GROUP PROJECT-3

Project Report: Data Analysis and Visualization

Course: IE6600 Computation and Visualization

Spring Semester 2024

Group Number - 10

Sri Sai Prabhath Reddy Gudipalli (002207631)

Deepthi Umesha (002661626)

Ashwini Mahadevaswamy (002661627)

Yashwant Dontam (002844794)

Introduction

CompViz Project 3 delved into the 2020-2021 Public School Characteristics dataset, employing data cleaning and visualization techniques to explore school attributes and demographics. Utilizing Python's Pandas and Plotly, the team efficiently managed over 100,000 entries, focusing on the geographic distribution of schools, the differentiation between charter and magnet schools, and the ethnic diversity within schools. Key findings were presented through interactive maps and cluster analysis, highlighting the diversity in the U.S. educational landscape. This project showcased the effectiveness of data science in uncovering insights from complex educational data.

Task: Data Acquisition and Inspection

In [121]:
```
import pandas as pd
df = pd.read_csv("Public_School_Characteristics_2020-21.csv", low_memory=False)
```

In [122]:
```
df.columns
```

Out[122]:
```
Index(['X', 'Y', 'OBJECTID', 'NCESSCH', 'SURVYEAR', 'STABR', 'LEAID',
       'ST_LEAID', 'LEA_NAME', 'SCH_NAME', 'LSTREET1', 'LSTREET2', 'LCITY',
       'LSTATE', 'LZIP', 'LZIP4', 'PHONE', 'CHARTER_TEXT', 'MAGNET_TEXT',
       'VIRTUAL', 'GSLO', 'GSHI', 'SCHOOL_LEVEL', 'TITLEI', 'STITLEI',
       'STATUS', 'SCHOOL_TYPE_TEXT', 'SY_STATUS_TEXT', 'ULOCALE', 'NMCNTY',
       'TOTFRL', 'FRELCH', 'REDLCH', 'PK', 'KG', 'G01', 'G02', 'G03', 'G04',
       'G05', 'G06', 'G07', 'G08', 'G09', 'G10', 'G11', 'G12', 'G13', 'UG',
       'AE', 'TOTMENROL', 'TOTFENROL', 'TOTAL', 'MEMBER', 'FTE', 'STUTERATIO',
       'AMALM', 'AMALF', 'AM', 'ASALM', 'ASALF', 'AS', 'BLALM', 'BLALF', 'BL',
       'HPALM', 'HPALF', 'HP', 'HIALM', 'HIALF', 'HI', 'TRALM', 'TRALF', 'TR',
       'WHALM', 'WHALF', 'WH', 'LATCOD', 'LONCOD'],
      dtype='object')
```

In [123]:
```
print("Shape of the dataset:", df.shape)
```

```
Shape of the dataset: (100722, 79)
```

In [124]:
```
print("Preview of the dataset:")
print(df.head())
```

```
Preview of the dataset:
          X        Y  OBJECTID      NCESSCH   SURVYEAR STABR   LEAID  \
0 -86.206200  34.2602         1  10000500870  2020-2021    AL  100005
1 -86.204900  34.2622         2  10000500871  2020-2021    AL  100005
2 -86.220100  34.2733         3  10000500879  2020-2021    AL  100005
3 -86.221806  34.2527         4  10000500889  2020-2021    AL  100005
4 -86.193300  34.2898         5  10000501616  2020-2021    AL  100005

  ST_LEAID          LEA_NAME                          SCH_NAME  ...  HIALF  \
0   AL-101  Albertville City         Albertville Middle School  ...  230.0
1   AL-101  Albertville City           Albertville High School  ...  371.0
2   AL-101  Albertville City    Albertville Intermediate School  ...  253.0
3   AL-101  Albertville City      Albertville Elementary School  ...  237.0
4   AL-101  Albertville City  Albertville Kindergarten and PreK  ...  137.0

      HI TRALM TRALF    TR  WHALM  WHALF     WH   LATCOD     LONCOD
0  469.0  19.0  10.0  29.0  187.0  184.0  371.0  34.2602 -86.206200
1  785.0  17.0  21.0  38.0  368.0  338.0  706.0  34.2622 -86.204900
2  481.0  17.0  12.0  29.0  177.0  168.0  345.0  34.2733 -86.220100
3  497.0   7.0   8.0  15.0  180.0  160.0  340.0  34.2527 -86.221806
4  288.0   6.0   7.0  13.0  108.0  108.0  216.0  34.2898 -86.193300
```

This Python code uses the pandas library to read a CSV file named "Public_School_Characteristics_2020-21.csv" into a DataFrame called df, ensuring that pandas doesn't use low memory mode during reading. The DataFrame df contains the data from the CSV file, allowing for analysis and manipulation of the dataset.

This output represents the column names (features) of the DataFrame df after reading the CSV file. Each column name corresponds to a specific attribute or piece of information about the public schools in the dataset. These attributes include information such as school location, name, grade levels, enrollment numbers, demographic breakdowns, and more.

The code prints the shape (number of rows and columns) of the dataset and displays a preview of the first few rows, providing an overview of the dataset's structure and content.

The output consists of two parts:

The shape of the dataset, which indicates the number of rows and columns. A preview of the dataset, displaying the first few rows along with their corresponding column values. This gives an insight into the data's structure and content.

Handling Missing Values

In [125]:

```
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
LSTREET1:          3 missing values
STUTERATIO:          1216 missing values
TOTAL:          2071 missing values
MEMBER:          2071 missing values
HI:          3698 missing values
WH:          3784 missing values
WHALM:          4354 missing values
WHALF:          4607 missing values
HIALM:          4686 missing values
HIALF:          4907 missing values
TR:          6700 missing values
BL:          8706 missing values
TRALM:          8943 missing values
TRALF:          9128 missing values
FTE:          9502 missing values
BLALM:          11599 missing values
BLALF:          12288 missing values
AS:          13828 missing values
ASALM:          17784 missing values
ASALF:          18097 missing values
TOTFRL:          23758 missing values
AM:          24373 missing values
FRELCH:          26605 missing values
REDLCH:          26605 missing values
AMALM:          31433 missing values
AMALF:          31687 missing values
HP:          34906 missing values
HPALM:          40038 missing values
HPALF:          40567 missing values
G02:          46742 missing values
G01:          46781 missing values
G03:          46790 missing values
G04:          47000 missing values
KG:          47185 missing values
G05:          48296 missing values
G06:          63502 missing values
G08:          68344 missing values
G07:          68572 missing values
PK:          69021 missing values
G09:          73736 missing values
G10:          73875 missing values
G11:          73903 missing values
```

This code identifies columns in the DataFrame df that contain missing values (NaNs) and prints the names of those columns along with the number of missing values in each column. It iterates over each column, calculates the count of missing values using the isnull().sum() function, and stores this count in a dictionary if it's greater than 0. It then sorts the dictionary based on the number of missing values and prints the column names along with their respective missing value counts.

This output lists the columns in the dataset that contain missing values, along with the number of missing values in each column. Each line represents a column name followed by the count of missing values. The count indicates how many entries in that column are NaN or null. This information helps identify which columns may require data cleaning or imputation techniques before analysis.

checking if we have any duplicates in the data

In [126]:
```
duplicates = df.duplicated().sum()
print("\nDuplicates:", duplicates)
```

```
Duplicates: 0
```

We are deleting the columns X and Y because the LATCOD and LONCOD represent the same values; the latitude and longitude co-ordinates respectively

This code checks for duplicated rows in the DataFrame df and counts the total number of duplicated rows. It uses the .duplicated() function to identify duplicate rows, and .sum() to count the occurrences of duplicated rows. Finally, it prints the total number of duplicated rows found in the DataFrame.

