This notebook will demonstrate how to perform EDA (Exploratory Data Analysis) and data visualization for the Real Estate Sales 2001-2022 GL dataset.

Table of Contents:

Hope it is helpful.

This notebook has been sourced from the following link: https://catalog.data.gov/dataset/real-estate-sales-2001-2018.

Import Library

- **missingno:** A library for visualizing missing data, helping us understand missing data structures.

- **plotly.express:** A library used to create interactive graphs.
- **folium:** A library for visualizing data on maps.
- **folium.plugins:** Allows us to use additional plugins for Folium.
- **plt.rcParams["figure.figsize"] = (6,4):** Sets the size of the plots created with Matplotlib. Here, the plot size is defined as 6 inches in width and 4 inches in height.
- **warnings.filterwarnings("ignore"):** Used to ignore Python warnings. This prevents unnecessary warning messages from appearing during code execution.
- **pd.set_option('display.max_columns', None):** Does not limit the number of columns in the Pandas DataFrame output, showing all columns.
- **pd.set_option('display.max_rows', None):** Does not limit the number of rows in the Pandas DataFrame output, showing all rows.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import missingno as msno
import plotly.express as px
import folium
from folium import plugins

plt.rcParams["figure.figsize"] = (6,4)

import warnings
warnings.filterwarnings("ignore")
warnings.warn("this will not show")

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Loading The Dataset

About the DataSet

Field Name	Description	Field Alias	Data Type
Serial Number	Serial number	serialnumber	Number
List Year	Year the property was listed for sale	listyear	Number
Date Recorded	Date the sale was recorded locally	daterecorded	Floating Timestamp
Town	Town name	town	Text

Address	Address	address	Text
Assessed Value	Value of the property used for local tax assessment	assessedvalue	Number
Sale Amount	Amount the property was sold for	saleamount	Number
Sales Ratio	Ratio of the sale price to the assessed value	salesratio	Text
Property Type	Type of property including: Residential, Commercial, Industrial, Apartments, Vacant, etc.	propertytype	Text
Residential Type	Indicates whether property is single or multifamily residential	residentialtype	Text
Non Use Code	Non-usable sale code typically means the sale price is not reliable for use in property valuation	nonusecode	Text
Assessor Remarks	Remarks from the assessor	remarks	Text
OPM Remarks	Remarks from OPM	opm_remarks	Text
Location	Lat / lon coordinates	geo_coordinates	Point

In [2]:

```
df0 = pd.read_csv('Real_Estate_Sales_2001-2022_GL.csv')
df = df0.copy()
```

Understanding the dataset-EDA

Skimpy Library:

It is a library used for quickly and effectively summarizing datasets in Python.

By displaying basic statistics on a column basis, it facilitates understanding the data and identifying potential issues.

It offers a simple solution that saves time for data analysts when exploring and inspecting large datasets.

In [3]:

```
from skimpy import skim
```

In [4]:

```
skim(df)
```

```
skimpy summary
        Data Summary                 Data Types
 dataframe            Values     Column Type   Count
 Number of rows       1097629    string        8
 Number of columns    13         float64       3
```

int64	2

number

column_name	NA	NA %	mean	sd	p0	p25
Serial Number	0	0	537000	7526000	0	30710
List Year	0	0	2011	6.773	2001	2005
Assessed Value	0	0	281800	1658000	0	89090
Sale Amount	0	0	405300	5143000	0	145000
Sales Ratio	0	0	9.604	1802	0	0.4779

string

column_name	NA	NA %	words p
Date Recorded	2	0	
Town	0	0	
Address	51	0	
Property Type	382446	34.84	
Residential Type	398389	36.3	
Assessor Remarks	926401	84.4	
OPM remarks	1084598	98.81	
Location	799518	72.84	

End

The info() method:

Provides basic information about a DataFrame. This method is useful for quickly understanding the structure of the DataFrame.

RangeIndex: - Information about indexing, such as start, end, and step size.

Data columns: - The names of all columns, how many non-null (non-empty) values there are, and the data type (dtype).

memory usage: - The amount of memory the DataFrame occupies in memory.

dtypes: - The count of different data types (e.g., integer, float, object, etc.).

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1097629 entries, 0 to 1097628
Data columns (total 14 columns):
 #   Column            Non-Null Count    Dtype
---  ------            --------------    -----
 0   Serial Number     1097629 non-null  int64
 1   List Year         1097629 non-null  int64
 2   Date Recorded     1097627 non-null  object
 3   Town              1097629 non-null  object
 4   Address           1097578 non-null  object
 5   Assessed Value    1097629 non-null  float64
 6   Sale Amount       1097629 non-null  float64
 7   Sales Ratio       1097629 non-null  float64
 8   Property Type     715183 non-null   object
 9   Residential Type  699240 non-null   object
```

```
 10  Non Use Code      313451 non-null   object
 11  Assessor Remarks  171228 non-null   object
 12  OPM remarks       13031 non-null    object
 13  Location          298111 non-null   object
dtypes: float64(3), int64(2), object(9)
memory usage: 117.2+ MB
```

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Serial Number', 'List Year', 'Date Recorded', 'Town', 'Address',
       'Assessed Value', 'Sale Amount', 'Sales Ratio', 'Property Type',
       'Residential Type', 'Non Use Code', 'Assessor Remarks', 'OPM remarks',
       'Location'],
      dtype='object')
```

.str.replace(" ", "_"): Replaces spaces in column names with underscores, thus reformatting the column names.

Helps in obtaining more useful and manageable column names. This is a common practice to keep column names consistent and make accessing the columns easier.

In [7]:
```
df.columns = df.columns.str.replace(" ","_")
```

In [8]:
```
df.columns
```

Out[8]:
```
Index(['Serial_Number', 'List_Year', 'Date_Recorded', 'Town', 'Address',
       'Assessed_Value', 'Sale_Amount', 'Sales_Ratio', 'Property_Type',
       'Residential_Type', 'Non_Use_Code', 'Assessor_Remarks', 'OPM_remarks',
       'Location'],
      dtype='object')
```

In [ ]:
```
```

df.shape, A property that returns the dimensions (number of rows and columns) of a DataFrame.

When this property is used, a tuple is returned that shows how many rows and columns the DataFrame consists of.

In [9]:
```
df.shape
```

Out[9]:
```
(1097629, 14)
```

dtypes Shows the data type (dtype) of each column.

This property helps you quickly understand what type of data exists in your DataFrame.

In [10]:
```
df.dtypes
```

Out[10]:
```
Serial_Number        int64
List_Year            int64
Date_Recorded       object
Town                object
```

```
Address              object
Assessed_Value      float64
Sale_Amount         float64
Sales_Ratio         float64
Property_Type        object
Residential_Type     object
Non_Use_Code         object
Assessor_Remarks     object
OPM_remarks          object
Location             object
dtype: object
```

duplicated

The duplicated function is used to identify duplicate rows within a pandas DataFrame. This function returns a Series containing boolean (True/False) values indicating whether each row is duplicated or not. The function provides flexibility with the keep parameter: keep=first (default): Marks all duplicates as True, except the first occurrence. keep=last: Marks all duplicates as True, except the last occurrence. keep=False: Marks all duplicates as True.

In [11]:

```
df.duplicated().sum()
```

Out[11]:

```
np.int64(0)
```

Mising Value

In [12]:

```
pd.DataFrame({
    'Count': df.count(),
    'Null': df.isnull().sum(),
    'Cardinality': df.nunique()
})
```

Out[12]:

	Count	Null	Cardinality
Serial_Number	1097629	0	96220
List_Year	1097629	0	22
Date_Recorded	1097627	2	6958
Town	1097629	0	170
Address	1097578	51	771931
Assessed_Value	1097629	0	99306
Sale_Amount	1097629	0	61075
Sales_Ratio	1097629	0	552974
Property_Type	715183	382446	11
Residential_Type	699240	398389	5
Non_Use_Code	313451	784178	105
Assessor_Remarks	171228	926401	75286
OPM_remarks	13031	1084598	6490

Location	298111	799518	216556

In [13]:

```
df.isnull().sum()
```

Out[13]:

```
Serial_Number             0
List_Year                 0
Date_Recorded             2
Town                      0
Address                  51
Assessed_Value            0
Sale_Amount               0
Sales_Ratio               0
Property_Type        382446
Residential_Type     398389
Non_Use_Code         784178
Assessor_Remarks     926401
OPM_remarks         1084598
Location             799518
dtype: int64
```

sns.set_theme(): - Uses Seaborn's default theme settings. This ensures that all generated plots have a consistent appearance.

sns.set(rc={"figure.dpi":300, "figure.figsize":(12,9)}): - This line adjusts the resolution and size of the generated plots.

"figure.dpi": 300: - Sets the DPI (dots per inch) to 300, which results in higher resolution plots.

"figure.figsize": (12, 9): - Sets the size of the generated plots to 12x9 inches.

