Load Packages

In [1]:
```
!pip install geopy
```

```
Requirement already satisfied: geopy in c:\users\thinithi\anaconda3\lib\site-packa
ges (2.4.1)
Requirement already satisfied: geographiclib<3,>=1.52 in c:\users\thinithi\anacond
a3\lib\site-packages (from geopy) (2.0)
```

In [9]:
```
import time #format date time variabls
from geopy.exc import GeocoderTimedOut # timeout when geocoding to save time
from geopy.geocoders import Nominatim # convert addresses to geocodes
import plotly.express as px #Bar graph plotting
import pandas as pd # Load pandas
import os #check directory info
import matplotlib.pyplot as plt #load plotly for barchart
```

In [4]:
```
import os

# Get the current working directory if needed
current_directory = os.getcwd()
```

Data Exploration

In [5]:
```
file_path = 'dv355-VIC All Schools Enrolments 2023.csv'

# Read the CSV file into a DataFrame with a different encoding
df = pd.read_csv(file_path, encoding='ISO-8859-1')

# describe data
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2290 entries, 0 to 2289
Data columns (total 26 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Education_Sector            2290 non-null   object 
 1   Entity_Type                 2290 non-null   int64  
 2   School_No                   2290 non-null   int64  
 3   School_Name                 2290 non-null   object 
 4   School_Type                 2290 non-null   object 
 5   School_Status               2290 non-null   object 
 6   "Prep Total"                2290 non-null   float64
 7   "Year 1 Total"              2290 non-null   float64
 8   "Year 2 Total"              2290 non-null   float64
 9   "Year 3 Total"              2290 non-null   float64
 10  "Year 4 Total"              2290 non-null   float64
 11  "Year 5 Total"              2290 non-null   float64
 12  "Year 6 Total"              2290 non-null   float64
 13  "Primary Ungraded Total"    2290 non-null   float64
 14  "Primary Total"             2290 non-null   float64
 15  "Year 7 Total"              2290 non-null   float64
 16  "Year 8 Total"              2290 non-null   float64
 17  "Year 9 Total"              2290 non-null   float64
 18  "Year 10 Total"             2290 non-null   float64
 19  "Year 11 Total"             2290 non-null   float64
 20  "Year 12 Total"             2290 non-null   float64
 21  "Secondary Ungraded Total"  2290 non-null   float64
 22  "Secondary Total"           2290 non-null   float64
 23  "Grand Total"               2290 non-null   float64
 24  Year                        2290 non-null   int64  
 25  CENSUS_TYPE                 2290 non-null   object 
dtypes: float64(18), int64(3), object(5)
memory usage: 465.3+ KB
None
```

In [6]:

```
# Check for nulls in columns
df.isnull().sum()
```

Out[6]:
```
Education_Sector            0
Entity_Type                 0
School_No                   0
School_Name                 0
School_Type                 0
School_Status               0
"Prep Total"                0
"Year 1 Total"              0
"Year 2 Total"              0
"Year 3 Total"              0
"Year 4 Total"              0
"Year 5 Total"              0
"Year 6 Total"              0
"Primary Ungraded Total"    0
"Primary Total"             0
"Year 7 Total"              0
"Year 8 Total"              0
"Year 9 Total"              0
"Year 10 Total"             0
"Year 11 Total"             0
"Year 12 Total"             0
"Secondary Ungraded Total"  0
"Secondary Total"           0
"Grand Total"               0
Year                        0
CENSUS_TYPE                 0
dtype: int64
```

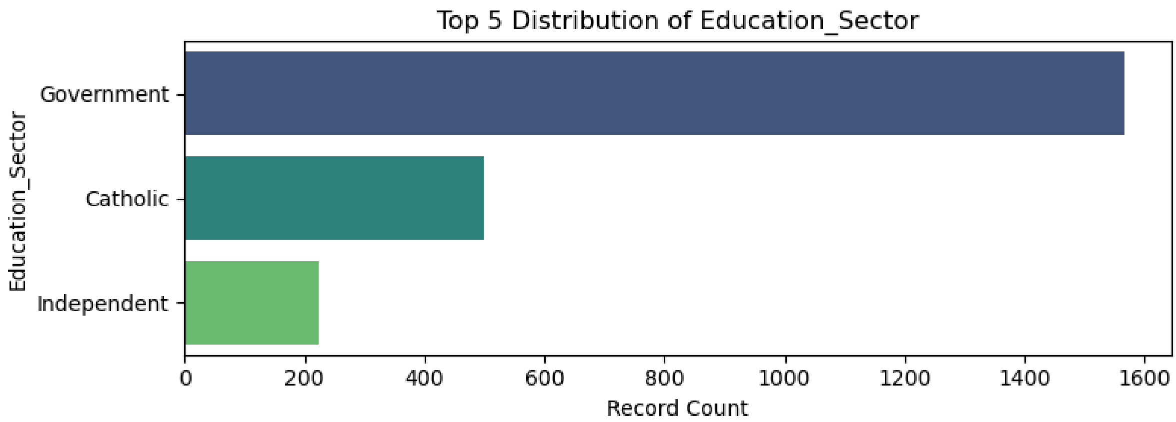
In [7]:
```
# describe data
print(df.describe())
```

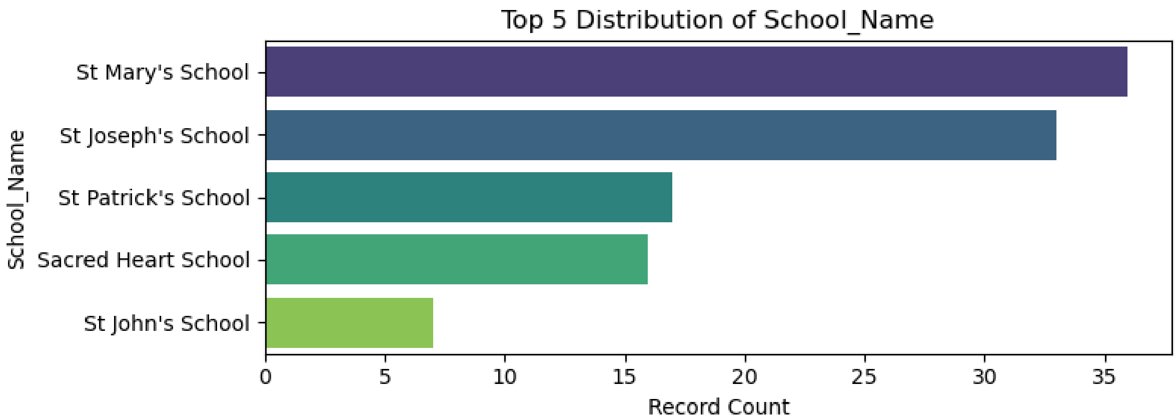
```
       Entity_Type    School_No  "Prep Total"  "Year 1 Total"  "Year 2 Total"  \
count  2290.000000  2290.000000   2290.000000     2290.000000     2290.000000
mean      1.316157  3531.783843     34.494236       35.045066       35.277773
std       0.465077  2518.851970     37.715966       38.869626       38.632585
min       1.000000     1.000000      0.000000        0.000000        0.000000
25%       1.000000  1554.000000      3.000000        3.000000        3.000000
50%       1.000000  2602.000000     24.000000       24.000000       25.000000
75%       2.000000  5239.750000     54.000000       54.000000       54.000000
max       2.000000  8917.000000    316.000000      355.000000      357.000000

       "Year 3 Total"  "Year 4 Total"  "Year 5 Total"  "Year 6 Total"  \
count     2290.000000     2290.000000     2290.000000     2290.000000
mean        35.431354       35.308734       35.812533       34.705153
std         38.808744       38.645317       39.338791       38.477236
min          0.000000        0.000000        0.000000        0.000000
25%          4.000000        3.125000        3.000000        3.000000
50%         25.000000       25.000000       26.000000       25.000000
75%         54.000000       54.000000       54.000000       52.000000
max        383.000000      394.000000      425.000000      385.000000

       "Primary Ungraded Total"  ...  "Year 7 Total"  "Year 8 Total"  \
count               2290.000000  ...     2290.000000     2290.000000
mean                   2.793100  ...       34.554891       34.626463
std                   18.821981  ...       77.200670       77.210002
min                    0.000000  ...        0.000000        0.000000
25%                    0.000000  ...        0.000000        0.000000
50%                    0.000000  ...        0.000000        0.000000
75%                    0.000000  ...        8.000000        8.000000
max                  305.200000  ...      600.000000      561.000000

       "Year 9 Total"  "Year 10 Total"  "Year 11 Total"  "Year 12 Total"  \
count     2290.000000      2290.000000      2290.000000      2290.000000
mean        34.373275        34.910480        33.142620        27.776638
std         76.201101        78.469754        79.575126        68.478003
min          0.000000         0.000000         0.000000         0.000000
25%          0.000000         0.000000         0.000000         0.000000
50%          0.000000         0.000000         0.000000         0.000000
75%          8.000000         5.000000         2.000000         0.000000
max        511.000000       600.000000       968.200000       776.100000

       "Secondary Ungraded Total"  "Secondary Total"  "Grand Total"    Year
count                 2290.000000        2290.000000    2290.000000  2290.0
mean                     2.481310         201.865677     450.733624  2023.0
std                     16.295483         437.426883     481.084346     0.0
min                      0.000000           0.000000       0.000000  2023.0
25%                      0.000000           0.000000     132.650000  2023.0
50%                      0.000000           0.000000     305.400000  2023.0
75%                      0.000000          98.400000     584.250000  2023.0
max                    253.000000        3317.000000    4610.000000  2023.0

[8 rows x 21 columns]
```

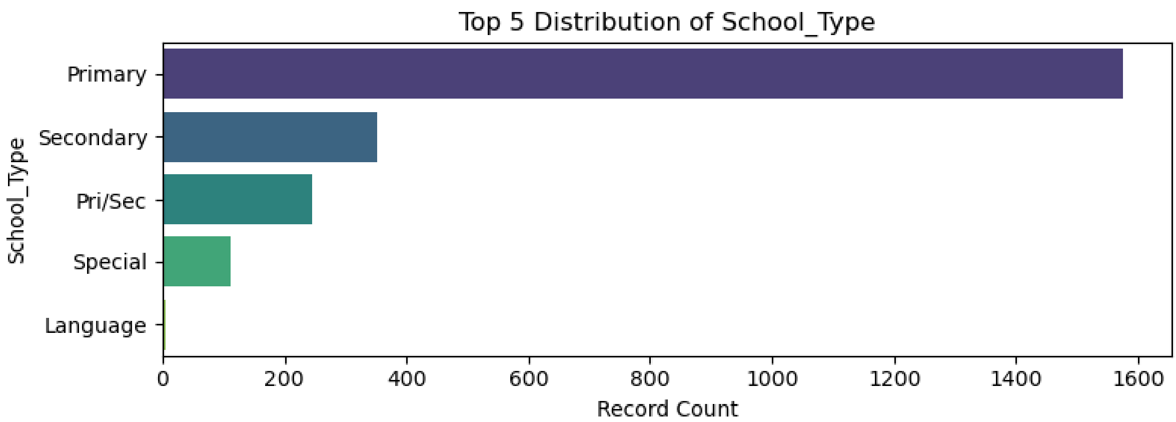
In [10]:

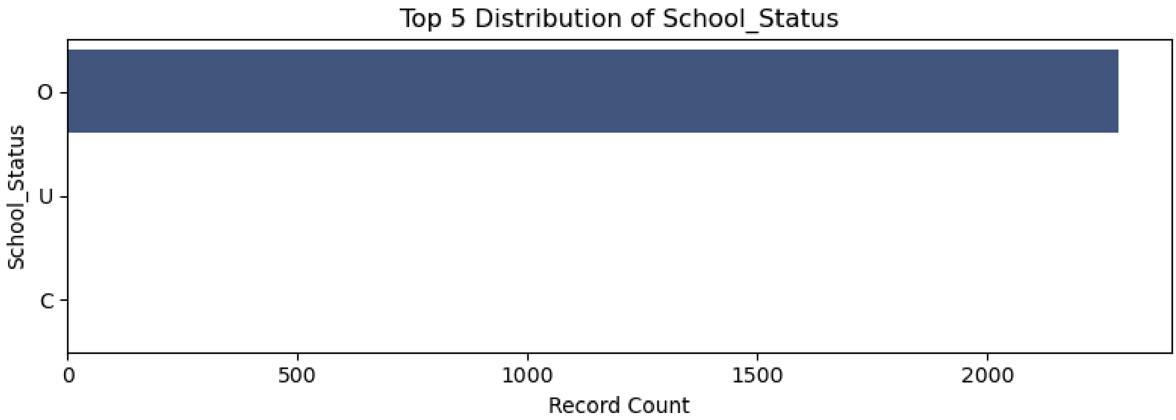
```python
import seaborn as sns

# Visualize the top 5 most frequent values for categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns

for column in categorical_columns:
    plt.figure(figsize=(8, 3))

    # Get the top 5 most frequent values in the column
    top_5_values = df[column].value_counts().nlargest(5)

    # Plot the top 5 values
```

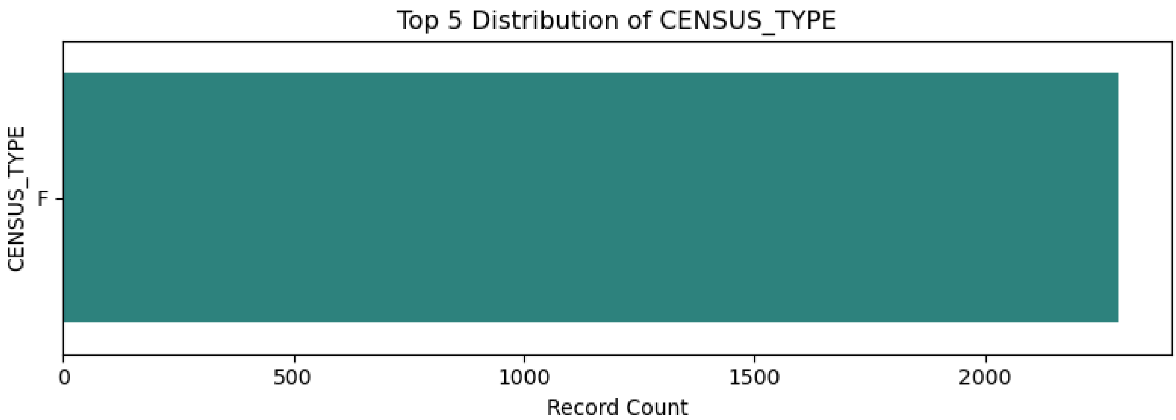
```
    sns.barplot(x=top_5_values.values, y=top_5_values.index, palette="viridis")

    # Add title and labels
    plt.title(f'Top 5 Distribution of {column}')
    plt.xlabel('Record Count')
    plt.ylabel(column)

    plt.tight_layout()
    plt.show()
```





