Data Engineering Nanodegree Capstone Project

STEP 1: Scope the Project and Gather Data

Scope

For the sake of this project, I have used Udacity provided US immigration dataset to build an ETL pipeline. I have used the following technologies from the course:

- Cloud Data Service Redshift
- Airflow
- PySpark

The purpose of this project is to analyse and explore the given datasets, clean them, define data modelling for the redshift tables, build an ETL pipeline and define an application of such pipeline. I personally preferred airflow to do this so that I can build different sub tasks, schedule them and then visualise the operators in Airflow's web interface.
Once the tables have been created in Redshift and the data is loaded, I have run some test queries to analyse the data further. This is one such application of this pipeline where we can query the Redshift tables to answer our business use case questions.

Main steps taken to complete the project:

1. Pre-process and understand the data (Identify .SAS/.CSV/.SAS7BDAT formats) and place them to s3 bucket.
2. Build the pipeline (preprocess, drop tables, create tables, load data, data quality checks)
3. Query the tables to demonstrate one of the applications of building the whole data pipeline.

Following are the end application based questions we would like to answer once the ETL pipeline has been run successfully:

Ques 1: From which country, maximum immigrants arrived to the US?
Ques 2: Which country/city has the maximum average temperature uncertainty?
Ques 3: Which type of airport has the least elevation feat?
Ques 4: Which city/state has the maximum average household size?

Note: *Of course, a lot many questions can be framed. Due to the limitations of this project, I have randomly chosen to address the above four questions. The whole idea is to be able to query the redshift tables once the entire ETL has been performed. To show that I have actually queried the Redshift tables, I have put the output of these queries as a snapshot taken from Redshift's editor console in AWS(last section of this document).*

Describe and Gather Data

Note: Further details are mentioned in data_dictionary.txt file as well as in the snapshot of data modelling in README.md file.

The data is already given by Udacity. There are 4 links provided for the data as follows:

- **I94 Immigration Data:** This data comes from the US National Tourism and Trade Office. A data dictionary is included in the workspace in the form of SAS file. I have pre-processed it to form separate csv files in the project. This preprocessing has also been scheduled in Airflow. This is where the data comes from.
- **World Temperature Data:** This dataset came from Kaggle. You can read more about it here.
- **U.S. City Demographic Data:** This data comes from OpenSoft. You can read more about it here.
- **Airport Code Table:** This is a simple table of airport codes and corresponding cities. It comes from here.

STEP 2: Explore and Assess the Data

There are 4 datasets for this project. I will now describe them one by one.

Explore the data:

The requisite has been done as a part of the pre-processing step. .SAS, .CSV, .SAS7BDAT format files have been explored using pandas and pyspark. Wherever, the data is small, I have used pandas. And wherever the data is big, I have used pyspark for pre-processing.

Cleaning Steps:

I have created the pre-processing python scripts for the respective data files as the first step in building a data pipeline. Please *refer airport_codes_preprocess.py, sas7bdat_to_parquet.py, sas_to_csv.py, temperature_preprocess.py and us_demographic.py.*

In general, the cleaning steps are:

- Check for null values
- Check for duplicate rows
- Clean the header names

- Check for the datatype for dates and extract the year and month. I have kept the data aggregated according to year.
- Convert .SAS7BDAT file to .parquet files.
- Convert .SAS files to .CSV files.

