Data Engineering Capstone Project

I chose to complete the Udacity provided project.

1. SCOPE THE PROJECT AND GATHER DATA

There are four datasets used in this project:

a. **Immigration (I-94) Data:** this is data on arrivals of passengers on flights into the United States in 2016. It includes information such as the visa type of the passenger, their I-94 arrival dates, times, countries of origin, ages, flight numbers, etc. There is a separate file per month of the year. However, for the purposes of this project, only the records from February 2016 will be analyzed. The total number of records in February is 2,570,544. This dataset was provided as a .sas7bdat file, but was converted into a CSV file in order to be consistent with the other major dataset (temperature).

b. Temperature Data: this is data on average land temperatures in cities around the world. It includes the date, average temperature, average temperature uncertainty, city, country, latitude, and longitude. The total number of records is 8,599,213. However, only the temperatures in US cities will be studied. This dataset is provided as a CSV file.

c. Airport Codes Data: This information was taken from the I94_SAS_Labels_Descriptions.SAS file that came along with the immigration dataset. This file includes only the values in the i94port field, which contains the airport codes of the ports of arrival of passengers into the United States. This information was copied over from the I94_SAS_Labels_Descriptions.SAS file and put into a text (.TXT) file.

d. US City Demographic Data: this includes demographic information on US cities, such as the total numbers of males, females, foreign-borns, and the predominant race in the city along with a count of the number of people who belong to that race. The total number of records is 2,892.

This analysis seeks to answer the following questions from the data:

- Which states were the most frequently visited by travelers to the United States during February 2016?
- What visa category was most common among travelers entering the United States during February 2016?
- Which airports did most travelers fly into?
- What were the average, minimum, and maximum temperatures in the most commonly visited city during February across the entire temporal range of the temperature data (i.e. 1743-2013)?

The end use-case will be to create a data warehouse within AWS Redshift that can be used as a source of truth database to run customized, ad-hoc queries on the data similar to and including the ones above.

2. EXPLORE AND ASSESS THE DATA

Data exploration and checks for data quality were made using a Spark cluster deployed on AWS. First, the individual data files were uploaded to a newly created S3 bucket called 'capstoneudacity' in a folder called 'source_data'. For the immigration data, only the file from February 2016 was used.

Several issues were found with each source dataset. A sequence of processing steps were followed to 'clean' each dataset. The processing steps to extract, transform, and load the datasets (back into S3) are given below:

Immigration Data:

1. The dataset was initially provided as files in the SAS7bdat format, and for ease of use and to be consistent with the other datasets, the file for February 2016 was converted into a CSV file with Pandas. This was then read in as a PySpark dataframe.

The dataset was converted into a CSV file with the function:

```
fh = input_data + "i94_feb16_sub.sas7bdat"
output_path = input_data + "i94_feb16_sub.csv"
df = pd.read_sas(fh, 'sas7bdat', encoding="ISO-8859-1")
df.to_csv(output_path)
```

With the *input_data* variable pointing to the folder containing the SAS file, and the *output_path* variable pointing to the folder where the output would be saved

2. Only the columns that would be used in the final analysis were kept, and the remaining columns were removed. The columns that were kept include *cicid, i94port, and visatype*.
3. Some of the columns had either empty string or NaN values, and these were replaced with None/Null.
4. A check was performed for duplicate values on the primary key field (*cicid*), but no duplicates were found.
5. The data types of certain columns were initially float (i.e. with a decimal point), but there is no need to have them stored as float when they can be stored as integers since they had a 0 after the decimal point. Therefore, these columns' values were converted from float to integer.
6. The dataframe was also filtered based on the *i94mode* variable to keep only air arrivals into the United States. 'Sea', 'Land', and 'Not reported' arrivals were removed from further processing.
7. Another check for duplicate values across all columns (i.e. rows that contain the same value across all columns) was performed, and no duplicate rows were found.
8. The output PySpark dataframe was written to parquet files named 'immigration.parquet' in the S3 bucket under the folder name 'formatted_data'. The parquet file will be split up

into multiple smaller files once this process is run, and these files will be visible in the 'formatted_data' folder under the immigration.parquet' folder.

Temperature Data: The temperature data goes from November 1, 1743 to September 1, 2013. It presents a historical monthly average temperature (and its associated uncertainty) for cities around the world.