In [127]:
```
df = df.drop(columns=['X', 'Y'])
```

dropping the columns with more than 70% missing values

In [128]:
```
missing_percentage = (df.isnull().sum() / len(df)) * 100
columns_to_drop = missing_percentage[missing_percentage > 70].index
df = df.drop(columns=columns_to_drop)
print(columns_to_drop)
```

```
Index(['LSTREET2', 'G09', 'G10', 'G11', 'G12', 'G13', 'UG', 'AE', 'TOTMENROL',
       'TOTFENROL'],
      dtype='object')
```

dropping the SURVYEAR column because it contains an obvious information we know about the dataset; the Survey year for the data

In [129]:
```
df = df.drop(columns=['SURVYEAR'])
```

In [130]:
```
print("Columns with missing values and data type string in the dataset:")
missing_string_columns = {}

for column in df.select_dtypes(include='object').columns:  # Select only columns with object (string) data typ
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_string_columns[column] = missing_count

sorted_missing_string_columns = sorted(missing_string_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_string_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values and data type string in the dataset:
LSTREET1:          3 missing values
```

In [131]:
```
#filling the null values with the word "Unknown" in LSTREET column
df['LSTREET1'].fillna('Unknown', inplace=True)
```

In [132]:
```
#dropping it cause it is irrelevant
df = df.drop(columns=['PHONE'])
```

This code snippet performs data preprocessing tasks on a DataFrame:

It drops columns with over 70% missing values and the 'SURVYEAR' column.
It creates an intermediate column ('HP_SUM') and updates the 'HP' column based on a specified condition.
It drops irrelevant columns like 'PHONE'.
It fills missing values in specific columns with 0 or the word "Unknown".

It converts certain columns to integer data type. These steps aim to clean the data and prepare it for analysis.

In [133]:
```
#filling the missing values with 0 because the number of students in these columns is mutually exclusive.
df['PK'].fillna(0, inplace=True)
df['G07'].fillna(0, inplace=True)
df['G08'].fillna(0, inplace=True)
df['G06'].fillna(0, inplace=True)
df['G05'].fillna(0, inplace=True)
df['G04'].fillna(0, inplace=True)
df['G03'].fillna(0, inplace=True)
df['G02'].fillna(0, inplace=True)
df['G01'].fillna(0, inplace=True)
df['KG'].fillna(0, inplace=True)
```

In [134]:
```
df['HPALM'].fillna(0, inplace=True)
df['HPALF'].fillna(0, inplace=True)

df['HPALM'] = df['HPALM'].astype(int)
df['HPALF'] = df['HPALF'].astype(int)
```

This code fills missing values in specified columns ('PK', 'G07', 'G08', etc.) with 0 because the number of students in these columns is considered mutually exclusive. It then converts the data type of columns 'HPALM' and 'HPALF' to integers after filling missing values with 0.

The following code creates an intermediate column 'HP_SUM' which contains the sum of 'HPALM' and 'HPALF'. Then, it replaces the values in the 'HP' column with 0 where the sum in 'HP_SUM' is 0. Finally, it drops the intermediate column 'HP_SUM'. After running this code, the 'HP' column will be updated according to the specified condition.

In [135]:

```python
# Calculate the sum of 'HPALM' and 'HPALF' columns
df['HP_SUM'] = df['HPALM'] + df['HPALF']

# Replace 'HP' with 0 where the sum is 0
df.loc[df['HP_SUM'] == 0, 'HP'] = 0

# Drop the intermediate column 'HP_SUM'
df.drop(columns=['HP_SUM'], inplace=True)

df['HP'] = df['HP'].astype(int)
```

In [136]:

```python
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
STUTERATIO:          1216 missing values
TOTAL:          2071 missing values
MEMBER:          2071 missing values
HI:          3698 missing values
WH:          3784 missing values
WHALM:          4354 missing values
WHALF:          4607 missing values
HIALM:          4686 missing values
HIALF:          4907 missing values
TR:          6700 missing values
BL:          8706 missing values
TRALM:          8943 missing values
TRALF:          9128 missing values
FTE:          9502 missing values
BLALM:          11599 missing values
BLALF:          12288 missing values
AS:          13828 missing values
ASALM:          17784 missing values
ASALF:          18097 missing values
TOTFRL:          23758 missing values
AM:          24373 missing values
FRELCH:          26605 missing values
REDLCH:          26605 missing values
AMALM:          31433 missing values
AMALF:          31687 missing values
```

This code calculates the sum of the 'HPALM' and 'HPALF' columns and stores the result in a new intermediate column 'HP_SUM'. It then replaces values in the 'HP' column with 0 where the sum in 'HP_SUM' is 0. Afterward, the intermediate column 'HP_SUM' is dropped. Finally, it prints the columns with missing values in the dataset along with their respective counts.

The output lists columns in the dataset that contain missing values, along with the count of missing values in each column. Each line represents a column name followed by the number of missing values present in that column. This information helps identify which columns require further attention or imputation techniques before analysis.

In [137]:
```
df['AMALF'].fillna(0, inplace=True)
df['AMALM'].fillna(0, inplace=True)

df['AMALF'] = df['AMALF'].astype(int)
df['AMALM'] = df['AMALM'].astype(int)
```

In [138]:
```
# Calculate the sum of 'HPALM' and 'HPALF' columns
df['AM_SUM'] = df['AMALM'] + df['AMALF']

# Replace 'HP' with 0 where the sum is 0
df.loc[df['AM_SUM'] == 0, 'AM'] = 0

# Drop the intermediate column 'HP_SUM'
df.drop(columns=['AM_SUM'], inplace=True)

df['AM'] = df['AM'].astype(int)
```

In [139]:
```
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
STUTERATIO:          1216 missing values
TOTAL:          2071 missing values
MEMBER:          2071 missing values
HI:          3698 missing values
WH:          3784 missing values
WHALM:          4354 missing values
WHALF:          4607 missing values
HIALM:          4686 missing values
HIALF:          4907 missing values
TR:          6700 missing values
BL:          8706 missing values
TRALM:          8943 missing values
TRALF:          9128 missing values
FTE:          9502 missing values
BLALM:          11599 missing values
BLALF:          12288 missing values
AS:          13828 missing values
ASALM:          17784 missing values
ASALF:          18097 missing values
TOTFRL:          23758 missing values
FRELCH:          26605 missing values
REDLCH:          26605 missing values
```

code fills missing values in the 'AMALF' and 'AMALM' columns with 0 and then converts these columns to integer data type. It then calculates the sum of 'AMALM' and 'AMALF' columns, storing the result in an intermediate column 'AM_SUM'. Values in the 'AM' column are replaced with 0 where the sum in 'AM_SUM' is 0. Finally, the intermediate column 'AM_SUM' is dropped, and the 'AM' column is converted to an integer data type. This process ensures consistency and prepares the data for further analysis.

code iterates through each column in the DataFrame 'df' and counts the number of missing values in each column. If a column has missing values (i.e., the count is greater than 0), it adds the column name and the corresponding count of missing values to a dictionary. It then sorts the dictionary based on the count of missing values and prints the column names along with their respective counts of missing values. This provides a summary of columns with missing data in the dataset.

The output lists columns in the dataset that contain missing values, along with the count of missing values in each column. Each line presents a column name followed by the number of missing values found in that column. This information helps identify which columns have missing data, aiding in further data cleaning or imputation processes before analysis.

