

Introduction

This document discusses the given task 'Data_Engineering_Challenge20230111.pdf' and add details for the each questions given in it.

The data referenced is from kaggle website with following links:

- https://www.kaggle.com/datasets/samyukthamurali/airbnb-ratings-dataset?select=LA_Listings.csv
- <https://www.kaggle.com/datasets/samyukthamurali/airbnb-ratings-dataset?select=airbnb-reviews.csv>

The data was downloaded through manual steps for this challenge, although for the purposes of data pipeline it is recommended to use Kaggle API.

Following are the location where the source data was extracted. It is to note that in order to execute the whole code without changing, the folder structure needs to be as follows.

- **Reviews:**
datasets\airbnb-reviews\airbnb-reviews.csv
- **Listing:**
datasets\LA_Listings\LA_Listings.csv

The datasets folder exist in the same path as the code.

Questions

Following are the observation and details regarding the questions asked in the challenge.

Data Familiarity

Following measures were taken in order to get myself familiarize with the dataset.

Pre-Processing:

As the size of reviews data was around 3 GB after being uncompressed. I decided to read it in chunks and use only those reviews that have corresponding listing. The code file '*pre_processing.py*' implements the same technique.

The data from the listing is first read, and then in chunks the data from the reviews. The listing_ids are filtered from the chunks using listing file as reference.

Preview the data:

The data was previewed to get basic insight using head(), info() and describe function.

Dimension Model

As per my analysis, the dimension model for the given data set would be as follows.