STEP 3: Define the Data Model

3.1 Conceptual Data Model

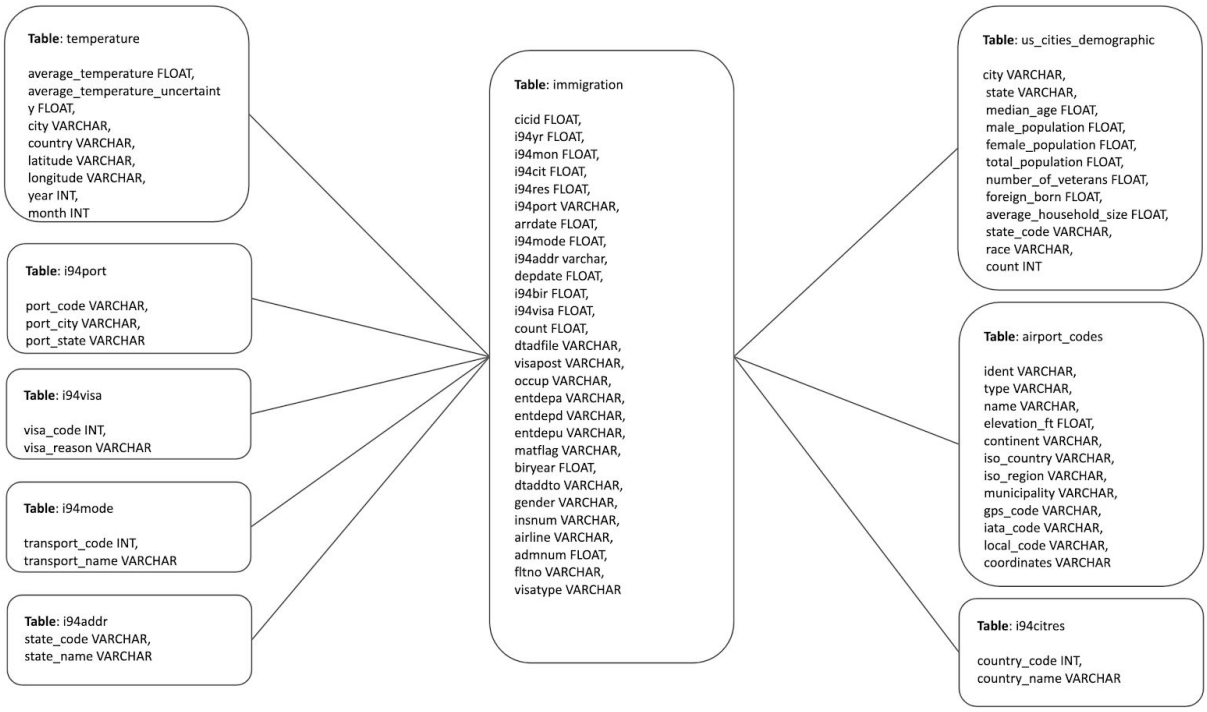
For data modelling, Star Schema has been used. Its advantages are:

- Easy to understand
- Clear join paths are defined using such a schema. Hence the query processing is quite faster.
- This schema is good if there are not a big number of tables in the database.
- Structural simplicity also reduces the time required to load large batches of data into a star schema database. By defining facts and dimensions and separating them into different tables, the impact of a load operation is reduced. Dimension tables can be populated once and occasionally refreshed.

Decision choices made for the given database:

- Dimension tables: us_cities_demographics, airport_codes, temperature data were given in csv format. This data acts well as dimension tables since they are quite domain specific. Along with this, the data was given as label description in the sas file. This also acts as a good candidate for dimension tables.
- Fact table: The data present in the format of sas7bdat, is used as a fact table and has data for the immigration in the US. This data has foreign keys which can act as the primary keys in the respective dimension tables.

  Below is a picture to describe the Star Schema for the given data.



3.2 Mapping Out Data Pipelines

Following are the data pipeline steps:

1. Preprocess all the data: This includes the following sub steps:
    a. Pre-process the us_cities_demographics.csv data. Check for null values, duplicates, size, data type formats. This is then built as a dimension table.
    b. Pre-process airport-codes_csv.csv file. Check for null values, duplicates, size, data type formats. This is then built as a dimension table.
    c. Pre-process GlobalTemperaturesByCity.csv file. Check for null values, duplicates, size, data type formats. This is then built as a dimension table.
    d. Process Label description sas file to the csv file. This file has been used to build the dimension tables. The tables are: i94addr, i94visa, i94port, i94mode, i94citres.
    e. Process sas7bdat files. The sas7bdat files have been converted to the parquet files as a pre-processing step. The table is called immigration. This is kept as a fact table since its columns act as foreign keys in the dimension tables.

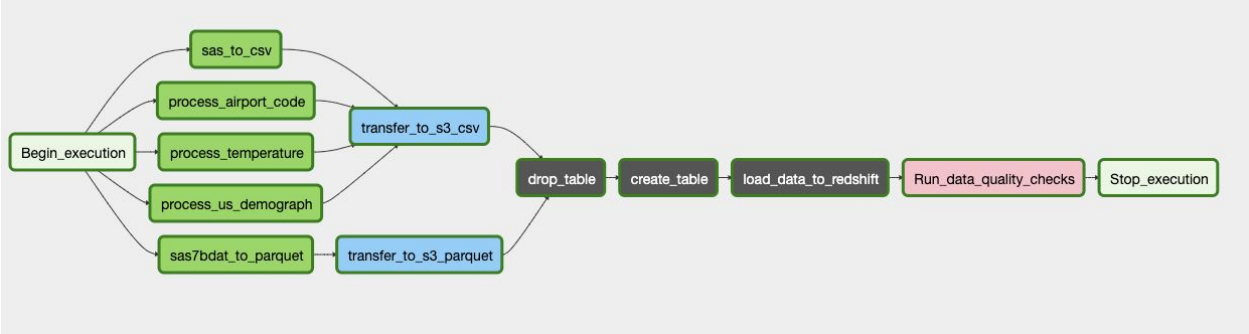
2. Transfer the csv and the parquet files to s3 bucket known as data-pipelines/private-dend. For this, s3 bucket has to be created by going to the s3 service in the amazon management console.
3. Drop tables: All the tables are dropped if they already exist in Redshift.