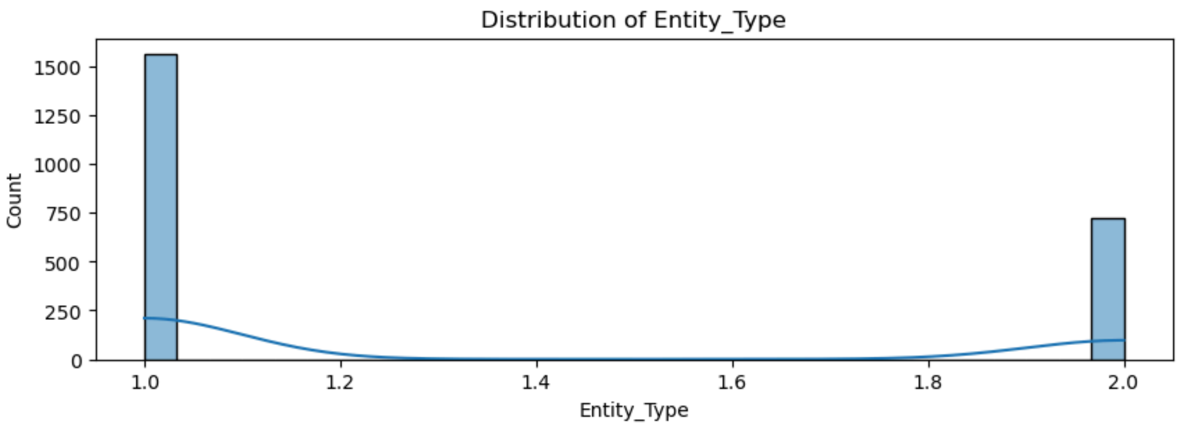


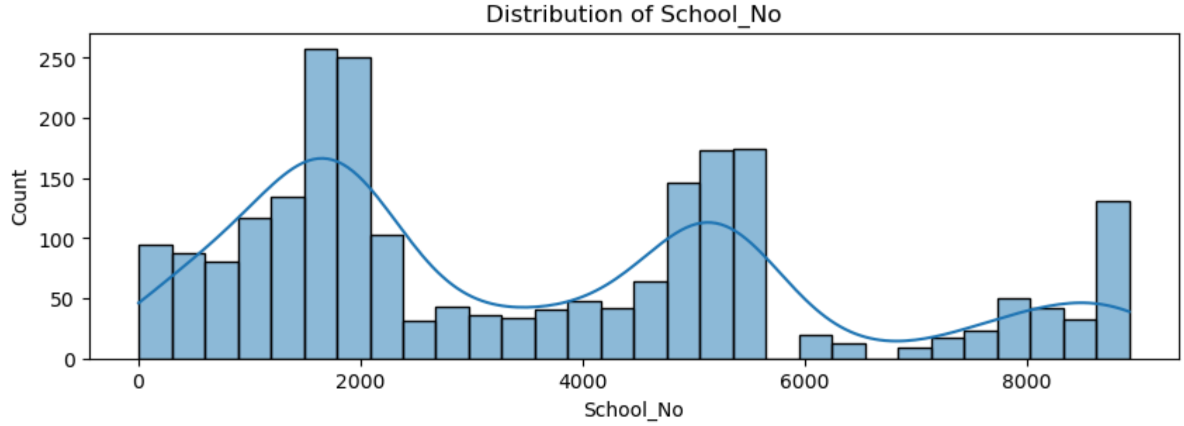


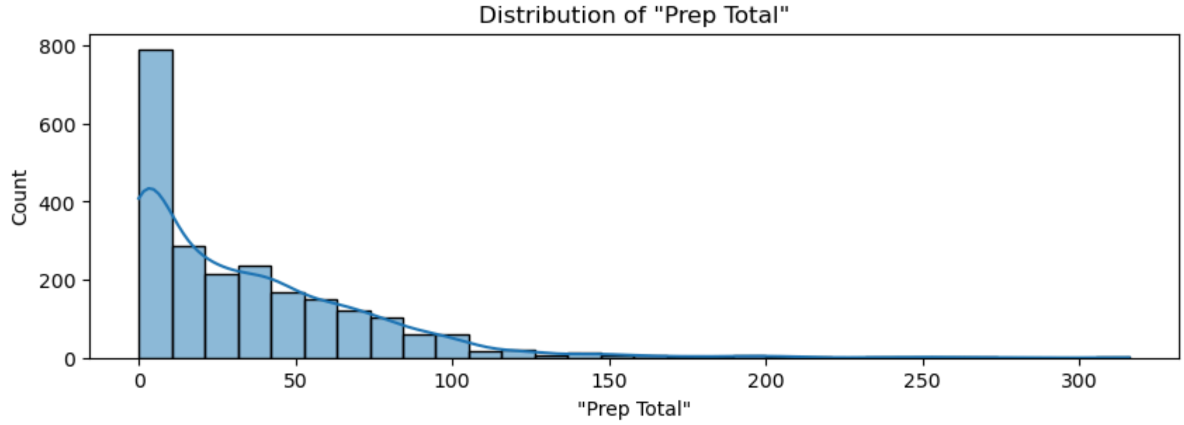


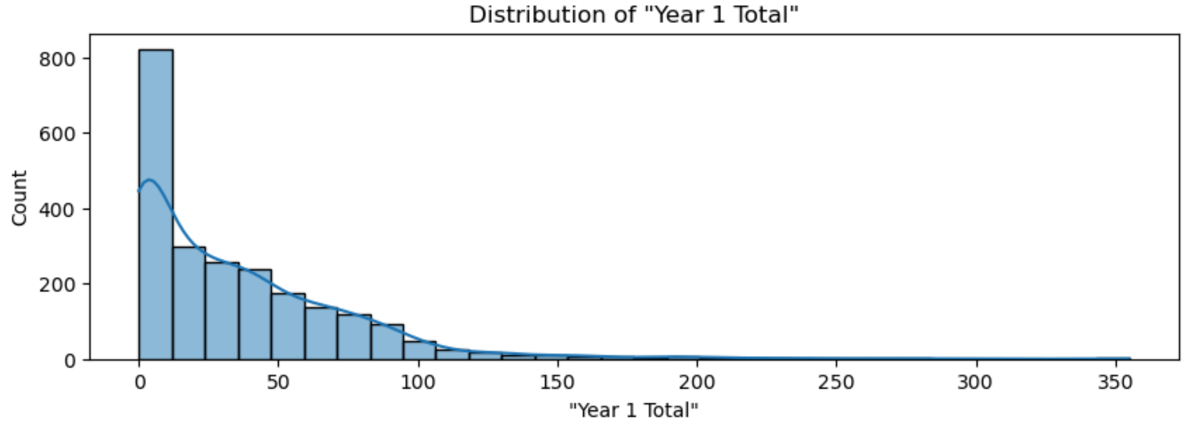
In [11]:

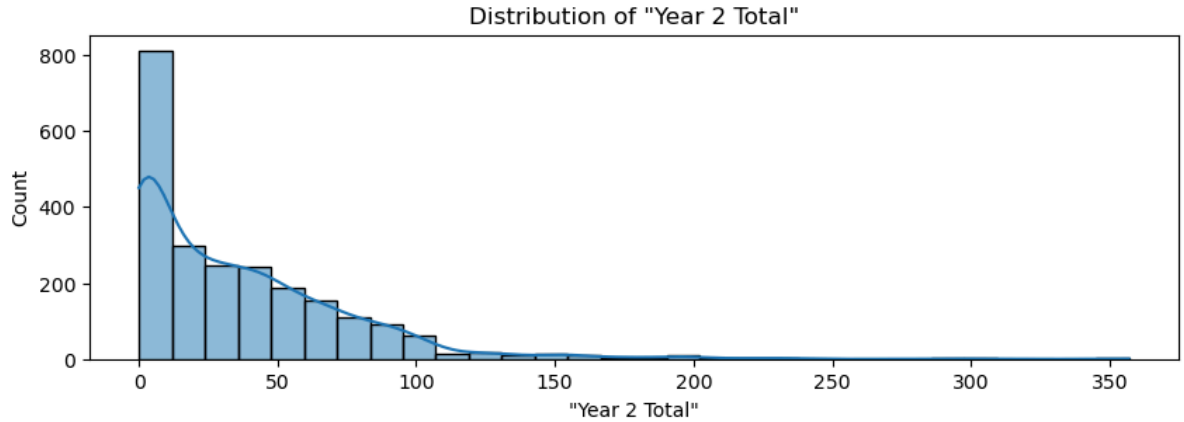
```
# Visualize the distribution of numerical columns
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns

for column in numerical_columns:
    plt.figure(figsize=(10, 3))
    sns.histplot(df[column], bins=30, kde=True)  # kde=True adds the Kernel Density
    plt.title(f'Distribution of {column}')
    plt.show()
```

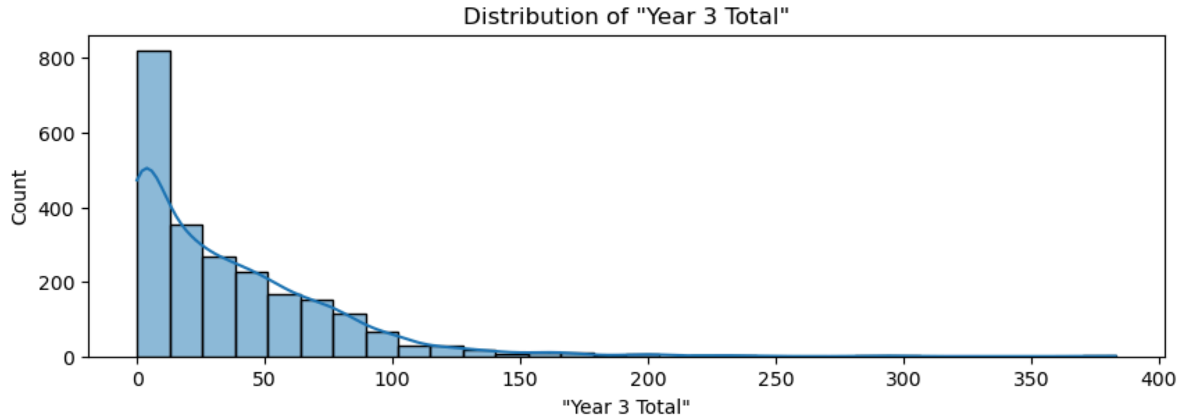


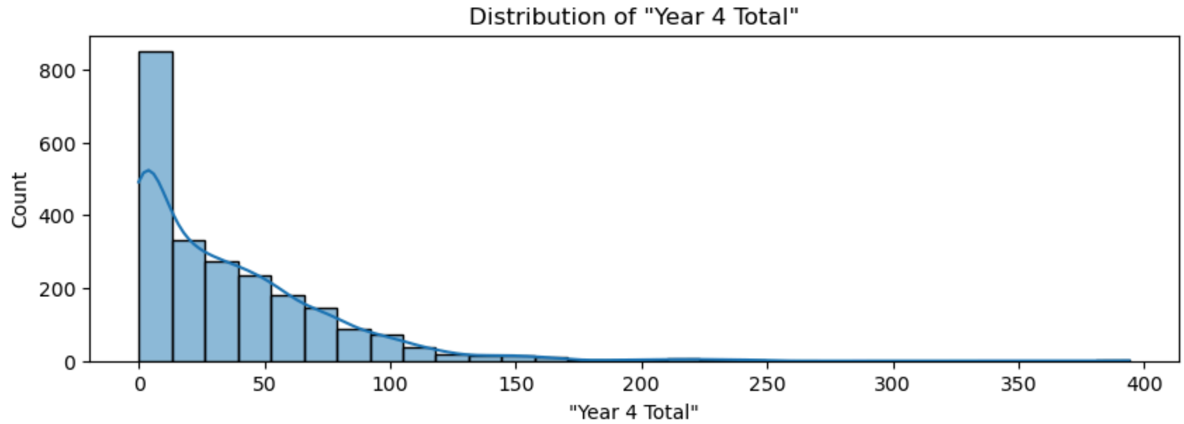

Distribution of School_No
Count
School_No

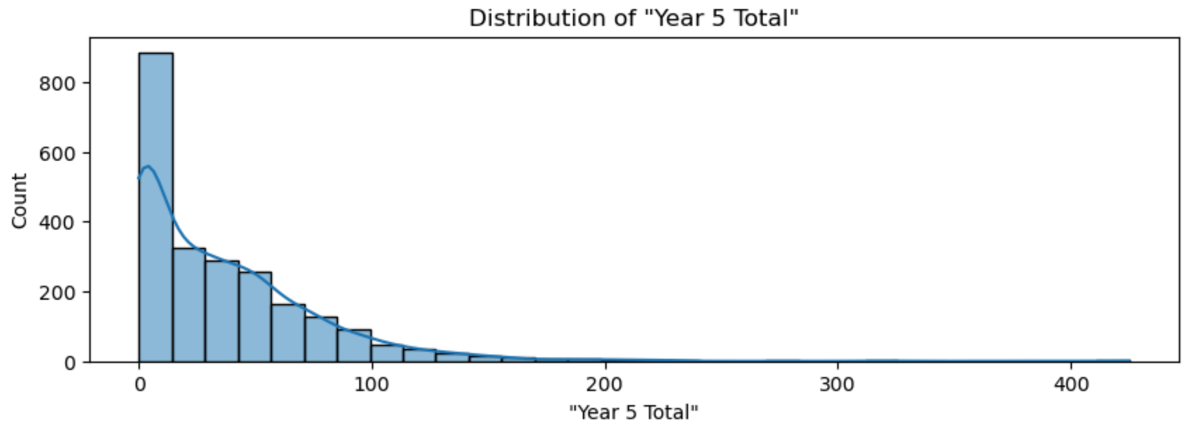

Distribution of "Prep Total"
Count
"Prep Total"

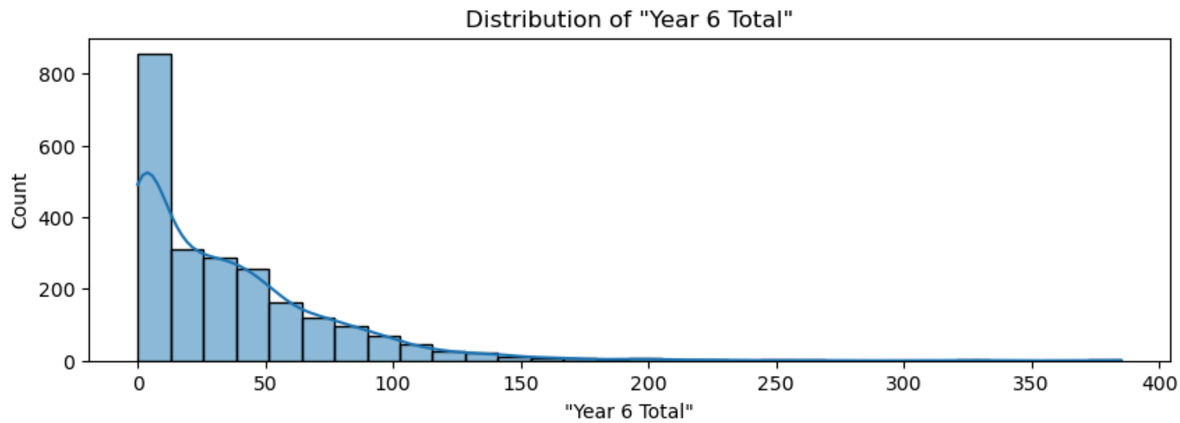

Distribution of "Year 1 Total"
Count
"Year 1 Total"

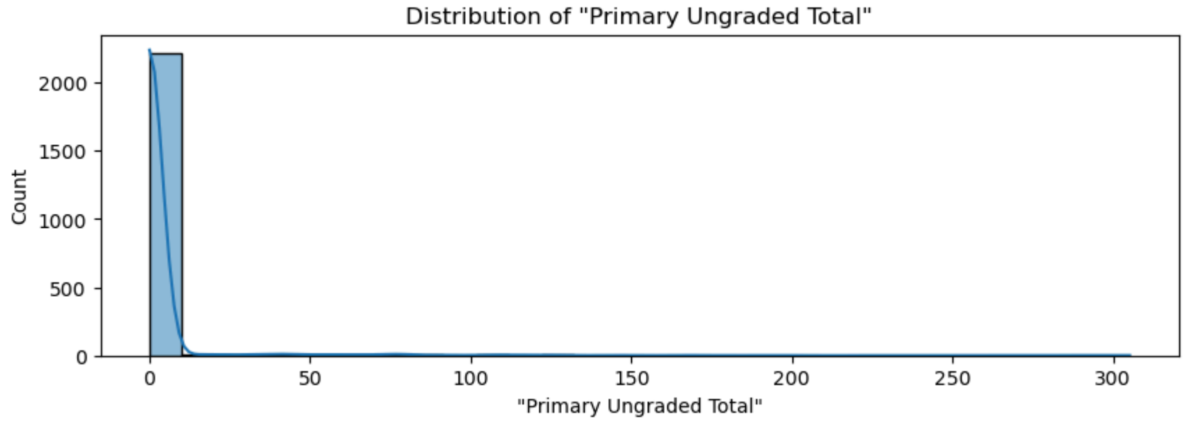

Distribution of "Year 2 Total"
Count
"Year 2 Total"