In [14]:

```
sns.set_theme()

sns.set(rc={"figure.dpi":300, "figure.figsize":(12,9)})
```

In [15]:

```
df.Town.value_counts().head(5)
```

Out[15]:

```
Town
Bridgeport     38158
Stamford       36629
Waterbury      32662
Norwalk        26939
New Haven      23705
Name: count, dtype: int64
```

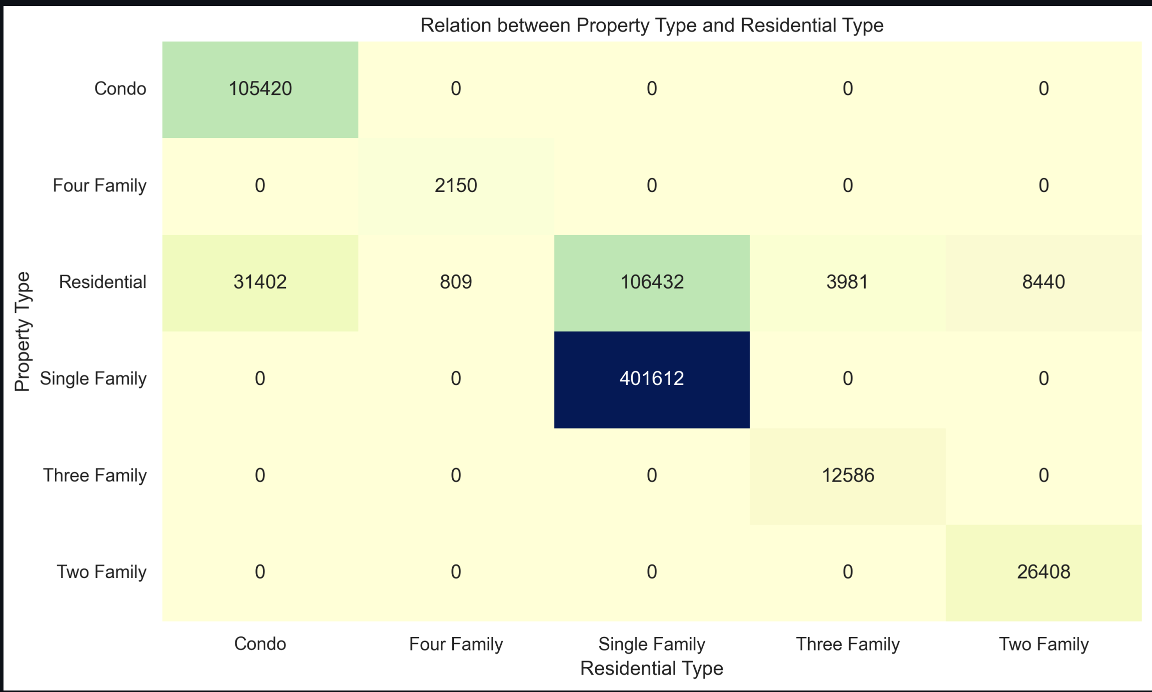
Data Manipulation Questions

Missing Values: How should we handle the missing values in the Property Type, Residential Type, and Location columns? Should we use imputation or remove these rows?

First, the relation between property type and residential type wil be checked

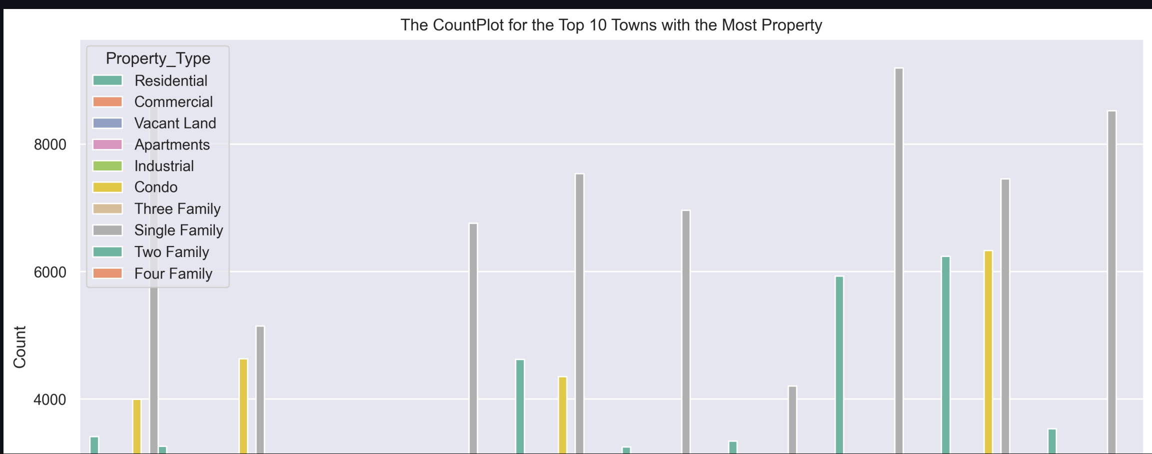
In [16]:

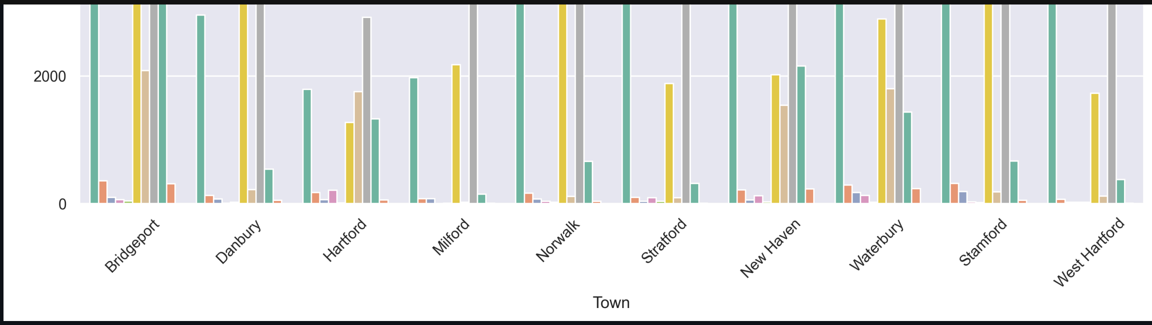
```
property_residential_table = df.pivot_table(index='Property_Type',
                                            columns='Residential_Type',
                                            aggfunc='size',
                                            fill_value=0)

plt.figure(figsize=(10, 6))
sns.heatmap(property_residential_table, annot=True, cmap='YlGnBu', fmt='d', c
plt.title('Relation between Property Type and Residential Type')
plt.xlabel('Residential Type')
plt.ylabel('Property Type')
plt.tight_layout()
plt.show()
```



In [17]:

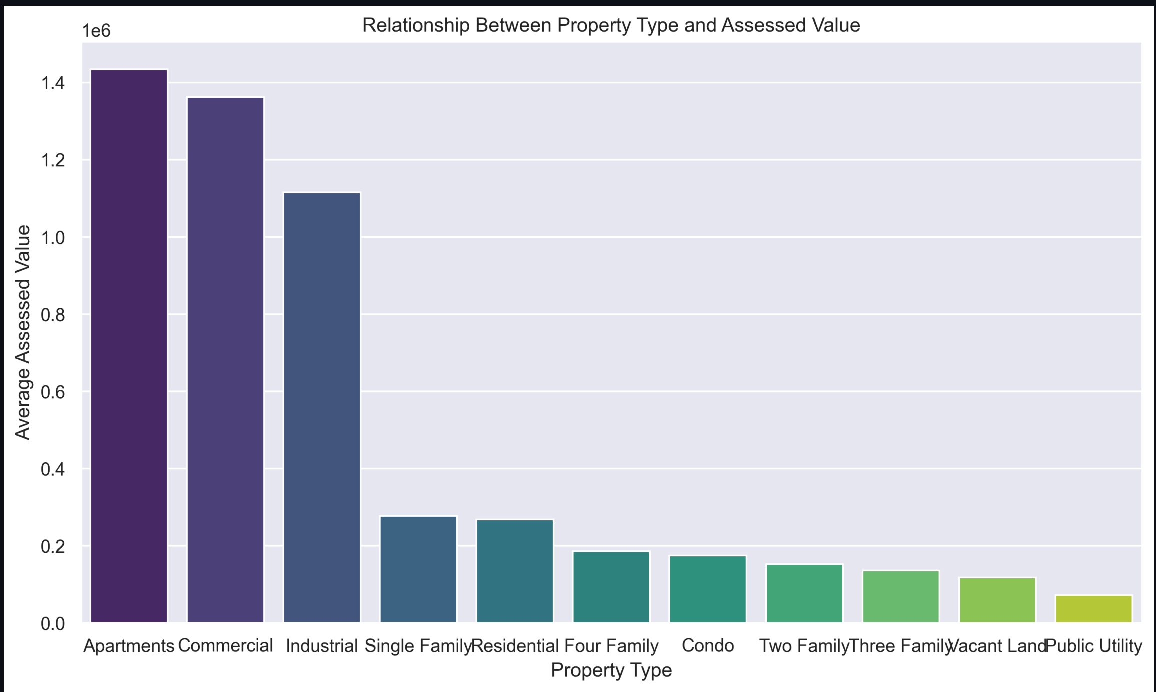
```
top_10_towns = df['Town'].value_counts().head(10).index
plt.figure(figsize=(12, 8))
sns.countplot(data=df[df['Town'].isin(top_10_towns)], x='Town', hue='Property
plt.title('The CountPlot for the Top 10 Towns with the Most Property')
plt.xlabel('Town')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```





In [18]:

```
property_value_table = df.groupby('Property_Type')['Assessed_Value'].mean().r
property_value_table = property_value_table.sort_values(by='Assessed_Value',

plt.figure(figsize=(10, 6))
sns.barplot(data=property_value_table, x='Property_Type', y='Assessed_Value',

plt.title('Relationship Between Property Type and Assessed Value')
plt.xlabel('Property Type')
plt.ylabel('Average Assessed Value')
plt.tight_layout()

plt.show()
```



Propert Type Missing Values

There will be **two different approaches** to fill in the missing values for Property Type and Residential Type.

In Property Type, the missing values will be filled based on the mean Assessed Value for each property type. If the Assessed Value of the property is between the specified values (these values were determined via mean value calculation), the property type will be assigned accordingly.

In Residential Type, the mean Assessed Value of the residential types per town will be calculated. The Assessed Value will be compared to those mean values, and the closest mean value residential type will be used to fill in the missing values.

The second approach might seem more logical to apply to the missing values in Property Type as well; however, for the sake of learning practices, both approaches will be retained in the notebook.

In [19]:
```
df.Property_Type.value_counts()
```

Out[19]:
```
Property_Type
Single Family     401612
Residential       151064
Condo             105420
Two Family         26408
Three Family       12586
Vacant Land         7824
Commercial          5987
Four Family         2150
Apartments          1327
Industrial           795
Public Utility        10
Name: count, dtype: int64
```

In [20]:
```
property_avg_value = df.groupby('Property_Type')['Assessed_Value'].mean().res
property_avg_value = property_avg_value.rename(columns={'Assessed_Value': 'Av

print(property_avg_value)
```

```
     Property_Type  Avg Assessed Value
0       Apartments        1.435348e+06
1       Commercial        1.363012e+06
2            Condo        1.745589e+05
3      Four Family        1.861389e+05
4       Industrial        1.116113e+06
5   Public Utility        7.247500e+04
6      Residential        2.687961e+05
7    Single Family        2.771346e+05
8     Three Family        1.367156e+05
9       Two Family        1.525845e+05
10     Vacant Land        1.179861e+05
```

In [21]:
```
df.loc[(df['Residential_Type'].notnull()) & (df['Property_Type'].isnull()), '


def assign_property_type_with_town(row):

    town_value = row['Town']


    if row['Assessed_Value'] < 150000:
        return 'Residential'
    elif 150000 <= row['Assessed_Value'] < 300000:
        if row['Property_Type'] in ['Single Family', 'Two Family', 'Three Fam
            return row['Property_Type']
        else:
            return 'Residential'
    elif 300000 <= row['Assessed_Value'] < 500000:
        if row['Property_Type'] in ['Single Family', 'Two Family', 'Four Fami
            return row['Property_Type']
        else:
            return 'Single Family'
```

```python
    elif 500000 <= row['Assessed_Value'] < 1000000:

        if row['Property_Type'] in ['Commercial', 'Industrial', 'Apartments']
            return row['Property_Type']
        else:
            return 'Commercial'
    else:

        return 'Commercial'


df['Property_Type'] = df.apply(lambda row: assign_property_type_with_town(row


print("Eksik Property Type sayısı:", df['Property_Type'].isnull().sum())
```

```
Eksik Property Type sayısı: 0
```

In [22]:

```python
df.Property_Type.value_counts()
```

Out[22]:

```
Property_Type
Residential       486634
Single Family     423247
Condo             105420
Commercial         31228
Two Family         26408
Three Family       12586
Vacant Land         7824
Four Family         2150
Apartments          1327
Industrial           795
Public Utility        10
Name: count, dtype: int64
```

Residential Type Missing Values

In Residential Type, the mean Assessed Value of the residential types per town will be calculated. The Assessed Value will be compared to those mean values, and the closest mean value residential type will be used to fill in the missing values.

The second approach might seem more logical to apply to the missing values in Property Type as well; however, for the sake of learning practices, both approaches will be retained in the notebook.

In [23]:

```python
df.Residential_Type.value_counts()
```

Out[23]:

```
Residential_Type
Single Family    508044
Condo            136822
Two Family        34848
Three Family      16567
Four Family        2959
Name: count, dtype: int64
```

In [24]:

```python
df.Residential_Type.isnull().sum()
```

Out[24]: np.int64(398389)

In [25]:

```
df['Residential_Type'] = np.where(
    df['Property_Type'].isin(['Residential', 'Single Family', 'Condo', 'Two F
    df['Residential_Type'],
    'Not Residence'
)
```

In [26]:

```
df.Residential_Type.value_counts()
```

Out[26]:

```
Residential_Type
Single Family    508044
Condo            136822
Not Residence     41184
Two Family        34848
Three Family      16567
Four Family        2959
Name: count, dtype: int64
```

In [27]:

```
df.Residential_Type.isnull().sum()
```

Out[27]:

```
np.int64(357205)
```

In [28]:

```
df[df['Residential_Type'].isna()][['Property_Type', 'Residential_Type']].head
```

Out[28]:

	Property_Type	Residential_Type
67	Residential	NaN
69	Residential	NaN
70	Residential	NaN
71	Residential	NaN
80	Residential	NaN

In [29]:

```
mean_sales = df.groupby(['Town', 'Residential_Type'])['Assessed_Value'].mean(
```

In [30]:

```
def fill_missing_property_type(row):
    if pd.isnull(row['Residential_Type']):

        town = row['Town']
        price = row['Assessed_Value']


        city_mean_values = mean_sales[mean_sales['Town'] == town]


        closest_type = city_mean_values.iloc[(city_mean_values['Assessed_Valu

        return closest_type['Residential_Type'].values[0]
    return row['Residential_Type']
```

```
df['Residential_Type'] = df.apply(fill_missing_property_type, axis=1)


print("The number of missing Residential_Type:", df['Residential_Type'].isnul
```

```
The number of missing Residential_Type: 0
```

In [31]:
```
df.Residential_Type.value_counts()
```

Out[31]:
```
Residential_Type
Single Family    531722
Condo            356120
Two Family        93916
Not Residence     46287
Three Family      41518
Four Family       28066
Name: count, dtype: int64
```

Location Missing Values

In [32]:
```
df.Town.unique()
```

Out[32]:
```
array(['Andover', 'Ansonia', 'Ashford', 'Avon', 'Beacon Falls', 'Berlin',
       'Branford', 'Bethany', 'Bethlehem', 'Bloomfield', 'Bethel',
       'Bridgeport', 'Bristol', 'Cheshire', 'Brookfield', 'Canaan',
       'Canton', 'Cornwall', 'Coventry', 'Chester', 'Colchester',
       'Columbia', 'Canterbury', 'Cromwell', 'Danbury', 'East Lyme',
       'Derby', 'Eastford', 'East Haddam', 'Greenwich', 'East Haven',
       'Farmington', 'Chaplin', 'Clinton', 'East Hampton', 'Easton',
       'Enfield', 'Ellington', 'Hamden', 'Fairfield', 'Essex', 'Hartford',
       'Durham', 'Franklin', 'Glastonbury', 'Killingly', 'Granby',
       'Ledyard', 'Guilford', 'Colebrook', 'Meriden', 'East Windsor',
       'Griswold', 'Bolton', 'Groton', 'Middlebury', 'Madison',
       'Mansfield', 'Harwinton', 'Milford', 'Killingworth', 'Lebanon',
       'Lisbon', 'Litchfield', 'Lyme', 'Manchester', 'New London',
       'New Britain', 'Putnam', 'Norwalk', 'New Milford', 'Norfolk',
       'Stafford', 'New Canaan', 'Sherman', 'North Haven', 'Stratford',
       'Roxbury', 'Oxford', 'New Haven', 'Old Lyme', 'Norwich', 'Sharon',
       'Monroe', 'Tolland', 'Torrington', 'Newtown', 'Naugatuck',
       'Ridgefield', 'Orange', 'New Fairfield', 'New Hartford', 'Somers',
       'Plainfield', 'Suffield', 'Plainville', 'Preston', 'West Haven',
       'Morris', 'Wallingford', 'Thompson', 'Stonington', 'Waterbury',
       'Stamford', 'Southbury', 'Newington', 'Vernon', 'Watertown',
       'West Hartford', 'Plymouth', 'Portland', 'Redding', 'Warren',
       'Rocky Hill', 'Salem', 'Winchester', 'Shelton', 'Simsbury',
       'Windsor', 'Woodbury', 'Montville', 'North Branford',
       'East Hartford', 'Hampton', 'Hebron', 'Darien', 'Burlington',
       'Middlefield', 'Brooklyn', 'Haddam', 'Barkhamsted', 'Kent',
       'East Granby', 'South Windsor', 'Deep River', 'Marlborough',
       'Hartland', 'Washington', 'Sterling', 'Thomaston', 'Waterford',
       'Trumbull', 'Old Saybrook', 'Westport', 'Westbrook', 'Middletown',
       'Goshen', 'Bridgewater', 'Wethersfield', 'Bozrah', 'Willington',
       'Wilton', 'Windsor Locks', 'Wolcott', 'Woodstock', 'Prospect',
       'Southington', 'Windham', 'Weston', 'Voluntown', 'North Canaan',
       'Scotland', 'Sprague', 'Pomfret', 'Seymour', 'Woodbridge', 'Union',
       'North Stonington', 'Salisbury', '***Unknown***'], dtype=object)
```

In [33]:
```
df = df[df['Town'] != '***Unknown***']

