ETL Report – Group 6

Dalton Bode, Hang Zhang Cao, Ted Korby, Raymond Sepulveda

January 16th, 2022

# Introduction

Through the process of recording a plethora of various survey types, the Census Bureau collects and stores massive quantities of data from these surveys onto a privately accessible API which can be accessed by the public if they obtain authorized access via an API key. This API contains survey data in many sectors including business, economy, security, manufacturing and more. For our project, we are focused on the Annual Business Surveys, ABS for short, which we can easily call upon and obtain data through the Census API.

After consuming the varying types of data that the Census API contains, we determined that we would base our questions around a topic such as employee demographics, annual pay rates, firm numbers, the number of US firm owners over time, and firm differentiation/patterns by geographical location. This allowed us to formulate questions applicable to the real world which can be answered using real-world data. The questions we are imposing on the data at hand are the following:

**Questions:**

1. Are there any major discrepancies between races and the average annual pay?
2. Did the number of owners of mid-size firms (50-99 employees) established between 2008 – 2012 increase or decrease in MN, NY, and WI when compared to the period 2000 – 2007? What seems to be the reason behind this?
3. Are there any major discrepancies between certain demographics, industries, and company sizes based on the average annual payroll?
4. Is there a trend of race densities varying by geographical location? If so, where are there more or less of a certain race and what might be the reason behind it?
5. Is there a trend of veteran status densities that vary by geographical location? If so, what are the trends?

We need to transform the raw data from the API due to each survey or item set acting as a single object. Since we need to obtain the entirety of the data due to data pulling restrictions with the API, we will need to extract specific information regarding our questions. Raw data is simply not enough to answer complex questions/problems.

# Data Sources

The data we use in this project is provided by the Census API which is hosted by the United States Census Bureau. We chose to use Annual Business Survey as our data focus due to the versatility of the information recorded through the surveys which will help us more accurately answer our questions. The main tools that we used when transforming and representing the data are the Pandas and matplotlib libraries for Python. We used these tools due to their simplicity and popularity in the realm of data which allowed us to get answers more easily to the questions that we had pertaining to how the libraries function and work with data.

**Data:**

US Census Bureau. (2021, October 14). *Annual Business Survey (ABS) APIs*. Census.gov. Retrieved January 14, 2022, from <https://www.census.gov/data/developers/data-sets/abs.html>

**Tools:**

API reference. API Reference - Matplotlib 3.5.1 documentation. (n.d.). Retrieved January 15, 2022, from <https://matplotlib.org/stable/api/index>

Hoefler, P. (n.d.). *Pandas.dataframe*. pandas.DataFrame - pandas 1.3.5 documentation. Retrieved January 15, 2022, from <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html>

# Extraction

There are two scales of data that we look at in this project, data on the state level and that of the country. Based on the question you aim to answer, follow the appropriate steps listed below to get the desired results.

**Base requirements:**

Start off by installing the following libraries via pip in the command line:

* requests, json, pandas, and matplotlib
  + Ex. ‘pip install requests’

Once these packages are installed, we can now import them via our IDE (VS Code, Jupyter Notebook, PyCharm, etc.) utilizing the import function. Import the following:

* Requests, json, pandas as pd, matplotlib.pyplot as plt
  + Ex. ‘import pandas as pd’

We will need a key to request data from the API.

* Obtain an API key from [https://api.census.gov/data/key\_signup.html,](https://api.census.gov/data/key_signup.html) the key will be sent to your email after being validated. Copy and store this in a variable (\_key)

**For US data:**

1. Set up a variable name (\_area), set the value to ‘us’
2. Using requests.get() function, call the HTML data from the Census API using the following URL formatted string and save it to a variable of your choice:

*\_url = f'https://api.census.gov/data/2018/abscs?get=GEO\_ID,NAME,NAICS2017,NAICS2017\_LABEL,SEX,SEX\_LABEL,ETH\_GROUP,ETH\_GROUP\_LABEL,RACE\_GROUP,RACE\_GROUP\_LABEL,VET\_GROUP,VET\_GROUP\_LABEL,EMPSZFI,EMPSZFI\_LABEL,YEAR,FIRMPDEMP,FIRMPDEMP\_F,RCPPDEMP,RCPPDEMP\_F,EMP,EMP\_F,PAYANN,PAYANN\_F,FIRMPDEMP\_S,FIRMPDEMP\_S\_F,RCPPDEMP\_S,RCPPDEMP\_S\_F,EMP\_S,EMP\_S\_F,PAYANN\_S,PAYANN\_S\_F&for={\_area}:\*&key={\_key}'*

1. Using the stored HTML data from the previous step, convert the data into an easily workable json format then store it in a variable.
2. You can save this data into a file for later use without needing to wait on the get request to fill every time. Write data to a json file of your choice using the ‘with open()’ and ‘json.dump()’ functions.
3. Using pandas.read\_json(), read in the variable from the previous step and store the Dataframe object into a separate, easily referenceable variable.