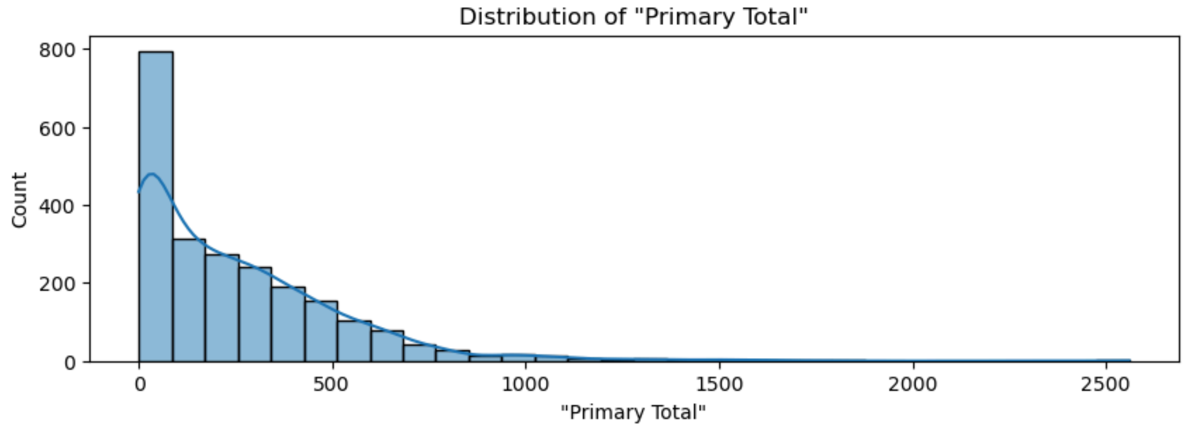

Distribution of "Year 3 Total"
Count
0
200
400
600
800
0
50
100
150
200
250
300
350
400
"Year 3 Total"

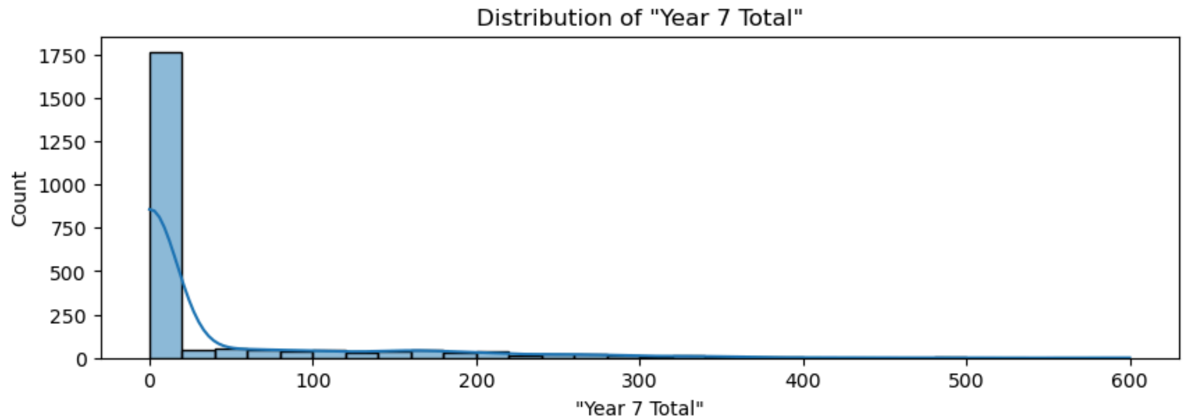

Distribution of "Year 4 Total"
Count
0
200
400
600
800
0
50
100
150
200
250
300
350
400
"Year 4 Total"

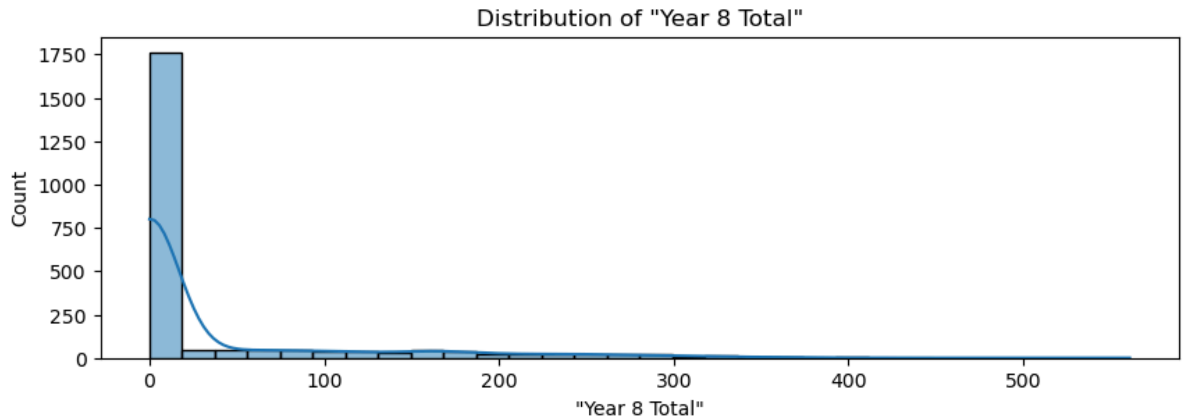

Distribution of "Year 5 Total"
Count
0
200
400
600
800
0
100
200
300
400
"Year 5 Total"

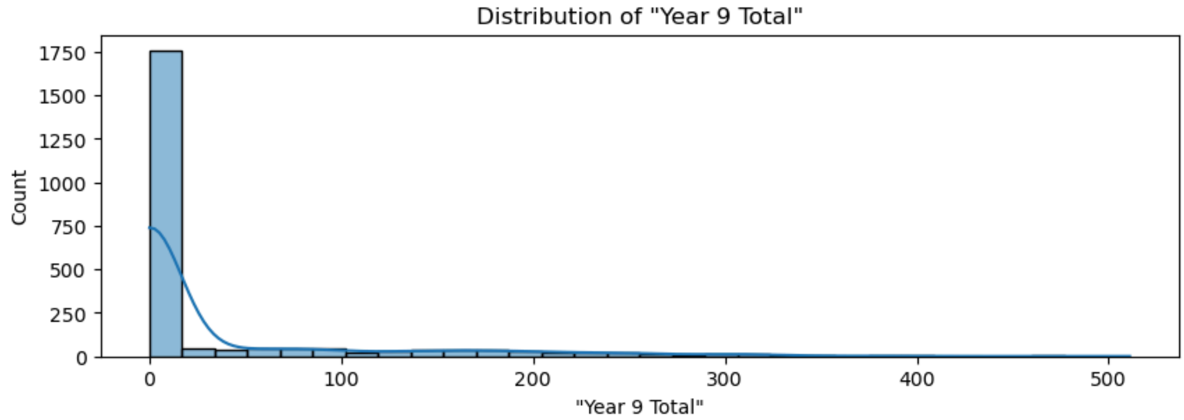

Distribution of "Year 6 Total"
Count
0
200
400
600
800
0
50
100
150
200
250
300
350
400
"Year 6 Total"

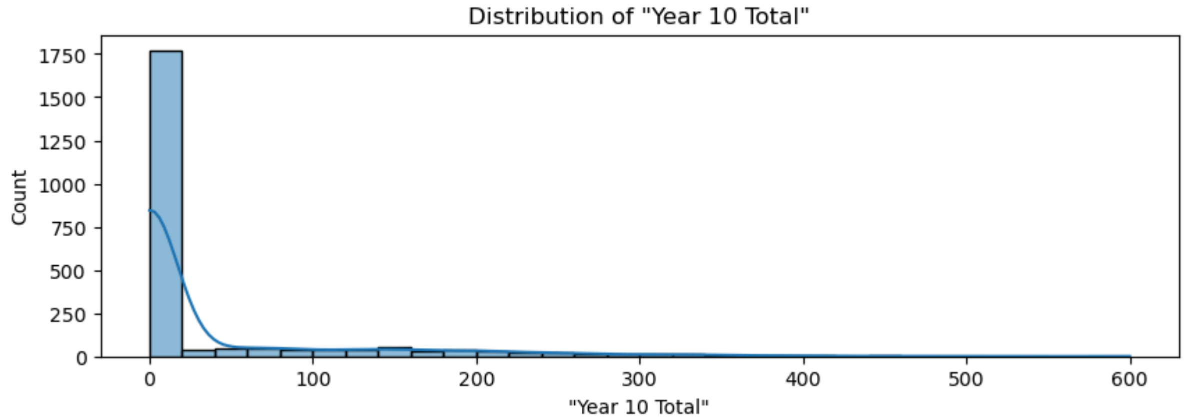

Distribution of "Primary Ungraded Total"
Count
2000
1500
1000
500
0
0
50
100
150
200
250
300
"Primary Ungraded Total"

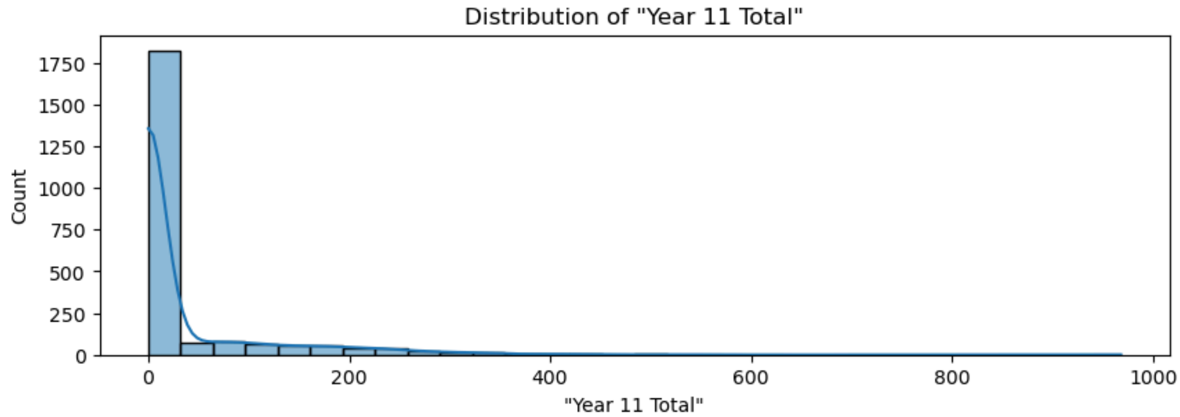

Distribution of "Primary Total"
Count
800
600
400
200
0
0
500
1000
1500
2000
2500
"Primary Total"

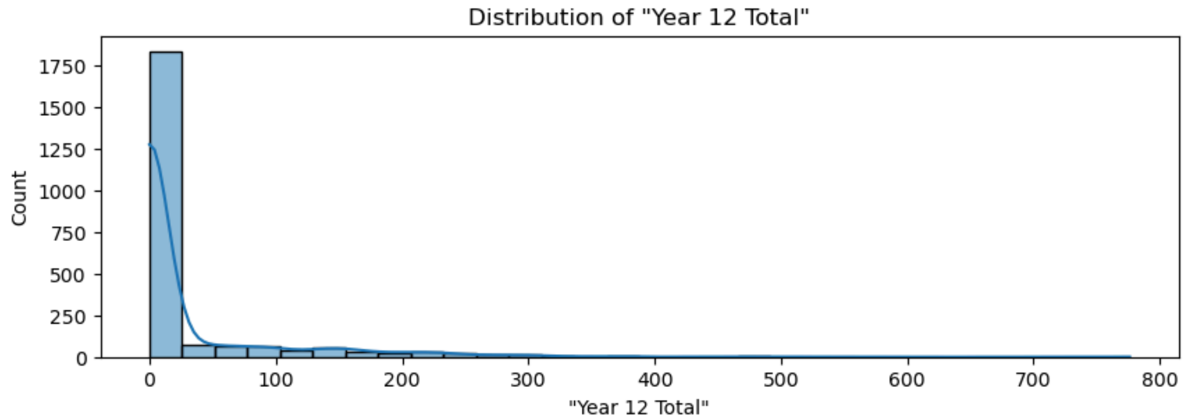

Distribution of "Year 7 Total"
Count
1750
1500
1250
1000
750
500
250
0
0
100
200
300
400
500
600
"Year 7 Total"

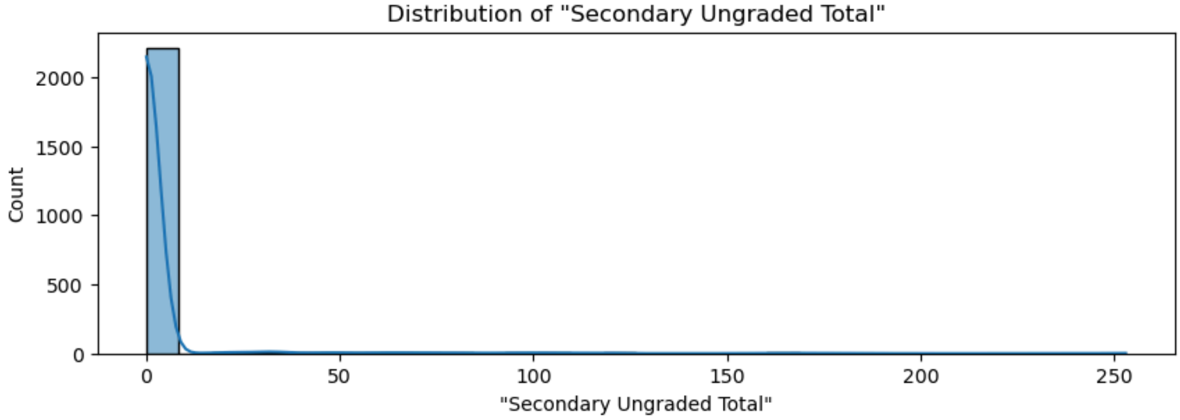

Distribution of "Year 8 Total"
Count
1750
1500
1250
1000
750
500
250
0
0
100
200
300
400
500
"Year 8 Total"

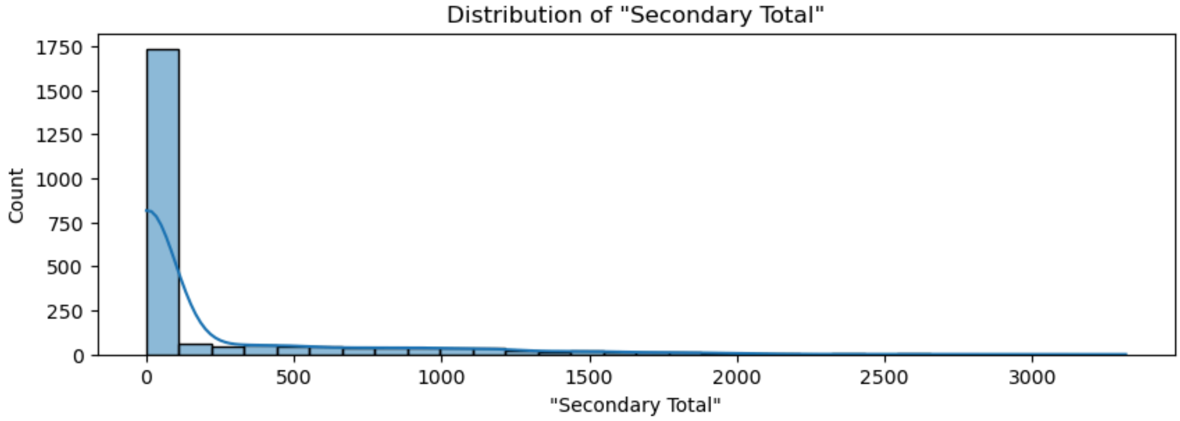

Distribution of "Year 9 Total"
Count
0
250
500
750
1000
1250
1500
1750
0
100
200
300
400
500
"Year 9 Total"

