```
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

[5 rows x 79 columns]
```

Handling Missing Values

In [125]:

```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:
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
G12:          73991 missing values
UG:          92569 missing values
LSTREET2:          100144 missing values
AE:          100540 missing values
G13:          100579 missing values
TOTMENROL:          100722 missing values
TOTFENROL:          100722 missing values
```

checking if we have any duplicates in the data

In [126]:

```python
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

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'])
```

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)
```

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]:

```
# 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]:

```
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
```

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
```

In [140]:

```
# 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
```

In [142]:

```
# 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]:

```
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
```

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
```

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
```

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
```

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
```

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
```

In [157]:

```python
# 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]:

```python
# 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
```

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
```

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)
```

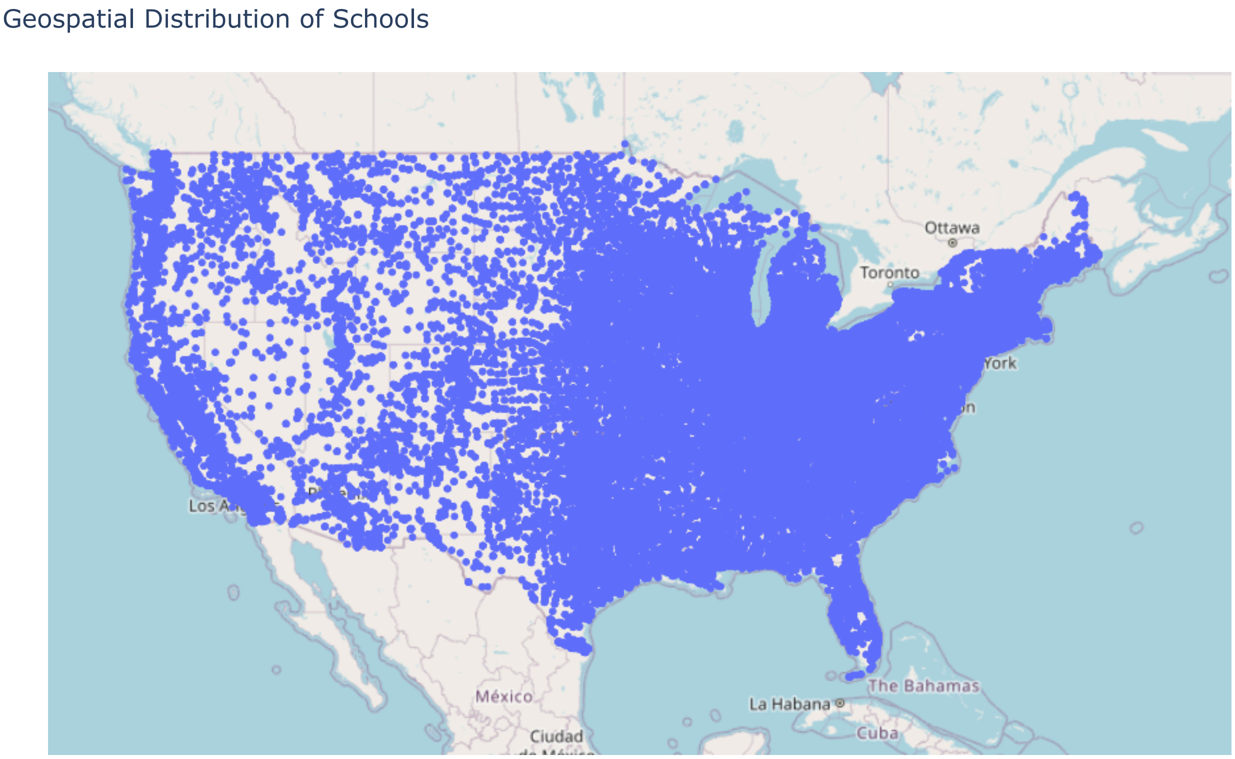
```
0
```

Visualizations

1. The basic scatter plot visualizing the locations of all the schools listed in the dataset

In [168]:

