

## Research report on Data Transformation with Python using Pandas:

The process of transforming, cleaning, and organizing data into a format that can be used for analysis and support decision-making processes to advance an organization's expansion is known as data transformation <sup>(1)</sup>. The process of data transformation can be constructive, destructive, aesthetic, or structural <sup>(1)</sup>. The data transformation procedure cleans up duplicate data, changes data types, removes unnecessary columns, deals with null values and outliers, renames the columns to easily understandable names and enhances the dataset to transform raw data into a format that can be used.

**Introduction:** The Demographics dataset chosen for the data transformation is from the Kaggle website <sup>(2)</sup> that indicates the population characteristics related to each state in the USA like county, strata determining factors, counties per strata, population size, population density, poverty, different age group population, different races population.

Demographic features like racial and population characteristics contribute to the enrichment and profiling of county-level observations. The steps involved in cleaning a dataset are as follows:

### Step 1: Data loading

- The demographics dataset is downloaded on local computer as a csv (comma-separated value) file and then loaded into the google colab platform by using the pd.read\_csv() function and giving the dataset URL as a parameter. Then, to perform the Data Transformation on the loaded dataset pandas, NumPy, Matplotlib, and Seaborn libraries are imported in the code. The code is as follows:

The screenshot shows a Google Colab interface with two tabs: 'Data Transformation.ipynb' and 'Copy of Untitled2.ipynb'. The 'Data Transformation.ipynb' tab is active. It displays the following code:

```
[365]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

[366]: demographics_df=pd.read_csv("https://raw.githubusercontent.com/vpallapolu/files/main/DEMOGRAPHICS.csv")

[367]: demographics_df.columns
```

The code imports necessary libraries (pandas, numpy, matplotlib, seaborn) and reads the 'DEMOGRAPHICS.csv' file into a DataFrame named 'demographics\_df'. Finally, it prints the column names of the DataFrame.

- The above picture also contains a code to view all the columns from the dataset by using columns attribute.

### Step 2: Dropping unnecessary columns

- The columns which are not useful are dropped by using drop function and specifying the not useful columns in a string format and mentioning the axis which is 1 for columns in this case.

```
[369] demographics_df=demographics_df.drop(['index','CHSI_State_Abbr','Min_Population_Size','Max_Population_Size','Min_Population_Density','Max_Population_Density','Min_Poverty','Max_Poverty'])
[370] demographics_df.info()
```

- The output for the above code shows the column names after removing unnecessary columns along with their datatypes which is as follows:

#	Column	Non-Null Count	Dtype
0	State_FIPS_Code	3141 non-null	int64
1	County_FIPS_Code	3141 non-null	int64
2	CHSI_County_Name	3141 non-null	object
3	CHSI_State_Name	3141 non-null	object
4	Strata_ID_Number	3141 non-null	int64
5	Strata_Determining_Factors	3141 non-null	object
6	Number_Counties	3141 non-null	int64
7	Population_Size	3141 non-null	int64
8	Population_Density	3141 non-null	int64
9	Poverty	3141 non-null	float64
10	Age_19_Under	3141 non-null	float64
11	Age_19_64	3141 non-null	float64
12	Age_65_84	3141 non-null	float64
13	Age_85_and_Over	3141 non-null	float64
14	White	3141 non-null	float64
15	Black	3141 non-null	float64
16	Native_American	3141 non-null	float64
17	Asian	3141 non-null	float64
18	Hispanic	3141 non-null	float64
	dtypes:	float64(10), int64(6), object(3)	
	memory usage:	466.4+ KB	

- This output shows the column names, datatype of all the columns and their non-null count after the removal of columns namely index, state abbreviation, min population size, max population size, min population density, max population density, min poverty, max poverty, min age 19 under, max age 19 under, min age 19-64, max age 19-64, min age 65-84, max age 65-84, min age 85 and over, max age 85 and over, min white, max white, min black, max black, min native American, max native American, min Asian, max Asian, min Hispanic, max Hispanic.

### Step 3: Renaming the columns

- The column names are renamed as a part of data transformation <sup>(5)</sup> to useful names by using the below function which uses rename () function and mentioning the column names along with the new names inside the dictionary as strings in key value pairs. The names changed for the columns are poverty-> poverty\_rate, CHSI\_State\_Name-> state, CHSI\_County\_Name->County, State\_FIPS\_Code->State\_FIPS, County\_FIPS\_Code->County\_FIPS, Number\_Counties-> CountiesPerStratum.

The screenshot shows a Jupyter Notebook interface with two tabs: "Data Transformation.ipynb" and "Copy of Untitled2.ipynb". The "Data Transformation.ipynb" tab is active. In the code editor, the following code is written:

```
demographics_df=demographics_df.rename(columns={"CHSI_State_Name": "State", "CHSI_County_Name": "county", "state_FIPS_Code": "State_FIPS", "County_FIPS_Code": "County_FIPS", "Poverty": "Poverty_rate"})
demographics_df
```

Below the code, a table preview of the "demographics\_df" DataFrame is shown. The table has 3141 rows and 19 columns. The columns are: State\_FIPS, County\_FIPS, county, State, Strata\_ID\_Number, Strata\_Determining\_Factors, CountiesPerStratum, Population\_Size, Population\_Density, Poverty\_rate, Age\_19\_Under, ... (truncated). The data includes various US states and their corresponding demographic statistics.

## Step 4: Checking and removing null values (if any)

- If the dataset contains any null value in any column those must be removed as a part of the data transformation. The loaded data frame is checked for null values by using isna() and notna() functions, represented in the below figures. The loaded dataset does not contain null value if isna function shows false and if notna function shows true. This dataset satisfies both the condition because it does not contain any null or missing value.

The screenshot shows a Jupyter Notebook interface with two tabs: "Data Transformation.ipynb" and "Copy of Untitled2.ipynb". The "Data Transformation.ipynb" tab is active. In the code editor, the following code is written:

```
demographics_df.isnull()
```

Below the code, a table preview of the "demographics\_df" DataFrame is shown. The table has 3141 rows and 19 columns. All values in every cell are displayed as "False", indicating that there are no null or missing values in the dataset.

The screenshot shows a Jupyter Notebook interface with two tabs: "Data Transformation.ipynb" and "Copy of Untitled2.ipynb". The "Data Transformation.ipynb" tab is active. In the code editor, the following code is written:

```
demographics_df.notna()
```

Below the code, a table preview of the "demographics\_df" DataFrame is shown. The table has 3141 rows and 19 columns. All values in every cell are displayed as "True", indicating that all values are non-null and valid.

## Step 5: Changing datatypes

- The data type of the columns is changed as a part of the data transformation. In this dataset, Counties per stratum, population size, population density columns are down casted from int64 to int32, poverty rate, age 19 under, age 19-64, age 65-84, age 85 and over, White, Black, Native American, Asian, Hispanic columns are also down casted from float 64 to float 32. There are many data type conversions like int to str, str to int, int to float, int to datetime and many more by using two different functions astype () and pd.to\_numeric() functions <sup>(3)</sup>. Operations between int32 and floats can sometimes upcast the int32s to int64 unnecessarily. Explicitly converting to int32 prevents this.

A screenshot of a Jupyter Notebook interface titled "Data Transformation.ipynb". The code cell contains the following Python code:

```
demographics_df["CountiesPerStratum"] = demographics_df["CountiesPerStratum"].astype("int32")
demographics_df["Population_Size"] = demographics_df["Population Size"].astype("int32")
demographics_df["Population_Density"] = demographics_df["Population Density"].astype("int32")
demographics_df["Poverty_Rate"] = demographics_df["Poverty_rate"].astype("float32")
demographics_df["Age_19_Under"] = demographics_df["Age_19_Under"].astype("float32")
demographics_df["Age_19_64"] = demographics_df["Age_19_64"].astype("float32")
demographics_df["Age_65_84"] = demographics_df["Age_65_84"].astype("float32")
demographics_df["Age_85_and_Over"] = demographics_df["Age_85_and_Over"].astype("float32")
demographics_df["White"] = demographics_df["white"].astype("float32")
demographics_df["Black"] = demographics_df["black"].astype("float32")
demographics_df["Native_American"] = demographics_df["Native_American"].astype("float32")
demographics_df["Asian"] = demographics_df["Asian"].astype("float32")
demographics_df["Hispanic"] = demographics_df["Hispanic"].astype("float32")
demographics_df.info()
```

- The output of the above code shows the changed data types of the columns which can be viewed by using the info () function as follows:

A screenshot of a Jupyter Notebook interface titled "Data Transformation.ipynb". The code cell contains the following Python code:

```
c = pd.core.frame.DataFrame
RangeIndex: 3341 entries, 0 to 3340
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   State_FIPS      3341 non-null    int64  
 1   County_FIPS     3341 non-null    int64  
 2   County          3341 non-null    object  
 3   State           3341 non-null    object  
 4   Strata_ID_Number 3341 non-null    int64  
 5   Strata_Determining_Factors 3341 non-null    object  
 6   CountiesPerStratum 3341 non-null    int32  
 7   Population_Size  3341 non-null    int32  
 8   Population_Density 3341 non-null    int32  
 9   Poverty_Rate    3341 non-null    float32 
 10  Age_19_Under    3341 non-null    float32 
 11  Age_19_64       3341 non-null    float32 
 12  Age_65_84       3341 non-null    float32 
 13  Age_85_and_Over 3341 non-null    float32 
 14  White           3341 non-null    float32 
 15  Black           3341 non-null    float32 
 16  Native_American 3341 non-null    float32 
 17  Asian            3341 non-null    float32 
 18  Hispanic         3341 non-null    float32 
dtypes: float32(10), int32(3), int64(3), object(3)
memory usage: 306.9+ KB
```

- Before changing the datatypes, the memory is 466kb. After changing down casting the memory dropped to 306kb. In such cases, where the memory problems arise the down casting is a good option.