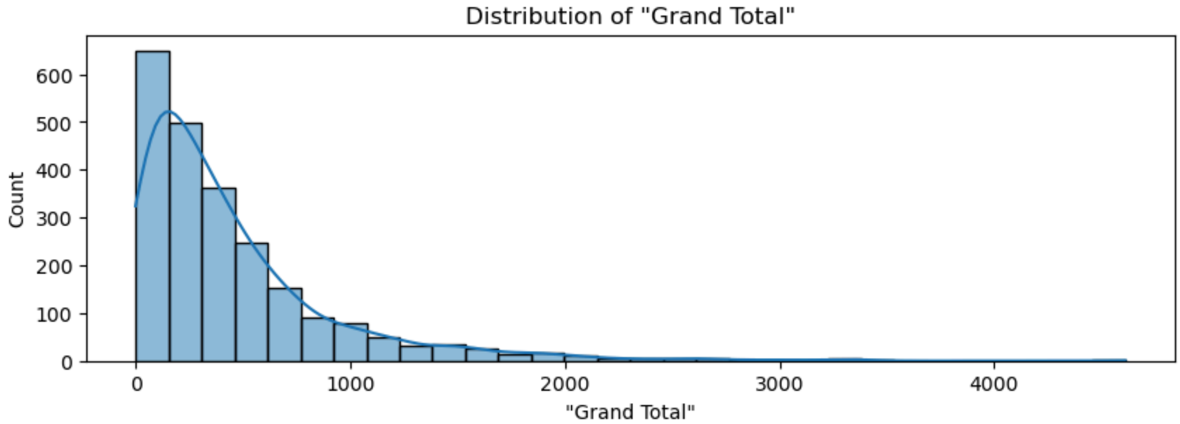

Distribution of "Year 10 Total"
Count
0
250
500
750
1000
1250
1500
1750
0
100
200
300
400
500
600
"Year 10 Total"

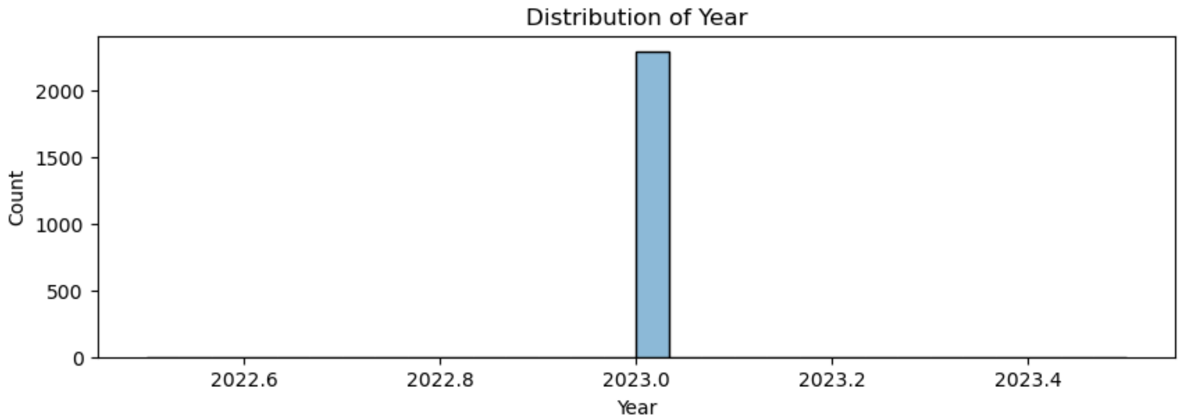

Distribution of "Year 11 Total"
Count
0
250
500
750
1000
1250
1500
1750
0
200
400
600
800
1000
"Year 11 Total"


Distribution of "Year 12 Total"
Count
0
250
500
750
1000
1250
1500
1750
0
100
200
300
400
500
600
700
800
"Year 12 Total"









Data Cleaning &Transformation

Student Enrollment Data

In [12]:
```
# Clean column names by stripping extra characters
df.columns = df.columns.str.strip().str.replace('"', '')

# Clean column names
df.columns = df.columns.str.strip().str.replace('"', '', regex=False)
```

In [13]:
```
# Convert the DataFrame from wide format to long format using the melt() function
long_df = df.melt(
    id_vars=['Education_Sector', 'Entity_Type', 'School_No', 'School_Name',
             'School_Type', 'School_Status', 'Year', 'CENSUS_TYPE'],
    value_vars=['Prep Total', 'Year 1 Total', 'Year 2 Total', 'Year 3 Total', 'Year
                'Year 6 Total', 'Primary Ungraded Total', 'Primary Total', 'Year 7
                'Year 9 Total', 'Year 10 Total', 'Year 11 Total', 'Year 12 Total',
                'Secondary Total', 'Grand Total'],
    var_name='Year_Level',
    value_name='Enrollment'
)

# Display the first few rows of the transformed DataFrame
print(long_df.head())
```

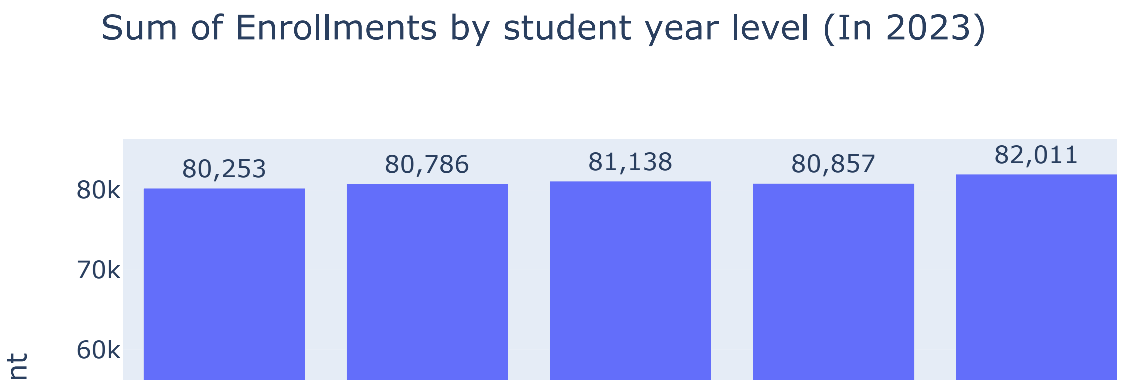
```
  Education_Sector  Entity_Type  School_No                     School_Name  \
0         Catholic            2         20                  Parade College
1         Catholic            2         25         Simonds Catholic College
2         Catholic            2         26      St Mary�s College Melbourne
3         Catholic            2         28  St Patrick's College Ballarat
4         Catholic            2         29             St Patrick's School

  School_Type School_Status  Year CENSUS_TYPE  Year_Level  Enrollment
0   Secondary             O  2023           F  Prep Total         0.0
1   Secondary             O  2023           F  Prep Total         0.0
2   Secondary             O  2023           F  Prep Total         0.0
3   Secondary             O  2023           F  Prep Total         0.0
4     Primary             O  2023           F  Prep Total        28.0
```

In [14]:
```
# Verify the column names
print(long_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41220 entries, 0 to 41219
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Education_Sector  41220 non-null  object
 1   Entity_Type       41220 non-null  int64
 2   School_No         41220 non-null  int64
 3   School_Name       41220 non-null  object
 4   School_Type       41220 non-null  object
 5   School_Status     41220 non-null  object
 6   Year              41220 non-null  int64
 7   CENSUS_TYPE       41220 non-null  object
 8   Year_Level        41220 non-null  object
 9   Enrollment        41220 non-null  float64
dtypes: float64(1), int64(3), object(6)
memory usage: 3.1+ MB
None
```

In [15]:
```
# Group by 'Year' and calculate the sum of 'Year 3 Total'
yearly_sum = long_df.groupby('Year_Level')['Enrollment'].sum().reset_index()

# Filter to include only 'Year' levels
yearly_sum = yearly_sum[yearly_sum['Year_Level'].str.contains('Year')]

# Define the order of categories
```

```
order = ['Year 1 Total', 'Year 2 Total', 'Year 3 Total', 'Year 4 Total', 'Year 5 To
         'Year 7 Total', 'Year 8 Total', 'Year 9 Total', 'Year 10 Total',
         'Year 11 Total', 'Year 12 Total']

# Convert 'Year_Level' to categorical with a specified order
yearly_sum['Year_Level'] = pd.Categorical(
    yearly_sum['Year_Level'], categories=order, ordered=True)

# Sort the DataFrame by 'Year_Level'
yearly_sum = yearly_sum.sort_values('Year_Level')

# Create the bar chart
fig = px.bar(yearly_sum, x='Year_Level', y='Enrollment',
             title='Sum of Enrollments by student year level (In 2023)',
             labels={'Year_Level': 'Year Level',
                     'Enrollment': 'Sum of Enrollment'},
             text='Enrollment')


# Update the text formatting to include commas
fig.update_traces(texttemplate='%{text:,.0f}', textposition='outside')


# Show the plot
fig.show()
```



School Bushfire risk Data

In [59]:
```
file_path = 'Website-BARR-2023-24-updated.xlsx'

# Read the Excel file into a DataFrame
df2 = pd.read_excel(file_path)
```

In [60]:
```
# Display the first few rows of the DataFrame
print(df2.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 868 entries, 0 to 867
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   row                      868 non-null    int64  
 1   SCHOOL_NO                868 non-null    int64  
 2   Fire Risk Category 2023-24 868 non-null    object 
 3   Facility Name            868 non-null    object 
 4   Education Sector         868 non-null    object 
 5   Facility Address         868 non-null    object 
 6   Town or Suburb           868 non-null    object 
 7   Local Government Area    868 non-null    object 
 8   Fire Weather District    867 non-null    object 
 9   LATITUDE                 868 non-null    float64
 10  LONGITUDE                868 non-null    float64
dtypes: float64(2), int64(2), object(7)
memory usage: 74.7+ KB
None
```

In [61]:
```
# describe data
print(df2.describe())
```

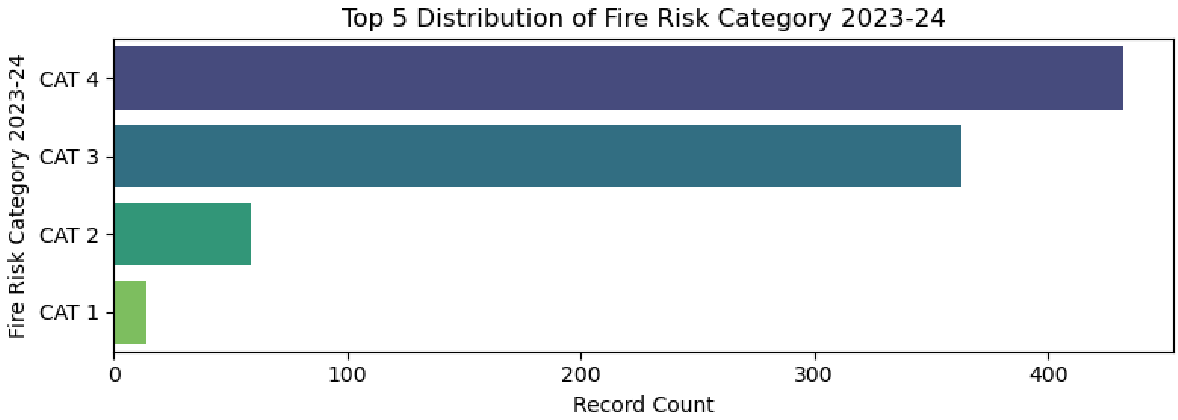
```
              row    SCHOOL_NO    LATITUDE   LONGITUDE
count  868.000000   868.000000  868.000000  868.000000
mean   434.500000  2590.906682  -37.364976  144.696656
std    250.714313  2438.589600    3.164138    5.164338
min      1.000000     0.000000  -38.701615   -0.805849
25%    217.750000   780.500000  -38.071129  144.057297
50%    434.500000  1866.000000  -37.695797  145.146556
75%    651.250000  3925.750000  -36.914017  145.618611
max    868.000000  8907.000000   52.941652  149.819539
```

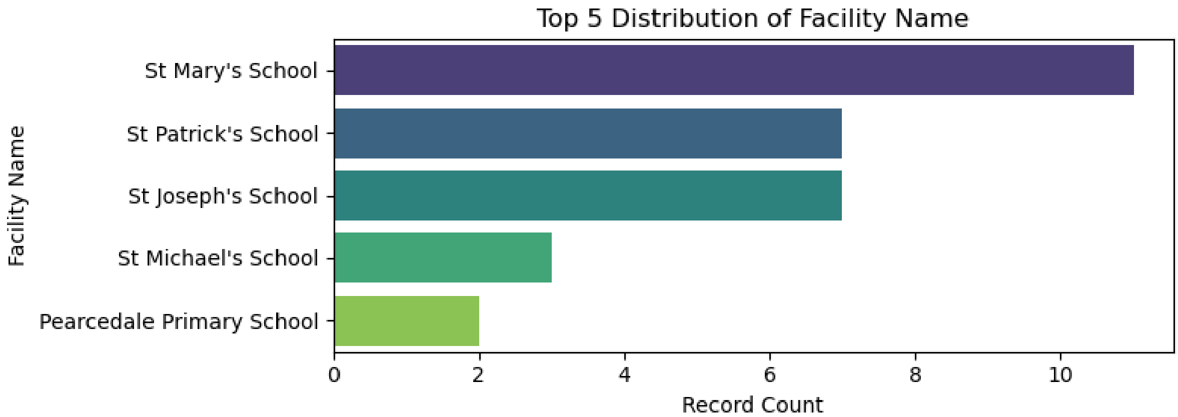
In [62]:
```
# Check for nulls in columns
df2.isnull().sum()
```

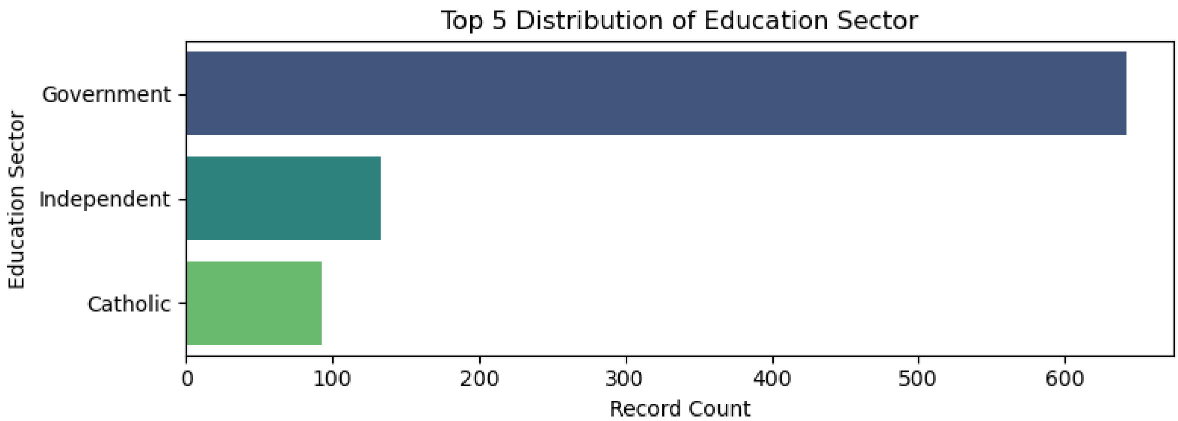
Out[62]:
```
row                           0
SCHOOL_NO                     0
Fire Risk Category 2023-24    0
Facility Name                 0
Education Sector              0
Facility Address              0
Town or Suburb                0
Local Government Area         0
Fire Weather District         1
LATITUDE                      0
LONGITUDE                     0
dtype: int64
```

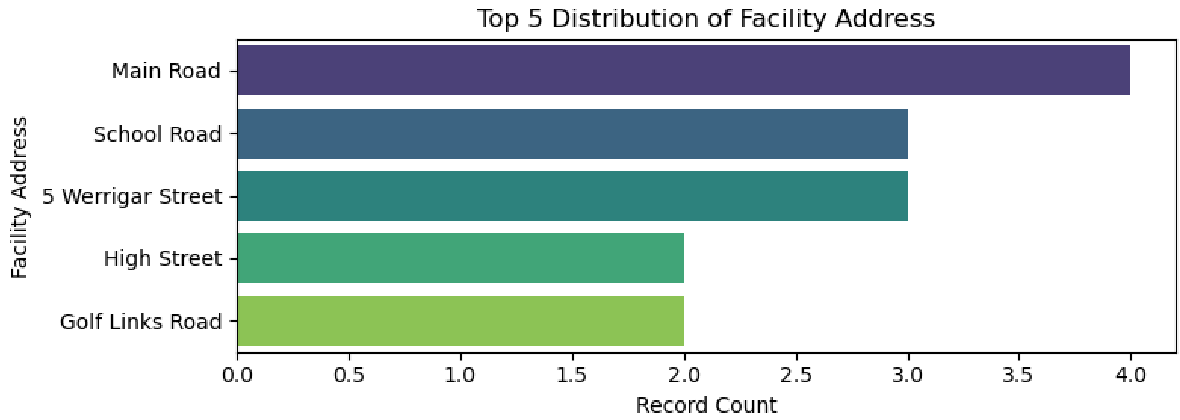
In [63]:
```
# Visualize the top 5 most frequent values for categorical columns
categorical_columns = df2.select_dtypes(include=['object']).columns

for column in categorical_columns:
    plt.figure(figsize=(8, 3))

    # Get the top 5 most frequent values in the column
```