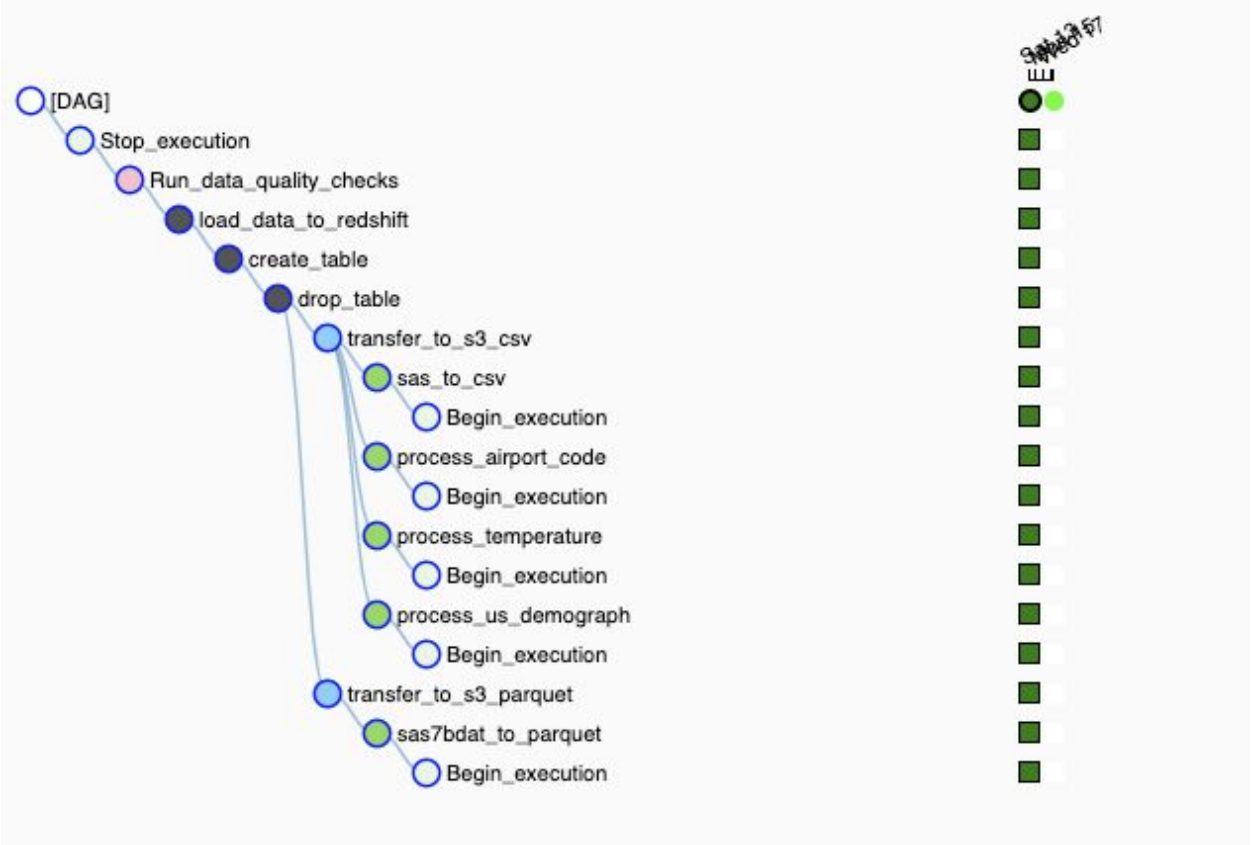
4. Create tables: All the tables are created in Redshift.. The names are: immigration, us_cities_demographics, airport_codes, temperature, i94addr, i94visa, i94port, i94mode and i94citres.
5. Load the data from s3 bucket to Redshift tables.
6. Perform the data quality checks as described in Section 4.2.

STEP 4: Run ETL to Model the Data

4.1 Create the data model

Data Model has been successfully created and ETL pipeline was run. Below are the snapshots taken from the Airflow's web UI.





4.2 Data Quality Checks

I have used the following two measures as data quality checks:

1. Check if the count of rows is 0 or negative: This is to ensure if the data is even loaded to the tables in the first place.
2. Check the total count of rows in each table: This is to ensure that the input data's rows should be equal to the number of rows in each table to ensure that all the rows have been loaded in the data. This measure checks the completeness of all the tables.

4.3 Data Dictionary

Please refer to data_dictionary.txt provided in the workspace.

STEP 5: Complete Project Write Up

Rationale behind the chosen technologies:

Airflow:

- Automates the pipeline.