## Step 6: Removing duplicates

- Prediction results may be skewed by duplicate data. To produce a more accurate prediction, it is crucial to look for and remove any duplicate rows from columns that should only contain unique values <sup>(4)</sup>.

- The data frame is checked for any duplicate values by using `duplicated()` function. If there are any duplicate values, it returns false. This loaded dataset has no duplicate values so, there is no necessity to use functions to remove the duplicates.

```
[ ] demographics_df.duplicated()
[0]    False
[1]    False
[2]    False
[3]    False
[4]    False
[...]
[3136] False
[3137] False
[3138] False
[3139] False
[3140] False
Length: 3141, dtype: bool
```

- If there are any duplicates in the data frame, they can be removed by using the `drop_duplicates()` function.

```
[ ] demographics_df.drop_duplicates()
[0]   State_FIPS County_FIPS County State Strata_ID_Number Strata_Determining_Factors CountiesPerStratum Population_Size Population_Density Poverty_rate Age_19_Under Ag
[1]       1          1     Autauga  Alabama        29 frontier status, population size, poverty, age...           37      48612         82  10.400000  26.900000  6:
[2]       1          3     Baldwin  Alabama        16 frontier status, population size, poverty, age...           27     162586        102  10.200000  23.500000  6:
[3]       1          5    Barbour  Alabama        51 frontier status, population size, poverty, age...           33      28414         32  22.100000  24.299999  6:
[4]       1          7     Bibb  Alabama        42 frontier status, population size, poverty, age...           53      21516         36  16.799999  24.600000  6:
[...]
[3136] 56          37  Sweetwater  Wyoming        77 frontier status, population size, poverty, age...           15      37975          4  8.600000  26.600000  6:
[3137] 56          39     Teton  Wyoming        78 frontier status, population size, poverty, age...           43     19032          5  5.600000  18.799999  7:
[3138] 56          41     Uinta  Wyoming        38 frontier status, population size, poverty, age...           35     19939         10  10.600000  29.100000  6:
[3139] 56          43  Washakie  Wyoming        82 frontier status, population size, poverty, age...           37      7933          4  11.100000  23.500000  5:
[3140] 56          45     Weston  Wyoming        78 frontier status, population size, poverty, age...           43      6671          3  9.900000  20.100000  6:
```

## Step 7: Removing whitespaces

- The whitespaces are a form of redundancies in the dataset to be removed in the data cleaning process. If a dataset contains whitespaces or any other unnecessary punctuation marks that causes the dataset to look confusing so, in order to remove redundant data, the below function `str.replace()`<sup>(6)</sup> is used to replace them. In this data frame to remove the whitespace in the county name column this function is used.

```
[ ] demographics_df['county']=demographics_df['County'].str.replace(' ','')
[0]   State_FIPS County_FIPS County State Strata_ID_Number Strata_Determining_Factors CountiesPerStratum Population_Size Population_Density Poverty_rate Age_19_Under Ag
[1]       1          1     Autauga  Alabama        29 frontier status, population size, poverty, age...           37      48612         82  10.400000  26.900000  6:
[2]       1          3     Baldwin  Alabama        16 frontier status, population size, poverty, age...           27     162586        102  10.200000  23.500000  6:
[3]       1          5    Barbour  Alabama        51 frontier status, population size, poverty, age...           33      28414         32  22.100000  24.299999  6:
[4]       1          7     Bibb  Alabama        42 frontier status, population size, poverty, age...           53      21516         36  16.799999  24.600000  6:
[...]
[3136] 56          37  Sweetwater  Wyoming        77 frontier status, population size, poverty, age...           15      37975          4  8.600000  26.600000  6:
[3137] 56          39     Teton  Wyoming        78 frontier status, population size, poverty, age...           43     19032          5  5.600000  18.799999  7:
[3138] 56          41     Uinta  Wyoming        38 frontier status, population size, poverty, age...           35     19939         10  10.600000  29.100000  6:
[3139] 56          43  Washakie  Wyoming        82 frontier status, population size, poverty, age...           37      7933          4  11.100000  23.500000  5:
[3140] 56          45     Weston  Wyoming        78 frontier status, population size, poverty, age...           43      6671          3  9.900000  20.100000  6:
```

0 completed at 4:59PM

## Step 8: Merging two columns into one field

- By using the data cleaning process, two columns which are of the same datatype can be combined into a single column <sup>(7)</sup>. Then, the previously combined columns must be removed to avoid redundancies. They can be removed by the already used drop() function and mentioning axis=1 (for columns). One must ensure that the two/ multiple columns which are to be combined should be of the same data type. Otherwise, errors are created. In the code below age 65-84 and age 85 and over columns are joined together to form new column: age 65 and over.

The screenshot shows a Jupyter Notebook interface with two tabs: 'Data Transformation.ipynb' and 'Copy of Untitled2.ipynb'. The code cell contains the following Python code:

```
demographics_df['Age_65_and_over']=demographics_df['Age_65_84']+demographics_df['Age_85_and_over']
demographics_df=demographics_df.drop(['Age_65_84','Age_85_and_over'],axis=1)
demographics_df
```

The output of the code is a table with 3138 rows and 13 columns. The columns are: State\_FIPS, County\_FIPS, County, State, Strata\_ID\_Number, Strata\_Determining\_Factors, CountiesPerStratum, Population\_Size, Population\_Density, Poverty\_rate, Age\_19\_Under\_Age, Age\_65\_and\_over, and Age\_85\_and\_over. The 'Age\_65\_and\_over' column is the result of the merge, and the original 'Age\_65\_84' and 'Age\_85\_and\_over' columns have been removed.

## Step 9: Changing the positions of columns

- The position of columns should be changed after the above step to avoid confusion <sup>(8)</sup>. In the above step, the created new column is by default at the end of the data frame. That should be placed after age group columns so, the position is reindexed by using new\_columns variable and specifying the order of the columns. In this step, one can arrange the columns as needed.

The screenshot shows a Jupyter Notebook interface with two tabs: 'Data Transformation.ipynb' and 'Copy of Untitled2.ipynb'. The code cell contains the following Python code:

```
new_columns=['State','State_FIPS','County','County_FIPS','Strata_ID_Number','Strata_Determining_Factors','CountiesPerStratum','Population_Size','Population_Density','Poverty_rate','Age_19_Under_Age','Age_65_and_over','Age_85_and_over']
demographics_df=demographics_df[new_columns]
demographics_df
```

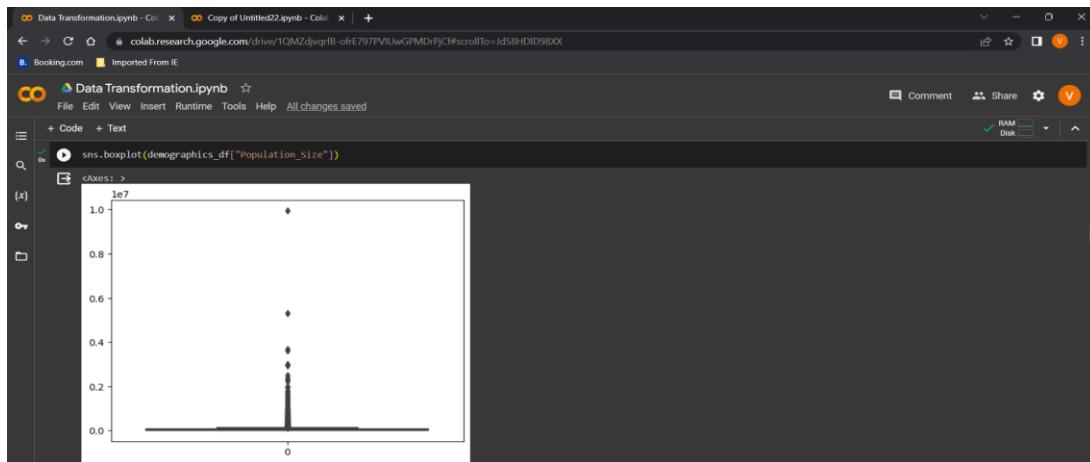
The output of the code is a table with 3138 rows and 13 columns. The columns are: State, State\_FIPS, County, County\_FIPS, Strata\_ID\_Number, Strata\_Determining\_Factors, CountiesPerStratum, Population\_Size, Population\_Density, Poverty\_rate, Age\_19\_Under\_Age, Age\_65\_and\_over, and Age\_85\_and\_over. The 'Age\_65\_and\_over' column has been moved to the second position from the last.

## Step 10: Identifying and Handling outliers

- Outliers are important data points which can sometimes be useful and sometimes useless. They can be capped, that is, replaced with minimum and maximum values. Whether the outliers are to be trimmed or capped depends upon the size of the dataset. Here, the decision is made to remove the outliers. Outliers can be removed by numerous methods and among them, z-score method using data visualization is used for the loaded dataset. It is a statistical measurement that helps us understand how close or far we are from the data set's arithmetic mean <sup>(9)</sup>. Methods like interquartile range, percentile method, etc. Can also be used.
- The describe() function is used to display the mean, standard deviation, minimum, maximum, 25%, 50%, 75% values for each column.