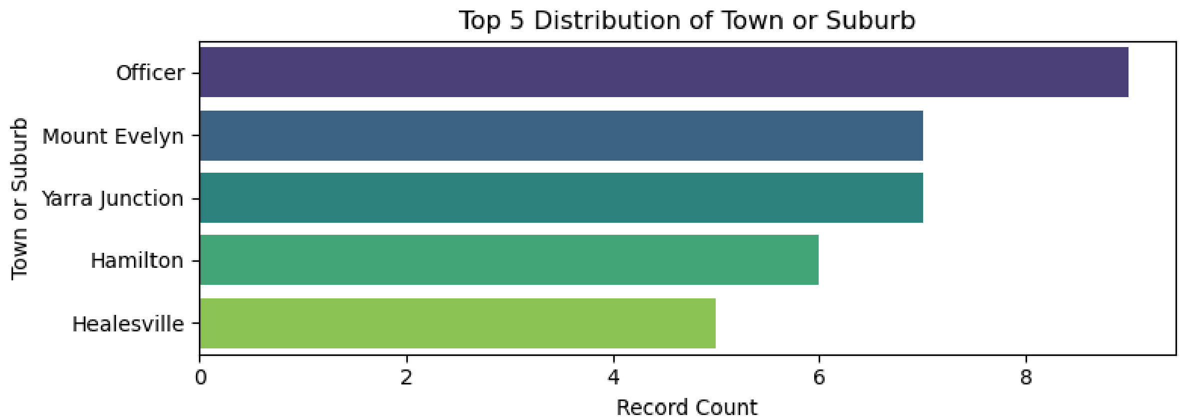
```
    top_5_values = df2[column].value_counts().nlargest(5)

    # Plot the top 5 values
    sns.barplot(x=top_5_values.values, y=top_5_values.index, palette="viridis")

    # Add title and labels
    plt.title(f'Top 5 Distribution of {column}')
    plt.xlabel('Record Count')
    plt.ylabel(column)

    plt.tight_layout()
    plt.show()
```

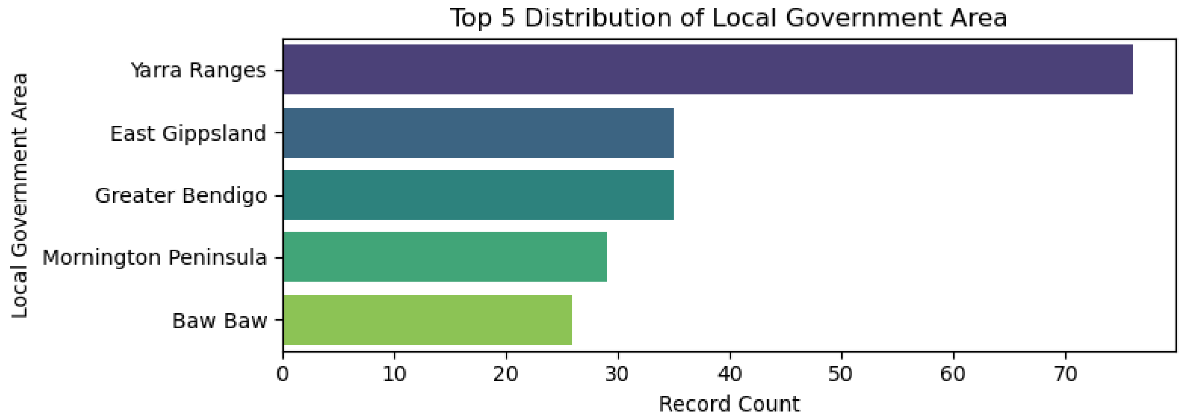


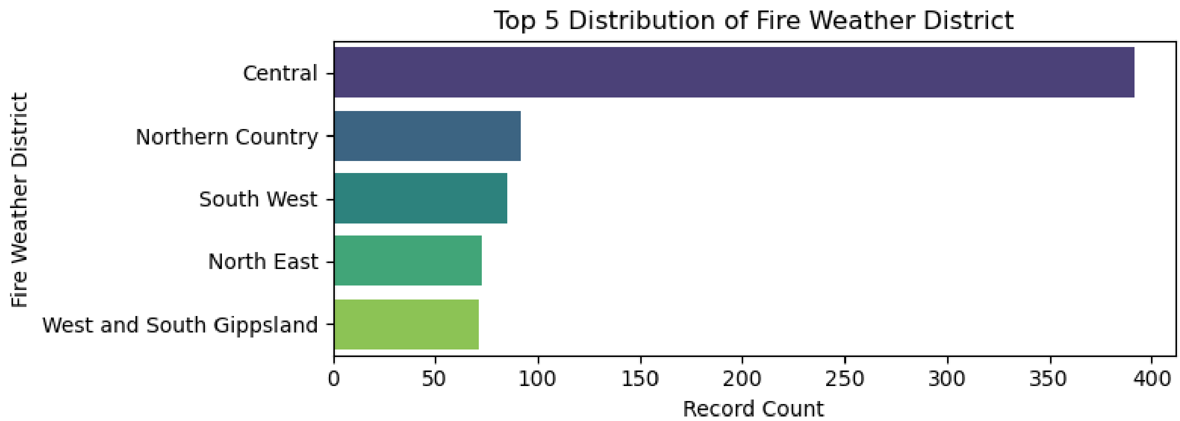






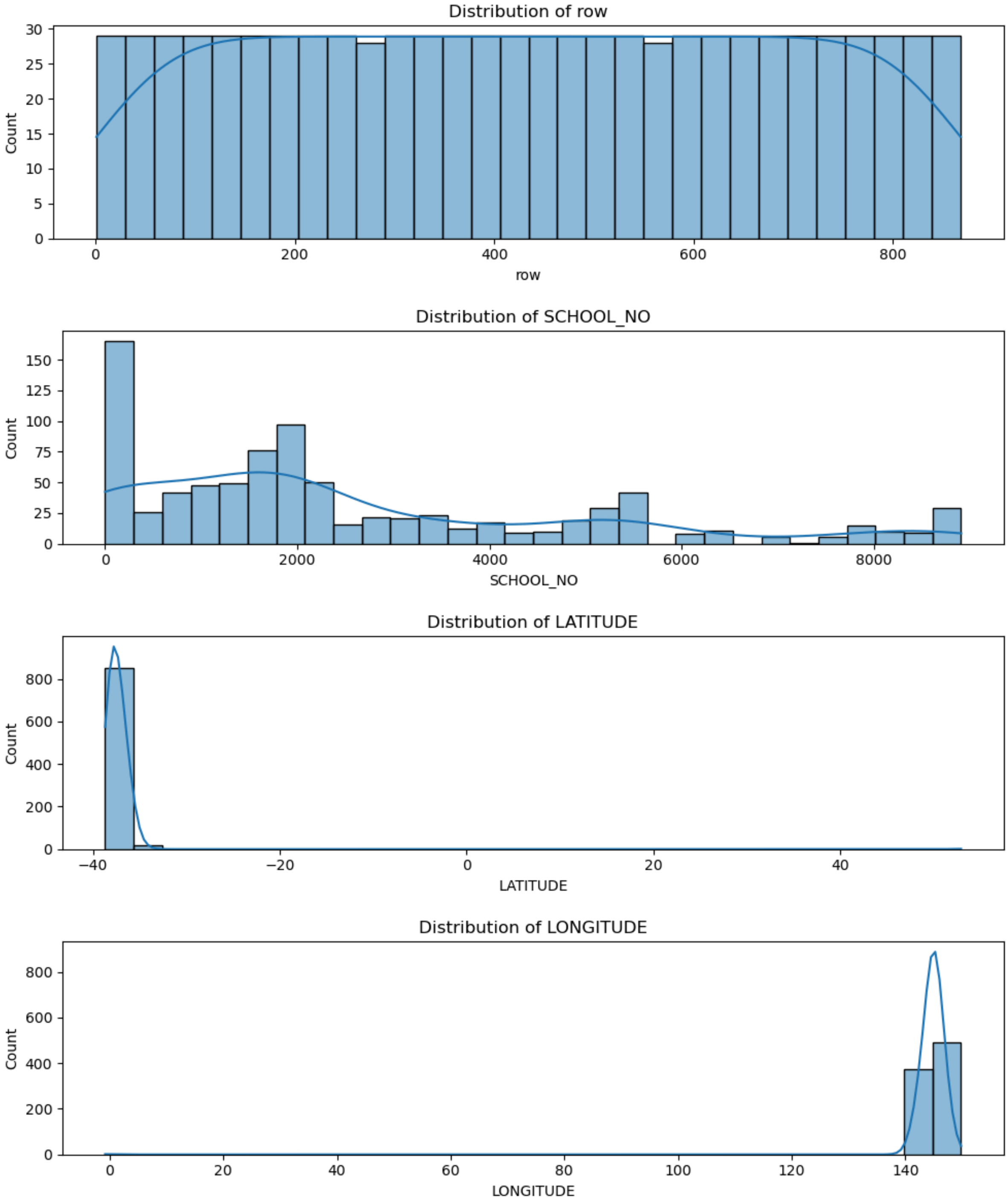






In [64]:
```
# Visualize the distribution of numerical columns
numerical_columns = df2.select_dtypes(include=['int64', 'float64']).columns

for column in numerical_columns:
    plt.figure(figsize=(10, 3))
    sns.histplot(df2[column], bins=30, kde=True)  # kde=True adds the Kernel Densit
    plt.title(f'Distribution of {column}')
```

```
    plt.tight_layout()
    plt.show()
```



A) Geocoding the School address

In [17]:
```
# Initialize geocoder
geolocator = Nominatim(user_agent="myGeocoder")

# Function to geocode addresses


def geocode_address(address):
    try:
        # Attempt to get the geographic coordinates (latitude and longitude) of the
        # Timeout is set to handle cases where the service is slow
        location = geolocator.geocode(address, timeout=10)
        if location:
            # If the location is found, return the latitude and longitude
            return location.latitude, location.longitude
```

```
        else:
            # If no location is found, return (None, None)
            return None, None
    except GeocoderTimedOut:
        # If the geocoding service times out, retry the geocoding request
        return geocode_address(address)  # Recursive call to retry
    except Exception as e:
        # If any other exception occurs, print the error and return (None, None)
        print(f"Error: {e}")
        return None, None
```

In [18]:

```
# Concatenate 'Facility Address' with 'Town or Suburb'
df2['Full Address'] = df2['Facility Address'].str.strip(
) + ', ' + df2['Town or Suburb'] + ', ' + df2['Local Government Area'].str.strip()

# Display the DataFrame to check the Full Address
df2.head(5)
```

Out[18]:

	row	SCHOOL_NO2	flag	Fire Risk Category 2023-24	Facility Name	Education Sector	Facility Address	Town or Suburb	Loca Governmen Are
0	1	1098	2	CAT 3	Advance College of Education Incorporated - Ha...	Independent	1973 Frankston Flinders Road	Hastings	Morningto Peninsul
1	2	5566	1	CAT 2	Aireys Inlet Primary School	Government	13 Anderson Street	Aireys Inlet	Surf Coas
2	3	2101	2	CAT 2	Alice Miller School	Independent	110 Bailey Road	Macedon	Macedo Range
3	4	366	1	CAT 3	Alice Miller School - Candlebark	Independent	83 Kerrie Road	Romsey	Macedo Range
4	5	1906	1	CAT 3	Al-Taqwa College - Camp	Independent	10 Cranswick Road	Banksia Peninsula	Eas Gippslan

In [ ]:

```
# Create a DataFrame to store the geocoded results
results = pd.DataFrame(df2['Full Address'], columns=['Full Address'])

# Apply geocoding with a delay to handle rate limits


def apply_geocoding(address):
```

```
        time.sleep(1)  # Adding delay to handle rate limits
        return geocode_address(address)


# Apply geocoding to addresses and create Latitude and Longitude columns
results[['Latitude', 'Longitude']] = results['Full Address'].apply(
    lambda x: pd.Series(apply_geocoding(x)))

# Merge the geocoded results with the original DataFrame
final_df = pd.concat([df2, results[['Latitude', 'Longitude']]], axis=1)

# Display the DataFrame with geocoded coordinates
final_df.head()
```

In [58]:

```
# Write the DataFrame to a CSV file
final_df.to_csv('geocoded_facilities.csv')
```

School register and location data

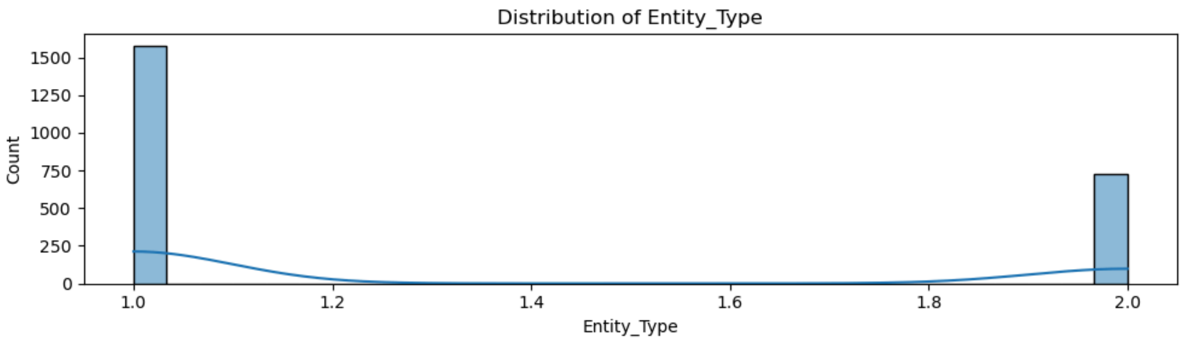
In [74]:

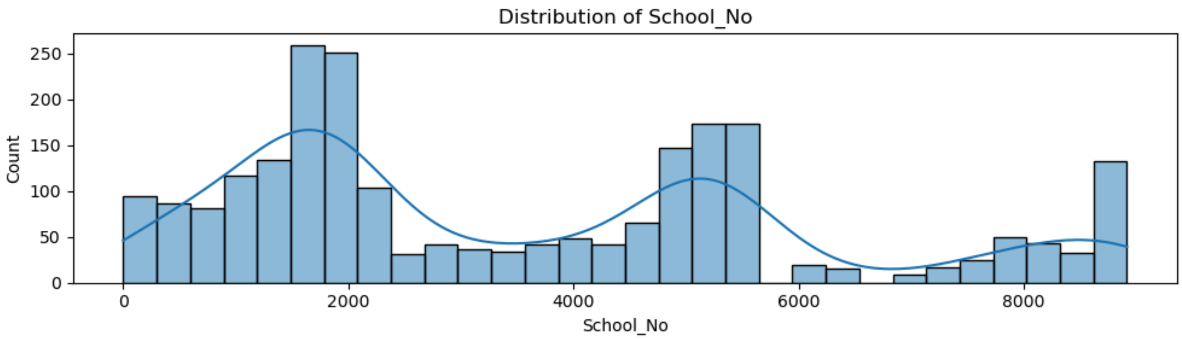
```
file_path = 'dv346-schoollocations2023.csv'

# Read the CSV file into a DataFrame with a different encoding
df3 = pd.read_csv(file_path, encoding='ISO-8859-1')

# Display the structure
df3.info()
```