- Helps to visualize, schedule and monitor data pipeline tasks
- Easy to integrate with AWS services.
- Allows back-fills.
- Easy interface to interact with logs.

Redshift:

- Easy to start querying large amounts of data quickly.
- On-premise data warehousing can be expensive.
- Higher performance for aggregation queries.
- Synced with the cloud ecosystem of AWS. So we can easily use other services of AWS along with Redshift.

Pyspark:

- With in-memory processing, it helps you increase the speed of processing. And the best part is that the data is being cached, allowing you not to fetch data from the disk every time thus the time is saved.
- High-processing speed.
  Follows a distributed processing system for faster speed.
- It provides Fault-tolerance.
- Real-time streaming is possible to handle in pyspark. So if a requirement comes, where the data needs to be processed on a real-time basis, pyspark can be of greater help.

How often the data should be updated and why?

If the data changes significantly every hour, the data can be either updated multiple times a day or at midnight. For example, these days the worldometer site for monitoring the COVID situation updates at midnight. If the data statistics does not change significantly in a short span of time, we can think of maybe updating the data periodically over a week/month or even a quarter.

Approaching the problem under following scenarios:

1. *The data is increased by 100x*
   I would continue to use Airflow, PySpark and Redshift even if the data is increased by 100x because of the following reasons:
   - Redshift has a columnar storage hence the querying should be faster.
   - Pyspark is used to do the heavy lifting for handling the data. It can be used by increasing the nodes in the cluster for more distributed processing of the data.
   - I can store the processed data in S3 since it is elastic in nature.
   - I would use Airflow since it provides a monitoring and managing interface, where it is possible to have a quick overview of the status of the different tasks, as well as have the possibility to trigger and clear tasks or DAGs runs.

2. *The data populates a dashboard that must be updated on a daily basis by 7am every day.*
   The scheduling is easily done using Airflow. I have also included the line of code in the main dag file. However, since the data needs to be populated daily in the morning I would take backup of the old data and include this task before the tables are dropped.

3. *The database needs to be accessed by 100+ people.*
   Redshift is fully capable of handling a lot of users. We can create and manage database users using the Amazon Redshift SQL commands such as CREATE USER and ALTER USER. The people who get the access to the database can only be able to query the Redshift tables.

STEP 6: Results

Following questions have been addressed by querying the Redshift tables:

Ques 1: From which country maximum immigrants arrived in the US?
Query:

```
SELECT A.country_code, A.country_name, COUNT(B.cicid) AS num_people
FROM i94citres AS A
LEFT JOIN immigration AS B
ON B.i94cit = A.country_code
GROUP BY A.country_code, A.country_name
ORDER BY COUNT(B.cicid) DESC;
```

country_code	country_name	num_people
245	China, Prc	275755
135	United Kingdom	213808
209	Japan	210156
689	Brazil	183107
582	Mexico Air Sea, And Not Reported (I-94, No Land Arrivals)	151242
438	Australia	95197
213	India	94540
111	France	79233
687	Argentina	67733
117	Italy	66268

Ques 2: Which country has the highest temperature uncertainty?

Query:

```
SELECT city, country, MAX(average_temperature_uncertainty) AS max_uncertainty
FROM temperature
GROUP BY city, country
ORDER BY MAX(average_temperature_uncertainty) DESC;
```

city	country	max_uncertainity
Copenhagen	Denmark	15.396
Odense	Denmark	15.396
Malmö	Sweden	15.359000000000002
Aalborg	Denmark	15.03
Århus	Denmark	15.03
Rostock	Germany	14.828
Bremerhaven	Germany	14.66
Kiel	Germany	14.66
Bremen	Germany	14.66
Hamburg	Germany	14.66

Ques 3: Which type of airport has the least elevation feat?

Query:

```
SELECT type, MIN(elevation_ft) AS min_elevation
FROM airport_codes
GROUP BY type
ORDER BY MIN(elevation_ft) ASC;
```

type	min_elevation
medium_airport	-1266.0
small_airport	-210.0
heliport	-99.0
closed	-15.0
large_airport	-11.0
seaplane_base	0.0
balloonport	37.0

Ques 4: What is the maximum average household size?

Query:

```
SELECT city, state, MAX(average_household_size) AS max_household_size
FROM us_cities_demographics
GROUP BY city, state
ORDER BY MAX(average_household_size) DESC;
```

city	state	max_household_size
Brentwood	New York	4.98
Perris	California	4.78
Santa Ana	California	4.58
Florence-Graham	California	4.57
Lynwood	California	4.43
Fontana	California	4.15
Baldwin Park	California	4.13
Oxnard	California	4.08
Compton	California	4.08
South Gate	California	3.97