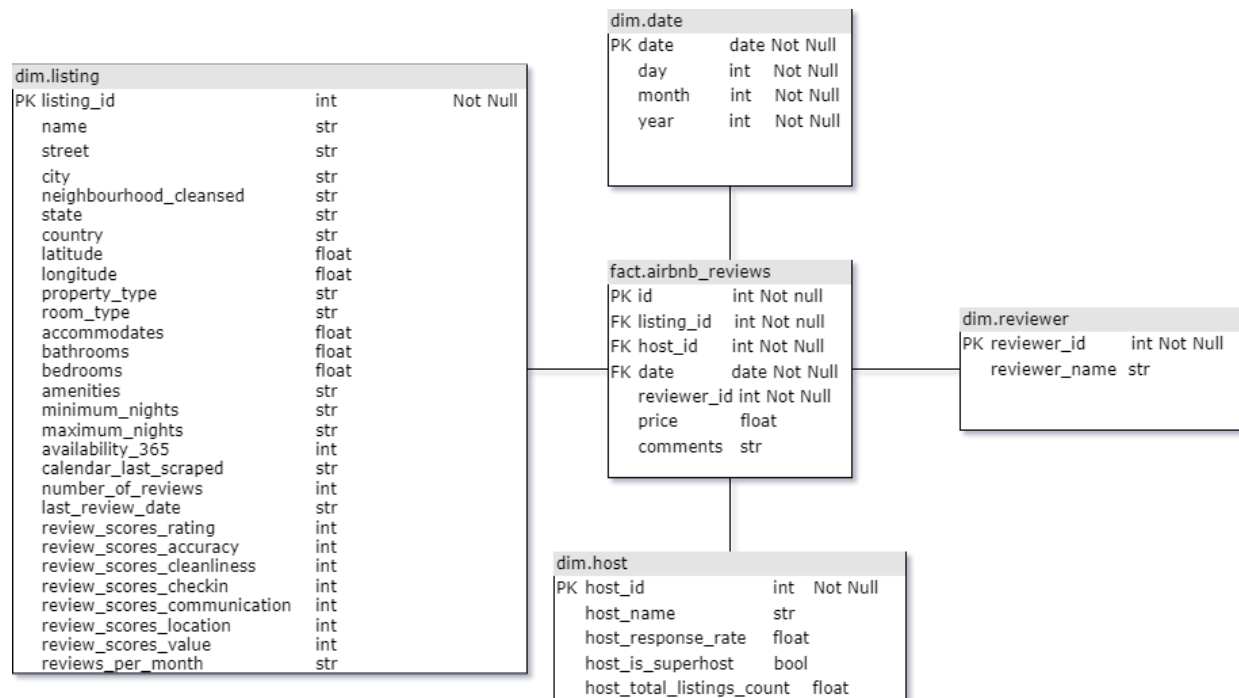


Figure 1.0

It is to be noted that the column in fact table could be added at any given time as per the requirement of the analysis, ease of querying data or increasing performance. The code has been designed in a way that it just needs configuration update without a single code line change and the columns will be added in the final datasets as per the requirements.

Application

This part requires an application that loads the data from the sources and transform it in the form of diagram represented in figure 1.0

Robustness

As part of the application development, it was mentioned that clean, maintainable and testable code is encouraged. I have created an application that would require just external parameter updates to accept any kind of change. As we observe that the schema of tables require continuous update, thus the application has developed to keep that requirement in mind.

Structure

The code is designed in object oriented format that is easily maintainable. It has external parameter functionality in yaml and command line parameter support of common requirements

Application folder structure details is as follows:

File\Folder Name	Usage	Responsible	Comments
init.py	Application Entry Point	To start all codes	Use this to execute
pre_processing.py	Library File	Gets subset of data from reviews as per LA listing's listing_id	This file is being imported in init
processing.py	Library File	Creates fact and dimension as per the configuration	This file is being imported in init
data_quality_checks.py	Library File	To apply data quality checks on facts and dimension	This file is being imported in init
utils. config_reader.py	Library File	To Parse Yaml Configs	This file is being imported in init
config.yml	Configuration File	Maintain parameter details	This file is being loaded in Config_Reader class
Datasets\	Downloaded data	Holds data directly downloaded from kaggle	Subfolders: airbnb-reviews\airbnb-reviews.csv LA_Listings\LA_Listings.csv
temp\	Folder for Temporary stored Files	Holds chunks of data while pre processing.py consolidates	Temp folder
pre_processed\	Source for processing.py	Holds subset of data after execution of pre_processing.py	la_airbnb-reviews.csv la_listing.csv
processed\	Fact and Dimension Data	Created after execution of processed	airbnb_reviews.csv date.csv host.csv listing.csv reviewer.csv
Requirements.txt	External Libraries Details	Pandas Pyyaml glob	pip3 install -r requirements.txt
logfile.txt	Auto Created Logging details	Holds details of pre_processing, processing and data quality checks	This can be used to understand and debug the execution of this applicaiton

Execution:

The execution can be done out of the box considering the requirements are fulfilled.

Requirements:

The only requirement is source data which needs to be made available through Kaggle download in the specified format. Moreover, the prior to executing the script, requirements.txt file should be executed to make the python libraries available.

Command:

The following command can be used to start the execution:

```
python init.py
```

It is to be noted that as per the requirements we are under the impression that the python version would be ≥ 3.7

We can use `--config` and `--log` parameters in case we need to give different config file, or get logs in file format respectively. By default the logs would be printed out on the console but using following command it can be saved in a file.

```
python init.py --log logfile.txt
```

Config:

The following parameters can be tweaked to run the application as per the requirements.

Logging:

This parameter is used to get more details from the application logs. Only 'DEBUG' and 'INFO' are supported.

```
logging: 'INFO'
```

Listing & Reviews:

Listing and Reviews has sub parameter that provides application details regarding the LA listing and Airbnb reviews source file respectively. User can tweak the source, delimiter, encoding and output destination. Moreover in case of reviews a chunk parameter is provided which can be used to tweak application parts formation during the pre-processing step. It is to be noted that the application was tested in windows environment thus the folder structure as escaped backslashes. In order to execute in Linux/Unix one must replace them with forward slashes.

```
listing:
  source: 'datasets\\LA_Listings\\LA_Listings.'
  deletemeter: ','
  encoding: 'ISO-8859-1'
  destination: 'pre_processed\\la_listings.csv'
```

```
reviews:
  source: 'datasets\\airbnb-reviews\\airbnb-reviews.'
  deleimeter: ';'
  chunk: 30000
  encoding: 'ISO-8859-1'
  destination: 'pre_processed\\la_airbnb-reviews.csv'
```