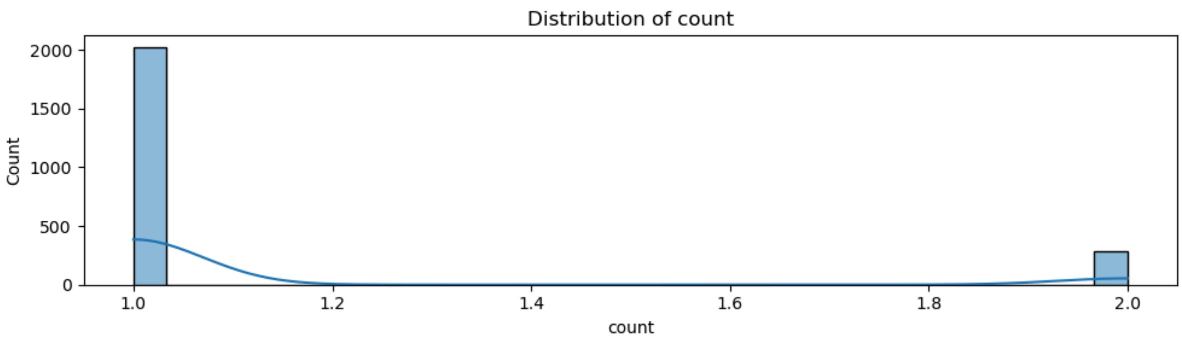
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2302 entries, 0 to 2301
Data columns (total 26 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Education_Sector      2302 non-null   object 
 1   Entity_Type           2302 non-null   int64  
 2   School_No             2302 non-null   int64  
 3   count                 2302 non-null   int64  
 4   School_Key            2302 non-null   object 
 5   School_Name           2302 non-null   object 
 6   School_Type           2302 non-null   object 
 7   School_Status         2302 non-null   object 
 8   Address_Line_1        2302 non-null   object 
 9   Address_Line_2        11 non-null     object 
 10  Address_Town          2302 non-null   object 
 11  Address_State         2302 non-null   object 
 12  Address_Postcode      2302 non-null   int64  
 13  Postal_Address_Line_1 2302 non-null   object 
 14  Postal_Address_Line_2 15 non-null     object 
 15  Postal_Town           2302 non-null   object 
 16  Postal_State          2302 non-null   object 
 17  Postal_Postcode       2302 non-null   int64  
 18  Full_Phone_No         2302 non-null   object 
 19  LGA_ID                2302 non-null   int64  
 20  LGA_Name              2302 non-null   object 
 21  X                     2301 non-null   float64
 22  Y                     2301 non-null   float64
 23  X.1                   2302 non-null   float64
 24  Y.1                   2302 non-null   float64
 25  lat-lon               2302 non-null   object 
dtypes: float64(4), int64(6), object(16)
memory usage: 467.7+ KB
```

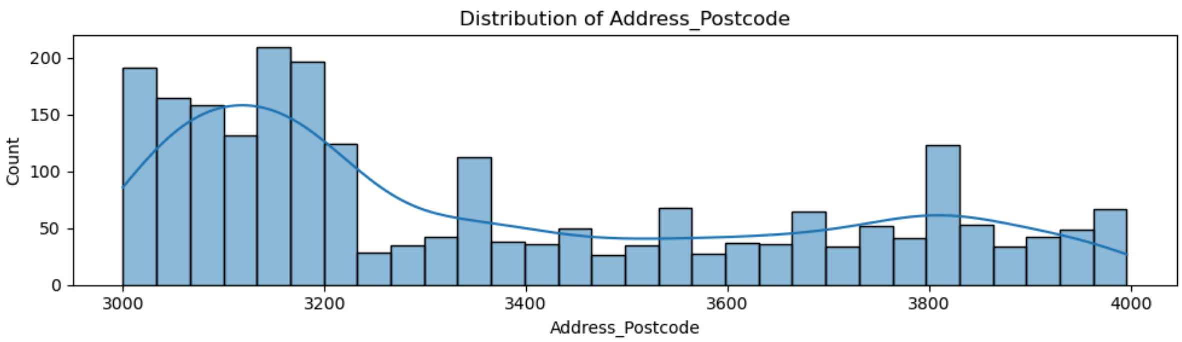
In [ ]:
```python
# Check for nulls in columns
df3.isnull().sum()
```

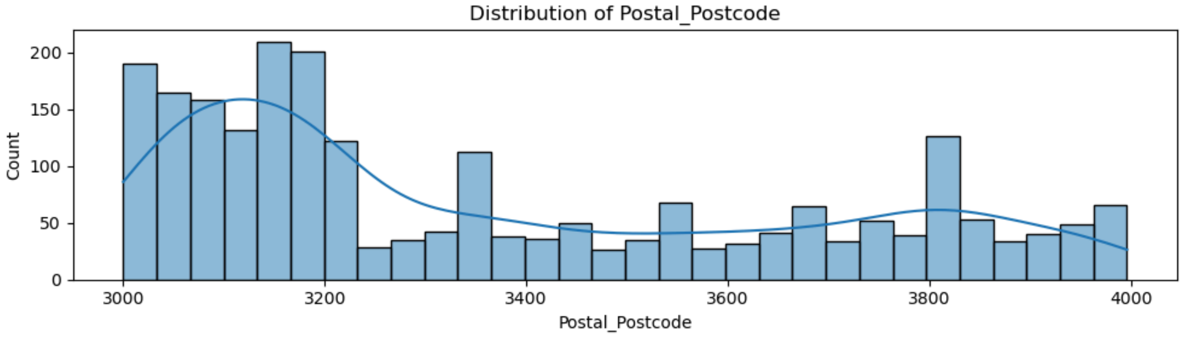
In [69]:
```python
# Visualize the distribution of numerical columns
numerical_columns = df3.select_dtypes(include=['int64', 'float64']).columns

for column in numerical_columns:
    plt.figure(figsize=(10, 3))
    sns.histplot(df3[column], bins=30, kde=True)  # kde=True adds the Kernel Densit
    plt.title(f'Distribution of {column}')
    plt.tight_layout()
    plt.show()
```

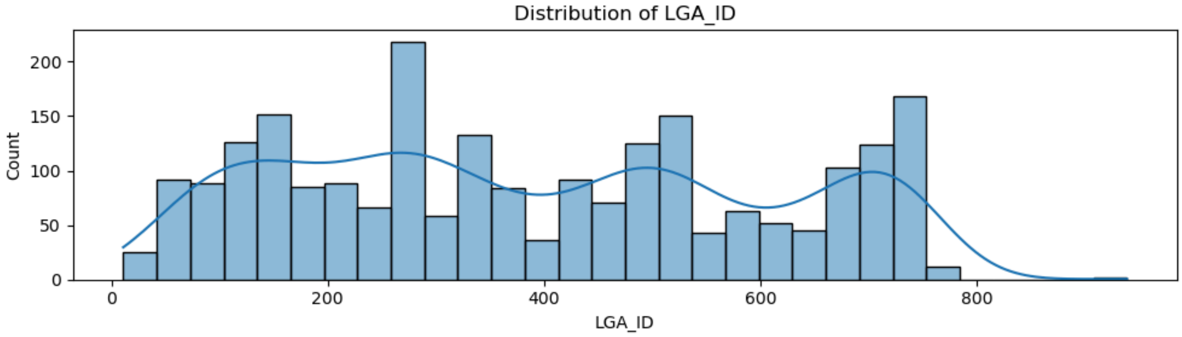


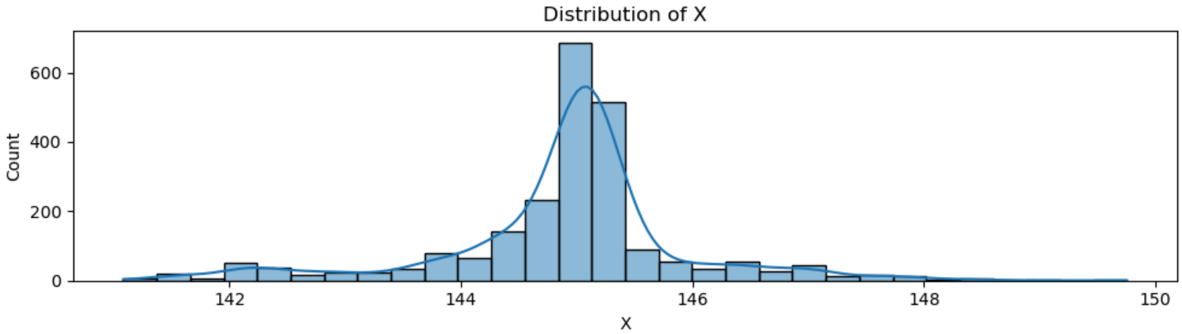


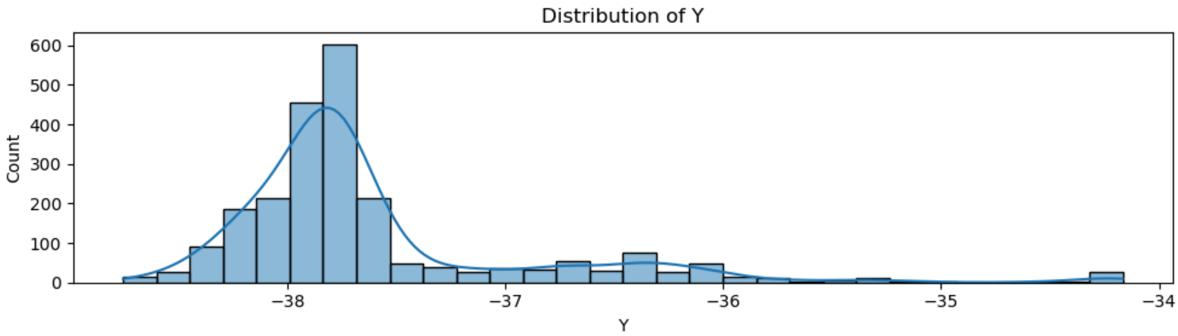


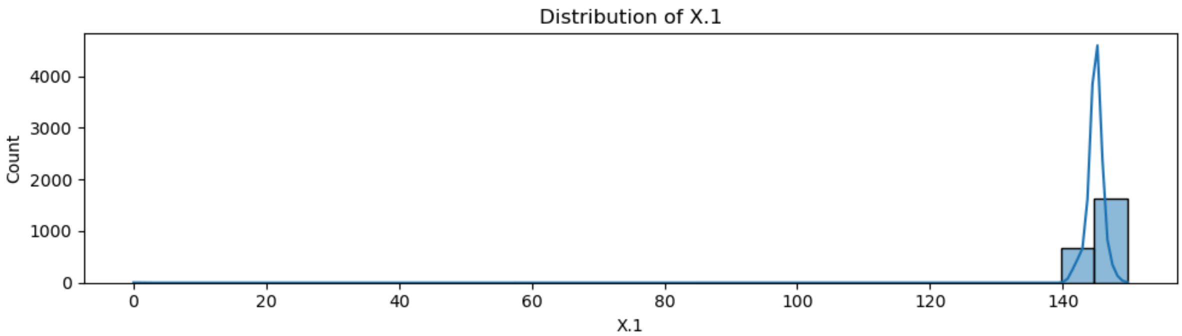


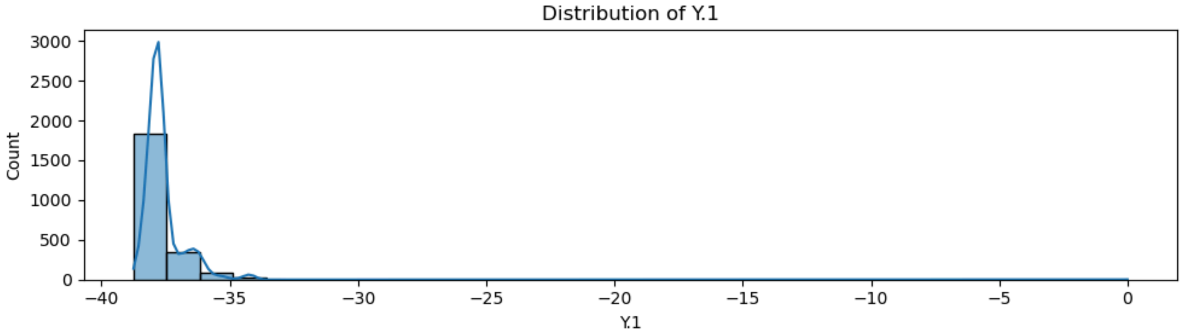

Distribution of Postal_Postcode
Count
200
150
100
50
0
3000
3200
3400
3600
3800
4000
Postal_Postcode


Distribution of LGA_ID
Count
200
150
100
50
0
0
200
400
600
800
LGA_ID

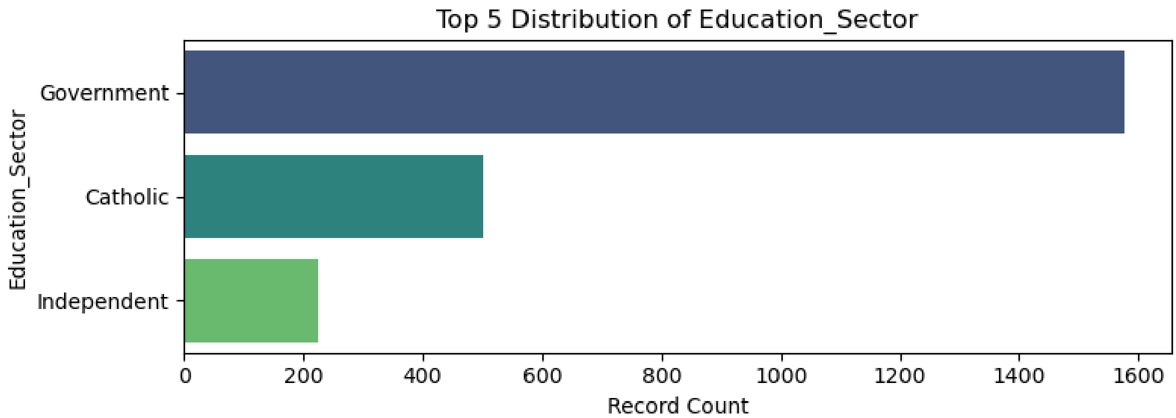

Distribution of X
Count
600
400
200
0
142
144
146
148
150
X

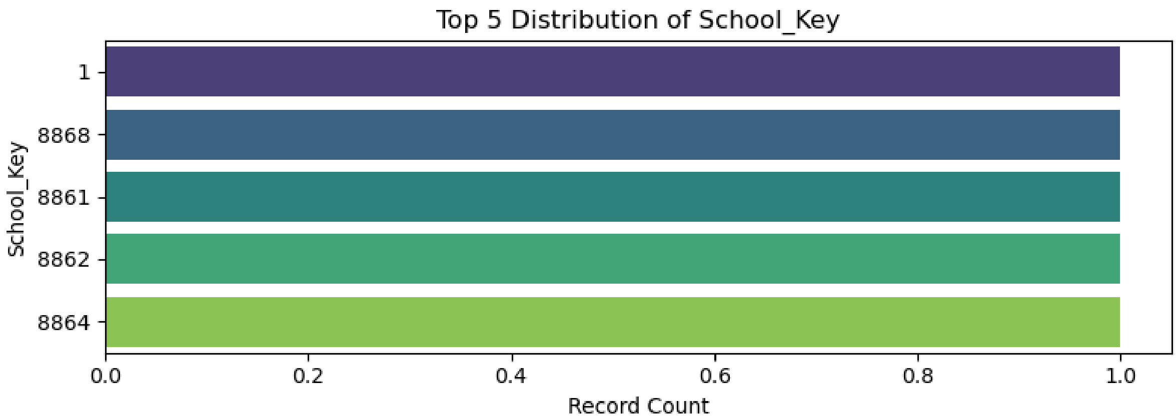

Distribution of Y
Count
600
500
400
300
200
100
0
−38
−37
−36
−35
−34
Y

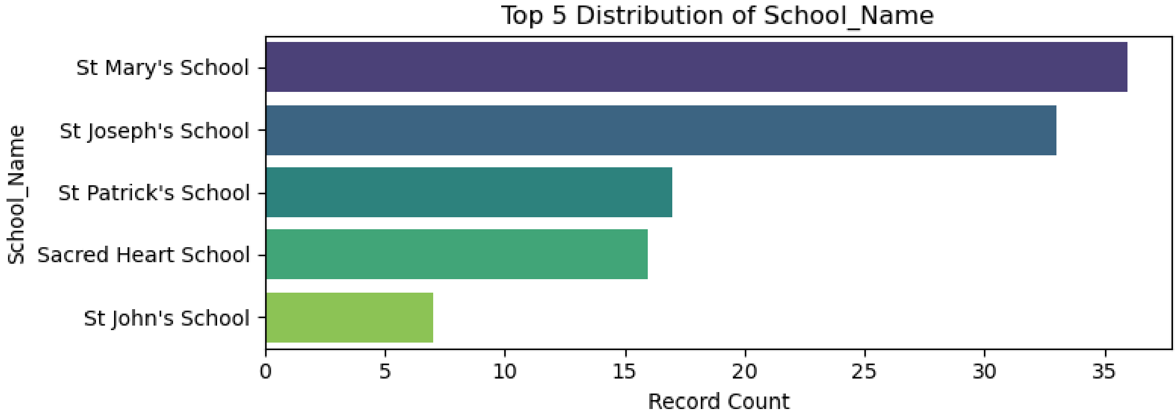

Distribution of X.1
Count
4000
3000
2000
1000
0
0
20
40
60
80
100
120
140
X.1

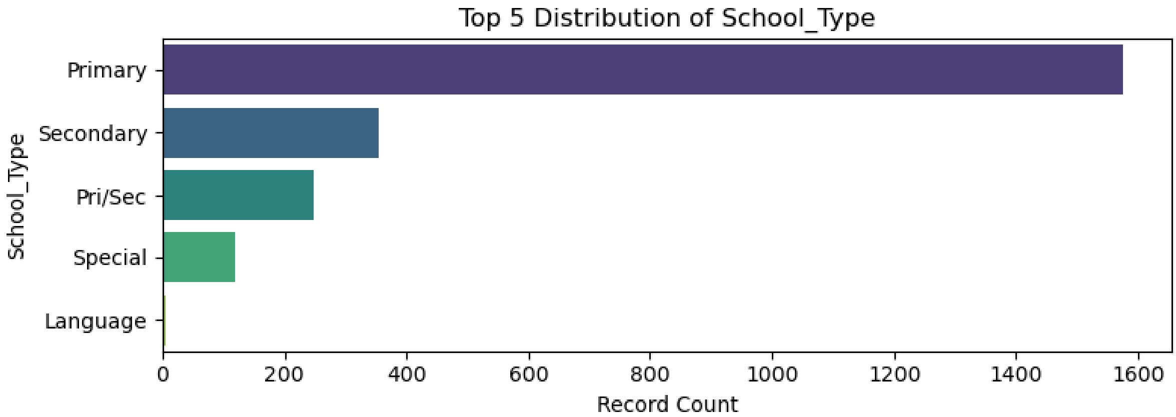


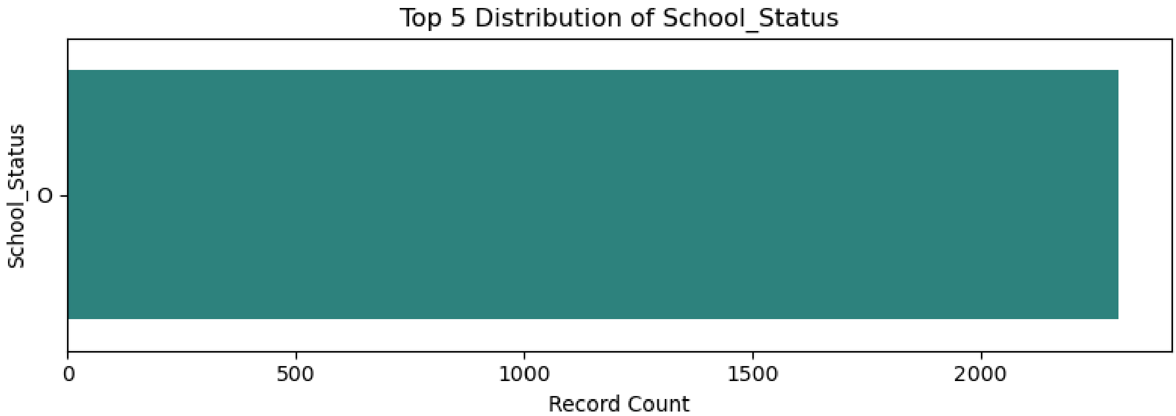
In [70]:
```
# Visualize the top 5 most frequent values for categorical columns
categorical_columns = df3.select_dtypes(include=['object']).columns