In [140]:

```python
# Group the DataFrame by LCITY and calculate the mean of FRELCH for each city
city_means = df.groupby('LCITY')['FRELCH'].mean()

# Define a function to fill missing values with the mean of the corresponding city
def fill_missing_frelch(row):
    if pd.isna(row['FRELCH']):  # Check if FRELCH is missing
        city_mean = city_means.get(row['LCITY'])  # Get the mean for the corresponding city
        if city_mean is not None:  # Check if mean exists for the city
            return city_mean
    return row['FRELCH']  # Return original value if no mean is found or if FRELCH is not missing

# Apply the function to fill missing values in FRELCH
df['FRELCH'] = df.apply(fill_missing_frelch, axis=1)

# Confirm that missing values in FRELCH have been filled with the mean of the corresponding city
print("Missing values in FRELCH filled with city-wise mean.")
```

```
Missing values in FRELCH filled with city-wise mean.
```

In [141]:

```
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
STUTERATIO:          1216 missing values
TOTAL:          2071 missing values
MEMBER:          2071 missing values
HI:          3698 missing values
WH:          3784 missing values
WHALM:          4354 missing values
WHALF:          4607 missing values
HIALM:          4686 missing values
HIALF:          4907 missing values
TR:          6700 missing values
BL:          8706 missing values
TRALM:          8943 missing values
TRALF:          9128 missing values
FTE:          9502 missing values
FRELCH:          11022 missing values
BLALM:          11599 missing values
BLALF:          12288 missing values
AS:          13828 missing values
ASALM:          17784 missing values
ASALF:          18097 missing values
TOTFRL:          23758 missing values
REDLCH:          26605 missing values
```

This code snippet calculates the mean of the 'FRELCH' column for each city by grouping the DataFrame by city. Then, it defines a function to fill missing values in 'FRELCH' with the mean of the corresponding city. The function checks if 'FRELCH' is missing, retrieves the mean for the corresponding city, and fills the missing value with the mean if available. Afterward, the function is applied to the DataFrame to fill missing values in 'FRELCH'. Finally, it confirms that missing values in 'FRELCH' have been filled with city-wise means. This process helps to impute missing values in 'FRELCH' based on city-specific averages.

code iterates through each column in the DataFrame 'df' and counts the number of missing values in each column. If a column has missing values (i.e., the count is greater than 0), it adds the column name and the corresponding count of missing values to a dictionary. It then sorts the dictionary based on the count of missing values and prints the column names along with their respective counts of missing values. This provides a summary of columns with missing data in the dataset.

output provides a list of columns in the dataset along with the count of missing values present in each column. Each line displays a column name followed by the number of missing values found in that column. This information helps identify which columns have missing data, aiding in further data cleaning or imputation processes before analysis.

In [142]:

```python
# Group the DataFrame by LCITY and calculate the mean of REDLCH for each city
city_means_redlch = df.groupby('LCITY')['REDLCH'].mean()

# Define a function to fill missing values with the mean of the corresponding city
def fill_missing_redlch(row):
    if pd.isna(row['REDLCH']):  # Check if REDLCH is missing
        city_mean = city_means_redlch.get(row['LCITY'])  # Get the mean for the corresponding city
        if city_mean is not None:  # Check if mean exists for the city
            return city_mean
    return row['REDLCH']  # Return original value if no mean is found or if REDLCH is not missing

# Apply the function to fill missing values in REDLCH
df['REDLCH'] = df.apply(fill_missing_redlch, axis=1)

# Confirm that missing values in REDLCH have been filled with the mean of the corresponding city
print("Missing values in REDLCH filled with city-wise mean.")
```

```
Missing values in REDLCH filled with city-wise mean.
```

In [143]:

```python
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
STUTERATIO:          1216 missing values
TOTAL:          2071 missing values
MEMBER:          2071 missing values
HI:          3698 missing values
WH:          3784 missing values
WHALM:          4354 missing values
WHALF:          4607 missing values
HIALM:          4686 missing values
HIALF:          4907 missing values
TR:          6700 missing values
BL:          8706 missing values
TRALM:          8943 missing values
TRALF:          9128 missing values
FTE:          9502 missing values
FRELCH:          11022 missing values
REDLCH:          11022 missing values
BLALM:          11599 missing values
BLALF:          12288 missing values
AS:          13828 missing values
ASALM:          17784 missing values
ASALF:          18097 missing values
TOTFRL:          23758 missing values
```

code calculates the mean of the 'REDLCH' column for each city by grouping the DataFrame by city. It then defines a function to fill missing values in 'REDLCH' with the mean of the corresponding city. The function checks if 'REDLCH' is missing, retrieves the mean for the corresponding city, and fills the missing value with the mean if available. Afterward, the function is applied to the DataFrame to fill missing values in 'REDLCH'. Finally, it confirms that missing values in 'REDLCH' have been filled with city-wise means. This process helps to impute missing values in 'REDLCH' based on city-specific

averages and iterates through each column in the DataFrame 'df' and counts the number of missing values in each column. If a column has missing values (i.e., the count is greater than 0), it adds the column name and the corresponding count of missing values to a dictionary. It then sorts the dictionary based on the count of missing values and prints the column names along with their respective counts of missing values. This provides a summary of columns with missing data in the dataset.

output provides a list of columns in the dataset along with the count of missing values present in each column. Each line displays a column name followed by the number of missing values found in that column. It highlights columns where data is missing, helping identify areas that may require further investigation or data imputation before analysis.

In [144]:
```
df['TOTFRL'] = df['FRELCH'] + df['REDLCH']
print("TOTFRL column filled with the sum of FRELCH and REDLCH values.")
```

```
TOTFRL column filled with the sum of FRELCH and REDLCH values.
```

In [145]:
```
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
STUTERATIO:          1216 missing values
TOTAL:          2071 missing values
MEMBER:          2071 missing values
HI:          3698 missing values
WH:          3784 missing values
WHALM:          4354 missing values
WHALF:          4607 missing values
HIALM:          4686 missing values
HIALF:          4907 missing values
TR:          6700 missing values
BL:          8706 missing values
TRALM:          8943 missing values
TRALF:          9128 missing values
FTE:          9502 missing values
TOTFRL:          11022 missing values
FRELCH:          11022 missing values
REDLCH:          11022 missing values
BLALM:          11599 missing values
BLALF:          12288 missing values
AS:          13828 missing values
ASALM:          17784 missing values
ASALF:          18097 missing values
```

This code iterates through each column in the DataFrame 'df' and counts the number of missing values in each column. If a column has missing values (i.e., the count is greater than 0), it adds the column name and the corresponding count of missing values to a dictionary. It then sorts the dictionary based on the

count of missing values and prints the column names along with their respective counts of missing values. This provides a summary of columns with missing data in the dataset.
output lists columns in the dataset that contain missing values, along with the count of missing values in each column. Each line presents a column name followed by the number of missing values found in that column. It highlights columns with missing data, aiding in identifying areas that may require further investigation or data imputation before analysis.

In [146]:
```
df['ASALM'].fillna(0, inplace=True)
df['ASALF'].fillna(0, inplace=True)

# Convert "ASALM" and "ASALF" columns to integers
df['ASALM'] = df['ASALM'].astype(int)
df['ASALF'] = df['ASALF'].astype(int)
```

In [147]:
```
# Fill AS column with the sum of ASALM and ASALF values in the same row
df['AS'] = df['ASALM'] + df['ASALF']

# Confirm that the AS column has been updated
print("AS column filled with the sum of ASALM and ASALF values.")
```

```
AS column filled with the sum of ASALM and ASALF values.
```

In [148]:
```
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
STUTERATIO:          1216 missing values
TOTAL:          2071 missing values
MEMBER:          2071 missing values
HI:          3698 missing values
WH:          3784 missing values
WHALM:          4354 missing values
WHALF:          4607 missing values
HIALM:          4686 missing values
HIALF:          4907 missing values
TR:          6700 missing values
BL:          8706 missing values
TRALM:          8943 missing values
TRALF:          9128 missing values
FTE:          9502 missing values
TOTFRL:          11022 missing values
FRELCH:          11022 missing values
REDLCH:          11022 missing values
BLALM:          11599 missing values
BLALF:          12288 missing values
```

code snippet fills missing values in the 'ASALM' and 'ASALF' columns with 0 and converts these columns to integer data type. Then, it calculates the sum of 'ASALM' and 'ASALF' for each row and assigns the result to the 'AS' column. Finally, it confirms that the 'AS' column has been updated with the

sum of 'ASALM' and 'ASALF' values. Additionally, it prints the columns with missing values in the dataset to provide a summary of columns requiring further attention or data imputation.

output lists columns in the dataset that contain missing values, along with the count of missing values in each column. Each line presents a column name followed by the number of missing values found in that column. It highlights columns with missing data, aiding in identifying areas that may require further investigation or data imputation before analysis.