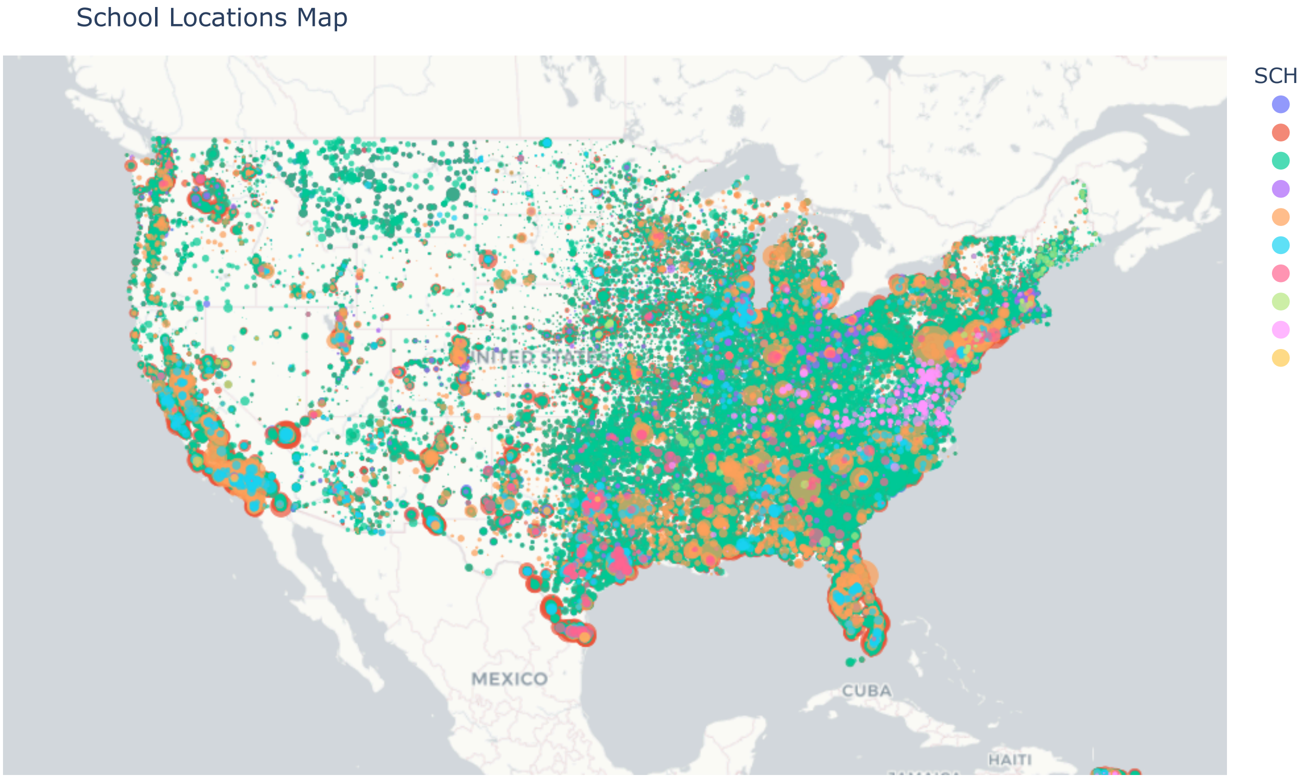
```
import plotly.express as px

# 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")
```