Relation:

The main relationship between the two dataset is described by relation parameter. Reviews dataset will be filtered on column which is similar in both the dataset as we are required extract subset of data from it.

```
relation: 'listing_id'
```

Dim Model:

This parameter has multiple different sub parameters which are being used to tweak the application output without making single change in the code.

Facts:

Facts list down all the fact tables that are required to be created through the application. Currently there is 1 but we can create multiple tables just by adding a name and related details.

```
facts: ['airbnb_reviews']
```

For each fact table name in the list, there must be 3 more parameters with suffix as '_columns', '_keys' and '_dtype'. These parameters are responsible to give fact its identity.

As **airbnb_reviews_columns** describe the number of required column in the fact table. It is to be noted that the code is fully dynamic to facilitate any change in the number of output columns without requiring any change.

The **airbnb_reviews_keys** column describe which column from the fact table are primary keys and foreign keys. This is only requires in the fact table implementation as dimension table would verify if their column exist in any of the fact table keys, and make it their own primary key automatically.

The **airbnb_reviews_dtype** provides column its datatypes. It is to be noted that as we are not delivering in format that keeps the type thus this is not fully enforced.

```
airbnb_reviews_columns: ['id','listing_id','host_id','date','reviewer_id',
'price', 'comments']
airbnb_reviews_keys: {'id':'primary_key', 'listing_id':'foreign_key',
'host_id':'foreign_key', 'date':'foreign_key', 'reviewer_id':'foreign_key'}
airbnb_reviews_dtype:
{'id':'int64','listing_id':'int64','host_id':'int64','date':'date','reviewer_id':
'int64', 'price':'float64', 'comments':'str'}
```

The output of facts would be copied in 'fact_save' parameter.

Dimensions:

Dimensions list down all the dimension tables that are required to be created through the application. Currently there is 4 but we can create multiple tables just by adding a name and related details.

For each dim table name in the list, there must be 2 more parameters with suffix as '_columns', and '_dtype'. These parameters are responsible to give dim its identity.

[dimension name]_column gives detail about the number of column in the dimension, whereas [dimension name]_dtype provides its datatypes.

```
dimensions: ['listing', 'host', 'date', 'reviewer']
host_columns: ['host_id', 'host_name', 'host_response_rate', 'host_is_superhost',
'host_total_listings_count']
host_dtype: {'host_id':'int64', 'host_name':'str', 'host_response_rate':'float64',
'host_is_superhost':'bool', 'host_total_listings_count':'float64'}
date_columns: ['date', 'day', 'month', 'year']
date_dtype: {'date':'date', 'day':'int64', 'month':'int64', 'year':'int64'}
reviewer_columns: ['reviewer_id', 'reviewer_name']
reviewer_dtype: {'reviewer_id':'int64', 'reviewer_name':'str'}
dim_save: 'processed\\'
```

'dim_save' would save the output in the given folder.

Required Column:

The required column of facts and dimension is changeable. Making the application welcoming any kind of schema change without code update. The user can increase or decrease the number of column from this yaml file. The columns from dimensions can be moved to facts or vice versa.

Dynamic Function:

This application is unique in a way that we can add new columns to the data which are either derived or self-created. If the column doesn't exist in source, it would **automatically require a function implementation** in processing.py file with [column name]_apply format. This function would provide a dataframe input that would be merged on the both source thus providing holistic view of the whole data.