**For State data:**

1. Set up a variable name (\_area), set the value to ‘state’
2. Using requests.get, scrape the data using the following string:

*\_url = f'https://api.census.gov/data/2018/abscs?get=GEO\_ID,NAME,NAICS2017,NAICS2017\_LABEL,SEX,SEX\_LABEL,ETH\_GROUP,ETH\_GROUP\_LABEL,RACE\_GROUP,RACE\_GROUP\_LABEL,VET\_GROUP,VET\_GROUP\_LABEL,EMPSZFI,EMPSZFI\_LABEL,YEAR,FIRMPDEMP,FIRMPDEMP\_F,RCPPDEMP,RCPPDEMP\_F,EMP,EMP\_F,PAYANN,PAYANN\_F,FIRMPDEMP\_S,FIRMPDEMP\_S\_F,RCPPDEMP\_S,RCPPDEMP\_S\_F,EMP\_S,EMP\_S\_F,PAYANN\_S,PAYANN\_S\_F&for={\_area}:\*&key={\_key}'*

1. Using the stored HTML data from the previous step, convert the data into an easily workable json format then store it in a variable.
2. You can save this data into a file for later use without needing to wait on the get request to fill every time. Write data to a json file of your choice using the ‘with open()’ and ‘json.dump()’ functions.
3. Using pandas.read\_json(), read in the file from the previous step and store the Dataframe object into a separate, easily referenceable variable.

# Transformation

**Average annual payroll for each race group in 2018**

1. Rename the headers of the Dataframe object (census data frame) using the data from the first row (iloc[0]).
   1. Set the ‘inplace’ parameter to True.
2. Drop the first row
3. Drop unnecessary columns from the Dataframe obj, then store it in a new variable.
   1. Columns to drop: ['us','SEX','GEO\_ID','ETH\_GROUP','VET\_GROUP','FIRMPDEMP\_F','RCPPDEMP\_F','EMP\_F','FIRMPDEMP\_S\_F','RCPPDEMP\_S\_F','EMP\_S\_F','PAYANN\_S\_F','PAYANN\_F']
   2. Parameters to use: columns set to list above, ‘axis’ equal to 1, inplace set to True.
4. Create a new data frame, ‘race\_groups\_only’, from the census data frame by selecting all rows in which the column header “RACE\_GROUP\_LABEL” does not equal ‘Total’.
5. We’ll need to do a few aggregations before we can create the data frame used for our data visualizations.
   1. First, we’ll need to figure out the average number of employees for each race group, as this will be used to normalize the average annual payroll.
6. Create new data frame, ‘race\_emps’, by selecting the following columns from the ‘race\_groups\_only’ data frame: [“RACE\_GROUP\_LABEL”, ”RACE\_GROUP”, ”EMP”]
7. Set the race\_emps column “EMP” to integer values by using the .to\_numeric() method from the pandas library.
8. Now select all rows from the race\_emps whose “RACE\_GROUP\_LABEL” values don’t equal the following categories: “Classifiable”, “Unclassifiable”, “Nonminority”, “Minority”, “Equally minority/nonminority”
   1. Use the & operator to string together multiple conditions.
9. Use the .groupby() method on “race\_emps” and aggregate these values by their mean.
   1. Set this equal to the variable name: avg\_num\_employees
10. Now we can create a new data frame named race\_payann. This is similar to what was done in step 6, but “EMP” is replaced with the “PAYANN” column instead.
11. Repeat steps 7 and 8 for the race\_payann data frame.
    1. Set the aggregation data frame to the variable avg\_pay\_by\_race
12. Create a function called “normalization” with the following parameters: indexlist, elementlist
    1. Inside the function:
    2. Set the variable named ‘new\_column’ to an empty list.
    3. Run a for loop over the indexlist
       1. Inside this for loop, run another for loop over the elementlist
       2. Set the following condition:

If (indexlist.index(index)==elementlist.index(element)):

* + 1. When this statement is true, append the following code to the new column list:

element[0]/(avg\_num\_employees.loc[index].values[0])

* 1. Once the for loops have finished cycling, return the new\_column list.