2. The following is an interactive map of school locations by Level and Enrollment

In [169]:

```
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")
```



3. Cluster Analysis of School Locations

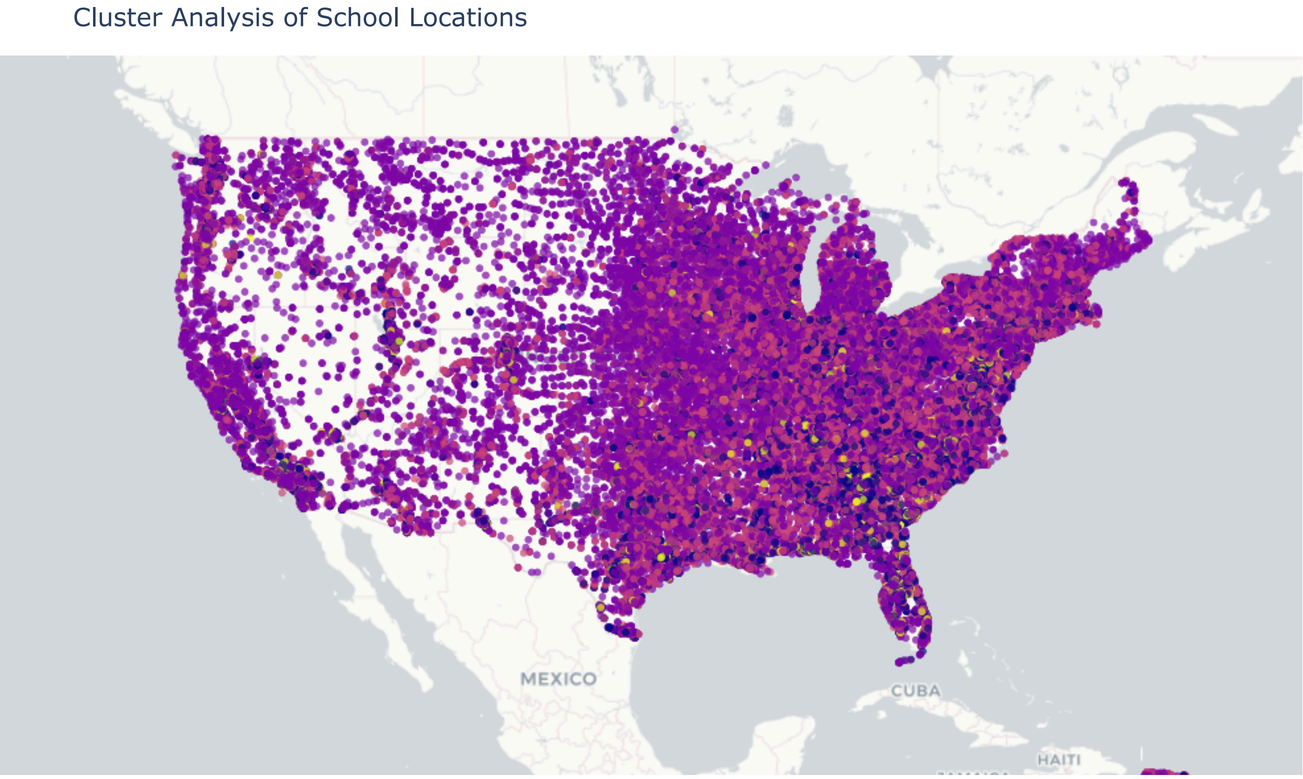
In [170]:

```python
from sklearn.cluster import KMeans  # Import KMeans class from sklearn.cluster module

# Perform one-hot encoding for categorical variables
df_encoded = pd.get_dummies(df, columns=['SCHOOL_LEVEL'])

# Select relevant features for clustering
features = ['LATCOD', 'LONCOD', 'TOTAL'] + [col for col in df_encoded.columns if 'SCHOOL_LEVEL' in col]

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

# Fit the model
kmeans.fit(df_encoded[features])

# Add cluster labels to the DataFrame
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
```