In [149]:
```
df['BLALM'].fillna(0, inplace=True)
df['BLALF'].fillna(0, inplace=True)

# Convert "BLALM" and "BLALF" columns to integers
df['BLALM'] = df['BLALM'].astype(int)
df['BLALF'] = df['BLALF'].astype(int)
```

In [150]:
```
# Fill BL column with the sum of BLALM and BLALF values in the same row
df['BL'] = df['BLALM'] + df['BLALF']

# Confirm that the BL column has been updated
print("BL column filled with the sum of BLALM and BLALF values.")
```

```
BL column filled with the sum of BLALM and BLALF values.
```

In [151]:
```
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
STUTERATIO:          1216 missing values
TOTAL:          2071 missing values
MEMBER:          2071 missing values
HI:          3698 missing values
WH:          3784 missing values
WHALM:          4354 missing values
WHALF:          4607 missing values
HIALM:          4686 missing values
HIALF:          4907 missing values
TR:          6700 missing values
TRALM:          8943 missing values
TRALF:          9128 missing values
FTE:          9502 missing values
TOTFRL:          11022 missing values
FRELCH:          11022 missing values
REDLCH:          11022 missing values
```

This code fills missing values in the 'BLALM' and 'BLALF' columns with 0 and converts these columns to integer data type. Then, it calculates the sum of 'BLALM' and 'BLALF' for each row and assigns the result to the 'BL' column. Finally, it confirms that the 'BL' column has been updated with the sum of

'BLALM' and 'BLALF' values. Additionally, it prints the columns with missing values in the dataset to provide a summary of columns requiring further attention or data imputation.
output lists columns in the dataset that contain missing values, along with the count of missing values in each column. Each line presents a column name followed by the number of missing values found in that column. It highlights columns with missing data, indicating areas that may require further investigation or data imputation before analysis.

In [152]:
```
#dropping the MEMBER column because it is a duplicate for the TOTAL column
df.drop(columns=['MEMBER'], inplace=True)
```

In [153]:
```
# Create the 'TOTAL' column by summing the values in columns 'PK' through 'G08'
df['TOTAL'] = df[['PK', 'KG', 'G01', 'G02', 'G03', 'G04', 'G05', 'G06', 'G07', 'G08']].sum(axis=1)

# Confirm that the 'TOTAL' column has been updated
print("TOTAL column updated with the sum of 'PK' through 'G08' values.")
```

```
TOTAL column updated with the sum of 'PK' through 'G08' values.
```

In [154]:
```
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
STUTERATIO:          1216 missing values
HI:          3698 missing values
WH:          3784 missing values
WHALM:          4354 missing values
WHALF:          4607 missing values
HIALM:          4686 missing values
HIALF:          4907 missing values
TR:          6700 missing values
TRALM:          8943 missing values
TRALF:          9128 missing values
FTE:          9502 missing values
TOTFRL:          11022 missing values
FRELCH:          11022 missing values
REDLCH:          11022 missing values
```

This code snippet removes the 'MEMBER' column from the DataFrame 'df'. Then, it calculates the sum of columns 'PK' through 'G08' and assigns the result to the 'TOTAL' column. It confirms that the 'TOTAL' column has been updated with the sum of 'PK' through 'G08' values. Finally, it prints the columns with missing values in the dataset to provide a summary of columns requiring further attention or data imputation.

output lists columns in the dataset that contain missing values, along with the count of missing values in each column. Each line presents a column name followed by the number of missing values found in that column. It highlights columns with missing data, indicating areas that may require further investigation or data imputation before analysis.

In [155]:

```python
# Calculate the mean of 'FRELCH' and 'REDLCH' within each 'LSTATE' class
state_means = df.groupby('LSTATE')[['FRELCH', 'REDLCH']].mean()

# Define a function to fill missing values with the mean of the corresponding state
def fill_missing_with_state_mean(row, column):
    state = row['LSTATE']
    mean_value = state_means.loc[state, column]
    if pd.isna(row[column]):
        return mean_value
    else:
        return row[column]

# Apply the function to fill missing values in 'FRELCH' and 'REDLCH'
df['FRELCH'] = df.apply(fill_missing_with_state_mean, axis=1, args=('FRELCH',))
df['REDLCH'] = df.apply(fill_missing_with_state_mean, axis=1, args=('REDLCH',))

# Confirm that missing values in 'FRELCH' and 'REDLCH' have been filled with the mean of the corresponding sta
print("Missing values in 'FRELCH' and 'REDLCH' filled with state-wise means.")
```

```
Missing values in 'FRELCH' and 'REDLCH' filled with state-wise means.
```

In [156]:

```python
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
FRELCH:          64 missing values
REDLCH:          64 missing values
STUTERATIO:          1216 missing values
HI:          3698 missing values
WH:          3784 missing values
WHALM:          4354 missing values
WHALF:          4607 missing values
HIALM:          4686 missing values
HIALF:          4907 missing values
TR:          6700 missing values
TRALM:          8943 missing values
TRALF:          9128 missing values
FTE:          9502 missing values
TOTFRL:          11022 missing values
```

code calculates the mean of columns 'FRELCH' and 'REDLCH' grouped by the states in the DataFrame. Then, it defines a function to fill missing values in these columns with the mean value of the corresponding state. The function takes a row and a column name as arguments and returns the mean value if the value in the column is missing; otherwise, it returns the original value. Finally, it applies this function to fill missing values in the 'FRELCH' and 'REDLCH' columns and prints a confirmation message. code prints a list of columns in the dataset that contain missing values along with the count of missing values in each column. It iterates over each column in the DataFrame 'df', calculates the number of missing values using the 'isnull().sum()' method, and stores the count in a dictionary 'missing_columns'. It then sorts the dictionary items based on the count of missing values and prints each column name followed by its corresponding count of missing values. This provides a summary of columns with missing data, facilitating further investigation or data imputation.

output provides a list of columns in the dataset with missing values, along with the count of missing values in each column. Each line presents a column name followed by the number of missing values found in that column. It highlights columns where data is missing, indicating areas that may require further investigation or data imputation before analysis.