for column in categorical_columns:
    plt.figure(figsize=(8, 3))

    # Get the top 5 most frequent values in the column
    top_5_values = df3[column].value_counts().nlargest(5)

    # Plot the top 5 values
    sns.barplot(x=top_5_values.values, y=top_5_values.index, palette="viridis")

    # Add title and labels
    plt.title(f'Top 5 Distribution of {column}')
    plt.xlabel('Record Count')
    plt.ylabel(column)

    plt.tight_layout()
    plt.show()
```

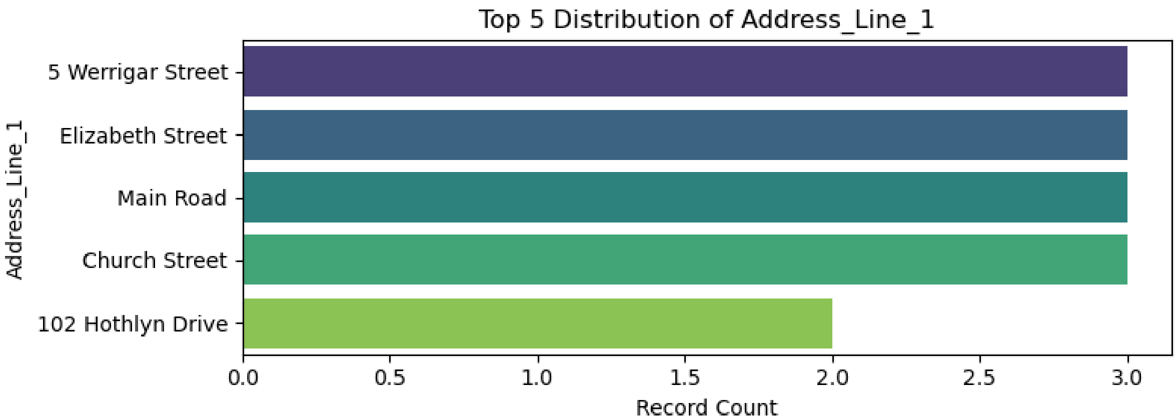


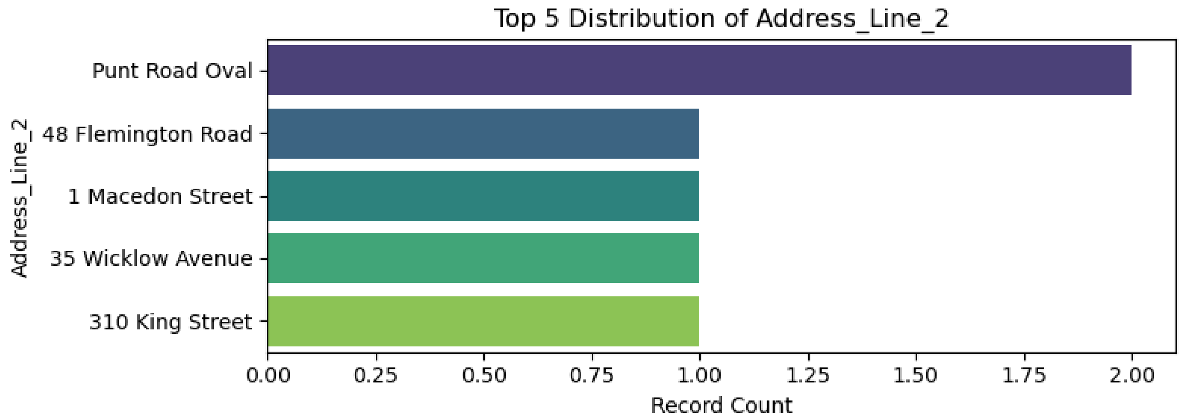


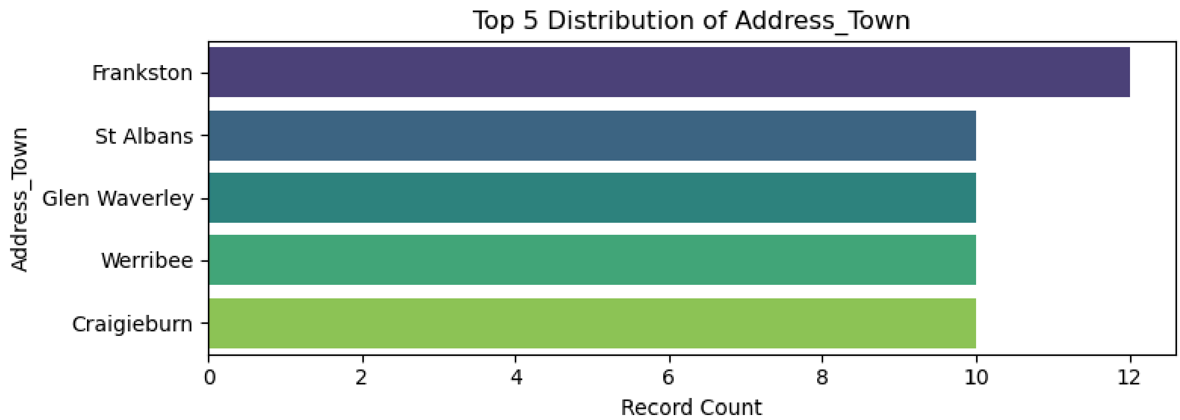

Top 5 Distribution of School_Name
School_Name
St Mary's School
St Joseph's School
St Patrick's School
Sacred Heart School
St John's School
0
5
10
15
20
25
30
35
Record Count

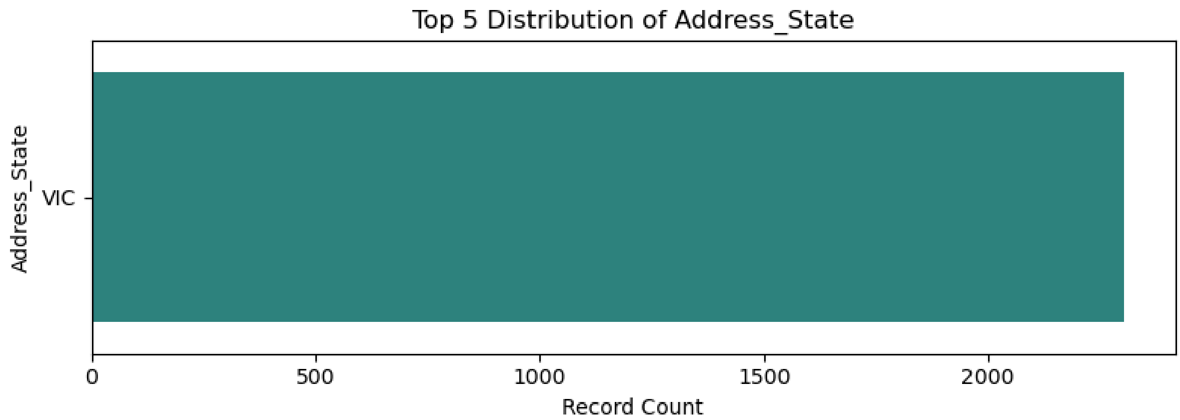

Top 5 Distribution of School_Type
School_Type
Primary
Secondary
Pri/Sec
Special
Language
0
200
400
600
800
1000
1200
1400
1600
Record Count

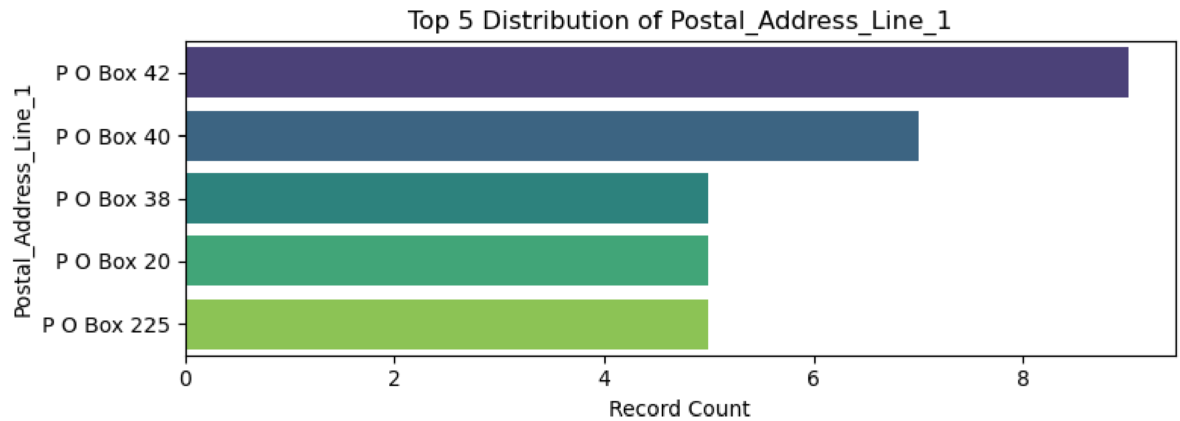

Top 5 Distribution of School_Status
School_Status
O
0
500
1000
1500
2000
Record Count

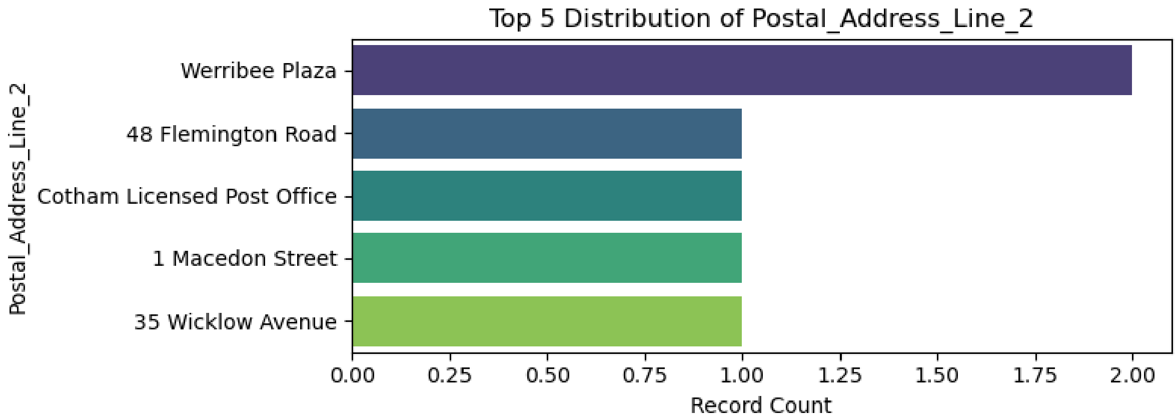

Top 5 Distribution of Address_Line_1
Address_Line_1
5 Werrigar Street
Elizabeth Street
Main Road
Church Street
102 Hothlyn Drive
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Record Count

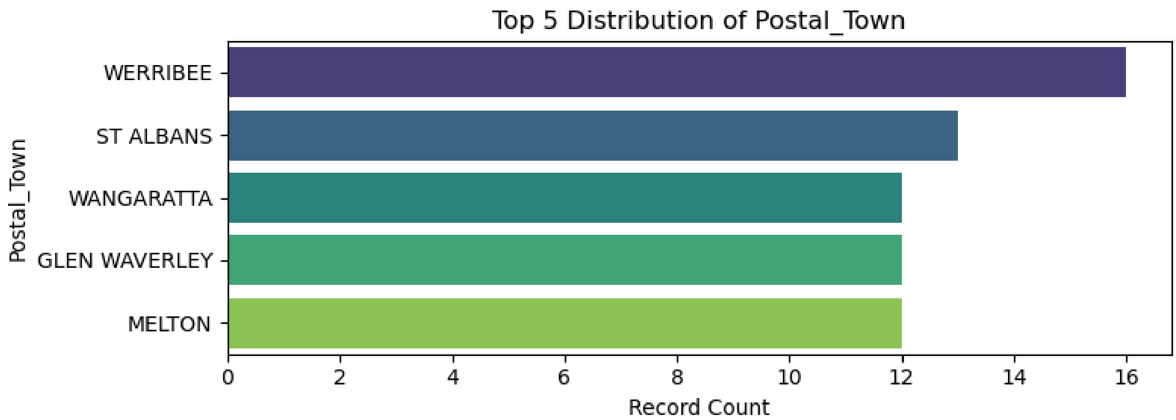

Top 5 Distribution of Address_Line_2
Address_Line_2
Punt Road Oval
48 Flemington Road
1 Macedon Street
35 Wicklow Avenue
310 King Street
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Record Count

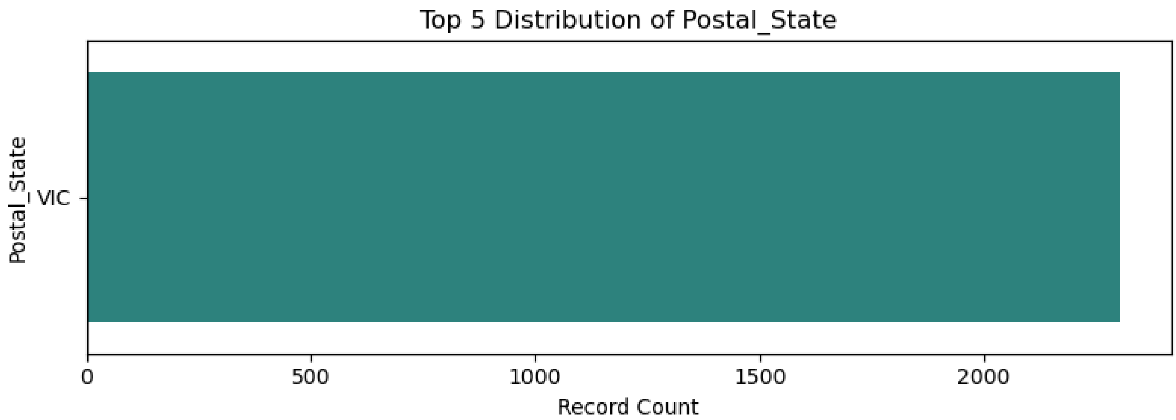

Top 5 Distribution of Address_Town
Address_Town
Frankston
St Albans
Glen Waverley
Werribee
Craigieburn
0
2
4
6
8
10
12
Record Count

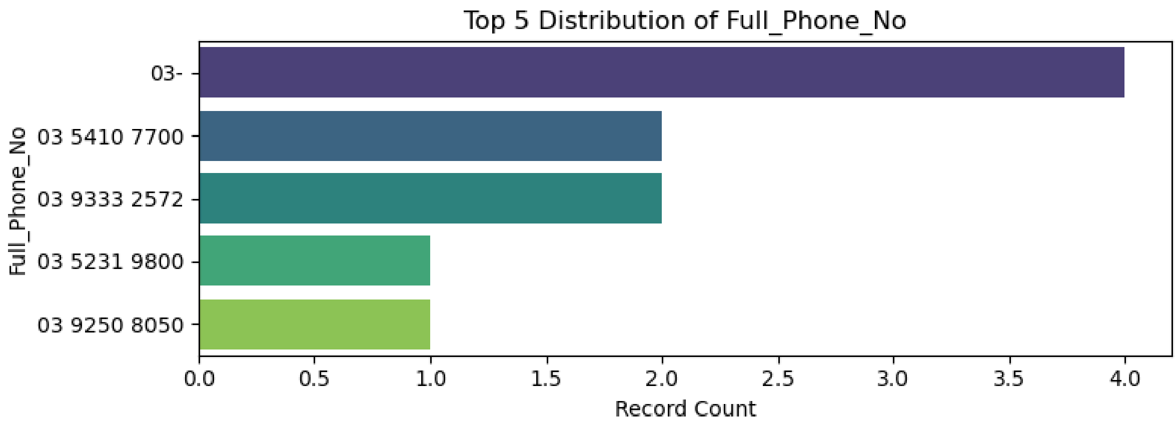

Top 5 Distribution of Address_State
Address_State
VIC
0
500
1000
1500
2000
Record Count

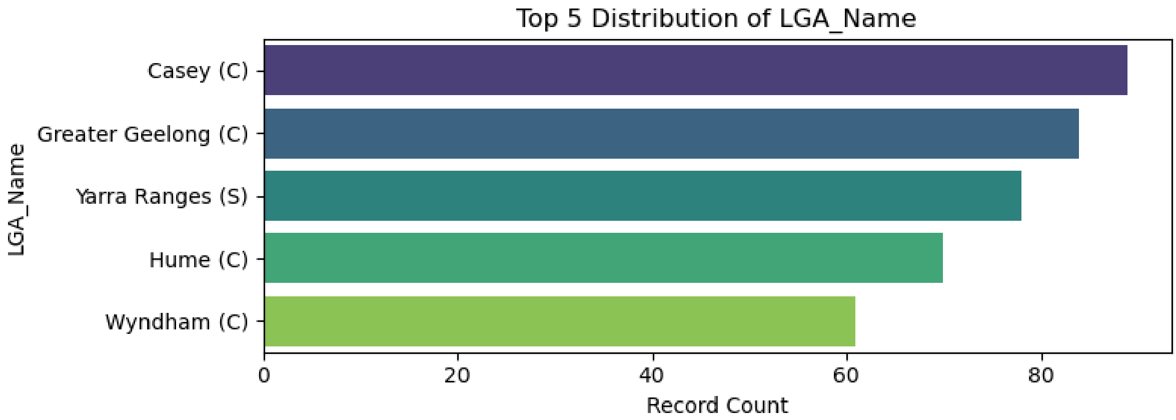

Top 5 Distribution of Postal_Address_Line_1
Postal_Address_Line_1
P O Box 42
P O Box 40
P O Box 38
P O Box 20
P O Box 225
0
2
4
6
8
Record Count

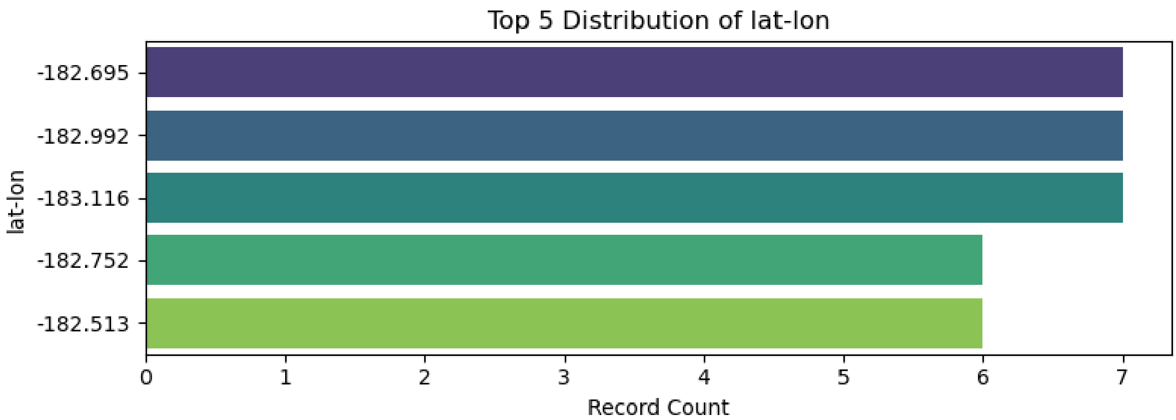

Top 5 Distribution of Postal_Address_Line_2
Postal_Address_Line_2
Werribee Plaza
48 Flemington Road
Cotham Licensed Post Office
1 Macedon Street
35 Wicklow Avenue
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
Record Count


Top 5 Distribution of Postal_Town
Postal_Town
WERRIBEE
ST ALBANS
WANGARATTA
GLEN WAVERLEY
MELTON
0 2 4 6 8 10 12 14 16
Record Count


Top 5 Distribution of Postal_State
Postal_State
VIC
0 500 1000 1500 2000
Record Count


Top 5 Distribution of Full_Phone_No
Full_Phone_No
03-
03 5410 7700
03 9333 2572
03 5231 9800
03 9250 8050
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
Record Count





In [31]:
```
# Display the first few rows of the DataFrame
df3.head()
```

Out[31]:

	Education_Sector	Entity_Type	School_No	count	School_Key	School_Name	School_Type	Scho
0	Government	1	1	2	1	Alberton Primary School	Primary	
1	Government	1	3	1	3	Allansford and District Primary School	Primary	
2	Government	1	4	1	4	Avoca Primary School	Primary	
3	Government	1	8	1	8	Avenel Primary School	Primary	
4	Government	1	12	1	12	Warrandyte Primary School	Primary	

5 rows × 26 columns

In [12]:
```
# Load geocoded data
file_path = 'geocoded_facilities.csv'