In [171]:

```
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")
```



Cluster Analysis of School Locations

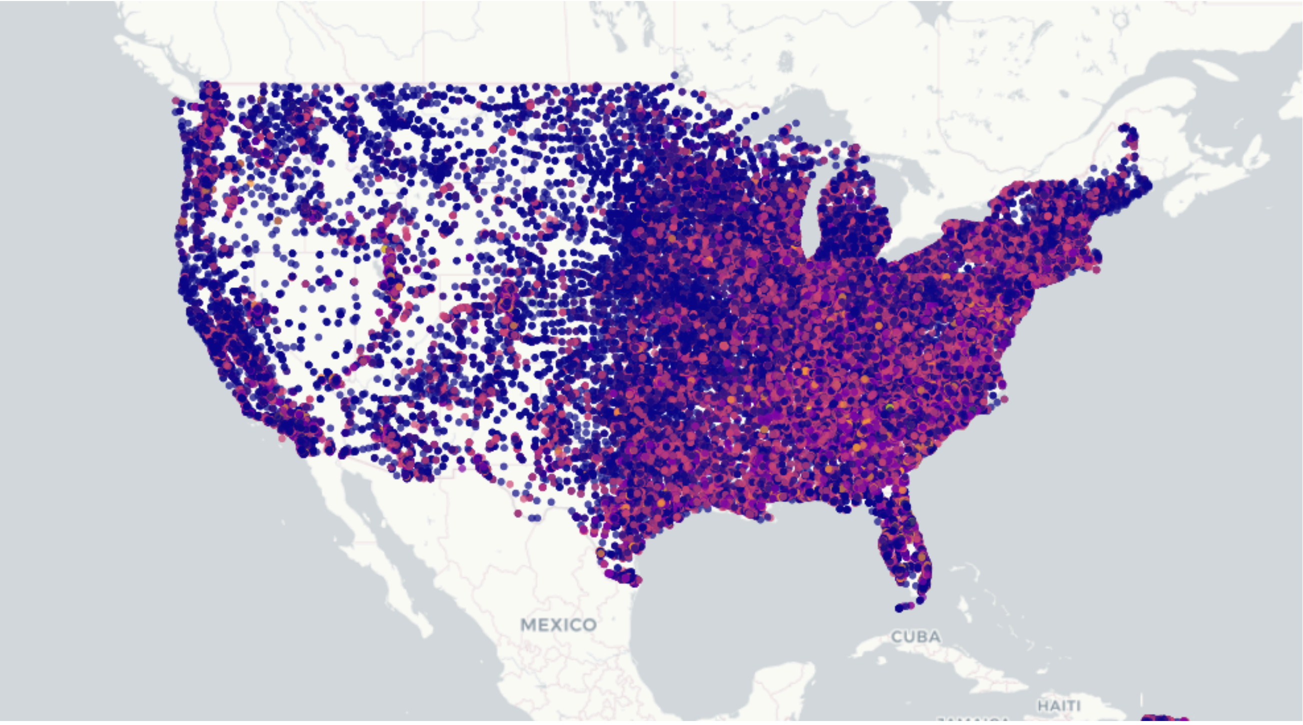
4. Cluster Analysis of School Locations based on Total Enrollment

