ETL Project - Final Report

Team Members:

Alexandra Taft
Anthony English
Arlette Varela
Nathan Bolt

Extract:Transform:Load - Automating Jobs Away

August 2020

Overview

The project was broken down into three main stages: 1) Extract, 2) Transform, and 3) Load.
The datasets used combine automation probability data with a breakdown of the number of jobs by salary in each occupation by state (within the US). For the purposes of our project we filtered down to the detailed level of occupation titles and assigned them an occupational group. We loaded the data into PgAdmin as a SQL database.

DATASET(S):

Kaggle Datasets: Occupation, Salary, and the Likelihood of Automation

Extract

Our original data sources consisted of two files:

1. automation_data_by_state.xlms: this dataset was acquired from the work of Carl Benedikt Freyand Michael A. Osborne;
2. occupation_salary.csv: State employment data is from the Bureau of Labor Statistics.

These files were downloaded from Kaggle (source listed above). The xlms file was converted to a csv. Both files were then extracted into DataFrames using pandas library in jupyter notebook.

All jobs where data was not available or there were less than 10 employees were marked as zero by the originating author of this dataset.

Transform

After extracting the CSV's into Pandas dataframes in Jupyter notebooks, we were ready to begin the data clean up and transformation process.

We first created an ERD to use as a blueprint for the creation of our tables that was uploaded to our SQL database in pgAdmin 4. We then created filtered dataframes from the specified columns in our ERD.

The following clean up and transform process was performed on the automation dataframe - one SQL table was generated from this data:

- Removed hyphens and commas from data to ensure proper loading into sql database;
- Renamed columns to best match SQL tables, including adding underscores to state names consisting of more than one word;
- Set the index to the detail_id;

- converted string integers to floats (pgAdmin required dbl--float was the Pandas equivalent).

As far as the occupation salary data file: The original data had categories for each job title, including detail, broad, minor, and major. In order to preserve this information and have it accessible in our database, we created a total of four tables - one for each category - which we later uploaded into our SQL database. The following clean up and transform process was performed to achieve this result:

Table: occupation_detail

- The purpose of this table is to house the broad id, total employees, and annual (a) & hourly (h) means and medians.
- Created a filtered dataframe from specific columns: detail id, occupation title (OCC_TITLE), broad id, total employees (total_emp), mean and median annual salary (a_mean and a_median), and the mean and median hourly rate (h_mean and h_median).
- Renamed columns to match SQL tables;
- Cast ID's as integers;
- Converted string integers to floats (pgAdmin required dbl--float was the pandas equivalent)

The purpose of the remaining three tables (occupation_broad, occupation_minor and occupation_major) is to house their respective category IDs: broad, minor and major. For each of these tables, we performed the same clean up and transform process:

- Created a filtered dataframe from specific columns: broad/minor/or major ID and occupation title;
- Renamed columns to match SQL tables;
- Cast id's as integers;
- Converted string integers to floats (pgAdmin required dbl--float was the pandas equivalent).

Load

In the final portion of the project, we loaded our tables into a SQL database using pgAdmin 4. The reason we utilized SQL to build our database was that the wanted

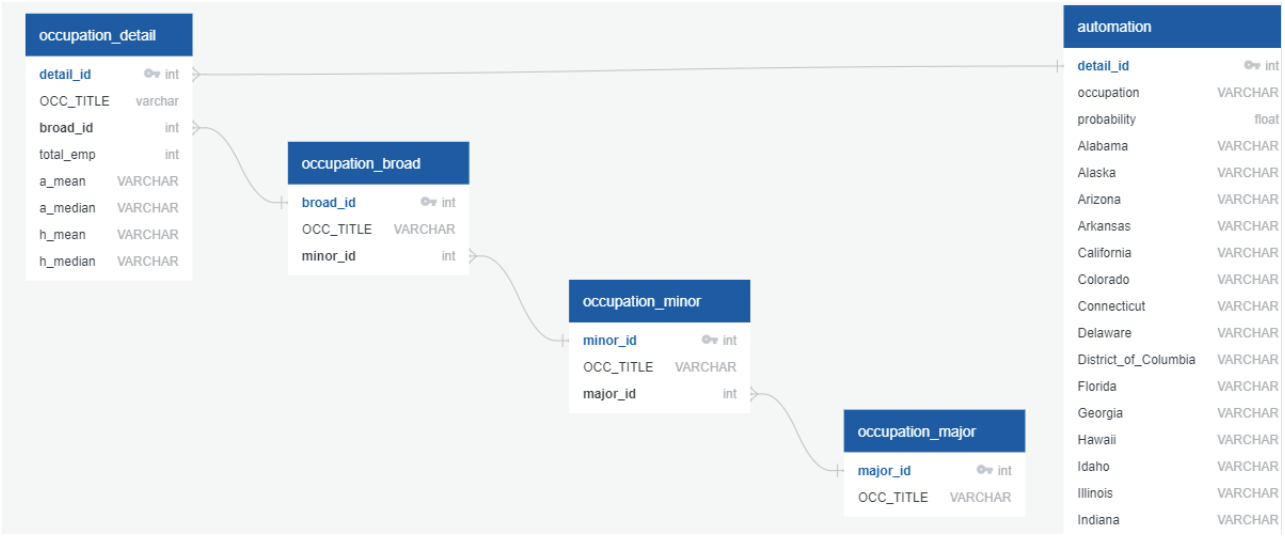
to use a relational database since our data was interconnected and dependent. The loading process was achieved via the following steps:

- A connection string was created to PostGres within Jupyter notebooks;
- A SQL database titled Automation_ETL was created in pgAdmin;
- Tables were created in our SQL database for each of the five dataframes;
- Loaded each dataframe into the SQL database using the .to_sql command in Jupyter notebooks.

The final SQL database consisted of the following tables:

- automation
- occupation_broad
- occupation_detail
- occupation_major
- occupation_minor

The automation table was linked to the occupation_detail table, and the remaining category tables were linked to their parent tables. For example, the broad id connects to the broad_id in the occupation table, the minor id connects to the minor id in the occupation_broad table, etc.



Summary

In this project we were able to successfully extract, transform, and load our data into a database. Using a relational database allows us to link information from different

tables based on common data. We can now filter our employee and salary information by category, type, and state as well as edit data in a table while maintaining the integrity of the data in the remaining tables.