# Read the CSV file into a DataFrame with a different encoding
df4 = pd.read_csv(file_path, encoding='ISO-8859-1')
```

```
# Display the first few rows of the DataFrame
print(df4.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 868 entries, 0 to 867
Data columns (total 12 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Unnamed: 0                    868 non-null    int64  
 1   row                           868 non-null    int64  
 2   Fire Risk Category 2023-24    868 non-null    object 
 3   Facility Name                 868 non-null    object 
 4   Education Sector              868 non-null    object 
 5   Facility Address              868 non-null    object 
 6   Town or Suburb                868 non-null    object 
 7   Local Government Area         868 non-null    object 
 8   Fire Weather District         867 non-null    object 
 9   Full Address                  868 non-null    object 
 10  Latitude                      818 non-null    float64
 11  Longitude                     818 non-null    float64
dtypes: float64(2), int64(2), object(8)
memory usage: 81.5+ KB
None
```

A) Matching schools using longitude and latitude

In [24]:

```
# Filter out rows with NaN in the Coordinates columns
#df4 = df4.dropna(subset=['Latitude', 'Longitude'])
#df3 = df3.dropna(subset=['Y', 'X'])

# Extract latitudes and longitudes
df2['Coordinates'] = list(zip(df2['LATITUDE'], df2['LONGITUDE']))
df3['Coordinates'] = list(zip(df3['Y'], df3['X']))
```

Reference: https://geopy.readthedocs.io/en/stable/#geopy.distance.GeodesicDistance

In [26]:

```
from geopy.distance import geodesic

# Filter out rows with Null in the Coordinates columns
df3 = df3.dropna(subset=['Y', 'X'])
df2 = df2.dropna(subset=['LATITUDE', 'LONGITUDE'])

# Extract latitudes and longitudes
df2['Coordinates'] = list(zip(df2['LATITUDE'], df2['LONGITUDE']))
df3['Coordinates'] = list(zip(df3['Y'], df3['X']))

# Ensure no extra spaces in column names
df3.columns = df3.columns.str.strip()
df2.columns = df2.columns.str.strip()

# Function to find the closest school in dataset A for a given facility in dataset
def find_closest_school(facility_coords, school_coords):
    closest_school = None
    min_distance = float('inf')
    for i, school_coord in enumerate(school_coords):
        distance = geodesic(facility_coords, school_coord).kilometers
        if distance < min_distance:
            min_distance = distance
            closest_school = i
    return closest_school

# Ensure 'School_No' exists in df3
```

```
if 'School_Key' not in df3.columns:
    raise KeyError("The column 'School_Key' is not present in df3")

# Find the closest school for each facility
df2['Closest_School_No'] = df2['Coordinates'].apply(
    lambda x: df3.iloc[find_closest_school(x, df3['Coordinates'])]['School_Key']
    if find_closest_school(x, df3['Coordinates']) is not None
    else None
)

# Define the path and filename for the CSV file
csv_file_path = 'output_data_with_school_no.csv'

# Save the DataFrame to a CSV file
df2[['row','Facility Name', 'Closest_School_No']].to_csv(csv_file_path, index=False
```

In [ ]:

```
# reload dataset with all long and lat, remap schools
file_path = 'Website-BARR-2023-24-updated.xlsx'

# Read the Excel file into a DataFrame
df5 = pd.read_excel(file_path)
```

In [ ]:

```
# Extract latitudes and longitudes
df5['Coordinates'] = list(zip(df5['LATITUDE'], df5['LONGITUDE']))

# Ensure no extra spaces in column names
df5.columns = df5.columns.str.strip()

# Find the closest school for each facility
df5['Closest_School_No'] = df5['Coordinates'].apply(
    lambda x: df3.iloc[find_closest_school(x, df3['Coordinates'])]['School_No']
    if find_closest_school(x, df3['Coordinates']) is not None
    else None
)

# Define the path and filename for the CSV file
csv_file_path = 'output_data_with_school_no2.csv'

# Save the DataFrame to a CSV file
df5[['row','Facility Name', 'Closest_School_No']].to_csv(csv_file_path, index=False
```

B) Matching schools between the datasets based on name and city

In [5]:

```
!pip install fuzzywuzzy
!pip install python-Levenshtein
```

```
Requirement already satisfied: fuzzywuzzy in c:\users\thinithi\anaconda3\lib\site-packages (0.18.0)
Requirement already satisfied: python-Levenshtein in c:\users\thinithi\anaconda3\lib\site-packages (0.25.1)
Requirement already satisfied: Levenshtein==0.25.1 in c:\users\thinithi\anaconda3\lib\site-packages (from python-Levenshtein) (0.25.1)
Requirement already satisfied: rapidfuzz<4.0.0,>=3.8.0 in c:\users\thinithi\anaconda3\lib\site-packages (from Levenshtein==0.25.1->python-Levenshtein) (3.9.6)
```

Reference: https://stackoverflow.com/questions/32055817/python-fuzzy-matching-fuzzywuzzy-keep-only-best-match

In [6]:

```
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
```

```python
import pandas as pd

# Ensure no extra spaces in column names
df2.columns = df2.columns.str.strip()
df3.columns = df3.columns.str.strip()

# Combine 'Facility Name' and 'Town or Suburb' into a single string
df2['Name_City'] = df2['Facility Name'] + ", " + df2['Town or Suburb']

# Combine 'School_Name' and 'Address_Town' into a single string
df3['Name_City'] = df3['School_Name'] + ", " + df3['Address_Town']

# Function to find the best match for a facility in df2 with the school in df3


def find_best_school_match(facility_name_city, school_name_cities, threshold=80):
    best_match = process.extractOne(
        facility_name_city, school_name_cities, scorer=fuzz.ratio)
    if best_match and best_match[1] >= threshold:
        # return the best matching school name if the match is above the threshold
        return best_match[0]
    else:
        return None


# Ensure 'School_No' and 'Name_City' exist in df3
if 'School_No' not in df3.columns or 'Name_City' not in df3.columns:
    raise KeyError(
        "The column 'School_No' or 'Name_City' is not present in df3")

# Create a dictionary to map Name_City to their corresponding School_No
name_city_to_no = df3.set_index('Name_City')['School_No'].to_dict()

# Find the best matching school name for each facility and get its School_No
df2['Closest_School_Name_City'] = df2['Name_City'].apply(
    lambda x: find_best_school_match(x, df3['Name_City'], threshold=90)
)
df2['Closest_School_No'] = df2['Closest_School_Name_City'].map(name_city_to_no)

# Define the path and filename for the CSV file
csv_file_path = 'output_data_with_school_no3.csv'

# Save the DataFrame to a CSV file
df2[['row', 'Facility Name', 'Town or Suburb', 'Closest_School_No']].to_csv(
    csv_file_path, index=False)
```

Longitude and latitude mappings were further validated through name-address similarity mapping to identify whether the same school appears in both datasets. A unique identifier was assigned to each school to facilitate this process. This approach ensured that schools with matching geographic coordinates were accurately linked, while discrepancies were addressed by verifying names and addresses to confirm or correct the data..

References:

Welcome to GeoPy's documentation!¶. Welcome to GeoPy's documentation! - GeoPy 2.4.1 documentation. (n.d.). https://geopy.readthedocs.io/en/stable/#geopy.distance.GeodesicDistance

GeeksforGeeks. (2024, September 29). Fuzzywuzzy Python Library. https://www.geeksforgeeks.org/fuzzywuzzy-python-library/

https://www.geeksforgeeks.org/how-to-get-geolocation-in-python/

GeeksforGeeks. (2022, September 6). How to get geolocation in python? https://www.geeksforgeeks.org/how-to-get-geolocation-in-python/

In [ ]: