Group XL ETL Report

Table of Contents:

Introduction

For our data capstone project we chose to examine the relationships between educational attainment and employment in the United States between the years of 2015 and 2020. In seraching for datasets relevant to our topic we found a wealth of information through the U.S. Bureau of Labor and Statistics (BLS), the U.S. Census Bureau and the National Center for Education Statistics. Using a variety of conventional tools we cleaned the data we gleaned from government servers and transformed every dataset to suit our needs. In each section we will breifly introduce the dataset we are interested in exctracting followed by an discussion of the cleaning steps involved with code provided to further illuminate our data transformation process.

Labor Statistics From The Current Population Survey

The Current Population Survey (CPS) is survey conducted each month by the U.S. Census Bureau for the BLS which contains data relevant to employment and educational attainment. In tables provided the BLS, we were able to capture this data for our target years of 2015 to 2020 by means of web-scrapping and pandas.To upload and clean the CPS tables, we begin by importing the neccessary packages. Here we

will be using numpy, pandas, re, requests, BeautifulSoup from bs4, sleep() from time and ascii_lowercase from string which we will use later to combine these datasets.

In [ ]:
```
import numpy as np
import pandas as pd
import re
import requests
from bs4 import BeautifulSoup
from time import sleep
from string import ascii_lowercase
```

Because the CPSAAT for years 2016 to 2019 were available in HTML format online, we chose to use a combination of requests and BeautifulSoup to extract all the data for those years. However, because much of the column data extracted via this method was untenable to clean, since the CPSAAT has only eight columns, we hard-coded in the column names. However, the index column names were in well enough shape to be extracted and transformed into a usable condition. After importing and cleaning the data, we saved each dataset as a pickle and a CSV.

In [ ]:
```
for i in range(2016, 2020):
    year = str(i)
    url = f"https://www.bls.gov/cps/aa{year}/cpsaat07.htm"
    request = requests.get(url)
    soup = BeautifulSoup(request.text, 'html.parser')


    table_data = []
    for i in soup.find_all('td'): # Finds all data with tag 'td'
        if i.get_text() == '':
            pass
        elif re.findall('[a-z]', i.get_text()): # Skip alphabetic content
            pass
        else:
            table_data.append(i.get_text())

    clean_data = []
    i = 0
    while i <= len(table_data):
        # We have eight columns in our dataframe so we will append data to raw data ur
        raw_data = []
        try:
            while len(raw_data) < 8:
                    raw_data.append(table_data[i])
                    i += 1
        except:
            break
        clean_data.append(raw_data)

    # We will grab all of the outer_index information.
    outer_index_to_clean = []
    for elements in soup.find_all(class_='sub0'):
            if elements.get_text() == 'Civilian noninstitutional population': #This el
                pass
            else:
```

```
            outer_index_to_clean.append(elements.get_text())

    # We now proceed to accumulate all items in the inner index
    inner_index = []
    for items in soup.find_all(class_=['sub0', 'sub1','sub2', 'sub3', 'sub4']):
        if items.get_text() in outer_index_to_clean:
            pass
        else:
            inner_index.append(items.get_text())

    # Because we are using pd.Multindex, length and order matter
    outer_index = []
    k = 0
    for i in range(0, len(inner_index)):
        try:
            if inner_index[i+1] == 'Civilian noninstitutional population': # The inner
                outer_index.append(outer_index_to_clean[k])
                k+=1
            else:
                outer_index.append(outer_index_to_clean[k])
        except:
            outer_index.append(outer_index_to_clean[k])

    assert len(outer_index) == len(inner_index)

    # We hard code the column names (which are shared for all years of the CPSAAT)
    column_level1 = np.array(((year + ',') * 8).split(',')[:-1])
    column_level2 = np.array(
        [
        'Less than a high school diploma',
        'High school graduates, no college',
        'Some college or associate degree',
        'Some college or associate degree',
        'Some college or associate degree',
        'Bachelor\'s degree and higher',
        'Bachelor\'s degree and higher',
        'Bachelor\'s degree and higher'
        ]
    )
    column_level3 = np.array(
        [
        'Less than a high school diploma',
        'High school graduates, no college',
        'Total',
        'Some college, no degree',
        'Associate degree',
        'Total',
        'Bachelor\'s degree only',
        'Advanced degree'
        ]
    )

    # Now we create a list of column and index arrays
    column_arrays = [column_level1, column_level2, column_level3]
    index_arrays = [outer_index, inner_index]

    # Create the dataframe
    df = pd.DataFrame(clean_data, index=index_arrays, columns=column_arrays)
    df.replace(',', '', regex=True, inplace=True) # We're replacing the commas in each
    df = df.astype(float) # Casting each value as a float data type
```

```
    # We pickle the dataframe and export a clean CSV
    df.to_pickle(f'clean-pickle/cpsaat-{year}.pkl')
    df.to_csv(f'clean-csv/cpsaat-{year}.csv')

    # Be polite to the government servers: sleep for 5 seconds
    sleep(5)
```

Now we will use pd.read_excel to import the final two datasets. In order to keep the dataframe in a format similiar to the previously uploaded tables, we will be using the optional arguments "skiprows," to skip the text at the top of the Excel document, "usecols," to designate that we only need that data between columns B and I, and "header," to tell pandas that this dataset does not have a header. Next we will drop rows containing all null values before cobbling together our dataframe using our previously defined list of index arrays, and a slightly modified list of column arrays altered to reflect the year of the data gathered. Finally, we will save the cleaned CSV and pickle the dataframe.

In [ ]:
```
for year in ['2015', '2020']:
    column_level1 = np.array(((year + ',') * 8).split(',')[:-1])
    column_arrays = [column_level1, column_level2, column_level3]
    xl = pd.read_excel(f"data-capstone\cpsaat{year}.xlsx", skiprows=[i for i in range(
    df = pd.DataFrame(xl.values, columns=column_arrays, index=index_arrays)
    df.to_pickle(f'clean-pickle/cpsaat-{year}.pkl')
    df.to_csv(f'clean-csv/cpsaat-{year}.csv')
```

Next, we'll create a dummy list of lowercase characters and unpickle each of the cleaned datasets to pd.concate them into a dataframe encompassing all the target years.

In [ ]:
```
dummy = list(ascii_lowercase)[0:6]
k = 0
for i in range(2015, 2021):
    year = str(i)
    dummy[k] = pd.read_pickle(f'clean-pickle/cpsaat-{year}.pkl')
    k += 1
```

Finally, we will create a list of our pickled dataframes and pass it into pd.concat and horizontally merge them together by passing the argument "axis=1."

In [ ]:
```
combine = [dummy[i] for i in range(0, 6)]
all_years = pd.concat(combine, axis=1)
```

The first five rows of the resulting dataframe:

In [ ]:
```
all_years.head()
```

Out[ ]:

		Less than a high school diploma	High school graduates, no college	Some college or associate degree			Bachelor's degree and hig		
		Less than a high school diploma	High school graduates, no college	Total	Some college, no degree	Associate degree	Total	Bachelor's degree only	Advan deg
TOTAL	Civilian noninstitutional population	24175.0	61712.0	56263.0	35326.0	20937.0	70061.0	44086.0	259
	Civilian labor force	10971.0	35322.0	37481.0	22706.0	14774.0	52133.0	32684.0	194
	Participation rate	45.4	57.2	66.6	64.3	70.6	74.4	74.1	
	Employed	10098.0	33402.0	35785.0	21573.0	14213.0	50792.0	31772.0	190
	Employment-population ratio	41.8	54.1	63.6	61.1	67.9	72.5	72.1	

5 rows × 48 columns

Now we will export the clean CSV and pickle the dataframe for later usage.

In [ ]:
```
all_years.to_csv('clean-csv/cpsaat-all-years.csv')
all_years.to_pickle('clean-pickle/cpsaat-all-years.pkl')
```

Median Household Income by State: Selected years 1990 through 2019

As always, we begin with importing the necessary packages: numpy, pandas, and re.

In [ ]:
```
import numpy as np
import pandas as pd
import re
```

We first proceed to read in the data and append each line to an empty list "to_clean." As we sort through each line in the file, we are interested in grabbing only those lines which contain numeric data. Before finally appending them to "to_clean," we will replace the characters ",," and "\n."

In [ ]:
```
to_clean = []
with open(r"data-capstone/tabn102.30.csv") as f:
```

```
    for lines in f:
        if re.findall('-[0-9]|[0-9]', lines):
            to_clean.append(lines.replace(',,', '').replace('\n', ''))
        else:
            pass
```

We those entries in our list which content table values. As demonstrated by "raw_data[0]," to clean the data we must seperate each list entry's non-numeric content, and eliminate the footnotes, quotation marks, and dollar signs.

In [ ]:
```
raw_data = to_clean[4:-3]
raw_data[0]
```

Out[ ]:
```
'   United States,"$60,000","$64,600","$60,700",(80),"$58,800",(50),"$58,000",(6
0),"$60,200",(60),"$61,400",(70),"$62,900",(50),"$63,100",(60),"$65,700",(70),'
```

Now we will write a for loop which will launder each data entry:

In [ ]:
```
index = []
clean_data = []
for i in range(0, len(raw_data)):
    clean_string = ''
    cleaner_data = []
    for k in range(0, len(raw_data[i])):
        try:
            if re.findall('[0-9],[0-9]', raw_data[i][k-1] + raw_data[i][k] + raw_data[
                pass
            elif raw_data[i][k] == '"': # Exclude quotation marks
                pass
            elif raw_data[i][k] == '$': # Exclude dollar signs
                pass
            else:
                clean_string = clean_string + raw_data[i][k]
        except:
            pass
    clean_string = clean_string.strip() # Remove extraneous white space
    list_to_clean = clean_string.split(',') # We now need to remove the footnotes
    for items in list_to_clean:
        if re.findall('[a-z]', items): # If the list item has non-numeric content that
            index.append(items)
        elif re.findall('\(', items): # If the list item has a '(', exclude that item
            pass
        else:
            cleaner_data.append(items)
    clean_data.append(cleaner_data)
```

Keeping with the format of the source document, we will use pd.Multindex to preserve the title of this dataset:

In [ ]:
```
title = 'Median Household Income by State: Selected years 1990 through 2019,'
column_level1 = (title * 10).split(',')[:-1] # the last entry is ''
column_level1 = np.array(column_level1)
column_level1.shape
```

Out[ ]:
```
(10,)
```

Next we hard-coded the columns for the next level.

In [ ]:
```
column_level2 = np.array([1990, 2000, 2005, 2010, 2014, 2015, 2016, 2017, 2018, 2019])
```

Now that we have each of our columns, we can now assemble a list of column arrays.

In [ ]:
```
column_arrays = [column_level1, column_level2]
```

After transforming and accumulating all of the data we can now assemble the dataframe. Finally, we will cast each data entry as an integer

In [ ]:
```
df = pd.DataFrame(clean_data, columns=column_arrays, index=index)
df = df.astype(int)
```

The first five rows of our dataframe:

In [ ]:
```
df.head()
```

Out[ ]:

	Median Household Income by State: Selected years 1990 through 2019									
	1990	**2000**	**2005**	**2010**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**
United States	60000	64600	60700	58800	58000	60200	61400	62900	63100	65700
Alabama	47100	52500	48400	47600	46300	48300	49300	50200	50800	51700
Alaska	82700	79400	73800	75900	77400	79200	81400	76300	75700	75500
Arizona	55000	62400	58100	55000	54100	55600	57100	59000	60300	62100
Arkansas	42200	49500	45900	45000	44600	45300	47200	47800	47900	49000

We save the resulting dataframe as both a CSV and a pickle.

In [ ]:
```
df.to_csv("clean-csv/tabn102-30.csv")
df.to_pickle("clean-pickle/tabn102-30.pkl")
```

Households by Total Income: 1990 through 2019

Keeping with the theme, to clean this data we will be using numpy, pandas, and re.

In [ ]:
```
import numpy as np
import pandas as pd
import re
```

Before proceeding with the cleaning process, from the source CSV file, we can observe that the table indicies contain the following identifiers: either the words "RACE" or "IN COMBINATION," or,

generally, a capital letter followed by a space and a number. To identify each index when we read in the data, we will use re to search for items which fit this description by way of a helper-function index_condition:

In [ ]:
```
def index_condition(string):
    index_condition = re.findall('[A-Z] [0-9]', string) or re.findall('RACE', string)
    return index_condition
```

Now we will read in each line of data. As we read in each line, we will be checking to see if it meets the requirements specified by our index_condition function. If so, we will append that item to our "outer_index_to_clean" list after replacing the characters ",,", "\n", and upper-casing the first letter of each word. If an item in the CSV does not meet the index_condition requirements, that content is probably numeric data. we will once again replace those characters and replace any "N" values with "np.nan."

In [ ]:
```
to_clean = []
outer_index_to_clean = []
with open(r"data-capstone/tableA2.csv") as f:
    for lines in f:
        if index_condition(lines):
            outer_index_to_clean.append(lines.replace(',,', '').replace('\n', '').titl
        else:
            to_clean.append(lines.replace(',,', '').replace('\n', '').replace('N', str
```

The resulting list, "to_clean," contains a combination of headers, footnotes, column data, and table values.

In [ ]:
```
# A sample of five row entries
to_clean[30:35]
```

Out[ ]:
```
['2015,"125,819",100,9.8,9.3,9.3,11.7,16.2,12.1,15.7,7.6,8.4,"64,631",604,"90,645",75
8',
 '2014,"124,587",100,10.7,9.7,9.5,12,16.6,11.9,14.7,7.1,7.8,"61,468",739,"86,763",84
0',
 '2013 4,"123,931",100,10.5,9.9,9.2,11.8,16.5,12.4,14.5,7.1,8,"62,425","1,253","87,59
9","1,272"',
 '2013 5,"122,952",100,10.4,10.1,9.5,12.1,16.9,12.9,14.2,6.9,6.9,"60,507",529,"84,62
4",956',
 '2012,"122,459",100,10.4,10.3,9.2,12.9,16.6,12.5,14.5,6.8,6.9,"60,313",406,"84,262",
819']
```

Observe that each table row entry begins with the year that data was recorded followed by a comma, a quotation mark, and finally the values collected for that year. We can now use re to sort our list based on this observation.

In [ ]:
```
data_to_clean = []
year_condition = '[0-9]\,\"'
for lines in to_clean:
```

```
    if re.findall(year_condition, lines):
        data_to_clean.append(lines)
    else:
        pass
```

Now we will extract the inner index of our dataframe and as well, isolate the data values for each entry. We begin by observing that, in this step, we need to accomplish a variety of cleaning tasks: we need to elimate, to the best of our ability, both footnotes and quotation marks which surround data values larger than 10^3; we need to erase all extraneous commas; and we need to isolate each year.

In [ ]:
```
inner_index = [] # This will be where each year lives
clean_data = [] # Our raw data will end up here

for i in range(0, len(data_to_clean)):
    dirty_data = data_to_clean[i]
    clean_string = ''
    for i in range(0, len(dirty_data)):
        try:
            if re.findall('\"', dirty_data[i+4]) and dirty_data[i] == ',': # If the ne
                pass
            elif re.findall(' ', dirty_data[i]): # If you are a blank space: pass
                pass
            elif re.findall(' ', dirty_data[i-1]) and dirty_data[i].isdigit() == True:
                pass
            elif re.findall(' ', dirty_data[i-2]) and dirty_data[i].isdigit() == True:
                pass
            elif dirty_data[i] == '"': # If you're a quotation mark, no thank you: pas
                clean_string = clean_string + ''
            else:
                clean_string = clean_string + dirty_data[i]
        except:
            clean_string = clean_string + dirty_data[i]
    clean_data.append(clean_string.split(',')[1:]) # The raw data is everything in thi
    inner_index.append(clean_string.split(',')[0]) # The year is the first entry
# An example of a "clean" row
clean_data[0]
```

Out[ ]:
```
['131202',
 '100',
 '9.3',
 '8.1',
 '7.8',
 '10.9',
 '16.2',
 '11.9',
 '15.9',
 '8.3',
 '11.6',
 '70784',
 '605',
 '102316',
 '1029"']
```

While this method succeeded in seperating these two groups of data, as evidenced from the print out, the last entry of every list in "clean_data" ends with a quotation mark. However, this issue can be quickly

resolving using pandas later. We now continue with our endeavor by pulling what will be our outer index. From the print-out below we see that we need to exclude in our clean list any footnotes, and any extraneous commas or quotations marks

In [ ]:
```
outer_index_to_clean
```

Out[ ]:
```
['All Races,',
 'White Alone 25,',
 'White 26,',
 '"White Alone, Not Hispanic 25",',
 '"White, Not Hispanic 26",',
 'Black Alone Or In Combination,',
 'Black Alone 27,',
 'Black 26,',
 'Asian Alone Or In Combination,',
 'Asian Alone 28,',
 'Asian And Pacific Islander 26,',
 'American Indian And Alaska Native Alone Or In Combination,',
 'American Indian And Alaska Native Alone 29,',
 'American Indian And Alaska Native 26,',
 'Two Or More Races,',
 'Hispanic (Any Race) 30,']
```

We will once again employ re to sort through our outer index:

In [ ]:
```
outer_index_clean = []
for k in range(0, len(outer_index_to_clean)):
    string_to_clean = outer_index_to_clean[k]
    clean_string = ''
    for letters in string_to_clean:
        if re.findall('[0-9]', letters): # Skip numbers
            pass
        elif letters == '\"': # Skip "
            pass
        elif letters == ',': # Skip ,
            pass
        else:
            clean_string = clean_string + letters
    clean_string = clean_string.strip()
    outer_index_clean.append(clean_string)
outer_index_clean
```

Out[ ]:
```
['All Races',
 'White Alone',
 'White',
 'White Alone Not Hispanic',
 'White Not Hispanic',
 'Black Alone Or In Combination',
 'Black Alone',
 'Black',
 'Asian Alone Or In Combination',
 'Asian Alone',
 'Asian And Pacific Islander',
 'American Indian And Alaska Native Alone Or In Combination',
 'American Indian And Alaska Native Alone',
 'American Indian And Alaska Native',
 'Two Or More Races',
 'Hispanic (Any Race)']
```

Now we are interested in matching the length of our outer index and our inner index. Because we will be using pd.Multindex to assemble our table, order and position matter. By conferring with the source CSV file, we first observe that each breakout group in the outer index begins either at year 2021 or year 2001 and extend backwards in time for an irregular period of time. Grouping for the first five breakout groups we find that the first two begin with the year 2021, followed by 2001 twice, and finally the fifth breakout group begins with the year 2021:

In [ ]:
```
outer_index = []
k, l, start = 0, 0, 0
# ========================================================================
# k will isolate the breakout group,
# l indicates how many times we encounter a target year
# start is the begining of each interval begining with the target year
#=========================================================================

# The first two breakout groups start at 2021
for i in range(start, len(inner_index)):
    try:
        if inner_index[i+1] == '2021': # if the year next in the list is 2021:
            outer_index.append(outer_index_clean[k])
            k += 1 # <-- move on to the next breakout group
        elif inner_index[i+1] == '2001':
            outer_index.append(outer_index_clean[k])

            l += 1
            if l == 2: # The second time in the loop that you encounter the year 2001:
                start = i + 1 # <-- your next interval to start at
                k += 1  # move to the next breakout group
                break
        else:
            outer_index.append(outer_index_clean[k])
    except:
        pass

# The next break out group begins at 2001
for i in range(start, len(inner_index)):
    try:
        if inner_index[i+1] == '2021': # If the next year is 2021:
```

```
            outer_index.append(outer_index_clean[k])
            start = i + 1 # <-- your next interval to start at
            k += 1 # <-- move to the next breakout group
            break
        else:
            outer_index.append(outer_index_clean[k])
    except:
        pass

# The next breakout group begins at 2021
for i in range(start, len(inner_index)):
    try:
        if inner_index[i+1] == '2001': # If the next year is 2001:
            outer_index.append(outer_index_clean[k])
            k += 1 # <-- move to the next breakout group
        elif inner_index[i+1] == '2021': # If the next year is 2021:
            outer_index.append(outer_index_clean[k])
            k += 1 # <-- move to the next breakout group
            start = i + 1 # <-- your next interval to start at
            break
        else:
            outer_index.append(outer_index_clean[k])
    except:
        pass
```

Now, past the first five breakout groups, the pattern regularizies: we have two breakout groups which begin at the year 2021 followed by one breakout group which begins at the year 2001. To model this behavior we will use modular arithmetic. We need to use two variables: $l \in \mathbb{Z}/3\mathbb{Z}$ to model the pattern $(2021, 2021, 2001)$ and $j \in \mathbb{Z}/2\mathbb{Z}$ to toggle between the two years.

In [ ]:
```
l, j = 2, 0
toggle = ['2021', '2001']
year = inner_index[start]
for i in range(start, len(inner_index)):
    try:
        if inner_index[i + 1] == year: # If the next year is the target year:
            outer_index.append(outer_index_clean[k])

            k += 1 # <-- move to the next breakout group
            l = (l + 1) % 3 # <-- increment our dummy variable

            if l == 0 or l == 1: # If 0, 1 = l (mod 3):
                j = (j + 1) % 2
                year = toggle[j] # <-- set the next target year
        else:
            outer_index.append(outer_index_clean[k])

    except:
        outer_index.append(outer_index_clean[k-1])


# Make sure that the lengths match up!
assert len(outer_index) == len(inner_index)
```

Our next task is to create arrays of our columns.

In [ ]:
```
dirty_column_level2 = to_clean[8:23]
dirty_column_level2
```

Out[ ]:
```
['Total,"Under $15,000","$15,000 ',
 'to',
 ' $24,999","$25,000 ',
 'to',
 ' $34,999","$35,000 ',
 'to',
 ' $49,999","$50,000 ',
 'to',
 ' $74,999","$75,000 ',
 'to',
 ' $99,999","$100,000 ',
 'to',
 ' $149,999","$150,000 ',
 'to',
 ' $199,999","$200,000 and over",Estimate,Margin of error1 (Â±),Estimate,Margin of er
ror1 (Â±)']
```

To clean these columns we will first convert this list into a string.

In [ ]:
```
dirty_string = ''
for strings in dirty_column_level2:
    dirty_string = dirty_string + strings
```

Now we will use re to launder our dirty string into a usable array of columns. Before proceeding, one column has been ommitted in the information gathered, "Number (Thousands)." Using our variable "final_wash," we will recover this column.

In [ ]:
```
final_wash = 'Number (Thousands),'
for k in range(0, len(dirty_string)):
    try:
        if re.findall('[0-9],[0-9]', dirty_string[k-1] + dirty_string[k] + dirty_strir
            pass
        elif re.findall('\"|\(|\)|Â|±', dirty_string[k]): # If you're a strange charac
            pass
        elif re.findall('r1', dirty_string[k-1] + dirty_string[k]): # If the string be
            pass
        elif re.findall('1 ', dirty_string[k-1] + dirty_string[k]): # If the string be
            pass
        else:
            final_wash = final_wash + dirty_string[k]
    except:
        pass
column_level2 = final_wash.split(',') # Split the string by a comma
column_level2 = np.array(column_level2) # Convert your list into a numpy array
column_level2
```

Out[ ]:
```
array(['Number (Thousands)', 'Total', 'Under $15000', '$15000 to $24999',
       '$25000 to $34999', '$35000 to $49999', '$50000 to $74999',
       '$75000 to $99999', '$100000 to $149999', '$150000 to $199999',
       '$200000 and over', 'Estimate', 'Margin of error', 'Estimate',
       'Margin of error'], dtype='<U18')
```

Now, we will produce the first level of columns by hard-coding them in. If this dataset were larger, there may be quicker methods. We first observe that in the source CSV we have four columns: Number (Thousands), Percent Distribution, Median Income, and Mean Income.

In [ ]:
```
a0 = 'Number (Thousands),'
a2 = 'Percent Distribution,'
a3 = 'Median Income,'
a4 = 'Mean Income,'
```

We'll use some fun string arithmetic to match the length of the columns in the level beneath it.

In [ ]:
```
column_level1 = (a0 + (a2 * 10) + (a3 * 2) + (a4 * 2)).split(',')[0:-1] # The last ent
column_level1 = np.array(column_level1) # Convert this column into a numpy array
```

Now that we have gathered all of our indicies and columns, we must now put them into a list which we can then use to create our dataframe.

In [ ]:
```
column_arrays = [column_level1, column_level2]
index_arrays = [np.array(outer_index), np.array(inner_index)]
```

We assemble:

In [ ]:
```
df = pd.DataFrame(clean_data, index=index_arrays, columns=column_arrays)
```

Now, our final cleaning step is to erase every quotation mark in our last column and convert each value to a float data type.

In [ ]:
```
df.replace('\"', "", inplace=True, regex=True)
df = df.astype(float)
df
```

Out[ ]:

		Number (Thousands)								
		Number (Thousands)	Total	Under $15000	15000to 24999	25000to 34999	35000to 49999	50000to 74999	75000to 99999	100000to 149999
All Races	2021	131202.0	100.0	9.3	8.1	7.8	10.9	16.2	11.9	15.9
	2020	129244.0	100.0	8.8	8.2	8.1	11.0	16.2	12.3	15.8
	2019	128451.0	100.0	8.5	7.5	8.0	11.3	16.1	12.1	16.3
	2018	128579.0	100.0	9.4	8.3	8.4	11.5	16.4	12.8	15.6
	2017	127669.0	100.0	9.4	8.6	8.6	11.8	15.8	12.6	15.2
...	...	...	...	...	...	...	...	...	...	...
Hispanic (Any Race)	1976	3081.0	100.0	14.7	14.2	13.6	17.2	20.3	11.6	6.2
	1975	2948.0	100.0	14.7	14.4	14.3	17.6	21.4	10.3	5.7
	1974	2897.0	100.0	11.9	14.2	13.5	18.2	22.1	11.3	6.9
	1973	2722.0	100.0	11.1	13.1	13.7	18.4	22.4	12.6	6.9
	1972	2655.0	100.0	10.8	14.7	13.3	19.6	23.7	10.1	6.1

437 rows × 15 columns

Finally, we will save this dataframe as both a pickle and a CSV.

In [ ]:
```
df.to_csv("clean-csv/table-A2.csv")
df.to_pickle("clean-pickle/table-A2.pkl")
```

Educational Attainment of People 25 Years and Over: Selected Years 1940 to 2021

As usual, we import numpy, pandas, and re.

In [ ]:
```
import numpy as np
import pandas as pd
import re
```

To import our data we begin by reading in the CSV. As we read in each line of the CSV we will use re to seperate lines which contain numeric and non numeric data. As well, before each line of data is appended to its list, we will replace the superfluous characters ",," and "\n."

In [ ]:
```
data = []
breakout_groups = []
years = []
outer_index_to_clean = []
```

```
with open(r"data-capstone/taba-2.csv") as f:
    for lines in f:
        if re.findall('[a-z]', lines): # If the element contains a lower-case letter:
            breakout_groups.append(lines.replace(',,', '').replace('\n', '')) # <-- ap
        if re.findall('[1-9],[1-9]', lines): # If the element contains a comma betweer
            data.append(lines.replace(',,', '').replace('\n', '')) # <-- append it to
        if re.findall('[0-9][0-9] [A-Z]', lines): # If the element contains two number
            outer_index_to_clean.append(lines.replace(',,', '').replace('\n', '')) # <
```

Next we will construct our first column level. In examining the source CSV file, we find that each column in the first level repeats three times.

In [ ]:
```
to_clean = breakout_groups[3].split(',') # This is where most of our first column leve
to_clean = to_clean[1:] # The 0th entry is irrelevant

column_level1 = []
for i in range(0, len(to_clean)):
    # Append each header three times
    column_level1.append(to_clean[i])
    column_level1.append(to_clean[i])
    column_level1.append(to_clean[i])

# We'll now retrieve the other column headers for this level
other_bit = breakout_groups[4].split(',')[1:-1]
for i in range(0, len(other_bit)):
    # Append each header three times
    column_level1.append(other_bit[i])
    column_level1.append(other_bit[i])
    column_level1.append(other_bit[i])

column_level1 = np.array(column_level1) # Convert this level into a numpy array
column_level1
```

Out[ ]:
```
array(['All races', 'All races', 'All races', 'White', 'White', 'White',
       'Non-Hispanic White', 'Non-Hispanic White', 'Non-Hispanic White',
       'Black', 'Black', 'Black', 'Asian', 'Asian', 'Asian', '"Hispanic ',
       '"Hispanic ', '"Hispanic ', 'White alone or in combination',
       'White alone or in combination', 'White alone or in combination',
       'Non-Hispanic White alone or in combination',
       'Non-Hispanic White alone or in combination',
       'Non-Hispanic White alone or in combination',
       'Black alone or in combination', 'Black alone or in combination',
       'Black alone or in combination', 'Asian alone or in combination',
       'Asian alone or in combination', 'Asian alone or in combination'],
      dtype='<U42')
```

We'll now assemble the second column level which is nearly intact.

In [ ]:
```
column_level2 = breakout_groups[5].split(',')[1:-1]
column_level2 = np.array(column_level2) # Convert to a numpy array
column_level2
```

Out[ ]:
```
array(['Total', 'Male', 'Female', 'Total', 'Male', 'Female', 'Total',
       'Male', 'Female', 'Total', 'Male', 'Female', 'Total', 'Male',
       'Female', 'Total', 'Male', 'Female', 'Total', 'Male', 'Female',
       'Total', 'Male', 'Female', 'Total', 'Male', 'Female', 'Total',
       'Male', 'Female'], dtype='<U6')
```

Now we will grab the inner index and the raw data for each row in the table. The data was very well preserved and the only cleaning needed to be done was to replace each "(NA)" with np.nan and to each row entry with its corresponding year.

In [ ]:
```
inner_index = []
clean_data = []
for k in range(0, len(data)):
    raw_data = []
    dirty_data = data[k].split(',')
    for i in range(0, len(dirty_data)):
        if re.findall('\(NA\)', dirty_data[i]): # If the element looks like (NA):
            raw_data.append(np.nan) # Replace it with np.nan
        elif dirty_data[i] == '': # If you're a blank element: Pass
            pass
        else:
            raw_data.append(dirty_data[i])
    inner_index.append(raw_data[0]) # The year is the first entry in the raw_data list
    raw_data = raw_data[1:]
    clean_data.append(raw_data)
# The first entry in clean_data
clean_data[0]
```

Out[ ]:
```
['90.9',
 '90.6',
 '91.3',
 '91.3',
 '90.8',
 '91.7',
 '95.1',
 '94.8',
 '95.4',
 '89.4',
 '88.6',
 '90.0',
 '91.6',
 '92.6',
 '90.7',
 '74.3',
 '73.8',
 '74.8',
 '91.2',
 '90.8',
 '91.6',
 '95.1',
 '94.8',
 '95.4',
 '89.5',
 '88.8',
 '90.1',
 '91.7',
 '92.7',
 '90.9']
```

Next we will grab the core elements of the outer index by slicing our outer_index_to_clean list.

In [ ]:
```
to_clean = outer_index_to_clean[1:3]
outer_index_to_extend = []
```

```
for items in to_clean:
    outer_index_to_extend.append(items.replace(',', '').title())
outer_index_to_extend
```

Out[ ]:
```
['25 Years And Older', '25 To 29 Years']
```

Similarly, we slice our breakout_groups list to isolate all of our core middile index items.

In [ ]:
```
to_clean = breakout_groups[6:10]
middle_index_to_extend = []
for items in to_clean:
    middle_index_to_extend.append(items.replace(',', '').title())
middle_index_to_extend
```

Out[ ]:
```
['Completed 4 Years Of High School Or More',
 'Completed 4 Years Of College Or More',
 'Completed 4 Years Of High School Or More',
 'Completed 4 Years Of College Or More']
```

We will now employ modular arithmetic to model the behavior of these indicies. Conferring with the source CSV we see that each element in the outer index cycles through the two unique middle index values and is periodic with respect to the inner index value 2020.

In [ ]:
```
middle_index = []
outer_index = []
i, j = 0, 0
for k in range(0, len(inner_index)):
    try:
        if inner_index[k + 1] == '2020': # If the next year is 2020:
            middle_index.append(middle_index_to_extend[i])
            outer_index.append(outer_index_to_extend[j])
            i += 1 # <-- move to the next element in the middle index list
            if i % 2 == 0: # <-- if you've visited the year 2019 twice:
                j += 1 # Move to the next element in the outer index list
        else:
            middle_index.append(middle_index_to_extend[i])
            outer_index.append(outer_index_to_extend[j])
    except:
        middle_index.append(middle_index_to_extend[i])
        outer_index.append(outer_index_to_extend[j])
```

Since all of our column and index levels are accounted for, we will now put together a list of our column and index arrays.

In [ ]:
```
column_arrays = [column_level1, column_level2]
index_arrays = [np.array(outer_index), np.array(middle_index), np.array(inner_index)]
```

We can now assemble the dataframe and convert all numeric data to float.

In [ ]:
```
df = pd.DataFrame(clean_data, index=index_arrays, columns=column_arrays)
df = df.astype(float)
```

The first five rows of our dataframe:

In [ ]:
```
df.head()
```

Out[ ]:

			All races			White			Non-Hispanic White		Black	...
			Total	Male	Female	Total	Male	Female	Total	Male	Female	Total	...
25 Years And Older	Completed 4 Years Of High School Or More	2020	90.9	90.6	91.3	91.3	90.8	91.7	95.1	94.8	95.4	89.4	...
		2019	90.1	89.6	90.5	90.5	89.9	91.0	94.6	94.2	95.0	87.9	...
		2018	89.8	89.4	90.2	90.2	89.6	90.8	94.3	93.9	94.7	87.9	...
		2017	89.6	89.1	90.0	90.1	89.5	90.6	94.1	93.7	94.5	87.3	...
		2016	89.1	88.5	89.6	89.5	88.8	90.1	93.8	93.4	94.3	87.1	...

5 rows × 30 columns

Save our dataframe to CSV and pickle it as well.

In [ ]:
```
df.to_csv('clean-csv/taba-2.csv')
df.to_pickle('clean-pickle/taba-2.pkl')
```

Mean Earnings of Workers 18 Years and Over: 1975 to 2020

We begin, as usual, with loading the required packages. In this case we will be using numpy, pandas, and re.

In [ ]:
```
import numpy as np
import pandas as pd
import re
```

Next we will reading the data which was downloaded in as a CSV. Simultaneously, as we read in the lines from the CSV we will be filtering out lines which contain numeric and text data into two sperate lists to clean. As well, before each element is added to the list, we will be replacing the characters ',,' and '\n' which are artifacts from the CSV irrelevant to our project.

In [ ]:
```
data = []
breakout_groups = []
years = []
to_match = '[a-z]'

with open(r"data-capstone/taba-3.csv") as f:
    for lines in f:
        if re.findall(to_match, lines):
            breakout_groups.append(lines.replace(',,', '').replace('\n', ''))
```

```
        if re.findall('[1-9],[1-9]', lines):
            data.append(lines.replace(',,', '').replace('\n', ''))

breakout_groups = breakout_groups[0:45] # Elements past index 45 contain footnote info
```

Beacuase of the multi-level style of CSV imported, to keep that structure intact, we will be using a multi-index within pandas. As such, it will be necessary to capture the appropriate length of the number of columns in are resulting data frame. In this case, we have eighteen columns and six different breakout groups so we will append each item three times.

In [ ]:
```
to_split = breakout_groups[3].split(',')
column_level1 = []
for i in range(3, len(to_split)):
    column_level1.append(to_split[i])
    column_level1.append(to_split[i])
    column_level1.append(to_split[i])
column_level1
```

Out[ ]:
```
['Total',
 'Total',
 'Total',
 'Not a high school graduate',
 'Not a high school graduate',
 'Not a high school graduate',
 'High school graduate',
 'High school graduate',
 'High school graduate',
 "Some college/associate's degree",
 "Some college/associate's degree",
 "Some college/associate's degree",
 "Bachelor's degree",
 "Bachelor's degree",
 "Bachelor's degree",
 'Advanced degree',
 'Advanced degree',
 'Advanced degree']
```

Now we will construct the second column level in a similar fashion.

In [ ]:
```
column_level2 = []
to_split = breakout_groups[4].split(',')
to_split.remove('')
for items in to_split:
    column_level2.append(items)
column_level2
```

Out[ ]:
```
['Mean',
 'Number with earnings',
 'Standard error',
 'Mean',
 'Number with earnings',
 'Standard error',
 'Mean',
 'Number with earnings',
 'Standard error',
 'Mean',
 'Number with earnings',
 'Standard error',
 'Mean',
 'Number with earnings',
 'Standard error',
 'Mean',
 'Number with earnings',
 'Standard error']
```

Now that we have pulled apart each column level, we can begin cleaning the data and isolating each index. In this dataset we will have three indicies: race, gender, and year.

In [ ]:
```
# We need to pull the years out of the unclean data
years = [i for i in range(1975, 2020)]
condition = ''
for i in years:
    condition = condition + str(i) + '|'


raw_data = []
inner_index = [] # This will contain the year
for i in range(0, len(data)):
    to_clean = []
    year = []
    to_filter = re.findall('"[^"]*"|,[0-9][0-9][0-9]', data[i]) # We want to grab ever
    year.append(re.findall(condition, data[i])[0])
    for items in to_filter:
        to_clean.append(items.replace(',', '').replace('\"', '')) # Now we will replac
    raw_data.append(to_clean) # Append the clean data
    inner_index.append(year[0]) # Append the year to the inner index list
```

Now that we have successfully filtered out the inner index and raw data, we will turn our attention to the middle and outer indicies. For each item in our initial breakout groups list, we can find our outer index at list indicies, i, where

$$i \in 1 + 4\mathbb{Z}.$$

As well we find that our middle indicies are located at list index, j, where

$$j \in \{\mathbb{Z}/4\mathbb{Z} - \{1 + 4\mathbb{Z}\}\}.$$

We also observe that for each index in the source CSV file that, although each breakout group has a variable number of years included in the dataset, the year data is periodic with respect to the year 2019.

This means that if we know that the next item in our inner index list is 2019 our break out group needs to change. Keeping in mind that becuase we will be using a multi-index for the resulting pandas dataframe, meaning that length and position matter, in light of both of these observations, we can now construct a for loop which will append items to a middle index and outer index list corresponding to their breakout group index value modulo four which will shift values when the next year added is 2019.

In [ ]:
```
middle_index = []
outer_index = []

i, j, l = 1, 1, 0
for k in range(0, len(inner_index)):
    try:
        if inner_index[k + 1] != '2019':
            middle_index.append(breakout_groups[(4 * i) + 1 + j].replace(',', '')) # (
            outer_index.append(breakout_groups[(4 * i) + 1]) # Our outer indicies appe
        else:
            middle_index.append(breakout_groups[(4 * i) + 1 + j].replace(',', ''))
            outer_index.append(breakout_groups[(4 * i) + 1])
            l = (1 + l) % 3 # We have three middle indicies to grab: Both Sexes, Male,
            j = (1 + j) % 4 # Each middle index appears three times per outer index
            if l == 0: # When we've captured all three middle indicies we increment ou
                i += 1
            if j == 0: # if j == 0 then we would accidentally append our outer index t
                j += 1

    except:
        middle_index.append(breakout_groups[(4 * i) + 1 + j].replace(',', ''))
        outer_index.append(breakout_groups[(4 * i) + 1])
        pass

assert len(inner_index) == len(middle_index) ==len(outer_index)
```

Now that we have captured all of the necessary data, to make use of pandas multi-index we will now create a list of column and index arrays.

In [ ]:
```
column_arrays = [np.array(column_level1), np.array(column_level2)]
index_arrays = [np.array(outer_index), np.array(middle_index), np.array(inner_index)]
```

We are now ready to compile the dataframe. After doing so, our final cleaning step is to convert our numeric data to float.

In [ ]:
```
df = pd.DataFrame(raw_data, index=index_arrays, columns=column_arrays)
df = df.astype(float)
```

The first five rows of our dataframe:

In [ ]:
```
df.head()
```

Out[ ]:

			Total			Not a high school graduate			High school grad		
			Mean	Number with earnings	Standard error	Mean	Number with earnings	Standard error	Mean	Number with earnings	Star
Total	Both Sexes	2019	58544.0	167215.0	329.0	29278.0	11413.0	552.0	39371.0	42598.0	
		2018	55619.0	165179.0	296.0	27037.0	12058.0	481.0	38936.0	42882.0	
		2017	53536.0	163871.0	218.0	26832.0	12240.0	383.0	38145.0	42816.0	
		2016	51893.0	162218.0	217.0	27800.0	12281.0	559.0	36702.0	42897.0	
		2015	49994.0	161074.0	209.0	25315.0	13159.0	422.0	35615.0	42404.0	

Now we will pickle our dataset and save it as a CSV

In [ ]:
```
df.to_csv('clean-csv/taba-3.csv')
df.to_pickle('clean-pickle/taba-3.pkl')
```

Recovering Dataframes

To recover any one of these dataframes, I would recommend either un pickling the dataframe or using optional arguments in the pd.read_csv function to preserve the intended structure. For example, to recover the "Mean Earnings of Workers 18 Years and Over: 1975 to 2020" dataframe, one could use pd.read_pickle to do so:

In [ ]:
```
df = pd.read_pickle("clean-pickle/taba-3.pkl")
df.head()
```

Out[ ]:

			Total			Not a high school graduate			High school grad		
			Mean	Number with earnings	Standard error	Mean	Number with earnings	Standard error	Mean	Number with earnings	Star
Total	Both Sexes	2019	58544.0	167215.0	329.0	29278.0	11413.0	552.0	39371.0	42598.0	
		2018	55619.0	165179.0	296.0	27037.0	12058.0	481.0	38936.0	42882.0	
		2017	53536.0	163871.0	218.0	26832.0	12240.0	383.0	38145.0	42816.0	
		2016	51893.0	162218.0	217.0	27800.0	12281.0	559.0	36702.0	42897.0	
		2015	49994.0	161074.0	209.0	25315.0	13159.0	422.0	35615.0	42404.0	

Equivalently, one could use pd.read_csv function to do so as well by specifying the header and index columns:

In [ ]:
```
df = pd.read_csv("clean-csv/taba-3.csv", index_col=[0, 1, 2], header=[0, 1])
df.head()
```

Out[ ]:

			Total			Not a high school graduate			High school grad		
			Mean	Number with earnings	Standard error	Mean	Number with earnings	Standard error	Mean	Number with earnings	Star
Total	Both Sexes	2019	58544.0	167215.0	329.0	29278.0	11413.0	552.0	39371.0	42598.0	
		2018	55619.0	165179.0	296.0	27037.0	12058.0	481.0	38936.0	42882.0	
		2017	53536.0	163871.0	218.0	26832.0	12240.0	383.0	38145.0	42816.0	
		2016	51893.0	162218.0	217.0	27800.0	12281.0	559.0	36702.0	42897.0	
		2015	49994.0	161074.0	209.0	25315.0	13159.0	422.0	35615.0	42404.0	