listing’s “neighbourhood\_cleansed”/"LGA"?

Yes, for the most part. 94% of the hosts with a unique listing has an estimated annualized revenue that exceed the median annual mortgage repayments.

# Appendix A: Common fields

# Appendix B: Shapefiles

* 2016 LGA level shape files: <https://www.abs.gov.au/AUSSTATS/subscriber.nsf/log?openagent&1270055003_lga_2016_aust_shape.zip&1270.0.55.003&Data%20Cubes&7951843398FB3F4ECA25833D000EAE34&0&July%202016&07.11.2018&Previous>
* 2011 SSC level shape files:

<https://www.abs.gov.au/AUSSTATS/subscriber.nsf/log?openagent&1270055003_ssc_2011_aust_shape.zip&1270.0.55.003&Data%20Cubes&D68DFFC14D31F4E1CA2578D40013268D&0&July%202011&22.07.2011&Previous>

1. Backfilling currently throws an error due to the design of the DAG, see Engine objects are not pickleable. [↑](#footnote-ref-1)