1. Create a list of the index values of the avg\_pay\_by\_race aggregated data frame and set this equal the variable name ‘race\_groups’.
2. With the normalization function we just defined, create a new column called [‘normalized\_avg\_pay’] and set it equal to this function.
   1. Parameters: indexlist = race\_groups, elementlist = list(avg\_pay\_by\_race.values), df = avg\_pay\_by\_race
   2. What this does is it checks to see if the index position of the race groups is equal to the indexed position of the values from the aggregated data frame (i.e., ‘PAYANN’ values). Those lists should be the same size. When that’s true, it divides the average number of employees for that race group from the average pay value.
3. Finally, create a “Race Group” column for the avg\_pay\_by\_race data frame, setting it equal to the race\_groups list.

**Number of owners of respondent employer firms for WI, MN, and NY for various time periods**

1. Read in the state data for Company Summary dataset and Characteristics of Business Owners dataset and store in separate variables.
2. Rename the headers of each dataset using the values from the first row (iloc[0]).
   1. Set ‘inplace’ to True.
3. Drop the first row for both datasets.
4. For the Company Summary dataset, filter out rows where the values in the column 'EMPSZFI\_LABEL' equal to 'Firms with 50 to 99 employees'.
5. For the Characteristics of Business Owners dataset, filter out rows where the values in the column OWNCHAR\_LABEL equal to 2000 to 2007, then filter out 2008 to 2012 data, store them in separate variables.
6. Apply inner merge on each of the variables from the previous step with the Company Summary dataset. Store them in new variables.
   1. Merge on ‘NAME’ column
7. Filter out rows where the values in the ‘NAME’ column equal to either ‘New York’, ‘Minnesota’, or ‘Wisconsin’.
8. Using astype function, convert the ‘OWNPDEMP’ column to numeric values.
9. Drop columns:
   1. GEO\_ID\_x, GEO\_ID\_y, FIRMPDEMP\_F, RCPPDEMP, RCPPDEMP\_F, state\_x, state\_y, OWNCHAR, OWNPDEMP\_F, EMPSZFI, EMPSZFI\_LABEL, FIRMPDEMP, EMP, PAYANN, OWNCHAR\_LABEL.
10. Apply steps 7-10 for both merged variables.

**Average annual payroll by gender, firm size, and industry**

1. Rename the headers of the Dataframe object (census data frame) using the data from the first row (iloc[0]).
   1. Set the ‘inplace’ parameter to True.
2. Drop the first row
3. Drop unnecessary columns from the Dataframe df\_census, then store it in a new variable, df\_census\_dropped
   1. Columns to drop: ['us','SEX','GEO\_ID','ETH\_GROUP','VET\_GROUP','FIRMPDEMP\_F','RCPPDEMP\_F','EMP\_F','FIRMPDEMP\_S\_F','RCPPDEMP\_S\_F','EMP\_S\_F','PAYANN\_S\_F','PAYANN\_F']
4. For the sex dataset:
   1. Create a new dataframe from df\_census\_dropped by selecting the ‘SEX\_LABEL’ column and storing it in a new variable, df\_census\_dropped\_sex
   2. Drop rows with values: Classifiable, Unclassifiable from the ‘SEX\_LABEL’ column
   3. Using the same datatype, select the ‘PAYANN’ column and convert it to an integer using astype(int) or pd.tonumeric()
   4. Using a groupby function on the ‘SEX\_LABEL’ column, use the mean() function to get the averages from the ‘PAYANN’ column grouped by sexes. Store the results in a variable avg\_annual\_pay\_sex.
5. For the company size dataset:
   1. Create a new dataframe from df\_census\_dropped by selecting the ‘EMPSZFI\_LABEL’ column and storing it in a new variable, df\_census\_dropped\_company\_size
   2. Drop rows with values: All firms from the ‘EMPSZFI\_LABEL’ column
   3. Using the same datatype, select the ‘PAYANN’ column and convert it to an integer using astype(int) or pd.tonumeric()
   4. Using a groupby function on the ‘EMPSZFI\_LABELS’ column, use the mean() function to get the averages from the ‘PAYANN’ column grouped by company size. Store the results in a variable avg\_annual\_pay\_company\_size.
6. For the industry dataset:
   1. Create a new dataframe from df\_census\_dropped by selecting the ‘NAICS2017\_LABEL’ column and storing it in a new variable, df\_census\_dropped\_industry
   2. Drop rows with values: Total for all sectors, Industries not classified from the ‘NAICS2017\_LABEL’ column
   3. Using the same datatype, select the ‘PAYANN’ column and convert it to an integer using astype(int) or pd.tonumeric()
   4. Using a groupby function on the ‘NAICS2017\_LABEL’ column, use the mean() function to get the averages from the ‘PAYANN’ column grouped by industry. Store the results in a variable avg\_annual\_pay\_industry.