As we are adding month column that is derived from the date, the code would automatically call the following method without the need of adding its method call anywhere.

```
def month_apply(self, df):
    df['date'] = pd.to_datetime(df['date'])
    df['month'] = df['date'].dt.month
    return df
```

We have additional functionality, in which if we need to make a change in already existing column, we can add its name in the 'apply_function_on_columns' list and its function call would automatically happen






as a result. The developer only need to provide method implementation without the need of adding an reference in the code.

```
apply_function_on_columns: ['host_response_rate']
```

```
def host_response_rate_apply(self, df):  
    df['host_response_rate'].interpolate(method='linear', inplace=True)  
    return df
```

The code is highly maintainable and robust.

Output:

Local Disk (E:) > Personal > GuidionTask > processed				
Name	Date modified	Type	Size	
 airbnb_reviews.csv	1/16/2023 01:16 PM	Microsoft Excel C...	238,880 KB	
 date.csv	1/16/2023 01:16 PM	Microsoft Excel C...	57 KB	
 host.csv	1/16/2023 01:16 PM	Microsoft Excel C...	1,071 KB	
 listing.csv	1/16/2023 01:16 PM	Microsoft Excel C...	20,424 KB	
 reviewer.csv	1/16/2023 01:16 PM	Microsoft Excel C...	8,765 KB	

Data Quality:

The third part of the application is data quality defined in data_quality_checks.py It is to make sure that in order to deliver data to the end user, the validation steps are performed beforehand.

Completeness Check:

We are validating data completeness by comparing data columns in fact and dimension with columns defined in yaml data dictionary.

Data is validated for this check.

Incorrect values:

We are maintaining data integrity by verifying key columns with null values. The primary key in data sets both facts and dimension should not have nulls.

Data is validated for this check.

Unexpected data:

We are maintaining data integrity by verifying key columns with duplicate values. The primary key in data sets both facts and dimension should not have duplicates. Although foreign key in a dataset can have duplicates but no null values.

Data is validated for this check.

Range check:

The data range for key columns was verified by checking any negative occurrence in it.

Data is validated for this check.

External Validity:

The source data was verified against the unique key value of the facts and dimensions. The columns such as id, host_id, listing_id, date, reviewer_id should have unique equal values as per the source.

All the checks for these datasets have been verified except listing_id as it exist in both the sources, thus source LA_Listing have more listing_ids than the unique listing_id in Airbnb_reviews. This was flagged during the data quality check as follows.

```
2023-01-16 13:16:36,612 - data_quality_checks.py - Table: listing Column listing_id defined as foreign_key
2023-01-16 13:16:36,612 - data_quality_checks.py - Column listing_id Exist in table listing
2023-01-16 13:16:36,891 - data_quality_checks.py -
-----
                        Data Quality Check Failure
-----
Check Details: check_primary_key_validity
Table:        listing
-----
2023-01-16 13:16:36,891 - data_quality_checks.py - Table: listing Column host_id defined as foreign_key
2023-01-16 13:16:36,891 - data_quality_checks.py - Table: listing Column date defined as foreign_key
```

The listing_id in fact table doesn't not have all the unique values that LA_Listing has because the fact table primary key is id based on which the data was de-duplicated.

Data Partitioning:

To increase query performance we can partition the fact table on **date** column. This will provide us with decrease lookup time when applying where clause.

SQL:

Following are the sqls for the questions written in the document.

What are the top 5 reviewer_id's who made the most reviews?

```
select * from
(select reviewer_id,count(*) as reviews from fact.airbnb_reviews group by
reviewer_id)
order by reviews desc limit 5
```

In which month were the most reviews done by reviewers?

```
select source.month, max(num_of_reviews) as review_count from
(select dates.month, count(*) num_of_reviews from fact.airbnb_reviews review
join dim.date dates
on dates.date = review.date
group by on date.month) source
```


Define relation between host_response_rate and host_is_superuser if you can find?

```
select host_is_superuser, AVG(host_response_rate) as avg_host_response_rate from dim.host group by host_is_superuser
```