print(df['Town'].unique())
```

```
['Andover' 'Ansonia' 'Ashford' 'Avon' 'Beacon Falls' 'Berlin' 'Branford'
 'Bethany' 'Bethlehem' 'Bloomfield' 'Bethel' 'Bridgeport' 'Bristol'
 'Cheshire' 'Brookfield' 'Canaan' 'Canton' 'Cornwall' 'Coventry' 'Chester'
 'Colchester' 'Columbia' 'Canterbury' 'Cromwell' 'Danbury' 'East Lyme'
 'Derby' 'Eastford' 'East Haddam' 'Greenwich' 'East Haven' 'Farmington'
 'Chaplin' 'Clinton' 'East Hampton' 'Easton' 'Enfield' 'Ellington'
 'Hamden' 'Fairfield' 'Essex' 'Hartford' 'Durham' 'Franklin' 'Glastonbury'
 'Killingly' 'Granby' 'Ledyard' 'Guilford' 'Colebrook' 'Meriden'
 'East Windsor' 'Griswold' 'Bolton' 'Groton' 'Middlebury' 'Madison'
 'Mansfield' 'Harwinton' 'Milford' 'Killingworth' 'Lebanon' 'Lisbon'
 'Litchfield' 'Lyme' 'Manchester' 'New London' 'New Britain' 'Putnam'
 'Norwalk' 'New Milford' 'Norfolk' 'Stafford' 'New Canaan' 'Sherman'
 'North Haven' 'Stratford' 'Roxbury' 'Oxford' 'New Haven' 'Old Lyme'
 'Norwich' 'Sharon' 'Monroe' 'Tolland' 'Torrington' 'Newtown' 'Naugatuck'
 'Ridgefield' 'Orange' 'New Fairfield' 'New Hartford' 'Somers'
 'Plainfield' 'Suffield' 'Plainville' 'Preston' 'West Haven' 'Morris'
 'Wallingford' 'Thompson' 'Stonington' 'Waterbury' 'Stamford' 'Southbury'
 'Newington' 'Vernon' 'Watertown' 'West Hartford' 'Plymouth' 'Portland'
 'Redding' 'Warren' 'Rocky Hill' 'Salem' 'Winchester' 'Shelton' 'Simsbury'
 'Windsor' 'Woodbury' 'Montville' 'North Branford' 'East Hartford'
 'Hampton' 'Hebron' 'Darien' 'Burlington' 'Middlefield' 'Brooklyn'
 'Haddam' 'Barkhamsted' 'Kent' 'East Granby' 'South Windsor' 'Deep River'
 'Marlborough' 'Hartland' 'Washington' 'Sterling' 'Thomaston' 'Waterford'
 'Trumbull' 'Old Saybrook' 'Westport' 'Westbrook' 'Middletown' 'Goshen'
 'Bridgewater' 'Wethersfield' 'Bozrah' 'Willington' 'Wilton'
 'Windsor Locks' 'Wolcott' 'Woodstock' 'Prospect' 'Southington' 'Windham'
 'Weston' 'Voluntown' 'North Canaan' 'Scotland' 'Sprague' 'Pomfret'
 'Seymour' 'Woodbridge' 'Union' 'North Stonington' 'Salisbury']
```

In [34]:

```python
from geopy.geocoders import Nominatim
import pandas as pd
import time


towns = ['Andover', 'Ansonia', 'Ashford', 'Avon', 'Beacon Falls', 'Berlin', '
         'Cheshire', 'Brookfield', 'Canaan', 'Canton', 'Cornwall', 'Coventry'
         'Derby', 'Eastford', 'East Haddam', 'Greenwich', 'East Haven', 'Farm
         'Fairfield', 'Essex', 'Hartford', 'Durham', 'Franklin', 'Glastonbury
         'Griswold', 'Bolton', 'Groton', 'Middlebury', 'Madison', 'Mansfield'
         'New London', 'New Britain', 'Putnam', 'Norwalk', 'New Milford', 'No
         'New Haven', 'Old Lyme', 'Norwich', 'Sharon', 'Monroe', 'Tolland', '
         'Somers', 'Plainfield', 'Suffield', 'Plainville', 'Preston', 'West H
         'Newington', 'Vernon', 'Watertown', 'West Hartford', 'Plymouth', 'Po
         'Windsor', 'Woodbury', 'Montville', 'North Branford', 'East Hartford
         'Kent', 'East Granby', 'South Windsor', 'Deep River', 'Marlborough',
         'Westport', 'Westbrook', 'Middletown', 'Goshen', 'Bridgewater', 'Wet
         'Southington', 'Windham', 'Weston', 'Voluntown', 'North Canaan', 'Sc


geolocator = Nominatim(user_agent="town_locator")


coordinates = []
for town in towns:
    location = geolocator.geocode(f"{town}, CT, USA")
    if location:
        coordinates.append((town, location.latitude, location.longitude))
    else:
        coordinates.append((town, None, None))
    time.sleep(1)


df_coords = pd.DataFrame(coordinates, columns=['Town', 'Latitude', 'Longitude
```

```python
df_coords = pd.DataFrame(coordinates, columns=['Town', 'Latitude', 'Longitude'


print(df_coords)
```

```
              Town   Latitude  Longitude
0          Andover  41.737321 -72.370360
1          Ansonia  41.342992 -73.078747
2          Ashford  41.873153 -72.121465
3             Avon  41.809821 -72.830654
4     Beacon Falls  41.442875 -73.062608
5           Berlin  41.621488 -72.745652
6         Branford  41.279541 -72.815099
7          Bethany  41.421764 -72.997050
8        Bethlehem  41.639363 -73.208080
9       Bloomfield  41.826488 -72.730095
10          Bethel  41.371206 -73.414010
11      Bridgeport  41.179269 -73.188786
12         Bristol  41.673521 -72.946486
13        Cheshire  41.498986 -72.900658
14      Brookfield  41.482595 -73.409565
15          Canaan  41.961667 -73.308333
16          Canton  41.824542 -72.893712
17        Cornwall  41.843706 -73.329285
18        Coventry  41.770099 -72.305080
19         Chester  41.403155 -72.450920
20      Colchester  41.575654 -72.332027
21        Columbia  41.702043 -72.301192
22      Canterbury  41.698206 -71.970982
23        Cromwell  41.594994 -72.645567
24         Danbury  41.394817 -73.454011
25       East Lyme  41.356991 -72.225871
26           Derby  41.322361 -73.089032
27        Eastford  41.902068 -72.079909
28     East Haddam  41.452922 -72.461390
29       Greenwich  41.026486 -73.628460
30      East Haven  41.276208 -72.868434
31      Farmington  41.719822 -72.832043
32         Chaplin  41.794821 -72.127299
33         Clinton  41.278710 -72.527590
34    East Hampton  41.575844 -72.502480
35          Easton  41.252874 -73.297339
36         Enfield  41.978939 -72.575511
37       Ellington  41.903986 -72.469807
38          Hamden  41.395930 -72.896857
39       Fairfield  41.141208 -73.263726
40           Essex  41.332314 -72.395194
41        Hartford  41.764582 -72.690855
42          Durham  41.481765 -72.681206
43        Franklin  41.608987 -72.145911
44     Glastonbury  41.712322 -72.608146
45       Killingly  41.821529 -71.841571
46          Granby  41.953830 -72.789313
47         Ledyard  41.438605 -72.017519
48        Guilford  41.282759 -72.681636
49       Colebrook  41.989539 -73.095665
50         Meriden  41.538153 -72.807044
51    East Windsor  41.915632 -72.613073
52        Griswold  41.603503 -71.962244
53          Bolton  41.768988 -72.433417
54          Groton  41.350160 -72.076200
55      Middlebury  41.527874 -73.127611
56         Madison  41.279428 -72.598315
57       Mansfield  41.778215 -72.213156
```

58	Harwinton	41.771209	-73.059830
59	Milford	41.222222	-73.057060
60	Killingworth	41.358154	-72.563702
61	Lebanon	41.636210	-72.212579
62	Lisbon	41.586685	-72.020781
63	Litchfield	41.747319	-73.188724
64	Lyme	41.394496	-72.351035
65	Manchester	41.783402	-72.523197
66	New London	41.355619	-72.099780
67	New Britain	41.661210	-72.779542
68	Putnam	41.915309	-71.909256
69	Norwalk	41.117597	-73.407897
70	New Milford	41.577099	-73.410580
71	Norfolk	41.993983	-73.202058
72	Stafford	41.985196	-72.289581
73	New Canaan	41.146763	-73.494845
74	Sherman	41.579261	-73.495679
75	North Haven	41.390931	-72.859545
76	Stratford	41.184542	-73.133165
77	Roxbury	41.556828	-73.308892
78	Oxford	41.435179	-73.117277
79	New Haven	41.308214	-72.925052
80	Old Lyme	41.315931	-72.328971
81	Norwich	41.524354	-72.075901
82	Sharon	41.879260	-73.476790
83	Monroe	41.332596	-73.207336
84	Tolland	41.870017	-72.367715
85	Torrington	41.800652	-73.121221
86	Newtown	41.413476	-73.308644
87	Naugatuck	41.486019	-73.050943
88	Ridgefield	41.281484	-73.498179
89	Orange	41.278430	-73.025661
90	New Fairfield	41.466483	-73.485679
91	New Hartford	41.882319	-72.977049
92	Somers	41.985374	-72.446195
93	Plainfield	41.676488	-71.915073
94	Suffield	41.981694	-72.650660
95	Plainville	41.671140	-72.867243
96	Preston	41.526802	-71.982138
97	West Haven	41.270653	-72.947047
98	Morris	41.684263	-73.196224
99	Wallingford	41.457042	-72.823155
100	Thompson	41.958709	-71.862572
101	Stonington	41.335933	-71.905904
102	Waterbury	41.553809	-73.043836
103	Stamford	41.053430	-73.538734
104	Southbury	41.481485	-73.213169
105	Newington	41.697878	-72.723706
106	Vernon	41.838292	-72.466333
107	Watertown	41.606208	-73.118166
108	West Hartford	41.762045	-72.742040
109	Plymouth	41.672032	-73.052889
110	Portland	41.572892	-72.640691
111	Redding	41.302596	-73.383453
112	Warren	41.742873	-73.348730
113	Rocky Hill	41.664822	-72.639259
114	Salem	41.491269	-72.276208
115	Winchester	41.918742	-73.104500
116	Shelton	41.271777	-73.177061
117	Simsbury	41.875915	-72.801221
118	Windsor	41.852598	-72.643702
119	Woodbury	41.544540	-73.209002
120	Montville	41.464981	-72.153818
121	North Branford	41.327597	-72.767320
122	East Hartford	41.767914	-72.644512