***Race and Veteran Status Densities by Geographical Location***

1. We will be using the States data, be sure to follow the steps in the Extraction section to yield the dataframe required.
   1. In these steps, I will refer to the main Dataframe as *df*.
2. Reference *df* and call the rename function on it, changing the column names to be set as the first row in the Dataframe via the columns parameter. Set the inplace parameter to True as well.
   1. This is because the first row in the data retrieved from the API are the column headers but is not directly formatted to the data, so we need to do the formatting ourselves.
   2. Ex. census\_df.rename(columns=census\_df.iloc[0],inplace=True)
3. Reference *df* and call the drop function, dropping the first index with the inplace parameter set to True.
   1. Ex. census\_df.drop(0, inplace=True)
4. Optional: You may print *df*.info() if you wish to see what your Dataframe looks like at the start as it may be helpful to reference later.
5. Since we are focusing on two categories, race and veteran status, we can cut out unnecessary data by referencing only the data we want from the Dataframe.
6. Create a new Dataframe from *df* and have it extract the following columns: “NAME”, “RACE\_GROUP\_LABEL”, “VET\_GROUP\_LABEL”
   1. You can reference a column or columns in a Dataframe by using a string value as an index call. We will reference this Dataframe as *df\_cut*.
   2. Ex. census\_df[["NAME","RACE\_GROUP\_LABEL","VET\_GROUP\_LABEL"]]
7. From this point, we split into two groups: Race and Veteran Status.
   1. Both follow the same set up convention, so you may choose to do one and copy the other processes.
8. Focusing on race to start, we don’t need the VET\_GROUP\_LABEL column, drop that column using the drop function. Save this Dataframe to a new variable name, reference as *df\_races\_cut*, we can use this later if we care to change which race categories we want to look at.
9. Reference the RACE\_GROUP\_LABEL column in *df\_cut,* accepting only a subset of items by excluding the following tags/race labels.
   1. Exclude list: “Classifiable”, “Unclassifiable”, “Nonminority”, “Minority”, “Equally minority/nonminority”, “Total”
   2. *df\_geo\_races = df\_races\_cut*[*(df\_races\_cut*[‘RACE\_GROUP\_LABEL’] != ‘Classifiable’) & (…) & ...]
10. Reference the previously made temporary Dataframe and call the groupby function on name with the index set to RACE\_GROUP\_LABEL. You can call the value\_counts function on the grouped Dataframe in the same line to get the desired product.
    1. *Df\_geo\_racesgrouped = df\_geo\_races*.groupby(“NAME”)[“RACE\_GROUP\_LABEL”].value\_counts(sort=False)
11. With this Series, we can now create the visualizations necessary to attempt answering the questions at hand.
    1. Be sure to repeat steps 6-10 while focusing on the VET\_GROUP\_LABEL, using all but “Classifiable” and “Unclassifiable” label tags in step 9.

# Load

1. Use the following code to save these data frames (df) to your desired file choice:

*with open(‘filename.ext’,’wt’) as alias:*

*df.to\_ext(alias, \*\*kwargs)*

1. Ext stands for extension, in which possible file types to replace ‘ext’ with include:
   1. csv, html, json, dict, excel, markdown, sql, stata
2. \*\*kwargs indicate optional arguments/parameters that you may want to set. This can vary based on the filetype you wish to save the data frame (df) to. For more information on those arguments, explore the [pandas.DataFrame documentation](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html) for the method related to the filetype you wish to save the data frame as.
   1. Example: .to\_csv() has a sep= parameter and a delimiter= parameter, which can be used to select a unique separator or delimiter.

# Conclusion

With these data frames now stored in a SQL database or a file type of your choosing, your team can use them to create visualizations needed for answering the 5 questions outlined in the introduction portion of this document. Check out our group’s project report to see a few examples of what visualizations could be made to answer these questions.