In [172]:

```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
```

Cluster Analysis of School Locations based on TOTAL

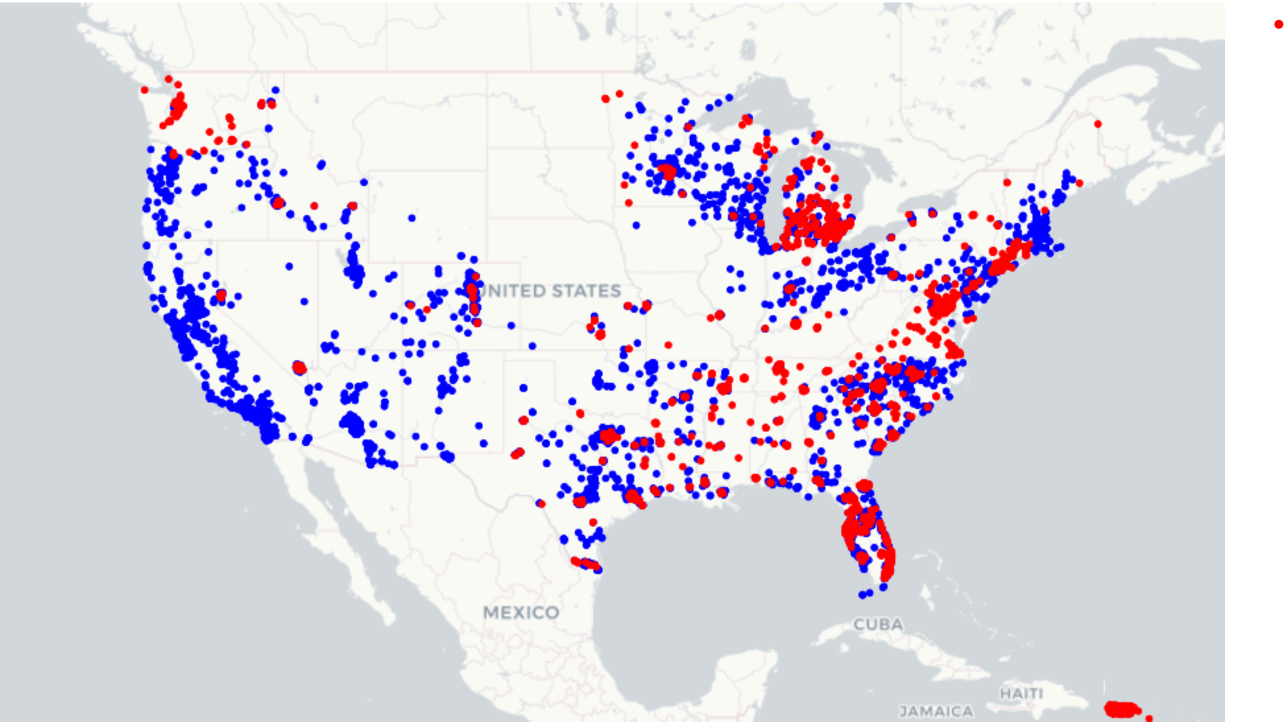


5. Spatial Distribution of Charter and Magnet Schools

In [173]:

```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")
```

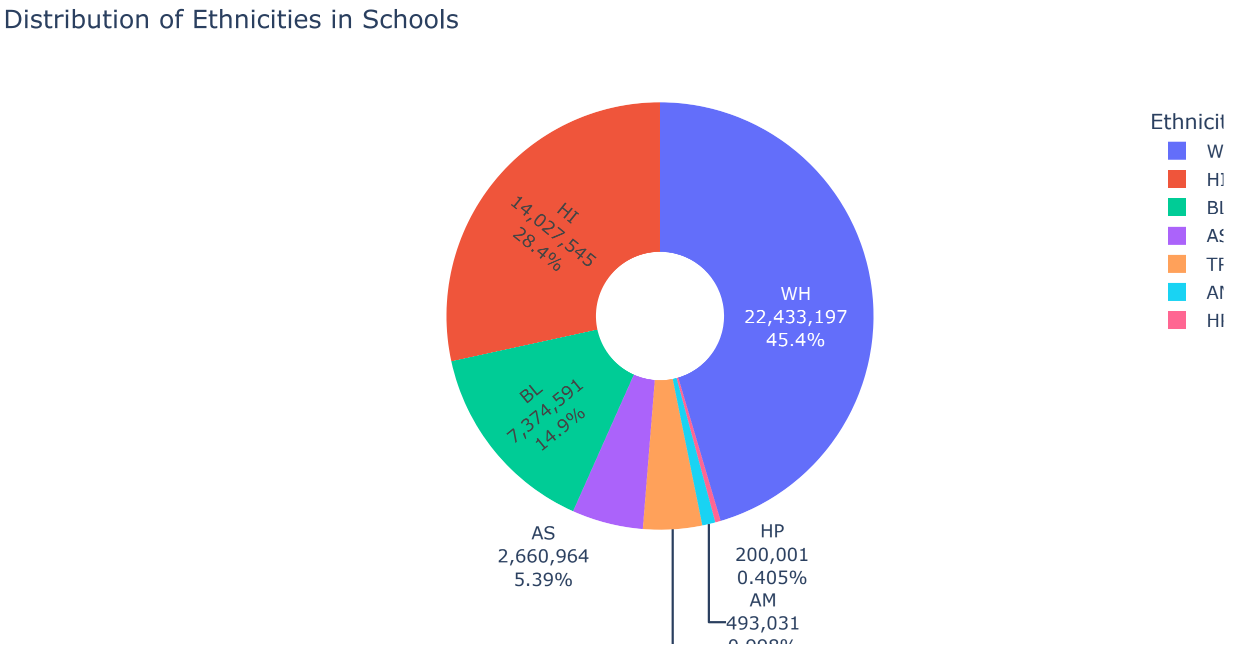
Spatial Distribution of Charter and Magnet Schools



6. Distribution of Ethnicities in Schools

In [174]:

```
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")
```


Distribution of Ethnicities in Schools
HI
WH
BL
AS
HP
AM
Ethnicit
W
HI
BI
AS
TI
AI
HI