In [157]:
```
# Add the columns 'FRELCH' and 'REDLCH' together and assign the sum to 'TOTFRL' column
df['TOTFRL'] = df['FRELCH'] + df['REDLCH']

# Confirm that the 'TOTFRL' column has been updated
print("TOTFRL column updated with the sum of FRELCH and REDLCH values.")
```

```
TOTFRL column updated with the sum of FRELCH and REDLCH values.
```

In [158]:
```
# Drop rows with missing values in 'TOTFRL', 'FRELCH', and 'REDLCH' columns
df.dropna(subset=['TOTFRL', 'FRELCH', 'REDLCH'], inplace=True)

# Confirm that rows with missing values have been deleted
print("Rows with missing values in 'TOTFRL', 'FRELCH', and 'REDLCH' columns have been deleted.")
```

```
Rows with missing values in 'TOTFRL', 'FRELCH', and 'REDLCH' columns have been deleted.
```

In [159]:
```
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
STUTERATIO:          1216 missing values
HI:          3637 missing values
WH:          3726 missing values
WHALM:          4293 missing values
WHALF:          4548 missing values
HIALM:          4624 missing values
HIALF:          4844 missing values
TR:          6636 missing values
TRALM:          8879 missing values
TRALF:          9064 missing values
FTE:          9438 missing values
```

This code updates the 'TOTFRL' column in the DataFrame by summing the values of 'FRELCH' and 'REDLCH' columns. Then, it removes rows with missing values in any of these three columns ('TOTFRL', 'FRELCH', and 'REDLCH') using the dropna() method with the subset parameter. Finally, it prints a message confirming the deletion of rows with missing values in the specified columns, followed by a summary of columns with missing values in the dataset.

output lists columns in the dataset with missing values along with the count of missing values in each column. It highlights areas where data is missing, indicating potential gaps in the dataset that may require further investigation or handling, such as imputation or deletion of missing values.

In [160]:
```
df.drop(columns=['FTE'], inplace=True)
```

In [161]:
```
df['TRALM'].fillna(0, inplace=True)
df['TRALF'].fillna(0, inplace=True)
```

In [162]:
```
df['TR'] = df['TRALM'] + df['TRALF']
```

In [163]:
```
print("Columns with missing values in the dataset:")
missing_columns = {}

for column in df.columns:
    missing_count = df[column].isnull().sum()
    if missing_count > 0:
        missing_columns[column] = missing_count

sorted_missing_columns = sorted(missing_columns.items(), key=lambda x: x[1])

for column, missing_count in sorted_missing_columns:
    print(f"{column}:          {missing_count} missing values")
```

```
Columns with missing values in the dataset:
STUTERATIO:          1216 missing values
HI:          3637 missing values
WH:          3726 missing values
WHALM:          4293 missing values
WHALF:          4548 missing values
HIALM:          4624 missing values
HIALF:          4844 missing values
```

This code snippet performs the following operations:

Drops the 'FTE' column from the DataFrame.
Fills missing values in the 'TRALM' and 'TRALF' columns with 0.
Calculates the total number of students ('TR') by adding the values of 'TRALM' and 'TRALF'.
Prints the columns with missing values in the dataset along with their respective counts of missing values.

The output indicates that there are missing values in several columns of the dataset, including 'STUTERATIO', 'HI', 'WH', 'WHALM', 'WHALF', 'HIALM', and 'HIALF'. The presence of these missing values suggests the need for data cleaning and imputation techniques to handle the missing data appropriately for analysis or modeling purposes.

In [164]:
```
df['HIALM'].fillna(0, inplace=True)
df['HIALF'].fillna(0, inplace=True)

df['HI'] = df['HIALM'] + df['HIALF']
```

In [165]:
```
df['WHALM'].fillna(0, inplace=True)
df['WHALF'].fillna(0, inplace=True)

df['WH'] = df['WHALM'] + df['WHALF']
```

In [166]:
```
df.drop(columns=['STUTERATIO'], inplace=True)
```

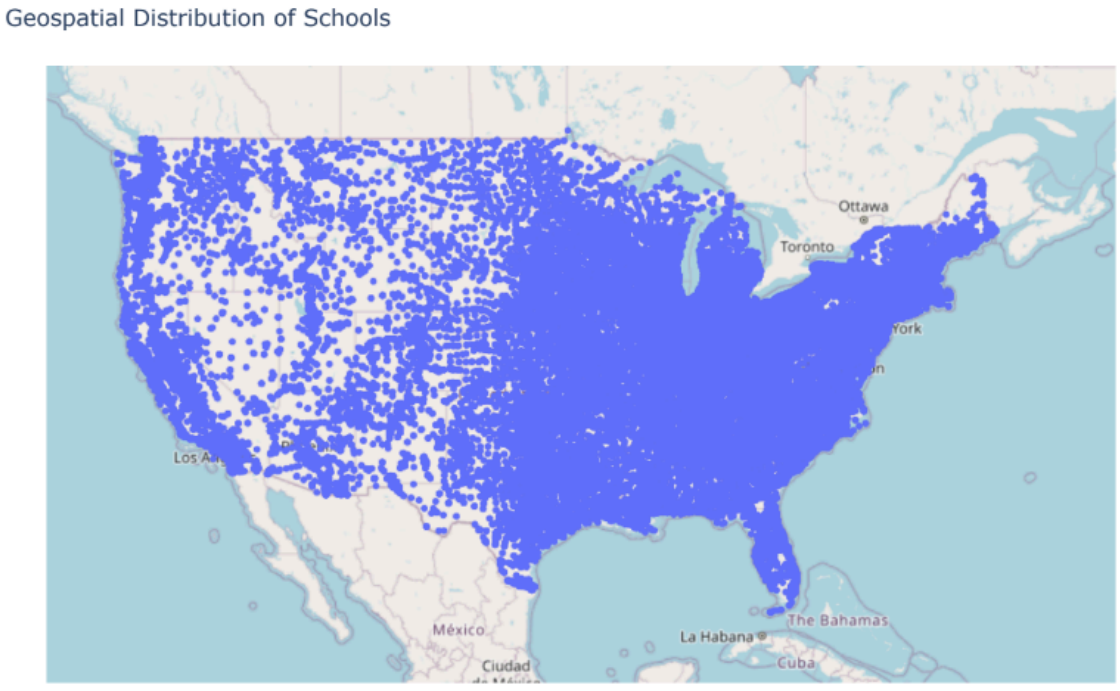
In [167]:
```
null_values = df.isnull().sum().sum()
print(null_values)
```

```
0
```

This code fills missing values in columns 'HIALM' and 'HIALF' with 0 and then computes the total for the 'HI' column. Similarly, it fills missing values in 'WHALM' and 'WHALF' with 0 and computes the total for the 'WH' column. It drops the 'STUTERATIO' column and finally calculates the total number of null values remaining in the DataFrame.

```
import plotly.express as px

# Create a scatter map plot of school locations
fig = px.scatter_mapbox(df, lat='LATCOD', lon='LONCOD', hover_name='SCH_NAME', hover_data=['LCITY'],
                        zoom=3, height=600)
fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(title='Geospatial Distribution of Schools')
fig.show()
fig.write_html("geospatial_distribution_of_schools.html")
```