1. The data was read as a CSV file into a PySpark dataframe.
2. Only temperature data for the US was kept (filtering based on 'Country' = 'United States').
3. The 'latitude', 'longitude', and 'AverageTemperatureUncertainty' columns were removed from the dataframe as they wouldn't be analyzed.
4. The AverageTemperature column was rounded to two decimal places.
5. There is a separate row for every month from 1743 to 2013, but since we are only analyzing February historical temperatures, only temperatures for February for all years were kept. This resulted in a historical record of February temperatures from 1743 to 2013 for each city.
6. A check for the count of missing values (i.e. empty strings, NaNs, 'NULL' strings) was performed but no such values were found, meaning that the temperature data did not contain any missing or empty values.
7. A check for duplicate rows was performed. Duplicate rows are rows where the value is the same for each column across all columns. 18 total duplicate rows were found and these were removed from the dataframe.
8. The dataframe was aggregated by the *City* attribute, and average, minimum, and maximum temperatures were found for each unique city.
9. The output dataframe was written to parquet files named 'temperature.parquet' in the S3 bucket under the folder name 'formatted_data'. The parquet file will be split up into multiple smaller files once this process is run, and these files will be visible in the 'formatted_data' folder under the 'temperature.parquet' folder.

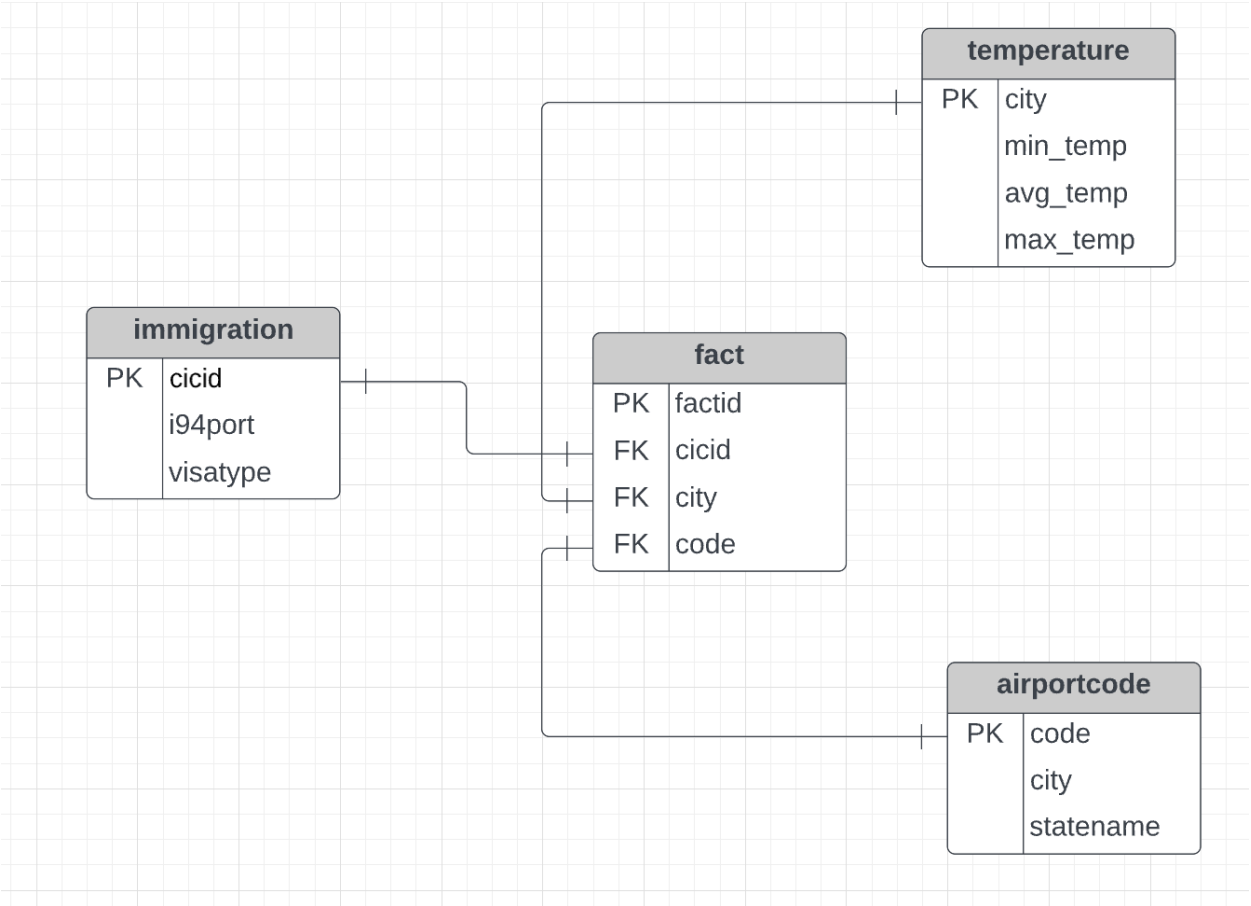
Airport Codes Data:

1. The codes and values for the *i94port* variable were copied over from the file I94_SAS_Labels_Descriptions.SAS and put into a separate text file named 'us_codes.txt', which was then uploaded to the S3 bucket.
2. This was then read as a text file into a PySpark dataframe with one column named 'value'.
3. Some of the airport codes have a corresponding value of 'No Port Code' and 'Collapsed'. These rows were removed from the dataframe.
4. The 'value' column was split into two columns, 'code' and 'citystate', on the "=" in the value string.
5. The 'citystate' column was then further split into two separate columns named 'city' and 'state' on the comma (",") character.

6. Spaces and single quotation marks were then removed from the “code”, “city”, and “state” columns.
7. The ‘code’ and ‘city’ columns had tab characters (i.e. the ‘\t’ character) in their string, and this was removed. Furthermore, the city name was converted into title case (first letter uppercase, all other letters lowercase for every word in the name).
8. Since the states were given as abbreviations and not full names (for example, ‘AK’ instead of ‘Alaska’), their full names were found by looking up the value from a dictionary mapping the abbreviation to the full state name. The full name was added as another attribute named ‘statename’.
9. The output dataframe was written to parquet files named ‘codes.parquet’ in the S3 bucket under the folder name ‘formatted_data’.

3. DEFINE THE DATA MODEL

The conceptual data model looks like below:



A **data dictionary** for the final data model can be found at the end of this writeup.

To pipeline the source data into this data model, we use Spark running on an EMR cluster. In the process, we clean the data from missing/duplicate values and other formatting errors.

Here are the pre-requisite steps for setting up Spark:

1. You should already have an EMR cluster created in AWS. The EMR cluster should have Spark installed on it.
2. Provide your AWS credentials for your AWS account where the EMR cluster was created within the config file named aws_creds.cfg.
3. Clone the Git repository to a directory on your local machine.
4. SSH into the master node of the EMR cluster using `ssh -i PATH_TO_KEY_PAIR_FILE hadoop@MASTER_NODE_PUBLIC_DNS`
5. Copy over the aws_creds.cfg and analyze_data.py files from the cloned Git repository onto the master node EC2 instance using:

```
scp -i PATH_TO_KEY_PAIR_FILE PATH_TO_FILE_TO_COPY
hadoop@MASTER_NODE_PUBLIC_DNS:~/FOLDER_ON_MASTER_NODE_WHERE_TO_COPY
```

where:

PATH_TO_KEY_PAIR_FILE is the path to where the private key file (.pem) is located on disk PATH_TO_FILE_TO_COPY is the path to the file on disk that you want to copy to the master node EC2 instance; this is either the path to the dl.cfg file or the etl.py file

MASTER_NODE_PUBLIC_DNS is the public DNS name of the master node e.g. ec2-35-90-245-54.us-west-2.compute.amazonaws.com
FOLDER_ON_MASTER_NODE_WHERE_TO_COPY is the folder under the master node's root directory where the two files will be copied

6. Then, navigate (`cd`) into the FOLDER_ON_MASTER_NODE_WHERE_TO_COPY directory on the master node and run this command:

```
spark-submit etl.py
```

7. View the output parquet files in the S3 bucket ‘capstonebucketudacity’ (or your own bucket).

This will run the PySpark job that will output cleaned and well-formatted parquet files in the S3 bucket.

4. RUN ETL TO MODEL THE DATA

The ETL to model the data was performed using Apache Airflow. A DAG was created that would read the parquet files from S3 and load them into their appropriate tables in a Redshift cluster that was created within AWS. From there, analytical queries could be run on the data to answer the questions posed in Section 1.

The Redshift cluster should have a database named ‘dev’. Make sure that this cluster is publically accessible, allows access on port 5439, and has VPC Routing enabled for faster access.

Then, you need to create an IAM user in AWS and record the user's Access Key ID and Secret Access Key.

The redshift connection information needs to be added as a Postgres Hook named 'redshift' in Airflow, whereas the IAM user's access key and secret access key need to be added as an Amazon Web Services hook named 'aws_credentials' in Airflow.

Both of these hooks need to be added to Airflow first in order for the DAG to execute since the DAG uses these hooks in its execution.