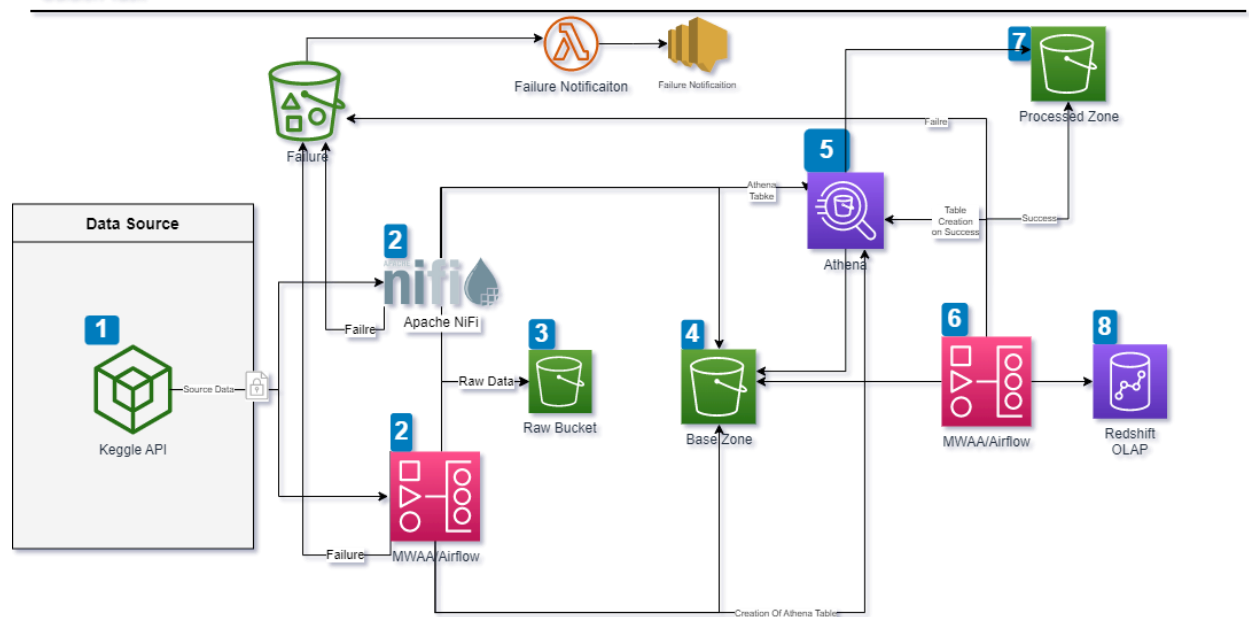
This query will group the records in the host dimension table by the host_is_superuser column, then calculate the average host_response_rate for each group. The result will give the average response rate for hosts that are superhosts and those hosts that are not.

Designing Data pipeline:

This section would explain how I would like to design a data pipeline using this data from beginning to reaching the end user along with technologies that I would prefer.

Kaggle Data Pipeline

Guidion Task



1. The source of data pipeline is kaggle which is accessible through API.
2. At Point two we have two option, either to go with Apache NiFi or Airflow DAG. Both of it has their pros and cons. We can schedule a time at which we require our data to be ingested in either of these
3. The Raw zone is where we store every data we ingest without making any change. This could be for recovery purposes or for compliance.
4. Base zone is where we would store the ingested data that would be readily consumed by the downstream pipeline. The recommended format for this zone is Parquet.

5. The ingestion step would create a table in Athena as that would provide easy visibility for the data exist in base zone. Business users can connect with Athena and verify their data before making any decision for the data warehouse
6. The MWAA Dag at this stage would read the base zone data apply required transformation and store it in processed zone. It will also create table in Athena for ease of verification and analysis before loading it downstream.
7. Processed zone would store processed data for downstream pipelines.
8. The Redshift data warehouse would be the final destination for our Fact and Dimension tables.

*All Failures would write failing dataset to the failure bucket. Which would trigger Lambda to provide sns/ses notification.