	State_FIPS	County_FIPS	Strata_ID_Number	CountiesPerStratum	Population_Size	Population_Density	Poverty_rate	Age_19_Under	Age_19_64	Age_65_and_over	White	Black
count	3141.000000	3141.000000	3141.000000	3.141000e+03	3141.000000	3141.000000	3141.000000	3141.000000	3141.000000	3141.000000	3141.000000	3141.000000
mean	30.304680	103.716651	44.696275	38.486151	9.436816e+04	249.119389	12.638428	24.806526	60.289398	14.904840	87.017891	8.98
std	15.134423	107.999484	25.118434	10.290195	3.064317e+05	1703.041884	40.186459	3.281777	3.356056	4.110967	16.150478	14.54
min	1.000000	1.000000	1.000000	15.000000	6.200000e+01	-2222.19951	1.400000	47.599998	2.200000	4.700000	0.00	
25%	18.000000	35.000000	23.000000	32.000000	1.121100e+04	17.000000	9.800000	22.700001	58.299999	12.200001	82.800003	0.50
50%	29.000000	79.000000	44.000000	37.000000	2.523500e+04	44.000000	12.600000	24.600000	60.299999	14.500000	94.099998	2.10
75%	45.000000	133.000000	66.000000	45.000000	6.404000e+04	109.000000	16.200001	26.400000	62.299999	17.200001	97.599998	10.30
max	56.000000	840.000000	88.000000	62.000000	9.935475e+06	69390.000000	36.200001	47.200001	83.300003	34.700001	100.000000	86.00

- The outliers are identified for each separate column by using z-score and visualization methods in the following way:
- o **Identifying and removing outliers from the 'Population size' column:**
  - Box plots are the best way of representing the outliers lying in a dataset, that is the reason box plots are chosen to represent the outliers out of so many visualization methods.
  - A box plot is plotted for the population size column using seaborn library. This plot indicates the 4 data points outside the whiskers as outliers.



- In the z-score method, “any value that is greater than **+3** and less than **-3** standard deviation is an outlier value” principle is followed<sup>(9)</sup>. To detect outliers in pandas using Z-score, set a limit for any values outside of **+3/-3** standard deviations from the mean to be considered outliers.
- The upper limit is the addition of mean and 3 times the standard deviation of the specific column and the lower limit is the subtraction of 3 times the standard deviation of the column from the mean<sup>(10)</sup>. The values falling outside the upper and lower limit range are considered as outliers. Below is another function loc to find the outliers where the condition satisfies the population size column greater than upper limit or the population size column less than the lower limit to get the outliers from that column.

```

Data Transformation.ipynb - Colab
Copy of Untitled22.ipynb - Colab + 

File Edit View Insert Runtime Tools Help All changes saved
Comment Share RAM Disk

+ Code + Text
[115]: upperlimit=demographics_df['Population_Size'].mean() + 3*demographics_df['Population_Size'].std()
lowerlimit=demographics_df['Population_Size'].mean() - 3*demographics_df['Population_Size'].std()
print('upperlimit:',upperlimit)
print('lowerlimit:',lowerlimit)
upperlimit: 1013663.131568267
lowerlimit: 824926.8030104828

[116]: #finding outliers
demographics_df.loc[(demographics_df['Population_Size']>upperlimit)|(demographics_df['Population_Size']<lowerlimit)]

```

State	State_FIPS	County	County_FIPS	Strata_ID_Number	Strata_Determining_Factors	CountiesPerStratum	Population_Size	Population_Density	Poverty_rate	Age_19
101	Arizona	4	Mariopa	13	1 frontier status, population size	34	3635528	395	12.800000	28.0
184	California	6	Alameda	1	1 frontier status, population size	34	1448905	1964	10.700000	26.2
190	California	6	ContraCosta	13	2 frontier status, population size, poverty	32	1017787	1414	7.500000	27.0
202	California	6	LosAngeles	37	1 frontier status, population size	34	9935475	2447	17.700001	28.3
213	California	6	Orange	59	1 frontier status, population size	34	2988072	3785	10.600000	28.0
216	California	6	Riverside	65	1 frontier status, population size	34	1946419	270	12.300000	29.1
217	California	6	Sacramento	67	1 frontier status, population size	34	1933482	1412	13.700000	28.5
219	California	6	SanBernardino	71	1 frontier status, population size	34	1963535	98	16.000000	32.2
220	California	6	SanDiego	73	1 frontier status, population size	34	2933462	698	11.600000	27.5
226	California	6	SantaClara	85	1 frontier status, population size	34	1699052	1316	8.800000	26.4

- The output for the above code gives the outliers in the specific rows for that column. Below is another code to represent the number of outliers in the population size column by modifying the above used function that is replacing the or condition with and condition and assigning it to a new variable new demographics. This represents the number of rows before removing the outliers and the number of rows after removing the outliers for this column.

```

Data Transformation.ipynb - Colab
Copy of Untitled22.ipynb - Colab + 

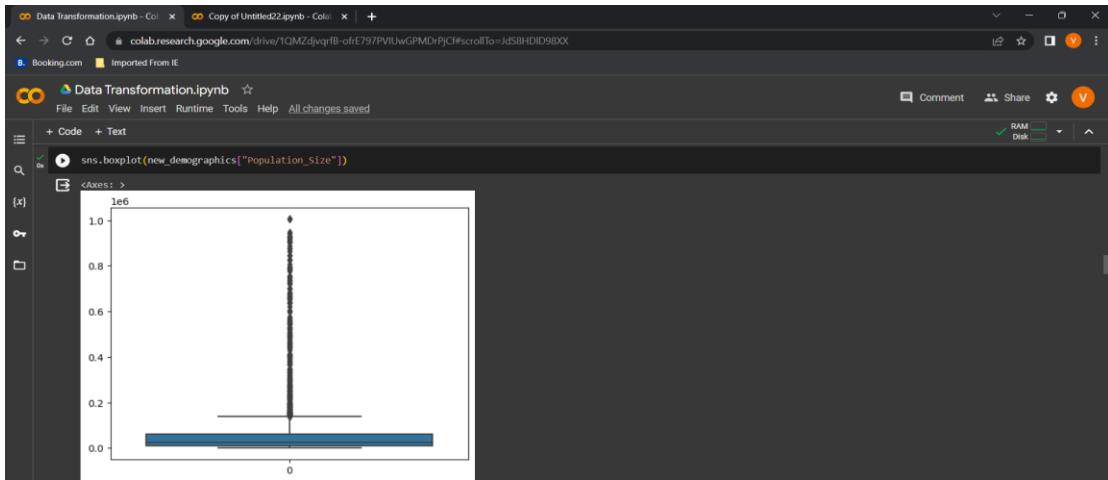
File Edit View Insert Runtime Tools Help All changes saved
Comment Share RAM Disk

+ Code + Text
[117]: #trimming the outliers
new_demographics=demographics_df.loc[(demographics_df['Population_Size']>upperlimit)&(demographics_df['Population_Size']<lowerlimit)]
len(new_demographics)
print('before removing outliers:',len(demographics_df))
print('after removing outliers:',len(new_demographics))
print('outliers:',len(demographics_df)-len(new_demographics))

before removing outliers: 3141
after removing outliers: 3105
outliers: 36

```

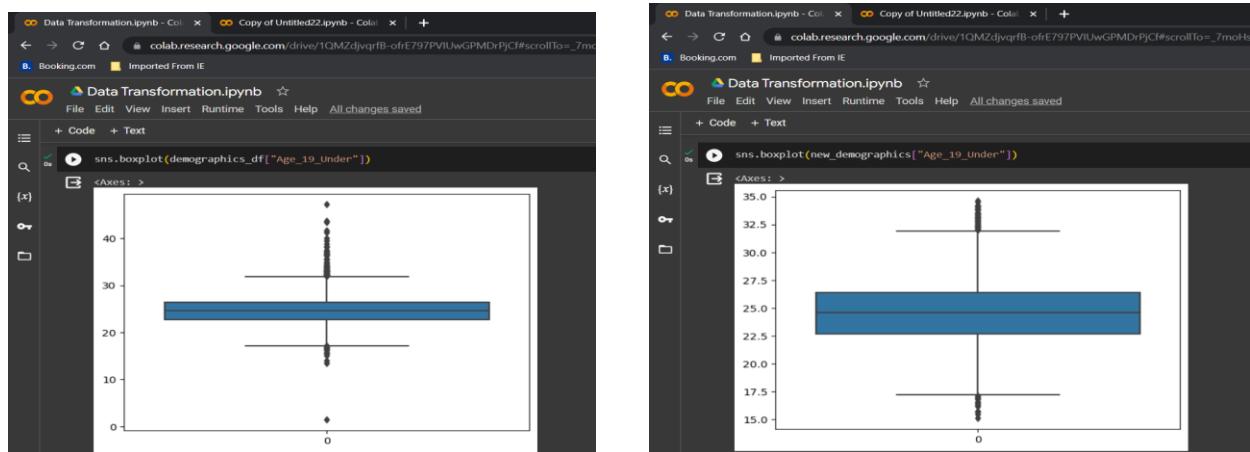
- Once again, the bar plot is plotted to check whether any data points are lying outside the whiskers. Comparing both the bar plots, that is the loaded data frame and the new data frame, one can tell that the outliers have been removed.



- The same procedure is followed for all the useful numeric columns to remove the outliers from the data frame by using the same z-score method which is mentioned in the appendix below.

### Comparing the boxplots before and after removal of outliers:

- Box plots for ‘Age 19 under’ column before and after removal of outliers



- Before removing outliers, the plot indicated the data points outside the whiskers which are potential outliers and after removing them, the plot is clear with no data points outside the whiskers and has a median value at 24.5, first quartile at 23 and third quartile at 26.
- The same procedure is followed for all the columns and the plots are compared to verify that the outliers are completely removed.

After removing all the outliers, by following the above-mentioned path to identify and remove outliers for all the columns the cleaned dataset without outliers is produced. The new data frame produced contains 3045 rows which when compared with the loaded data frame are free of outliers.

The describe () function is used once again to check the variations in the mean, standard deviation, minimum, maximum, 25%, 50%, 75% values for the new data frame from that of the previously loaded data frame.

**Repository link to access the cleaned dataset:** the GitHub repository publicly accessible link to the new cleaned dataset (after applying all the above-mentioned data cleaning tools) is mentioned below in the references<sup>(11)</sup>.

## Conclusion:

The demographics dataset was cleaned and made ready for analysis through extensive data transformation techniques. The steps included loading the data into a Pandas data frame, examining the columns and data types, removing redundant columns, renaming columns for clarity, checking for and handling missing values, down casting data types to reduce memory usage, merging relevant columns, rearranging columns, removing duplicates, standardizing text data, and identifying and removing outliers. Using the z-score method, visualizations such as boxplots were utilized to systematically identify outliers across several numerical columns. The dataset was reduced from 3141 to 3045 rows after the data transformations.