```
122      East Hartford  41.767914 -72.644312
123            Hampton  41.783987 -72.054798
124             Hebron  41.657877 -72.365916
125             Darien  41.078708 -73.469287
126         Burlington  41.769265 -72.964548
127        Middlefield  41.516516 -72.712079
128           Brooklyn  41.788154 -71.949796
129             Haddam  41.477321 -72.512033
130        Barkhamsted  41.929263 -72.913990
131               Kent  41.724689 -73.476921
132        East Granby  41.941208 -72.727316
133      South Windsor  41.848987 -72.571755
134        Deep River   41.385655 -72.435642
135        Marlborough  41.631488 -72.459808
136           Hartland  41.996206 -72.979549
137         Washington  41.631484 -73.310673
138           Sterling  41.707599 -71.828682
139          Thomaston  41.673986 -73.073164
140          Waterford  41.358659 -72.151937
141           Trumbull  41.242874 -73.200669
142       Old Saybrook  41.291765 -72.376196
143           Westport  41.141486 -73.357895
144          Westbrook  41.286031 -72.448787
145         Middletown  41.562318 -72.650906
146             Goshen  41.831762 -73.225115
147        Bridgewater  41.535095 -73.366231
148       Wethersfield  41.714267 -72.652592
149             Bozrah  41.550596 -72.168238
150         Willington  41.874428 -72.259894
151             Wilton  41.195374 -73.437899
152      Windsor Locks  41.928131 -72.643631
153            Wolcott  41.602320 -72.986772
154          Woodstock  41.948431 -71.973963
155           Prospect  41.502319 -72.978716
156        Southington  41.600544 -72.878294
157            Windham  41.699354 -72.157824
158             Weston  41.202130 -73.381274
159          Voluntown  41.570654 -71.870350
160       North Canaan  42.027014 -73.329595
161           Scotland  41.698200 -72.082083
162            Sprague  41.621407 -72.066367
163            Pomfret  41.897598 -71.962574
164            Seymour  41.394358 -73.074170
165         Woodbridge  41.352597 -73.008438
166              Union  41.990930 -72.157299
167  North Stonington  41.441184 -71.881270
168          Salisbury  41.983426 -73.421232
```

In [35]:

```python
df = df.merge(df_coords, on='Town', how='left')
```

In [36]:

```python
df.head()
```

Out[36]:

	Serial_Number	List_Year	Date_Recorded	Town	Address	Assessed_Value	Sal
0	220008	2022	01/30/2023	Andover	618 ROUTE 6	139020.0	
1	2020348	2020	09/13/2021	Ansonia	230 WAKELEE AVE	150500.0	
					390		

2	20002	2020	10/02/2020	Ashford	TURNPIKE RD	253000.0
3	210317	2021	07/05/2022	Avon	53 COTSWOLD WAY	329730.0
4	200212	2020	03/09/2021	Avon	5 CHESTNUT DRIVE	130400.0

In [37]:

```
df.loc[df['Location'].isna(), 'Location'] = df['Longitude'].astype(str) + ',
```

In [38]:

```
df.head()
```

Out[38]:

	Serial_Number	List_Year	Date_Recorded	Town	Address	Assessed_Value	Sal
0	220008	2022	01/30/2023	Andover	618 ROUTE 6	139020.0	
1	2020348	2020	09/13/2021	Ansonia	230 WAKELEE AVE	150500.0	
2	20002	2020	10/02/2020	Ashford	390 TURNPIKE RD	253000.0	
3	210317	2021	07/05/2022	Avon	53 COTSWOLD WAY	329730.0	
4	200212	2020	03/09/2021	Avon	5 CHESTNUT DRIVE	130400.0	

In [39]:

```
df.Location.isnull().sum()
```

Out[39]:

```
np.int64(0)
```

Date Format: Can we convert Date Recorded into a proper datetime format for better analysis?

In [40]:

```
df.Date_Recorded.value_counts().head(5)
```

Out[40]:

```
Date_Recorded
07/01/2005    877
08/01/2005    859
07/01/2004    840
06/30/2005    828
09/30/2005    781
Name: count, dtype: int64
```

In [41]:
```
df.Date_Recorded.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 1097628 entries, 0 to 1097627
Series name: Date_Recorded
Non-Null Count    Dtype
--------------    -----
1097626 non-null  object
dtypes: object(1)
memory usage: 8.4+ MB
```

In [42]:
```
df.Date_Recorded.isnull().sum()
```

Out[42]:
```
np.int64(2)
```

In [43]:
```
df.dropna(subset=['Date_Recorded'], inplace=True)
```

In [44]:
```
df['Date_Recorded'] = pd.to_datetime(df['Date_Recorded'])
```

In [45]:
```
df.Date_Recorded.info()
```

```
<class 'pandas.core.series.Series'>
Index: 1097626 entries, 0 to 1097627
Series name: Date_Recorded
Non-Null Count    Dtype
--------------    -----
1097626 non-null  datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 16.7 MB
```

String Cleaning: Should the Address and Town columns be cleaned for uniformity (e.g., removing excess spaces, fixing capitalization)?

Address and town columns were examined and it was concluded that no data manipulation is necessary.

In [46]:
```
df.Town.unique()
```

Out[46]:
```
array(['Andover', 'Ansonia', 'Ashford', 'Avon', 'Beacon Falls', 'Berlin',
       'Branford', 'Bethany', 'Bethlehem', 'Bloomfield', 'Bethel',
       'Bridgeport', 'Bristol', 'Cheshire', 'Brookfield', 'Canaan',
       'Canton', 'Cornwall', 'Coventry', 'Chester', 'Colchester',
       'Columbia', 'Canterbury', 'Cromwell', 'Danbury', 'East Lyme',
       'Derby', 'Eastford', 'East Haddam', 'Greenwich', 'East Haven',
       'Farmington', 'Chaplin', 'Clinton', 'East Hampton', 'Easton',
       'Enfield', 'Ellington', 'Hamden', 'Fairfield', 'Essex', 'Hartford',
       'Durham', 'Franklin', 'Glastonbury', 'Killingly', 'Granby',
       'Ledyard', 'Guilford', 'Colebrook', 'Meriden', 'East Windsor',
       'Griswold', 'Bolton', 'Groton', 'Middlebury', 'Madison',
       'Mansfield', 'Harwinton', 'Milford', 'Killingworth', 'Lebanon',
       'Lisbon', 'Litchfield', 'Lyme', 'Manchester', 'New London',
       'New Britain', 'Putnam', 'Norwalk', 'New Milford', 'Norfolk',
       'Stafford', 'New Canaan', 'Sherman', 'North Haven', 'Stratford',
       'Roxbury', 'Oxford', 'New Haven', 'Old Lyme', 'Norwich', 'Sharon',
```

```
       'Monroe', 'Tolland', 'Torrington', 'Newtown', 'Naugatuck',
       'Ridgefield', 'Orange', 'New Fairfield', 'New Hartford', 'Somers',
       'Plainfield', 'Suffield', 'Plainville', 'Preston', 'West Haven',
       'Morris', 'Wallingford', 'Thompson', 'Stonington', 'Waterbury',
       'Stamford', 'Southbury', 'Newington', 'Vernon', 'Watertown',
       'West Hartford', 'Plymouth', 'Portland', 'Redding', 'Warren',
       'Rocky Hill', 'Salem', 'Winchester', 'Shelton', 'Simsbury',
       'Windsor', 'Woodbury', 'Montville', 'North Branford',
       'East Hartford', 'Hampton', 'Hebron', 'Darien', 'Burlington',
       'Middlefield', 'Brooklyn', 'Haddam', 'Barkhamsted', 'Kent',
       'East Granby', 'South Windsor', 'Deep River', 'Marlborough',
       'Hartland', 'Washington', 'Sterling', 'Thomaston', 'Waterford',
       'Trumbull', 'Old Saybrook', 'Westport', 'Westbrook', 'Middletown',
       'Goshen', 'Bridgewater', 'Wethersfield', 'Bozrah', 'Willington',
       'Wilton', 'Windsor Locks', 'Wolcott', 'Woodstock', 'Prospect',
       'Southington', 'Windham', 'Weston', 'Voluntown', 'North Canaan',
       'Scotland', 'Sprague', 'Pomfret', 'Seymour', 'Woodbridge', 'Union',
       'North Stonington', 'Salisbury'], dtype=object)
```

In [47]:

```python
df.Address.sample(50)
```

Out[47]:

```
504792                  19 VILLAGE RD
627099                 78 TEMPLE DRIVE
1004873              63 SHERWOOD PLACE
509550                 68 BATTERSON DR
935089                 13 RIVERSIDE DR
55415        300 FLAX HILL RD UNIT 12B
379467                    13F BYRNE CT
245617                 CALKINSTOWN RD
200532                    25 C AMATO DR
143801               335 WESTPORT ROAD
592589              18 STADLEY ROUGH RD
748552                 6-13 BAMFORTH RD
144478                   23 LAKE RIDGE
534731                   7 SILVER BRK RD
108533           118 WASHINGTON ST 309
870046               8 SOUTHEAST TRAIL
160230                   71 MAXWELL DR
811850                 48 ESSEX STREET
7132                      114 FITCH ST
732614                 372 MAIN STREET
2110                     110 BENHAM ST
579013                 198 BROOKSIDE RD
309226                  2 HOMESTEAD LN
533840                     20 TERRIE RD
990961                7 DECUBELLIS CRT
253789          11 OLD STAGECOACH CROSS
171499                     205 WELLS RD
954770                      63 LYMAN RD
451465                   4 CRAWFORD RD
555854                  11 CLOVERLY CIR
972632                  16 TEN STREET
364184                  31 HIGH ST 7104
1055731                  25 SYLVAN TERR
150051                    8 FLICKER LN
680448               942 RIVERBANK ROAD
241350                  2300 GLASGO RD
34773                  41 ALLENDALE RD
1017663                     78 TOSUN RD
37551                   41 COLONIAL DR
1083004              122 WATERBURY RD
629599         84 PERRY STREET UNIT 220
973095             71 LANTERN RIDGE RD
491786           1022-1024 CAPITOL AVE
```

```
871429              455 WESTPORT ROAD
384656            4 SAUGATUCK RDG RD
45978                    25 ALDEN PL
437251             61-63 GRIDLEY ST
689757             28 WALNUT STREET
881474              50 ANTHONY ROAD
1072896             117 MICHALEC RD
Name: Address, dtype: object
```

Location Parsing: Should we split the Location column into latitude and longitude for easier geospatial analysis?

The Location column was populated based on the data in the Town column as a result of the operations performed above, and two new columns, Latitude and Longitude, were added to the DataFrame

In [48]:

```
df[['Town', 'Location', 'Latitude', 'Longitude']].head(10)
```

Out[48]:

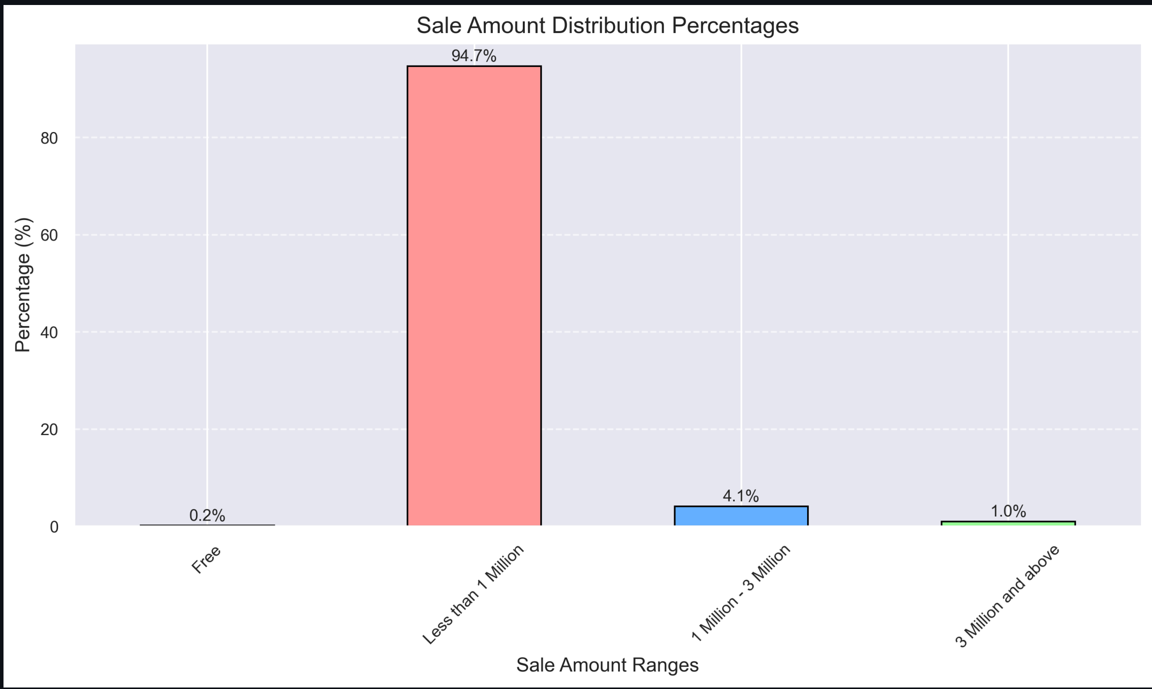
	Town	Location	Latitude	Longitude
0	Andover	POINT (-72.343628962 41.728431984)	41.737321	-72.370360
1	Ansonia	-73.0787468, 41.3429922	41.342992	-73.078747
2	Ashford	-72.1214653, 41.8731532	41.873153	-72.121465
3	Avon	POINT (-72.846365959 41.781677018)	41.809821	-72.830654
4	Avon	-72.8306541, 41.8098209	41.809821	-72.830654
5	Avon	-72.8306541, 41.8098209	41.809821	-72.830654
6	Avon	-72.8306541, 41.8098209	41.809821	-72.830654
7	Beacon Falls	POINT (-73.053071989 41.439434021)	41.442875	-73.062608
8	Avon	-72.8306541, 41.8098209	41.809821	-72.830654
9	Berlin	-72.7456519, 41.621488	41.621488	-72.745652

Outlier Detection: Are there outliers in Sale Amount or Assessed Value, and how should they be handled?

In [49]:

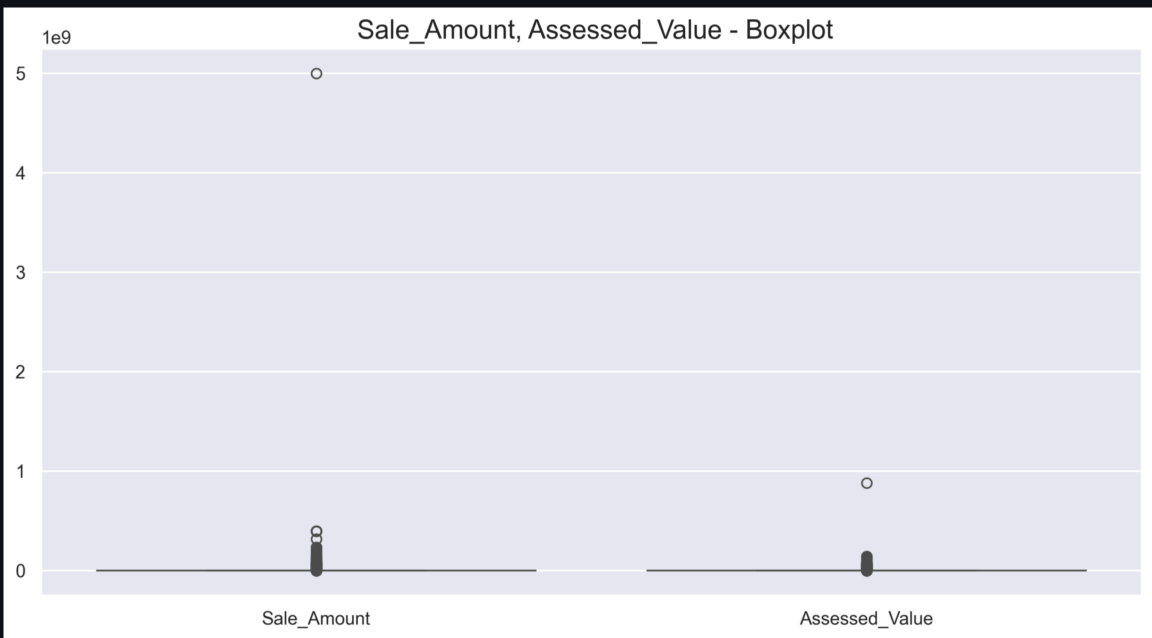
```
bins = [0, 1, 1000000, 3000000, float('inf')]
labels = ['Free', 'Less than 1 Million', '1 Million - 3 Million', '3 Million

category_counts = pd.cut(df['Sale_Amount'], bins=bins, labels=labels, right=F

plt.figure(figsize=(10, 6))
colors = ['#FFCC99', '#FF9999', '#66B3FF', '#99FF99']
category_counts.sort_index().plot(kind='bar', color=colors, edgecolor='black'

plt.title('Sale Amount Distribution Percentages', fontsize=14)
plt.xlabel('Sale Amount Ranges', fontsize=12)
plt.ylabel('Percentage (%)', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)

for i, value in enumerate(category_counts.sort_index()):
```

```python
    plt.text(i, value + 1, f'{value:.1f}%', ha='center', fontsize=10)

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



In [50]:

```python
plt.figure(figsize=(12, 6))

sns.boxplot(data=df[['Sale_Amount', 'Assessed_Value']])


plt.title(" Sale_Amount, Assessed_Value - Boxplot", fontsize=16)

plt.show()
```



In [51]:

```python
print(df[['Sale_Amount', 'Assessed_Value']].describe())
```

```
        Sale_Amount  Assessed_Value
count  1.097626e+06    1.097626e+06
mean   4.053154e+05    2.818023e+05
```

```
std    5.143499e+06   1.657892e+06
min    0.000000e+00   0.000000e+00
25%    1.450000e+05   8.909000e+04
50%    2.330000e+05   1.405800e+05
75%    3.750000e+05   2.282700e+05
max    5.000000e+09   8.815100e+08
```

In [52]:

```
Q1 = df[['Sale_Amount', 'Assessed_Value']].quantile(0.25)
Q3 = df[['Sale_Amount', 'Assessed_Value']].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 3 * IQR
upper_bound = Q3 + 3 * IQR

lower_bound = lower_bound.reindex(df[['Sale_Amount', 'Assessed_Value']].colum
upper_bound = upper_bound.reindex(df[['Sale_Amount', 'Assessed_Value']].colum

outliers = (df[['Sale_Amount', 'Assessed_Value']] < lower_bound) | (df[['Sale

print("Number of Outliers - Sale Amount:", outliers['Sale_Amount'].sum())
print("Number of Outliers - Assessed Value:", outliers['Assessed_Value'].sum(
```