The Redshift cluster should also be given an IAM role that allows it to access the parquet files in the S3 bucket. This role should be manually created in the AWS console. Line 36 in the *load_redshift.py* file needs to be changed to use the ARN of this role, for example, '*arn:aws:iam::851835621396:role/myRedshiftRole*'. The AWS Hook defined in the DAG will use this name to get the ARN associated with the IAM role within the COPY command that loads the parquet files into the Redshift cluster.

Make sure to create the Redshift cluster in the same AWS region as the S3 bucket. Since the S3 bucket was created in ‘us-west-2’, the Redshift cluster was also created in us-west-2.

5. CONCLUSIONS

We also have two data quality checks in place that, respectively, 1) check for the count of rows in all tables and 2) whether the *cicid* column in the fact table is unique.

To answer the questions posed in Section 1, we can use simple SQL queries on the data.

- Which states were the most frequently visited by travelers to the United States during February 2016?

```
SELECT
 a.statename,
 COUNT(a.statename) AS times_visited

FROM
public.immigration i join public.airportcode a
on a.code = i.i94port
GROUP BY
a.statename

ORDER BY
 times_visited DESC

LIMIT 1;
```

The answer is **Florida**, with a total of 566,191 passengers visiting during February 2016.

- What visa category was most common among travelers entering the United States during February 2016?

```
SELECT
 i.visatype,
 COUNT(i.visatype) AS num_visa

FROM
public.immigration i
GROUP BY
i.visatype

ORDER BY
 num_visa DESC

LIMIT 1;
```

WT, or "waiver-tourist" or "temporary tourist visitor", was the most common visa type for passengers entering the US by air in February 2016.

- Which airports did most travelers fly into?

Looking at the fact table, we can get this information. Each row in the fact table represents a separate passenger, and tells us the airport and city they visited.

SELECT code, COUNT(*) AS numVisited
FROM public.fact
GROUP BY code
ORDER BY numVisited DESC
LIMIT 1;

Miami, Florida (code MIA) was the most visited airport/city in February 2016.

- What were the average, minimum, and maximum historical February temperatures in the most commonly visited city across the entire temporal range of the temperature data (i.e. 1743-2013)?

SELECT min_temp, avg_temp, max_temp
FROM public.temperature
WHERE city = 'Miami'

The minimum average Feb historical temperature was 13.84, average was 18.66, and maximum was 22.49 degrees Fahrenheit.

6. **ALTERNATE SCENARIOS**

- If the data was increased by 100x.
    - Since the code runs on a Spark AWS EMR cluster, the size of the input data is not an issue. Spark can scale to support petabytes of data, since it is a big data processing framework. You can extract data from a variety of sources, process it at scale, and make it available for applications and users. The total size of the input data currently is 1.2 GB. Even if the input data was increased 100 times, this would still yield a total of only 120 GB of data, which is far less than the maximums EMR can support.
- If the pipelines were run on a daily basis by 7am.
    - Spark jobs can be scheduled to run at certain times through EMR.

    - The Airflow pipeline that reads the output of these Spark jobs from S3 can also be scheduled to run at a certain scheduled interval, like hourly or daily at a certain time. Currently, it is set to run only once.
    - Therefore, the entire data pipeline can be orchestrated to run at 7 am every day.
- If the database needed to be accessed by 100+ people.
    - The Redshift cluster, which contains the output data stored in fact and dimension tables, can be made accessible to end users through permissions via a certain IAM role or it can be made accessible to a certain set of IAM users directly. If it is made accessible via a role, end users can then 'assume' the role using AWS Security Token Service. You can also configure inbound rules for the cluster's security group so that only certain IP's can access the cluster on port 5439. AWS makes managing permissions easy with IAM, VPC's, subnets and security groups.

Data Dictionary for final data model

public.immigration

Column Name	Data Type (Redshift)	Definition
cicid	int4	cicid of passenger (unique ID)
i94port	varchar	Port of arrival into the US
visatype	varchar	Type of US visa

public.temperature

Column Name	Data Type (Redshift)	Definition
city	varchar	City name
min_temp	float8	Minimum aggregate February temperature
avg_temp	float8	Average aggregate February temperature
max_temp	float8	Maximum aggregate February temperature

public.airportcode

Column Name	Data Type (Redshift)	Definition
code	varchar	Airport code
city	varchar	City where airport is located
statename	varchar	Full name of state

public.fact (FACT TABLE)

Column Name	Data Type (Redshift)	Definition
factid	int4	Unique ID
cicid	int4	cicid (foreign key)
city	varchar	city (foreign key)
code	varchar	code (foreign key)