ETL Report

Dennis Kelly, Eduard Stalmakov, Tarick Mehanna

9/4/2022

Introduction

As a group, what do American business owners look like? What are their demographic profiles? What are their educational backgrounds? At the broadest level, our group wanted to analyze these questions, as well as investigate any possible relationships between these characteristics and the characteristics of the firms they own. We consider these questions interesting in their own right, and furthermore deem their understanding an important prerequisite to examining more specific phenomena affecting the American business landscape.

However, to answer even these preliminary questions, it was first necessary to find data and ensure they were in a format suitable for analysis and visualizations. This report outlines where the data were found, as well as the transformative steps we applied to them.

Data Sources

All data came from the US Census Annual Business Survey (ABS), specifically the Characteristics of Businesses (CB) and Characteristics of Business Owners, and were accessed using the Census-provided APIs. The first dataset records information about the number of firms, the number of employees at those firms, and the annual payroll and revenue of those firms, broken down along the demographic profile of the owners, as well as specific characteristics of the firm such as if it is family-owned or how long it has been in business.

The second table records information about the number of owners with a particular demographic profile also having a particular characteristic, such as what level of education they have received or what motivated them to start a business. We did not pull in information on every variable in our analysis, omitting in both queries the ethnicity of the firm owner, whether the firm owner is a veteran, and geographic information. Even though state-level data are available in these datasets, our group only examined data aggregated to the nationwide level.

Extraction

We performed all ETL steps in Python, making specific use of the requests and pandas libraries. The extraction phase comprised two main steps: querying the API to get the CBO data, and converting it to a pandas dataframe, and then doing the same for the CB data. We outline specifics below:

1. Using the requests library, we sent a GET request to the following URL (https://api.census.gov/data/2018/abscbo?get=NAICS2017,NAICS2017_LABEL,QDESC,QDESC_LABEL,OWNER_RACE,OWNER_RACE_LABEL,OWNER_SEX,OWNER_SEX_LABEL,OWNCHAR,OWNCHAR_LABEL,OWNPDEMP,OWNPDEMP_PCT&for=us:*) to obtain the CBO data. We omit the portion of the URL containing the API key; anyone seeking to recreate these steps will need to obtain one to make this GET request in Python.

2. The data are formatted as a JSON, so we use the built in .json() method of the response object in Python to convert this to a dictionary. We then use pandas.DataFrame.from_dict() to convert this dictionary into a pandas dataframe.
3. Similarly, to obtain the CB data we sent a GET request to the following URL (https://api.census.gov/data/2018/abscb?get=NAICS2017,NAICS2017_LABEL,RACE_GROUP,RACE_GROUP_LABEL,SEX,SEX_LABEL,QDESC,QDESC_LABEL,BUSCHAR,BUSCHAR_LABEL,FIRMPDEMP,FIRMPDEMP_PCT,EMP,EMP_PCT,PAYANN,PAYANN_PCT,RCPPDEMP,RCPPDEMP_PCT&for=us:*), and then performed identical steps to create a pandas dataframe as well, completing the extraction phase.

Transformation

The data collected by the Census are very complete and sensibly formatted, so we did not have to do much transforming of the data in order to conduct our analysis. A detailed break down of the steps is given below:

1. In the dataframe containing the CBO data, replace the default columns with the values contained in the first row, which pandas did not automatically recognize as the header, and then drop the first row.
2. Drop the 'us' column, which is redundant and records the same value for all observations.
3. The 'OWNPDEMP' and 'OWNPDEMP_PCT' fields need to be converted to numerical data types- int and float respectively. Use the .apply() method of a pandas dataframe in concert with the pd.to_numeric() function to perform this conversion.
4. We then apply similar steps to the dataframe containing the CB data.
5. First replace the columns with the values contained in the first row, then drop the first row as well as the 'us' column.
6. Using the same combination of .apply() and pd.to_numeric(), we then convert the "FIRMPDEMP", "FIRMPDEMP_PCT", "EMP", "EMP_PCT", "PAYANN", "PAYANN_PCT", "RCPPDEMP", and "RCPPDEMP_PCT" fields to the appropriate numerical types. The pd.to_numeric() function automatically gets the "_PCT" fields to floats and the others to integers.
7. Our group wished to see how certain business owner characteristics covaried with firm characteristics at the industry level. In order to do this, we needed to combine the two extant dataframes with a merge operation.
8. We merged the CBO dataframe with the CB dataframe, the former being the left and the latter the right, although the order did not matter as we used an inner join. We joined on the equivalence of the "NAICS2017", "OWNER_RACE", and "OWNER_SEX" fields in the CBO dataframe with the "NAICS2017", "RACE_GROUP" and "SEX" fields in the CB dataframe.
9. This merge creates some redundant columns, and some awkwardly named columns. We drop the "NAICS2017_LABEL_y", "RACE_GROUP", "RACE_GROUP_LABEL", "SEX", and "SEX_LABEL" columns, as these reproduce exactly the information contained in the conjugate columns from the CBO dataframe.
10. As for new names, we renamed the "NAICS2017_LABEL_x" column to "NAICS2017_LABEL", the "QDESC_x" and "QDESC_LABEL_x" columns to "CBO_QDESC" and "CBO_QDESC_LABEL" respectively, and finally the "QDESC_y" and "QDESC_LABEL_y" columns to "CB_QDESC" and "CB_QDESC_LABEL".

11. We note here that the resulting merged dataframe is quite large, with well over 1 million rows, and that it takes some time for this operation and all subsequent operations on this dataframe to run. In our group's case, we prioritized the flexibility provided by retaining as much of the data as possible. However, we advise anyone seeking to work with this data to think carefully if such flexibility is needed for their own purposes, and if not, to filter down each constituent table to only the data they are interested in before performing the merge described above.
12. At this, the transformations applied to the data are complete.

Load

In order to obviate the need for any future API calls, as well as to ensure that all group members were working off of identically-processed data, we saved all three dataframes created in the above phases of the ETL process as CSV files, using the .to_csv() method of pandas dataframes. We could easily read these files back into Python with the pd.read_csv() function, and they served as the basis for subsequent visualizations.

Conclusion

This report describes in words the main thrust of the ETL process, however it is often very useful to see the actual code in which the work was done. For this reason, we include in our GitHub repository the get_daya.iypnb Jupyter notebook, so all steps can be seen exactly as they were performed.

Moreover, some additional transformations have been applied on an incidental basis in the production of individual graphics. In this report we have not described these transformations, typically such small things as the addition of a derived column to, the selection of particular rows or columns from, or the application of a group-by or pivot operation to one of the pre-existing dataframes whose creation is outlined above. They are almost always easily reverse-engineered from the graphic itself, and in any case we also include the code performing these operations in the GitHub repo, in the three Jupyter notebooks named after each of the three group members.