The data transformation process used here is an example of the essential preprocessing needed to draw valuable insights from raw datasets. Clean, high-quality data serves as the basis for further analysis and modeling, which can improve accuracy and have greater impact on data-driven business strategy and decision making.

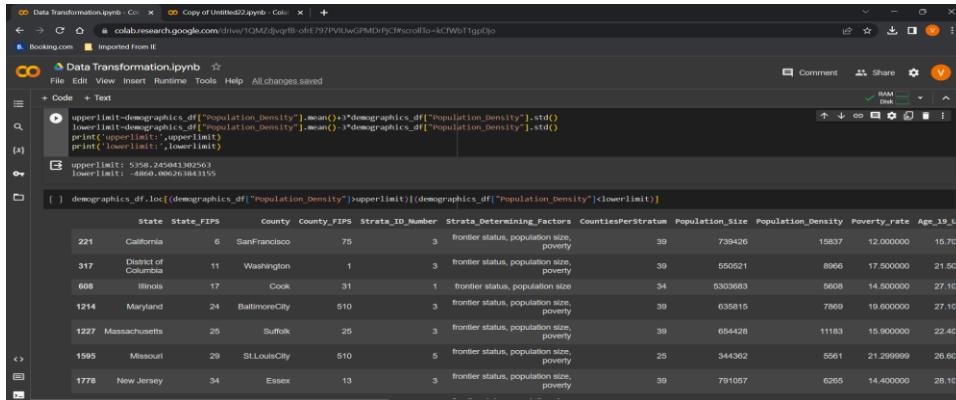
## References:

1. What is data transformation?. TIBCO Software. (n.d.). <https://www.tibco.com/reference-center/what-is-data-transformation#:~:text=Data%20transformation%20is%20the%20process,that%20of%20the%20destination%20system>
2. Demographic trends and Health Outcomes in the U.S. Kaggle. Accessed October 19, 2023. <https://www.kaggle.com/datasets/thedevastator/demographic-trends-and-health-outcomes-in-the-u> .
3. Sekan, F. (2023, February 3). 3 ways how to update data type of columns in Pandas. Medium. <https://medium.com/@filip.sekan/3-ways-how-to-update-data-type-of-columns-in-pandas-97ddb5f32ae4>
4. Data Cleaning - remove duplicates. Mage. (n.d.). <https://www.mage.ai/blog/data-cleaning-remove-duplicates>
5. Pandas.dataframe.rename#. pandas.DataFrame.rename - pandas 2.1.2 documentation. (n.d.). <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rename.html>
6. How to remove space from columns in pandas a data scientists guide. Saturn Cloud Blog. (2023, October 25). [https://saturncloud.io/blog/how-to-remove-space-from-columns-in-pandas-a-data-scientists-guide#:~:text=strip\(\)%20method,.The%20str.,column%20names%20or%20column%20values](https://saturncloud.io/blog/how-to-remove-space-from-columns-in-pandas-a-data-scientists-guide#:~:text=strip()%20method,.The%20str.,column%20names%20or%20column%20values).
7. Concatenate two columns into a single column in Pandas dataframe. Concatenate two columns into a single column in pandas dataframe. (n.d.). <https://net-informations.com/ds/pd/comb.htm>
8. (NNK), N. (2023, October 5). Pandas - how to change position of a column. Spark By {Examples}. <https://sparkbyexamples.com/pandas/pandas-change-position-of-a-column/>
9. Zafar, A. (2022, February 25). Handling outliers in Pandas. Medium. [https://medium.com/@arsalan\\_zafar/handling-outliers-in-pandas-5cd872eef508](https://medium.com/@arsalan_zafar/handling-outliers-in-pandas-5cd872eef508)
10. YouTube. (2022). YouTube. Retrieved November 5, 2023, from <https://www.youtube.com/watch?v=Cw2IvmWRcXs>.
11. [https://github.com/vpallapolu/ORES/blob/main/cleaned\\_demographics.csv](https://github.com/vpallapolu/ORES/blob/main/cleaned_demographics.csv)

## Appendix:

- **Handling Outliers for population density column**

The code below in the picture calculates the upper and lower limits for “Population density” column using mean +/- 3 standard deviations by using z-score method. This gives a threshold range to define outliers. `demographics_df.loc[]` function Identifies rows where 'Population density' column values are outside the limits. These are outlier rows.



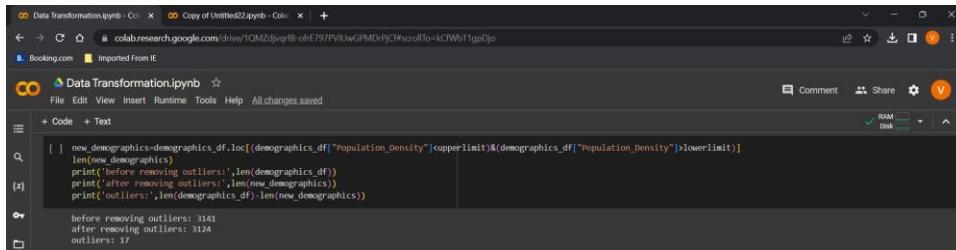
```

Data Transformation.ipynb Colaboratory Import From IE
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[ ] upperlimit=demographics_df["Population_Density"].mean() + 3*demographics_df["Population_Density"].std()
lowerlimit=demographics_df["Population_Density"].mean() - 3*demographics_df["Population_Density"].std()
print('upperlimit:',upperlimit)
print('lowerlimit:',lowerlimit)
[ ] upperlimit: 5358.254853032563
lowerlimit: -4866.086263843155
[ ] demographics_df.loc[(demographics_df["Population_Density"]>upperlimit)|(demographics_df["Population_Density"]<lowerlimit)]

```

State	State_FIPS	County	County_FIPS	Strata_ID_Number	Strata_Determining_Factors	CountiesPerStratum	Population_Size	Population_Density	Poverty_rate	Age_19_u
221	California	6	SanFrancisco	75	3 frontier status, population size, poverty	39	739426	15837	12.000000	15.70
317	District of Columbia	11	Washington	1	3 frontier status, population size, poverty	39	559521	8966	17.500000	21.50
608	Illinois	17	Cook	31	1 frontier status, population size	34	5303663	5606	14.500000	27.10
1214	Maryland	24	BaltimoreCity	510	3 frontier status, population size, poverty	39	635815	7869	19.800000	27.10
1227	Massachusetts	25	Suffolk	25	3 frontier status, population size, poverty	39	654428	11183	15.900000	22.40
1595	Missouri	29	St.LouisCity	510	5 frontier status, population size, poverty	25	344362	5561	21.299999	22.60
1778	New Jersey	34	Essex	13	3 frontier status, population size, poverty	39	791057	6265	14.400000	28.10

The code in the below picture uses `new_demographics = demographics_df.drop()` function to drop the outlier rows in the Population density column to create a new dataframe without outliers.

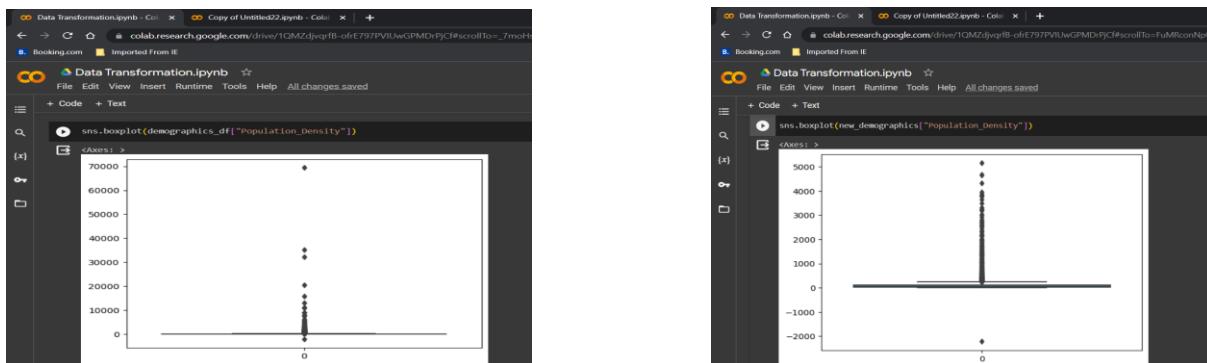


```

Data Transformation.ipynb Colaboratory Import From IE
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[ ] new_demographics=demographics_df.loc[(demographics_df["Population_Density"]<upperlimit)&(demographics_df["Population_Density"]>lowerlimit)]
[ ] len(new_demographics)
print('before removing outliers:',len(demographics_df))
print('after removing outliers:',len(new_demographics))
print('outliers:',len(demographics_df)-len(new_demographics))
[ ] before removing outliers: 3141
after removing outliers: 3124
outliers: 17

```

The left side picture depicts a code to Generate a boxplot to visually identify potential outliers in the 'Age 19 under' column while the right-side picture depicts Boxplot on new dataframe to verify outliers have been removed. This code uses the z-score method to systematically identify outlier values outside 3 standard deviations from the mean in the 'Age 19 under' column. It drops these outliers to clean the data for further analysis. The boxplots before and after validate the outliers were removed.



- **Handling outliers for poverty rate column**

The code below in the picture calculates the upper and lower limits for “Poverty rate” column using mean +/- 3 standard deviations by using z-score method. This gives a threshold range to define outliers. `demographics_df.loc[]` function Identifies rows where 'Poverty rate' column values are outside the limits. These are outlier rows. The code in the same picture uses `new_demographics = demographics_df.drop()` function to drop the outlier rows in the Age 19 under column to create a new dataframe without outliers.

```

Data Transformation.ipynb - Colab | Copy of Untitled2.ipynb - Colab | + 
← → C ⌂ colab.research.google.com/drive/1QMZdjvqrIB-ofrE797PVUlwGPMDrPjC#scrollTo=FuMrIconNpOzs
Booking.com Imported From IE

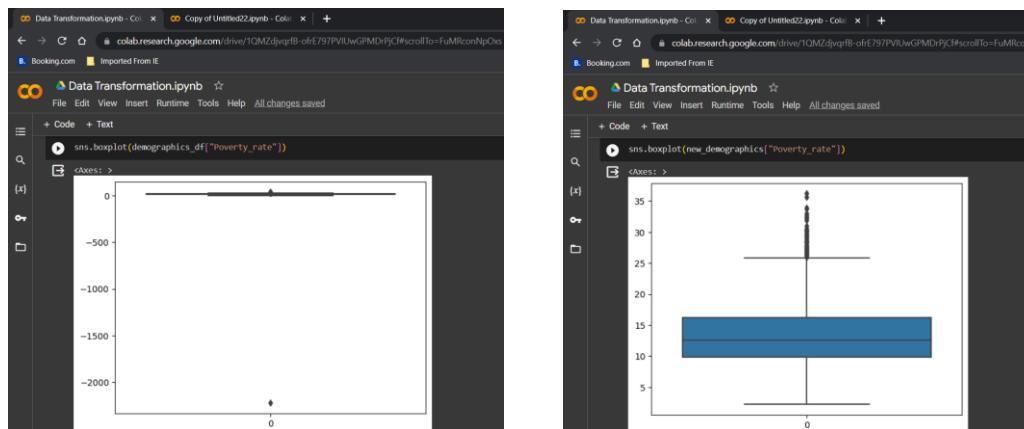
Data Transformation.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Q
[ ] upperlimit=demographics_df["Poverty_rate"].mean()+3*demographics_df["Poverty_rate"].std()
lowerlimit=demographics_df["Poverty_rate"].mean()-3*demographics_df["Poverty_rate"].std()
print("upperlimit:",upperlimit)
print("lowerlimit:",lowerlimit)
[+] upperlimit: 133.978834973145
lowerlimit: -107.92894880286445

[ ] demographics_df.loc[(demographics_df["Poverty_rate"]>upperlimit)|(demographics_df["Poverty_rate"]<lowerlimit)]
state state_FIPS county county_FIPS Strata_ID_Number Strata_Determining_Factors CountiesPerStratum Population_Size Population_Density Poverty_rate Age_19_Under Age_19_J
546 Hawaii 15 Kalawao 5 37 frontier status, population size, poverty, age 30 111 8 -2222.196991 1.4 82.3000
[ ] new_demographics=demographics_df.loc[(demographics_df["Poverty_rate"]>upperlimit)&(demographics_df["Poverty_rate"]<lowerlimit)]
len(new_demographics)
print('before removing outliers:',len(demographics_df))
print('after removing outliers:',len(new_demographics))
print('outliers:',len(demographics_df)-len(new_demographics))

before removing outliers: 3141
after removing outliers: 3140
outliers: 1

```

The left side picture depicts a code to Generate a boxplot to visually identify potential outliers in the 'Age 19 under' column while the right-side picture depicts Boxplot on new dataframe to verify outliers have been removed. This code uses the z-score method to systematically identify outlier values outside 3 standard deviations from the mean in the 'Age 19 under' column. It drops these outliers to clean the data for further analysis. The boxplots before and after validate the outliers were removed.



- **Handling outliers for age 19 under column**

The code below in the picture calculates the upper and lower limits for “Age 19 under” column using mean +/- 3 standard deviations by using z-score method. This gives a threshold range to define outliers. `demographics_df.loc[]` function Identifies rows where 'Age 19 under' column values are outside the limits. These are outlier rows.

```

Data Transformation.ipynb
[ ] upperlimit=desgraphics_df["Age_19_Under"].mean() + 3*desgraphics_df["Age_19_Under"].std()
lowerlimit=desgraphics_df["Age_19_Under"].mean() - 3*desgraphics_df["Age_19_Under"].std()
print("Upper limit:",upperlimit)
print("Lower limit:",lowerlimit)

upperlimit: 34.691858397295
lowerlimit: 14.90113840891113

[ ] demographics_df.loc[(demographics_df["Age_19_Under"]>upperlimit)|(demographics_df["Age_19_Under"]<lowerlimit)]

```

The screenshot shows a Jupyter Notebook cell with Python code. The code calculates the mean and standard deviation for the 'Age\_19\_Under' column and prints the upper and lower limits. It then uses the `loc` method to filter rows where the 'Age\_19\_Under' value is greater than the upper limit or less than the lower limit. The resulting DataFrame is displayed below the code cell.

The code in the below picture uses new\_demographics = demographics\_df.drop() function to drop the outlier rows in the Age 19 under column to create a new dataframe without outliers.

```

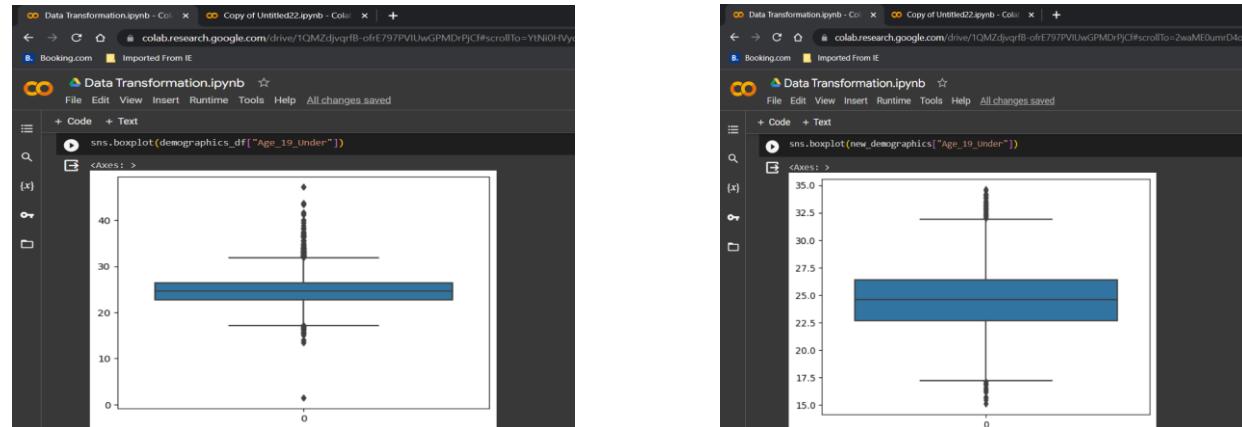
Data Transformation.ipynb
[ ] len(new_demographics)
len(demographics)
print('before removing outliers:',len(demographics))
print('after removing outliers:',len(new_demographics))
print('outliers:',len(demographics)-len(new_demographics))

before removing outliers: 3141
after removing outliers: 3109
outliers: 32

```

The screenshot shows a Jupyter Notebook cell with Python code. The code prints the lengths of the original 'demographics' DataFrame and the new 'new\_demographics' DataFrame. It then calculates the number of outliers by subtracting the length of 'new\_demographics' from the length of 'demographics'. The output shows there were 32 outliers removed, leaving 3109 data points.

The left side picture depicts a code to Generate a boxplot to visually identify potential outliers in the 'Age 19 under' column while the right-side picture depicts Boxplot on new dataframe to verify outliers have been removed. This code uses the z-score method to systematically identify outlier values outside 3 standard deviations from the mean in the 'Age 19 under' column. It drops these outliers to clean the data for further analysis. The boxplots before and after validate the outliers were removed.



- Handling outliers for age 19-64 column

The code below in the picture calculates the upper and lower limits for “Age 19-64” column using mean +/- 3 standard deviations by using z-score method. This gives a threshold range to define outliers. demographics\_df.loc[] function Identifies rows where 'Age 19-64' column values are outside the limits. These are outlier rows.

```

Data Transformation.ipynb
+ Code + Text
[ ] upperlimit= demographics_df["Age_19_64"].mean() + 3*demographics_df["Age_19_64"].std()
lowerlimit= demographics_df["Age_19_64"].mean() - 3*demographics_df["Age_19_64"].std()
print('upperlimit:',upperlimit)
print('lowerlimit:',lowerlimit)

upperlimit: 70.357568336961
lowerlimit: 56.2212995322656
[ ] demographics_df.loc[(demographics_df['Age_19_64']>upperlimit)|(demographics_df['Age_19_64']<lowerlimit)]

```

The screenshot shows a Jupyter Notebook cell containing Python code to calculate the mean and standard deviation for the 'Age\_19\_64' column, then print the resulting upper and lower limits. Below the code, a preview of the 'demographics' DataFrame is shown, filtered to show rows where the 'Age\_19\_64' value is either greater than the upper limit or less than the lower limit.

The code in the below picture uses new\_demographics = demographics\_df.drop() function to drop the outlier rows in the Age 19-64 column to create a new dataframe without outliers.

```

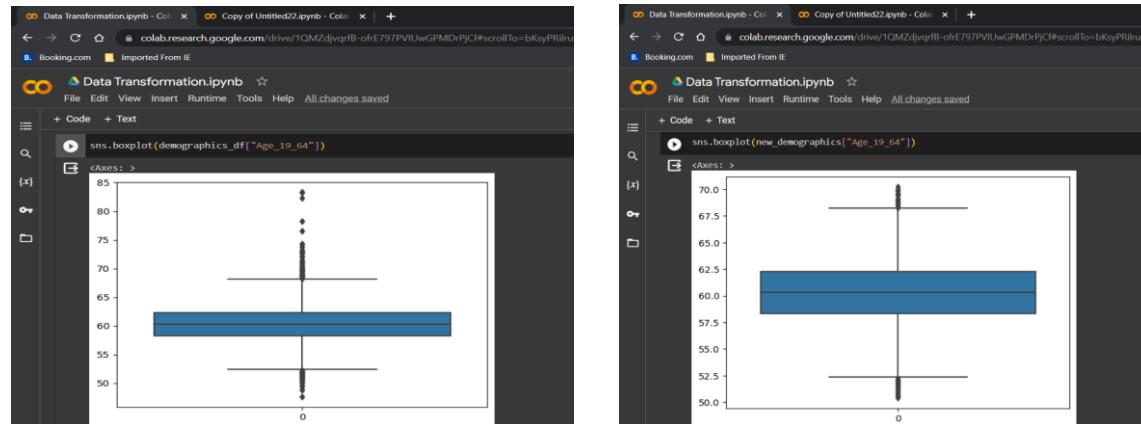
Data Transformation.ipynb
+ Code + Text
[ ] new_demographics=demographics_df.loc[(demographics_df['Age_19_64']>upperlimit)&(demographics_df['Age_19_64']<lowerlimit)]
len(new_demographics)
print('before removing outliers:',len(demographics_df))
print('after removing outliers:',len(new_demographics))
print('outliers:',len(demographics_df)-len(new_demographics))

before removing outliers: 3141
after removing outliers: 3113
outliers: 28

```

The screenshot shows a Jupyter Notebook cell containing Python code that drops rows from the 'demographics' DataFrame where the 'Age\_19\_64' value is outside the specified range. It then prints the total number of rows before and after the removal, and the count of outliers removed.

The left side picture depicts a code to Generate a boxplot to visually identify potential outliers in the 'Age 19-64' column while the right-side picture depicts Boxplot on new dataframe to verify outliers have been removed. This code uses the z-score method to systematically identify outlier values outside 3 standard deviations from the mean in the 'Age 19-64' column. It drops these outliers to clean the data for further analysis. The boxplots before and after validate the outliers were removed.



- Handling outliers for age 65 and over column

The code below in the picture calculates the upper and lower limits for “Age 65 and over” column using mean +/- 3 standard deviations by using z-score method. This gives a threshold range to define outliers. demographics\_df.loc[] function Identifies rows where 'Age 65 and over' column values are outside the limits. These are outlier rows.

```

Data Transformation.ipynb
[ ] upperlimit=demographics_df["Age_65_and_over"].mean() + 3*demographics_df["Age_65_and_over"].std()
lowerlimit=demographics_df["Age_65_and_over"].mean() - 3*demographics_df["Age_65_and_over"].std()
print('upperlimit:',upperlimit)
print('lowerlimit:',lowerlimit)

upperlimit: 27.23777956298828
lowerlimit: 2.571939468181789

```

[ ] demographics\_df.loc[(demographics\_df["Age\_65\_and\_over"]>upperlimit)|(demographics\_df["Age\_65\_and\_over"]<lowerlimit)]

State	State_FIPS	County	County_FIPS	Strata_ID_Number	Strata_Determining_Factors	CountiesPerStratum	Population_Size	Population_Density	Poverty_rate	Age_19_Under	
100	Arizona	4	LaPaz	12	88	frontier status, population size, poverty, age	23	20238	4	17.200001	20.600
325	Florida	12	Charlotte	15	13	frontier status, population size, poverty, age	16	157536	227	8.300000	16.200
326	Florida	12	Citrus	17	6	frontier status, population size, poverty	53	134370	230	12.400000	17.400
344	Florida	12	Highlands	55	24	frontier status, population size, poverty, age	35	95496	93	13.900000	20.000
375	Florida	12	Sarasota	115	10	frontier status, population size, poverty, age	36	366256	641	8.400000	17.500
410	Georgia	13	Chattahoochee	53	39	frontier status, population size, poverty, age	51	14679	59	14.500000	33.099
977	Kansas	20	Smith	183	81	frontier status, population size, poverty, age	58	4121	5	10.300000	19.299

The code in the below picture uses new\_demographics = demographics\_df.drop() function to drop the outlier rows in the Age 65 and over column to create a new dataframe without outliers.

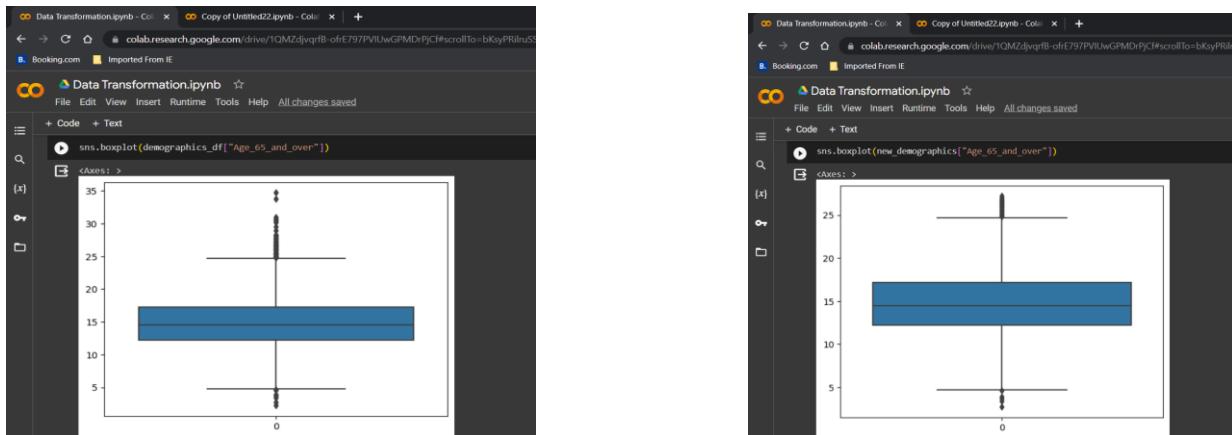
```

new_demographics=demographics_df.loc[(demographics_df["Age_65_and_over"]>upperlimit)&(demographics_df["Age_65_and_over"]<lowerlimit)]
len(new_demographics)
print('before removing outliers:',len(demographics_df))
print('after removing outliers:',len(new_demographics))
print('outliers:',len(demographics_df)-len(new_demographics))

before removing outliers: 3141
after removing outliers: 3120
outliers: 21

```

The left side picture depicts a code to Generate a boxplot to visually identify potential outliers in the 'Age 65 and over' column while the right-side picture depicts Boxplot on new dataframe to verify outliers have been removed. This code uses the z-score method to systematically identify outlier values outside 3 standard deviations from the mean in the 'Age 65 and over' column. It drops these outliers to clean the data for further analysis. The boxplots before and after validate the outliers were removed.



- Handling outliers for White column

The code below in the picture calculates the upper and lower limits for “White” column using mean +/- 3 standard deviations by using z-score method. This gives a threshold range to define outliers. demographics\_df.loc[] function Identifies rows where 'White' column values are outside the limits. These are outlier rows.

```

Data Transformation.ipynb Colaboratory Copy of Untitled22.ipynb - Colaboratory
File Edit View Insert Runtime Tools Help All changes saved
Comment Share RAM Disk
+ Code + Text
[ ] upperlimit=demographics_df["white"].mean()+(demographics_df["white"].std())
lowerlimit=demographics_df["white"].mean()-(demographics_df["white"].std())
print('upperlimit:',upperlimit)
print('lowerlimit:',lowerlimit)
[ ] upperlimit: 135.469260192871
lowerlimit: 38.56645584106445
[ ] demographics_df.loc[(demographics_df["white"]>upperlimit)|(demographics_df["white"]<lowerlimit)]

```

State	State_FIPS	County	County_FIPS	Strata_ID_Number	Strata_Determining_Factors	CountiesPerStratum	Population_Size	Population_Density	Poverty_rate	Age_19_Under
5	Alabama	1	Bullock	11	75	frontier status, population size, poverty, age...	37	11055	18	26.200001
23	Alabama	1	Dallas	47	52	frontier status, population size, poverty, age...	40	44366	45	25.000000
31	Alabama	1	Greene	63	75	frontier status, population size, poverty, age...	37	9651	15	24.900000
42	Alabama	1	Lowndes	85	66	frontier status, population size, poverty, age...	37	13076	18	23.299999
43	Alabama	1	Macon	87	44	frontier status, population size, poverty, age...	33	22810	37	25.799999
...	...	...	...	...	...	...	...	...	...	...
2419	South Dakota	46	Todd	121	66	frontier status, population size, poverty, age...	37	9882	7	33.700001

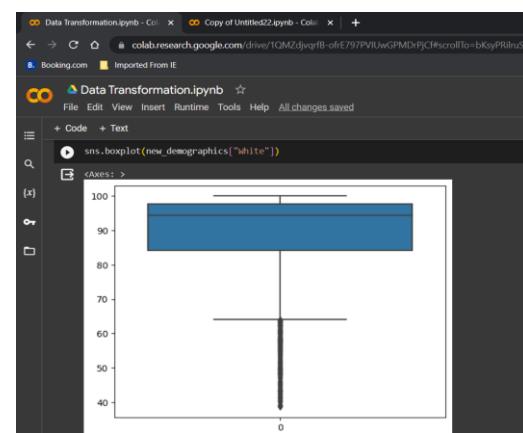
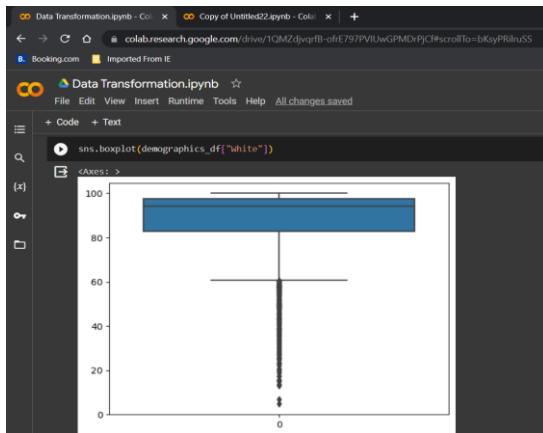
The code in the below picture uses new\_demographics = demographics\_df.drop() function to drop the outlier rows in the White column to create a new dataframe without outliers.

```

Data Transformation.ipynb Colaboratory Copy of Untitled22.ipynb - Colaboratory
File Edit View Insert Runtime Tools Help All changes saved
Comment Share RAM Disk
+ Code + Text
[ ] new_demographics=demographics_df.loc[(demographics_df["white"]>upperlimit)&(demographics_df["white"]<lowerlimit)]
len(new_demographics)
print('before removing outliers:',len(demographics_df))
print('after removing outliers:',len(new_demographics))
print('outliers:',len(demographics_df)-len(new_demographics))
[ ] before removing outliers: 3141
after removing outliers: 3067
outliers: 74

```

The left side picture depicts a code to Generate a boxplot to visually identify potential outliers in the 'White' column while the right-side picture depicts Boxplot on new dataframe to verify outliers have been removed. This code uses the z-score method to systematically identify outlier values outside 3 standard deviations from the mean in the 'White' column. It drops these outliers to clean the data for further analysis. The boxplots before and after validate the outliers were removed.



### • Handling outliers for Black column

The code below in the picture calculates the upper and lower limits for Black column using mean +/- 3 standard deviations by using z-score method. This gives a threshold range to define outliers.

`demographics_df.loc[]` function Identifies rows where 'Black' column values are outside the limits. These are outlier rows.

Data Transformation.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

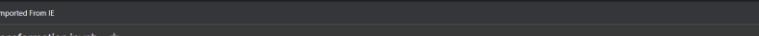
0 upperlimit=demographics\_df[["Black"]].mean() + 3\*demographics\_df[["Black"]].std()  
lowerlimit=demographics\_df[["Black"]].mean() - 3\*demographics\_df[["Black"]].std()  
print("upperlimit:",upperlimit)  
print("lowerlimit:",lowerlimit)

upperlimit: 52.623668760543  
lowerlimit: -34.65026572082519

[ ] demographics\_df.loc[(demographics\_df[["Black"]]>upperlimit) | (demographics\_df[["Black"]]<lowerlimit)]

	State	State_FIPS	County	County_FIPS	Strata_ID_Number	Strata_Determining_Factors	CountiesPerStratum	Population_Size	Population_Density	Poverty_rate	Age_19_Under
5	Alabama	1	Bullock	11	75	frontier status, population size, poverty, age...	37	11055	18	26.200001	24.700001
23	Alabama	1	Dallas	47	52	frontier status, population size, poverty, age...	40	44366	45	25.000000	29.000000
31	Alabama	1	Greene	63	75	frontier status, population size, poverty, age...	37	9661	15	24.900000	28.000000
32	Alabama	1	Hale	65	76	frontier status, population size, poverty, age...	38	18316	28	20.600000	26.799999
42	Alabama	1	Lauderdale	65	66	frontier status, population size, poverty, age...	37	13076	18	23.299999	29.000000
...	...	...	...	...	...	...	...	...	...	...	...
2905	Virginia	51	Sussex	183	71	frontier status, population size, poverty, age...	33	12071	25	17.000000	19.200001

The code in the below picture uses new\_demographics = demographics\_df.drop() function to drop the outlier rows in the Black column to create a new dataframe without outliers.

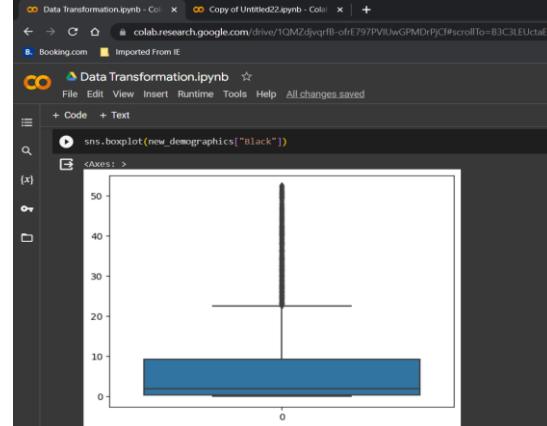
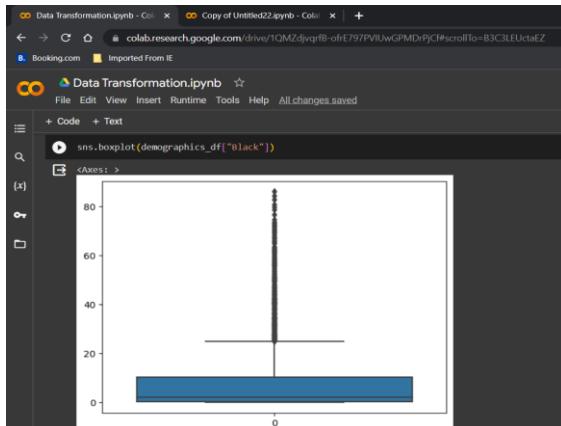


The screenshot shows a Jupyter Notebook interface with the title "Data Transformation.ipynb". The code cell contains Python code to remove outliers from a DataFrame named "new\_demographics" based on the "Black" column. It prints the number of rows before and after removing outliers, and the count of remaining outliers. The output shows that 3141 rows were removed, leaving 3061 rows, with 89 outliers remaining.

```
new_demographics=demographics_df.loc[(demographics_df["Black"]>upperlimit)&(demographics_df["Black"]<lowerlimit)]
len(new_demographics)
print('before removing outliers:',len(demographics_df))
print('after removing outliers:',len(new_demographics))
print('outliers:',len(demographics_df)-len(new_demographics))

before removing outliers: 3141
after removing outliers: 3061
outliers: 89
```

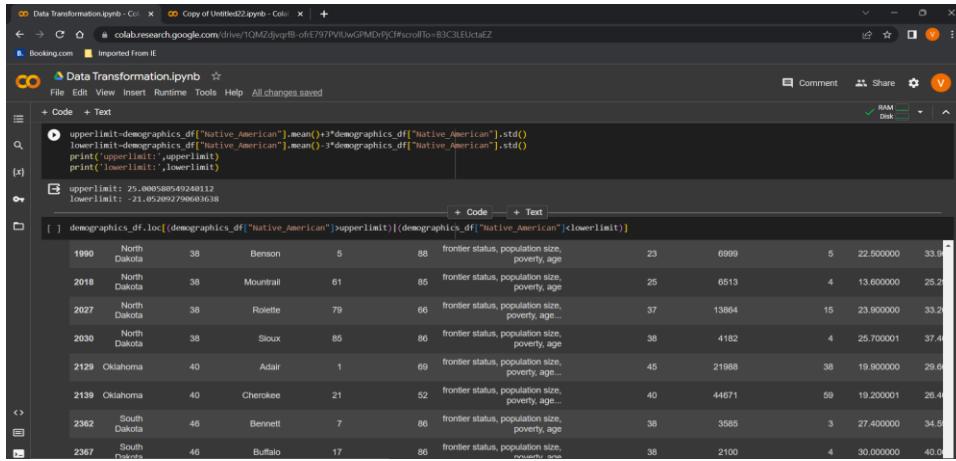
The left side picture depicts a code to Generate a boxplot to visually identify potential outliers in the 'Black' column while the right-side picture depicts Boxplot on new dataframe to verify outliers have been removed. This code uses the z-score method to systematically identify outlier values outside 3 standard deviations from the mean in the 'Black' column. It drops these outliers to clean the data for further analysis. The boxplots before and after validate the outliers were removed.



- Handling outliers for Native American column

The code below in the picture calculates the upper and lower limits using mean  $\pm$  3 standard deviations by using z-score method. This gives a threshold range to define outliers.

`demographics_df.loc[]` function Identifies rows where 'Native American' column values are outside the limits. These are outlier rows.



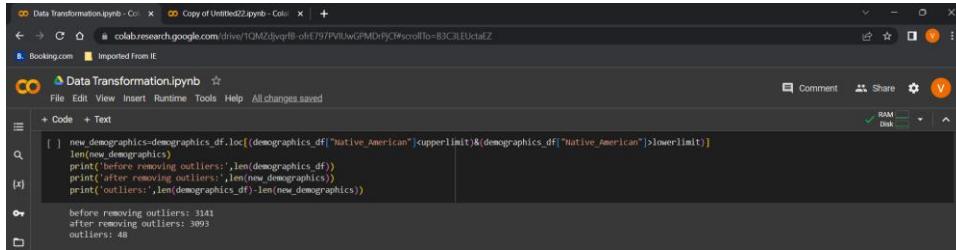
```

Data Transformation.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[1] upperlimit=demographics_df["Native_American"].mean()+3*demographics_df["Native_American"].std()
lowerlimit=demographics_df["Native_American"].mean()-3*demographics_df["Native_American"].std()
print('upperlimit:',upperlimit)
print('lowerlimit:',lowerlimit)
[2] upperlimit: 25.000588549240112
lowerlimit: -21.0209279003638
[3] demographics_df.loc[(demographics_df["Native_American"]>upperlimit)|(demographics_df["Native_American"]<lowerlimit)]

```

Year	State	Age	Native American Population	Frontier Status	Poverty Rate	Average Age	Median Income	Population Density	Median Household Income	Unemployment Rate	Population
1990	North Dakota	38	Benson	5	88	frontier status, population size, poverty, age	23	6999	5	22.500000	33.5
2018	North Dakota	38	Monteal	61	85	frontier status, population size, poverty, age	25	6513	4	13.600000	25.2
2027	North Dakota	38	Rolette	79	66	frontier status, population size, poverty, age...	37	13864	15	23.900000	33.2
2030	North Dakota	38	Sioux	85	86	frontier status, population size, poverty, age...	38	4182	4	25.700001	37.4
2129	Oklahoma	40	Adair	1	69	frontier status, population size, poverty, age...	45	21988	38	19.900000	29.6
2139	Oklahoma	40	Cherokee	21	52	frontier status, population size, poverty, age...	40	44671	59	19.200001	26.4
2362	South Dakota	46	Bennett	7	86	frontier status, population size, poverty, age...	38	3585	3	27.400000	34.5
2367	South Dakota	46	Buffalo	17	86	frontier status, population size, poverty, age...	38	2100	4	30.000000	40.0

The code in the below picture uses `new_demographics = demographics_df.drop()` function to drop the outlier rows in the Native American column to create a new dataframe without outliers.

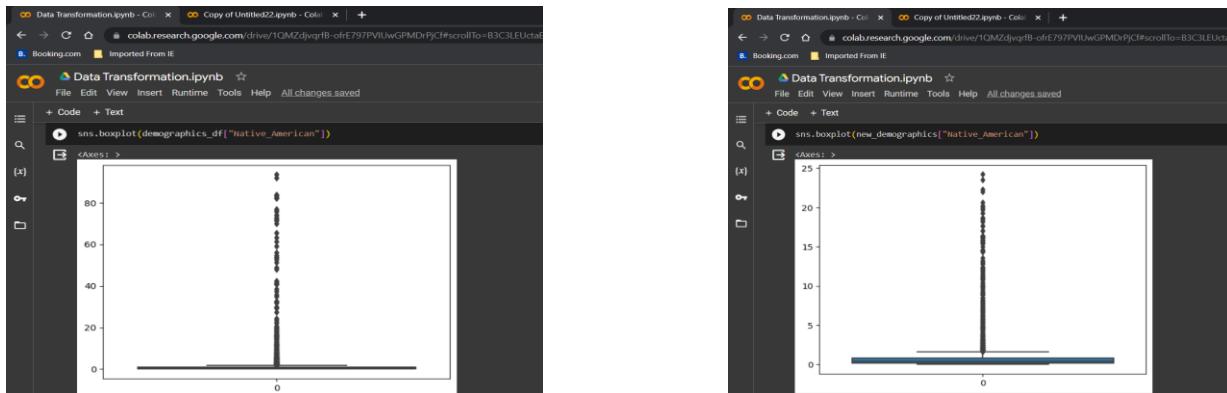


```

Data Transformation.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[1] new_demographics=demographics_df.loc[(demographics_df["Native_American"]<upperlimit)&(demographics_df["Native_American"]>lowerlimit)]
len(new_demographics)
print('before removing outliers:',len(demographics_df))
print('after removing outliers:',len(new_demographics))
print('outliers:',len(demographics_df)-len(new_demographics))
[2] before removing outliers: 3141
after removing outliers: 3093
outliers: 48

```

The left side picture depicts a code to Generate a boxplot to visually identify potential outliers in the 'Native American' column while the right-side picture depicts Boxplot on new dataframe to verify outliers have been removed. This code uses the z-score method to systematically identify outlier values outside 3 standard deviations from the mean in the 'Native American' column. It drops these outliers to clean the data for further analysis. The boxplots before and after validate the outliers were removed.



- Handling outliers for Asian column

The code below calculates the upper and lower limits using mean +/- 3 standard deviations. This gives a threshold range to define outliers. `demographics_df.loc[]` function Identifies rows where 'Asian' column values are outside the limits. These are outlier rows.

```

In [1]: upperlimit=demographics_df["Asian"].mean() + 3*demographics_df["Asian"].std()
lowerlimit=demographics_df["Asian"].mean() - 3*demographics_df["Asian"].std()
print('upperlimit:',upperlimit)
print('lowerlimit:',lowerlimit)

Out[1]: upperlimit: 9.394759415892
lowerlimit: -7.148659467697136

In [2]: demographics_df.loc[(demographics_df["Asian"]>upperlimit)|(demographics_df["Asian"]<lowerlimit)]

```

The screenshot shows a Jupyter Notebook interface with two code cells and a preview of a DataFrame. The first cell calculates the mean and standard deviation for the 'Asian' column and prints the upper and lower limits. The second cell uses the `loc` method to filter rows where the 'Asian' value is outside these limits. The preview shows a portion of the 'demographics\_df' DataFrame with columns like State, State\_FIPS, County, County\_FIPS, Strata\_ID, Number, Strata, Determining\_Factors, CountiesPerStratum, Population\_Size, Population\_Density, Poverty\_rate, Age\_19\_U, and more.

`new_demographics = demographics_df.drop()` function Drops the outlier rows to create a new dataframe without outliers.

```

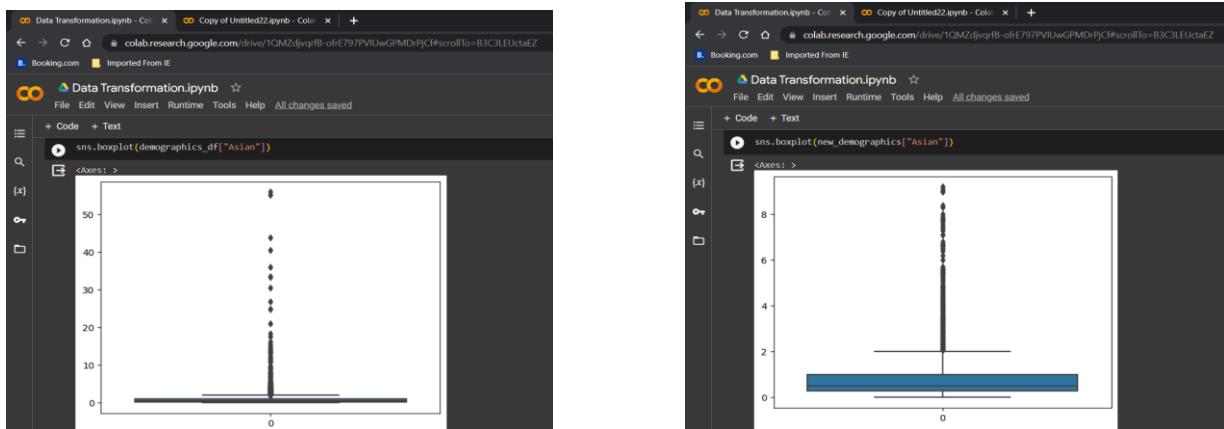
In [1]: new_demographics=demographics_df.loc[(demographics_df["Asian"]<upperlimit)&(demographics_df["Asian"]>lowerlimit)]
len(new_demographics)
print('before removing outliers:',len(demographics_df))
print('after removing outliers:',len(new_demographics))
print('outliers:',len(demographics_df)-len(new_demographics))

Out[1]: before removing outliers: 3141
after removing outliers: 3104
outliers: 37

```

The screenshot shows a Jupyter Notebook interface with one code cell. The code filters the 'demographics\_df' DataFrame to keep only rows where the 'Asian' value is between the calculated upper and lower limits. It then prints the total number of rows before and after removal, and the count of outliers removed.

The left side picture depicts a code to Generate a boxplot to visually identify potential outliers in the 'Asian' column while the right-side picture depicts Boxplot on new dataframe to verify outliers have been removed. This code uses the z-score method to systematically identify outlier values outside 3 standard deviations from the mean in the 'Asian' column. It drops these outliers to clean the data for further analysis. The boxplots before and after validate the outliers were removed.



- Handling outliers for Hispanic column

The below picture indicates the code to Calculate upper and lower limits of 'Hispanic' column using mean +/- 3 standard deviations. This gives a threshold range to define outliers. `demographics_df.loc[]` function Identifies rows where 'Hispanic' column values are outside the limits. These are outlier rows.

```

Data Transformation.ipynb - Colab
Copy of Untitled2.ipynb - Colab | + 
colab.research.google.com/drive/1QMZdjvqrIB-ofE797PVUlwGPMDrPjC#scrollTo=B3C3LEuctaEZ
Booking.com Imported From E
Data Transformation.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[ ] 

```
upperlimit=demographics_df["Hispanic"].mean() + 3*demographics_df["Hispanic"].std()
lowerlimit=demographics_df["Hispanic"].mean() - 3*demographics_df["Hispanic"].std()
print('upperlimit:',upperlimit)
print('lowerlimit:',lowerlimit)
```


[ ] 

```
upperlimit: 44.41216707229614
lowerlimit: -30.37619161695825
```


[ ] 

```
demographics_df.loc[(demographics_df["Hispanic"]>upperlimit)|(demographics_df["Hispanic"]<lowerlimit)]
```


State State_FIPS County County_FIPS Strata_ID_Number Strata_Determining_Factors CountiesPerStratum Population_Size Population_Density Poverty_rate Age_19_Under_Age_5
99 Arizona 4 Greenlee 11 78 frontier status, population size, poverty, age... 45 7521 4 11.200000 30.000000
106 Arizona 4 SantaCruz 23 51 frontier status, population size, poverty, age...
108 Arizona 4 Yuma 27 7 frontier status, population size, poverty...
189 California 6 Colusa 11 42 frontier status, population size, poverty, age...
183 California 6 Fresno 19 3 frontier status, population size, poverty...
... ...
2771 Texas 48 Yoakum 501 42 frontier status, population size, poverty, age...

```

The below picture uses new\_demographics = demographics\_df.drop() function to drop the outlier rows to create a new dataframe without outliers.

```

Data Transformation.ipynb - Colab
Copy of Untitled2.ipynb - Colab | + 
colab.research.google.com/drive/1QMZdjvqrIB-ofE797PVUlwGPMDrPjC#scrollTo=B3C3LEuctaEZ
Booking.com Imported From E
Data Transformation.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[ ] 

```
new_demographics=demographics_df[(demographics_df["Hispanic"]>upperlimit)&(demographics_df["Hispanic"]<lowerlimit)]
len(new_demographics)
print('before removing outliers:',len(demographics_df))
print('after removing outliers:',len(new_demographics))
outliers=len(demographics_df)-len(new_demographics)
print('outliers:',outliers)
```


[ ] 

```
before removing outliers: 3141
after removing outliers: 3045
outliers: 96
```


```

The left side picture depicts a code to Generate a boxplot to visually identify potential outliers in the 'Hispanic' column while the right-side picture depicts Boxplot on new dataframe to verify outliers have been removed. This code uses the z-score method to systematically identify outlier values outside 3 standard deviations from the mean in the 'Hispanic' column. It drops these outliers to clean the data for further analysis. The boxplots before and after validate the outliers were removed.