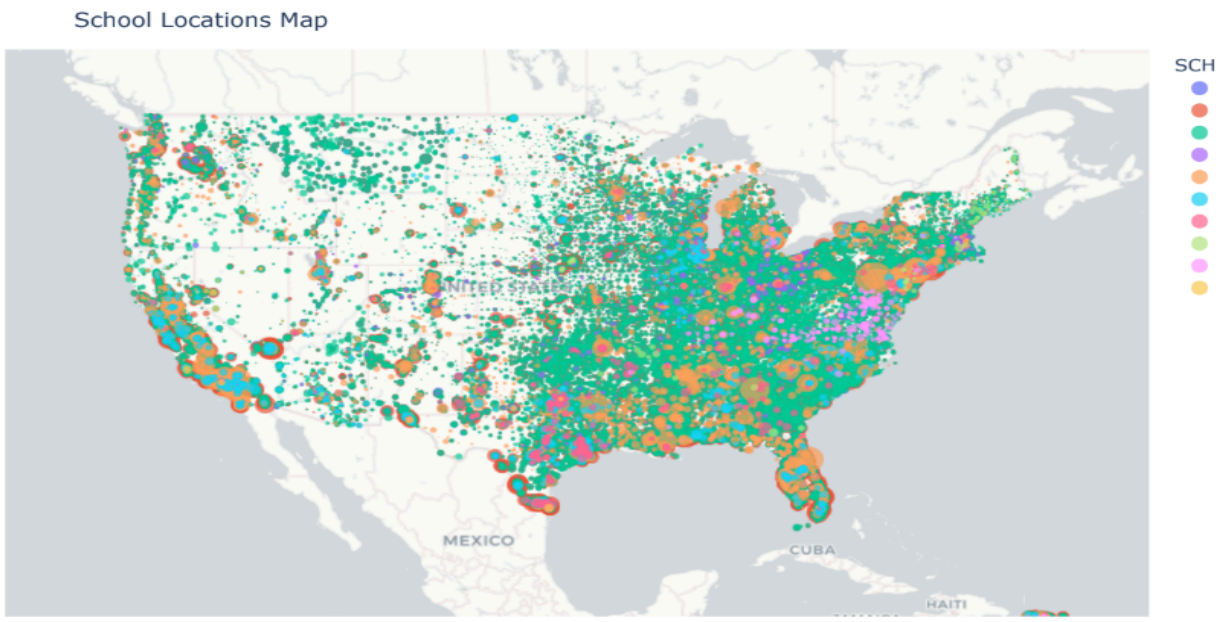
The above Python code uses Plotly Express to create a map with dots representing school locations, utilizing their latitude and longitude coordinates. The interactive map allows users to see school names and cities by hovering over the dots. The visual output is a dense scatter of blue points across the U.S., showing the distribution of schools, with a higher concentration of dots indicating more schools in that region. The map is styled with 'open-street-map' and saved as an HTML file for web viewing.



The visualization is a scatter plot map representing the geospatial distribution of schools across the United States. Each blue dot on the map corresponds to a school's location, with the latitude and longitude dictating the placement of the dots. The concentration of dots is higher in more populated areas, indicating a greater number of schools, which can be observed in regions such as the Northeast, the West Coast, and urban areas throughout the country. The map provides an at-a-glance view of how schools are spread out geographically, highlighting areas of both high and low school density.

```
df_selected = df[['LATCOD', 'LONCOD', 'SCH_NAME', 'SCHOOL_LEVEL', 'TOTFRL']]

fig = px.scatter_mapbox(df_selected,
                        lat='LATCOD',
                        lon='LONCOD',
                        hover_name='SCH_NAME',
                        hover_data={'SCHOOL_LEVEL': True, 'TOTFRL': True},  # Additional info on hover
                        color='SCHOOL_LEVEL',  # Color points based on school level
                        size='TOTFRL',  # Size points based on total enrollment
                        opacity=0.7,
                        mapbox_style='carto-positron',
                        zoom=3,
                        center={'lat': 37.0902, 'lon': -95.7129},  # Centered on the US
                        title='School Locations Map'
                       )

# Update layout
fig.update_layout(margin={'r':0,'t':40,'l':0,'b':0})

# Show the plot
fig.show()
fig.write_html("school_locations_map.html")
```

The code here uses Plotly Express to create an interactive map titled "School Locations Map" that visualizes schools with their respective levels and enrollment data. The map displays points sized by the total free lunch data ('TOTFRL'), indicating enrollment, and colored by school level to differentiate between elementary, middle, and high schools. Additional details are revealed when hovering over the points. The map is set with a semi-transparent style, centered on the U.S., and the layout is adjusted for optimal viewing. It is then rendered in the browser and saved as an HTML file.



This map displays schools across the U.S., with dots colored by school level and sized by the number of students receiving free lunches. Larger, more colorful clusters typically indicate denser, more diverse, or lower-income student populations. The map is interactive, allowing for more detailed exploration of individual school data.

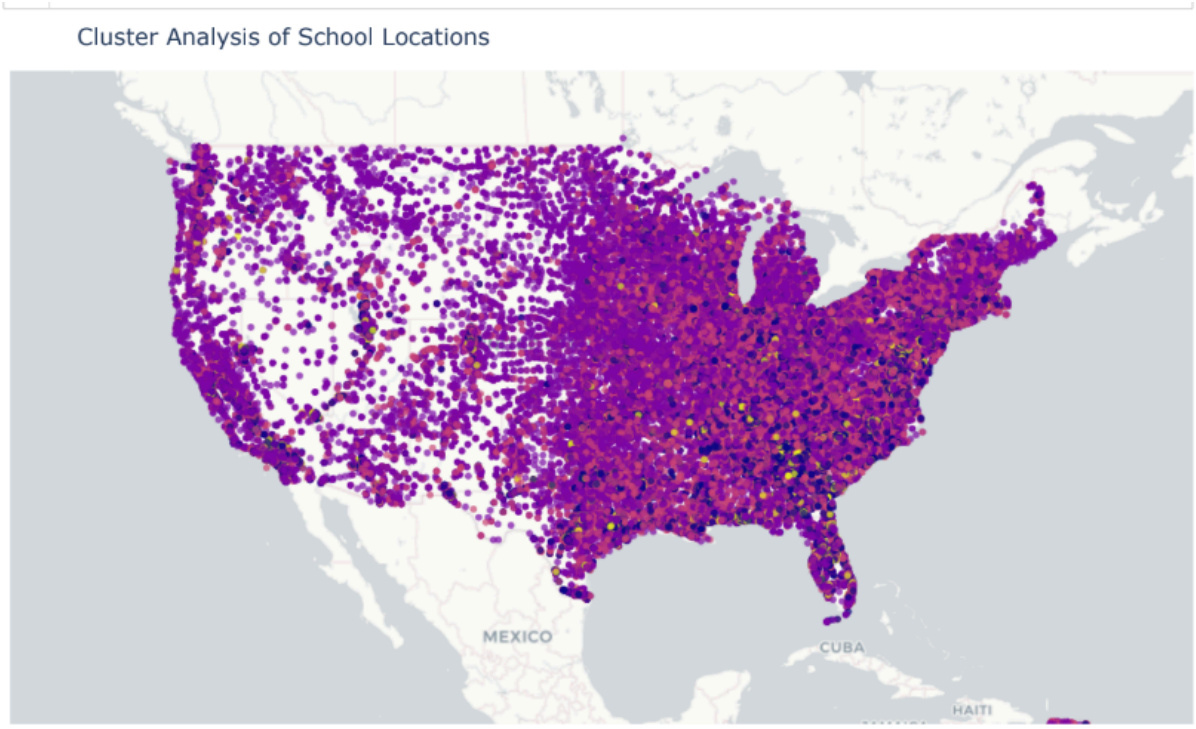
```
:
1  from sklearn.cluster import KMeans  # Import KMeans class from sklearn.cluster module
2
3  # Perform one-hot encoding for categorical variables
4  df_encoded = pd.get_dummies(df, columns=['SCHOOL_LEVEL'])
5
6  # Select relevant features for clustering
7  features = ['LATCOD', 'LONCOD', 'TOTAL'] + [col for col in df_encoded.columns if 'SCHOOL_LEVEL' in col]
8
9  # Initialize K-means model
10 kmeans = KMeans(n_clusters=5, random_state=0)  # Adjust the number of clusters as needed
11
12 # Fit the model
13 kmeans.fit(df_encoded[features])
14
15 # Add cluster labels to the DataFrame
16 df['Cluster'] = kmeans.labels_
```

```
C:\Users\prabh\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning
```

This code performs a K-Means cluster analysis on school data. It encodes the 'SCHOOL_LEVEL' as numerical data, selects geographic and school level features for clustering, initializes a K-Means model with 5 clusters, fits the model to the data, and assigns a cluster label to each school in the DataFrame.

```python
import plotly.express as px

# Plot clusters on a map
fig = px.scatter_mapbox(df,
                        lat='LATCOD',
                        lon='LONCOD',
                        hover_name='SCH_NAME',
                        hover_data={'SCHOOL_LEVEL': True, 'TOTAL': True, 'Cluster': True},
                        color='Cluster',
                        opacity=0.7,
                        mapbox_style='carto-positron',
                        zoom=3,
                        center={'lat': 37.0902, 'lon': -95.7129},
                        title='Cluster Analysis of School Locations'
                       )

# Update layout
fig.update_layout(margin={'r':0,'t':40,'l':0,'b':0})

# Show the plot
fig.show()
fig.write_html("cluster_analysis_school_locations_map.html")
```

This code creates a map visualization using Plotly Express, plotting data points based on school latitude ('LATCOD') and longitude ('LONCOD'). Each point is interactive, showing the school's name, level, total enrollment, and cluster assignment on hover. The points are colored by their cluster group to visually differentiate the clusters. The map is styled with 'carto-positron', centered on a specific latitude and longitude, and given a title 'Cluster Analysis of School Locations'. The layout margins are adjusted for presentation, and the map is displayed and saved as an HTML file for web use.



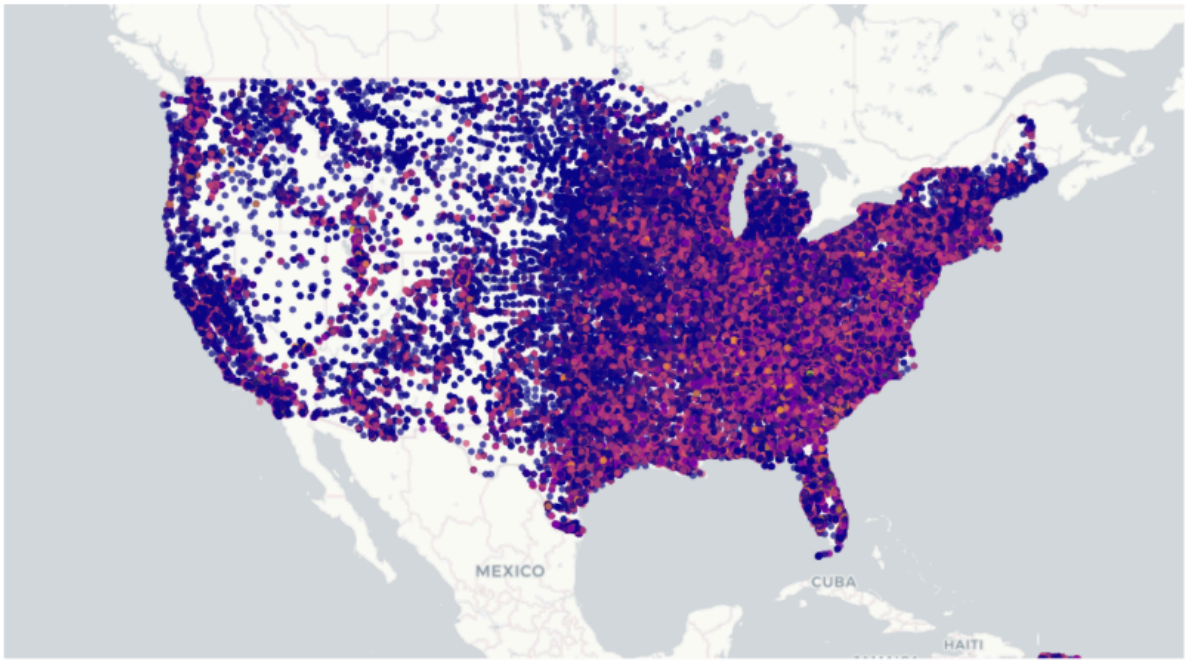
The map visualization shows the result of a cluster analysis of school locations across the U.S. The schools are represented by points, with colors indicating the cluster each school belongs to. This clustering may reflect similarities in geography, school level, and enrollment size. The visual pattern of clustering can help identify regions with schools that share certain characteristics.

```python
import pandas as pd
import plotly.express as px
from sklearn.cluster import KMeans

# Load your dataset
# Assuming df is your DataFrame

# Preprocess data (handle missing values, encode categorical variables, etc.)

# Select the attribute for clustering
attribute = 'TOTAL'  # Total enrollment

# Reshape the attribute data for clustering (reshape for KMeans input)
X = df[[attribute]].values

# Initialize K-means model
kmeans = KMeans(n_clusters=5, random_state=0)  # Adjust the number of clusters as needed

# Fit the model
kmeans.fit(X)

# Add cluster labels to the DataFrame
df['Cluster'] = kmeans.labels_

# Visualize clusters on a map
fig = px.scatter_mapbox(df,
                        lat='LATCOD',
                        lon='LONCOD',
                        hover_name='SCH_NAME',
                        hover_data={attribute: True, 'Cluster': True},
                        color='Cluster',
                        opacity=0.7,
                        mapbox_style='carto-positron',
                        zoom=3,
                        center={'lat': 37.0902, 'lon': -95.7129},
                        title=f'Cluster Analysis of School Locations based on {attribute}'
                       )

# Update layout
fig.update_layout(margin={'r':0,'t':40,'l':0,'b':0})

# Show the plot
fig.show()
fig.write_html("cluster_analysis_school_locations_based_on_total.html")
```

```
C:\Users\prabh\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning
```

This code combines Python's Pandas for data handling, scikit-learn for K-Means clustering, and Plotly Express for visualization. It selects 'TOTAL' as the clustering attribute, likely representing total enrollment, and performs K-Means clustering on the school data. The schools are assigned to five clusters based on this attribute. It then visualizes the clusters on a map with points colored according to their cluster label, allowing for geographic patterns in school size or enrollment to be discerned. The map is interactive, with tooltips for school names and additional data, centered on the U.S., and styled for clarity. After plotting, the map is saved as an HTML file for sharing or offline use.

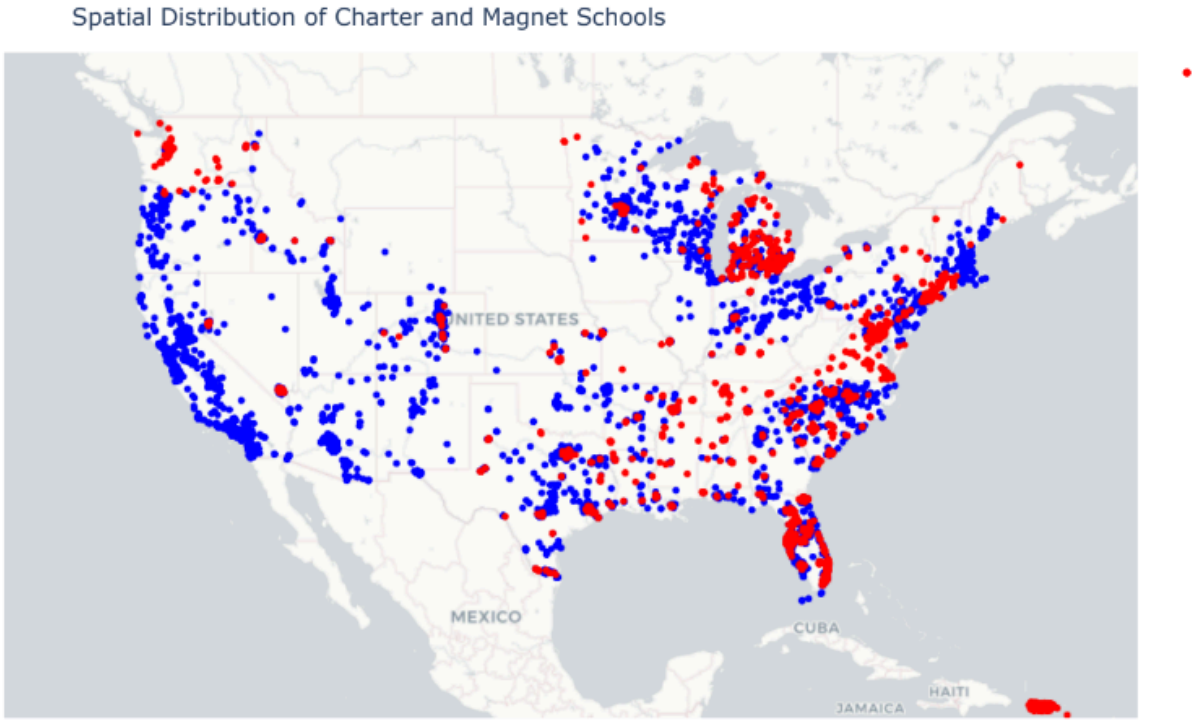
Cluster Analysis of School Locations based on TOTAL



This is a cluster analysis map of school locations in the U.S., categorized by total enrollment. Each dot represents a school, with the color indicating its cluster grouping. This visual differentiation likely reflects varying enrollment sizes, illustrating regional enrollment patterns and potentially highlighting disparities in school size across different areas. The map enables a quick visual assessment of how schools of similar sizes are distributed geographically.

```python
import pandas as pd
import plotly.express as px

# Load your dataset
# Assuming df is your DataFrame

# Filter data for charter and magnet schools
charter_schools = df[df['CHARTER_TEXT'] == 'Yes']
magnet_schools = df[df['MAGNET_TEXT'] == 'Yes']

# Create separate DataFrame for each school type
charter_df = charter_schools[['LATCOD', 'LONCOD', 'SCH_NAME']]
magnet_df = magnet_schools[['LATCOD', 'LONCOD', 'SCH_NAME']]

# Plot charter schools
fig = px.scatter_mapbox(charter_df,
                        lat='LATCOD',
                        lon='LONCOD',
                        hover_name='SCH_NAME',
                        color_discrete_sequence=['blue'],
                        zoom=3,
                        title='Spatial Distribution of Charter and Magnet Schools'
                       )

# Add magnet schools
fig.add_scattermapbox(lat=magnet_df['LATCOD'],
                      lon=magnet_df['LONCOD'],
                      hovertext=magnet_df['SCH_NAME'],
                      mode='markers',
                      marker=dict(color='red'),
                      name='Magnet Schools'
                     )

# Update layout
fig.update_layout(mapbox_style='carto-positron',
                  margin={'r':0,'t':40,'l':0,'b':0}
                 )

# Show the plot
fig.show()
fig.write_html("charter_and_magnet_schools_map.html")
```

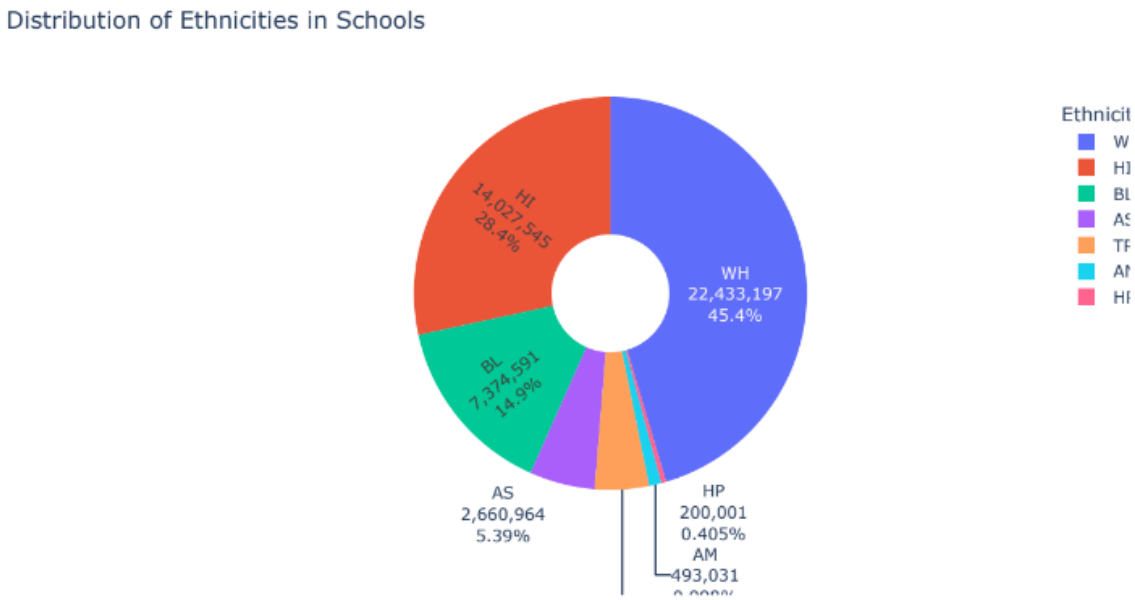
This code creates a map visualization that differentiates between charter and magnet schools. It filters the dataset to create two subsets for charter and magnet schools. Each subset is plotted on the map with distinct colors: charter schools in blue and magnet schools in red. The schools are represented as points on the map with their names displayed on hover. The final map, titled "Spatial Distribution of Charter and Magnet Schools," is interactive and saved as an HTML file.



The map illustrates the distribution of charter and magnet schools across the United States, with blue dots representing charter schools and red dots indicating magnet schools. The spread of the dots shows where these types of schools are concentrated, with denser clusters likely in urban areas. This visual comparison allows for an immediate understanding of the relative prevalence and geographical spread of these specialized schools.

```python
import plotly.graph_objects as go

# Assuming df is your DataFrame containing ethnicity data

# Calculate the total count for each ethnicity
ethnicity_counts = df[['AM', 'AS', 'BL', 'HP', 'HI', 'TR', 'WH']].sum()

# Create a DataFrame for ethnicity counts
ethnicity_df = pd.DataFrame({'Ethnicity': ethnicity_counts.index, 'Count': ethnicity_counts.values})

# Create a pie chart
fig = go.Figure(data=[go.Pie(labels=ethnicity_df['Ethnicity'],
                             values=ethnicity_df['Count'],
                             hoverinfo='label+percent+value',
                             hole=0.3,
                             textinfo='label+percent',
                             insidetextorientation='radial'
                            )
                      ]
                )

# Add custom hover effect to lift up the slice
fig.update_traces(hoverinfo='label+percent+value',
                  hovertemplate='%{label}: %{percent}<br>Total: %{value}',
                  textinfo='label+percent+value'
                 )

# Update layout
fig.update_layout(title='Distribution of Ethnicities in Schools',
                  showlegend=True,
                  legend_title_text='Ethnicity'
                 )

# Show the plot
fig.show()
fig.write_html("ethnicity_distribution_pie_chart.html")
```

This code creates a pie chart visualization of ethnicity distribution in schools. It calculates the total count for each ethnicity category within a DataFrame and then creates a new DataFrame for these counts. With Plotly's graph objects module, it constructs a pie chart with labels, values, and a hover template that displays the ethnicity, its percentage, and total count upon hovering. The pie chart features a donut-style hole in the middle and radial text orientation for the labels. The final layout includes a title and a legend. After displaying the chart, the code saves it as an HTML file for easy access and sharing.



The visualization displays a donut-shaped pie chart showing the ethnic composition within schools. Each slice represents a different ethnicity, with the size indicating the proportion of that ethnicity in the dataset. The largest segment is labeled 'WH', suggesting it represents the majority ethnicity. Smaller slices represent other ethnicities like 'HI', 'BL', 'AS', among others. Percentages and exact counts are provided, offering a quantitative view of the diversity in schools. The chart allows for an immediate visual comparison of the relative sizes of each ethnic group.