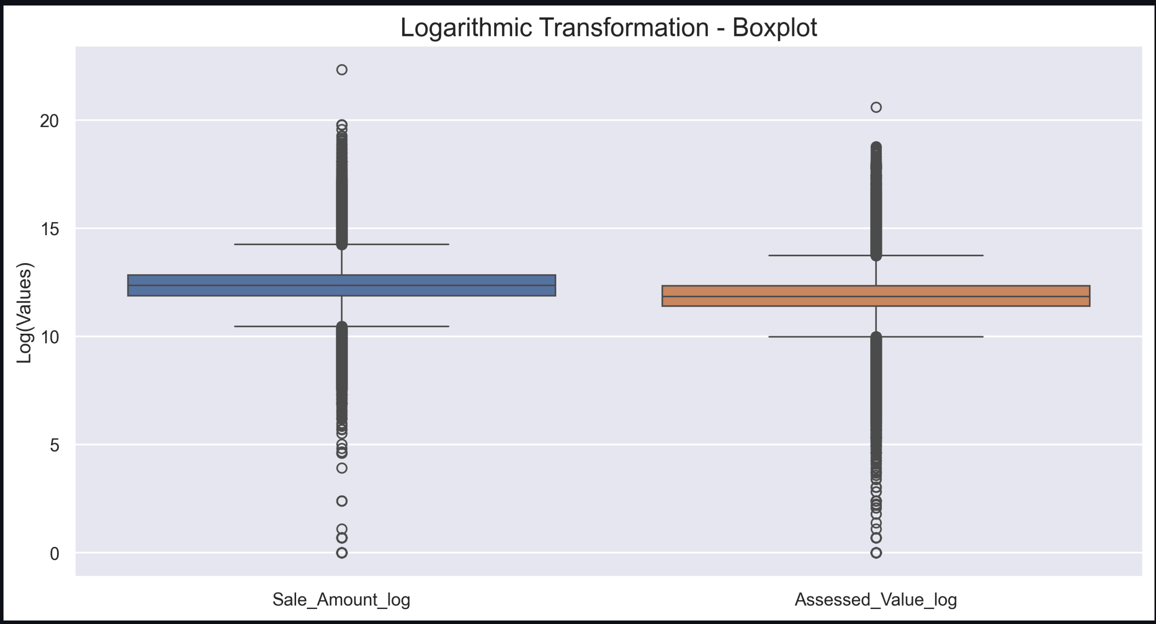
```
Number of Outliers - Sale Amount: 51858
Number of Outliers - Assessed Value: 57407
```

In [53]:

```
df['Sale_Amount_log'] = np.log1p(df['Sale_Amount'])
df['Assessed_Value_log'] = np.log1p(df['Assessed_Value'])
```

In [54]:

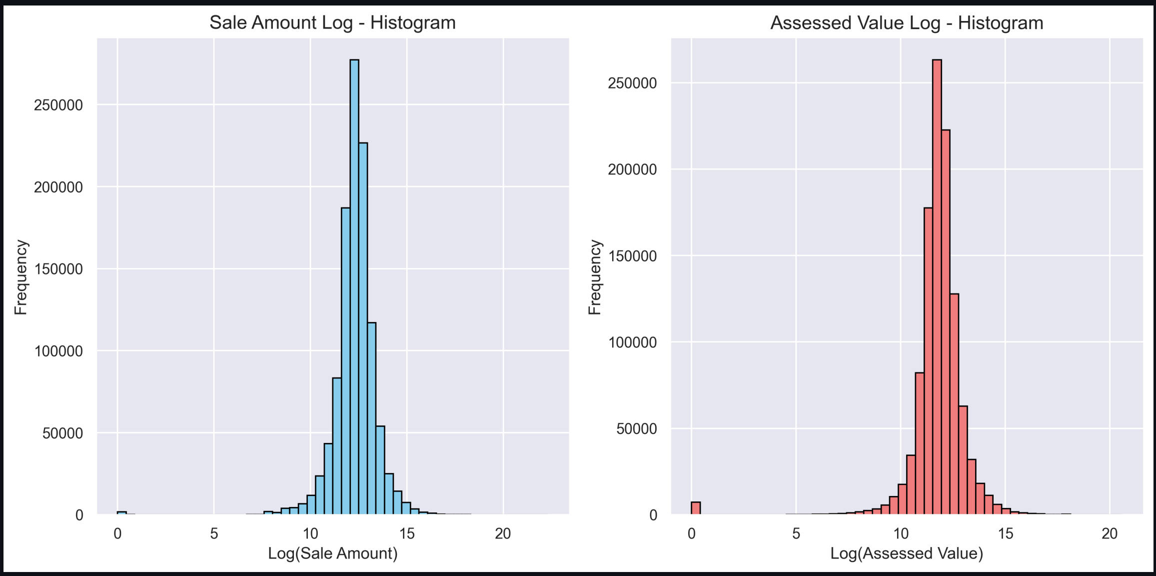
```
plt.figure(figsize=(12, 6))

sns.boxplot(data=df[['Sale_Amount_log', 'Assessed_Value_log']])

plt.title("Logarithmic Transformation - Boxplot", fontsize=16)
plt.ylabel("Log(Values)", fontsize=12)
plt.show()
```



In [55]:

```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(df['Sale_Amount_log'], bins=50, color='skyblue', edgecolor='black')
```

```
plt.title("Sale Amount Log - Histogram", fontsize=14)
plt.xlabel("Log(Sale Amount)", fontsize=12)
plt.ylabel("Frequency", fontsize=12)

plt.subplot(1, 2, 2)
plt.hist(df['Assessed_Value_log'], bins=50, color='lightcoral', edgecolor='bl
plt.title("Assessed Value Log - Histogram", fontsize=14)
plt.xlabel("Log(Assessed Value)", fontsize=12)
plt.ylabel("Frequency", fontsize=12)

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



In [56]:

```
# Sale_Amount_log ve Assessed_Value_log için describe
log_summary = df[['Sale_Amount_log', 'Assessed_Value_log']].describe()
print(log_summary)
```

```
       Sale_Amount_log  Assessed_Value_log
count     1.097626e+06        1.097626e+06
mean      1.231562e+01        1.180209e+01
std       1.095200e+00        1.365230e+00
min       0.000000e+00        0.000000e+00
25%       1.188450e+01        1.139741e+01
50%       1.235880e+01        1.185354e+01
75%       1.283468e+01        1.233829e+01
max       2.233270e+01        2.059715e+01
```

Based on the missing data analysis, the sale amount higher than 3 million are specified as outliers, and deleted from data for cleaning purpose

Eksik veri analizine dayanarak, 3 milyondan fazla satış tutarı uç değer olarak belirlenir ve temizleme amacıyla verilerden silinir.

In [57]:

```
data = df[(df['Sale_Amount'] <= 3000000) & (df['Assessed_Value'] <= 3000000)]

print("Rows with values above 3 million in either Sale_Amount or Assessed_Val
```

```
Rows with values above 3 million in either Sale_Amount or Assessed_Value have b
een removed. New DataFrame shape: (1083606, 18)
```

In [58]:

```
bins = [0, 1, 1000000, 3000000, float('inf')]
labels = ['Free', 'Less than 1 Million', '1 Million - 3 Million', '3 Million
```

```
category_counts = pd.cut(data['Sale_Amount'], bins=bins, labels=labels, right

plt.figure(figsize=(10, 6))
colors = ['#FFCC99', '#FF9999', '#66B3FF', '#99FF99']
category_counts.sort_index().plot(kind='bar', color=colors, edgecolor='black'

plt.title('Sale Amount Distribution Percentages', fontsize=14)
plt.xlabel('Sale Amount Ranges', fontsize=12)
plt.ylabel('Percentage (%)', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)

for i, value in enumerate(category_counts.sort_index()):
    plt.text(i, value + 1, f'{value:.1f}%', ha='center', fontsize=10)

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

In [59]:

```
data.Sale_Amount.describe()
```

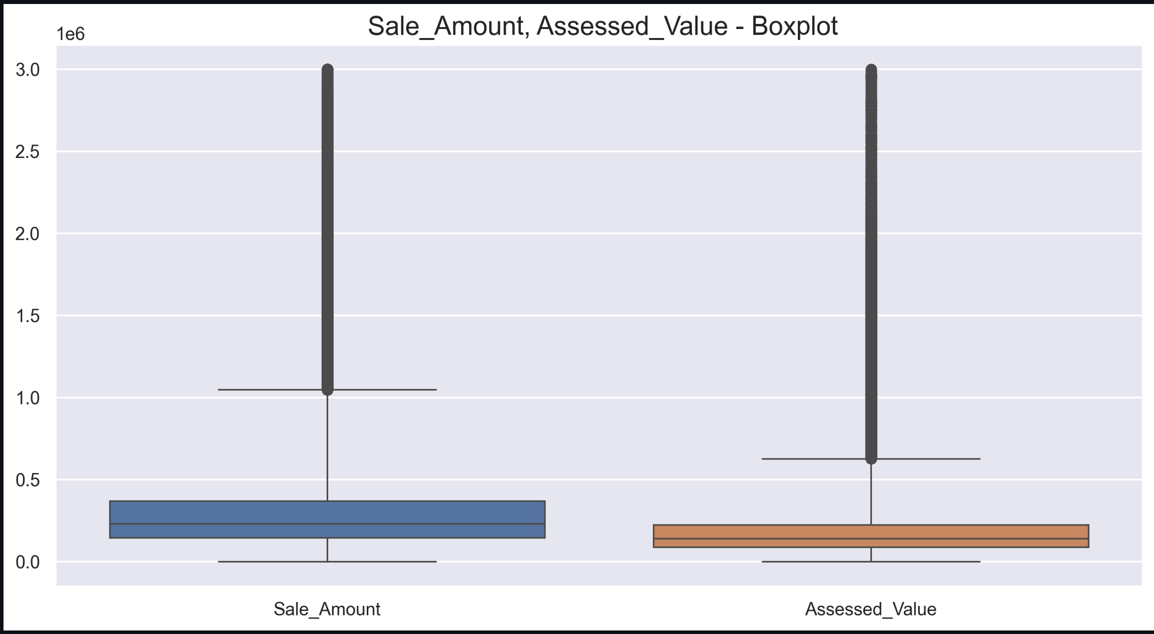
Out[59]:

```
count    1.083606e+06
mean     3.220228e+05
std      3.403882e+05
min      0.000000e+00
25%      1.442000e+05
50%      2.300000e+05
75%      3.700000e+05
max      3.000000e+06
Name: Sale_Amount, dtype: float64
```

In [60]:

```
plt.figure(figsize=(12, 6))

sns.boxplot(data=data[['Sale_Amount', 'Assessed_Value']], whis = 3)


plt.title(" Sale_Amount, Assessed_Value - Boxplot", fontsize=16)

plt.show()
```

```
plt.show()
```



In [61]:

```
print(data[['Sale_Amount', 'Assessed_Value']].describe())
```

```
        Sale_Amount  Assessed_Value
count  1.083606e+06    1.083606e+06
mean   3.220228e+05    2.017832e+05
std    3.403882e+05    2.348004e+05
min    0.000000e+00    0.000000e+00
25%    1.442000e+05    8.852000e+04
50%    2.300000e+05    1.392300e+05
75%    3.700000e+05    2.231800e+05
max    3.000000e+06    2.999710e+06
```

In [62]:

```
Q1 = data[['Sale_Amount', 'Assessed_Value']].quantile(0.25)
Q3 = data[['Sale_Amount', 'Assessed_Value']].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 3 * IQR
upper_bound = Q3 + 3 * IQR

lower_bound = lower_bound.reindex(data[['Sale_Amount', 'Assessed_Value']].col
upper_bound = upper_bound.reindex(data[['Sale_Amount', 'Assessed_Value']].col

outliers = (data[['Sale_Amount', 'Assessed_Value']] < lower_bound) | (data[['

print("Number of outliers - Sale Amount:", outliers['Sale_Amount'].sum())
print("Number of outliers - Assessed Value:", outliers['Assessed_Value'].sum(
```

```
Number of outliers - Sale Amount: 42230
Number of outliers - Assessed Value: 46604
```

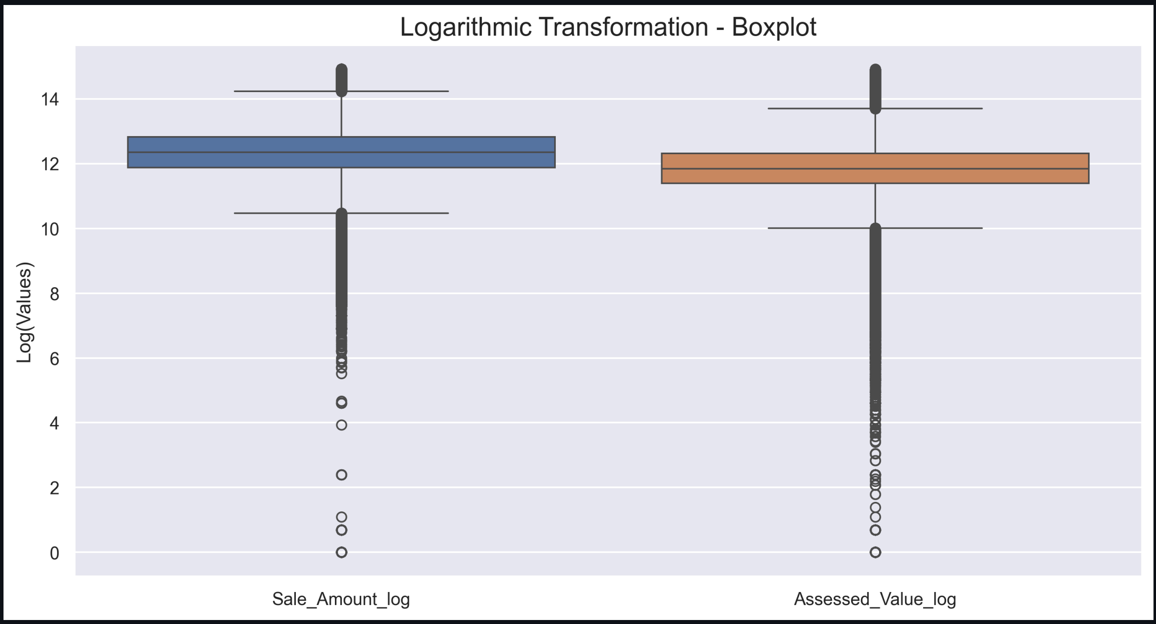
In [63]:

```
data['Sale_Amount_log'] = np.log1p(data['Sale_Amount'])
data['Assessed_Value_log'] = np.log1p(data['Assessed_Value'])
```

In [64]:

```
plt.figure(figsize=(12, 6))

sns.boxplot(data=data[['Sale_Amount_log', 'Assessed_Value_log']])

plt.title("Logarithmic Transformation - Boxplot", fontsize=16)
```

```
plt.ylabel("Log(Values)", fontsize=12)
plt.show()
```



Data Type Correction: Are any columns incorrectly typed? For instance, Non Use Code might be categorical instead of object.

There is no column with an incorrect data type; no adjustments are needed.

In [65]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1083606 entries, 0 to 1097626
Data columns (total 18 columns):
 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   Serial_Number       1083606 non-null  int64
 1   List_Year           1083606 non-null  int64
 2   Date_Recorded       1083606 non-null  datetime64[ns]
 3   Town                1083606 non-null  object
 4   Address             1083558 non-null  object
 5   Assessed_Value      1083606 non-null  float64
 6   Sale_Amount         1083606 non-null  float64
 7   Sales_Ratio         1083606 non-null  float64
 8   Property_Type       1083606 non-null  object
 9   Residential_Type    1083606 non-null  object
 10  Non_Use_Code        306445 non-null   object
 11  Assessor_Remarks    166614 non-null   object
 12  OPM_remarks         12889 non-null    object
 13  Location            1083606 non-null  object
 14  Latitude            1083606 non-null  float64
 15  Longitude           1083606 non-null  float64
 16  Sale_Amount_log     1083606 non-null  float64
 17  Assessed_Value_log  1083606 non-null  float64
dtypes: datetime64[ns](1), float64(7), int64(2), object(8)
memory usage: 157.1+ MB
```

Derived Features: Should we create a new feature, such as the ratio of Assessed Value to Sale Amount, for further insights?

There is a repetitive sale of the same properties.A new dataset was created for a more detailed analysis based on the first and last listing years of repeated addresses. New data was produced for price changes and repeat sales of real estate over time.

In [66]:

```
duplicate_addresses = data[data.duplicated(subset='Address', keep=False)]
town_data = data[['Address', 'Town','Property_Type','Residential_Type']].drop
```

In [67]:

```
first_year_data = duplicate_addresses[duplicate_addresses['List_Year'] == dup
last_year_data = duplicate_addresses[duplicate_addresses['List_Year'] == dupl

first_year_data = first_year_data[['Address', 'List_Year', 'Sale_Amount']].re
last_year_data = last_year_data[['Address', 'List_Year', 'Sale_Amount']].rena

merged_data = pd.merge(first_year_data, last_year_data, on='Address', how='in

duplicated_adress = merged_data[merged_data['First_Year'] != merged_data['Las



duplicated_adress.head (10)
```

Out[67]:

	Address	First_Year	First_Year_Sale_Amount	Last_Year	Last_Year_Sale_Amount
2	1963 MAIN ST	2021	500000.0	2022	500000.0
3	2075 MAIN ST	2020	25000.0	2021	49900.0
4	GENERAL LYON RD	2020	20000.0	2022	70000.0
5	5 SPARROW LN	2001	302866.0	2003	400000.0
6	50 FOURTH ST	2001	50000.0	2005	169900.0
7	FEDERAL RD	2001	35000.0	2021	295000.0
8	42 PRATT RD	2001	230000.0	2017	178000.0
9	50 RIVERSIDE LANE	2020	300000.0	2021	1185000.0
10	WESTFORD RD	2001	5000.0	2021	40000.0
11	KASSON RD	2001	105000.0	2002	65000.0

In [68]:

```
#Adding the Town Column
merged_data = pd.merge(duplicated_adress, town_data, on='Address', how='left'
```

In [69]:

```
merged_data.head(5)
```

Out[69]:

	Address	First_Year	First_Year_Sale_Amount	Last_Year	Last_Year_Sale_Amount	
0	1963 MAIN ST	2021	500000.0	2022	500000.0	Br
1	2075 MAIN ST	2020	25000.0	2021	49900.0	
2	GENERAL LYON RD	2020	20000.0	2022	70000.0	
3	5 SPARROW LN	2001	302866.0	2003	400000.0	Fa
4	5 SPARROW LN	2001	302866.0	2003	400000.0	Fa

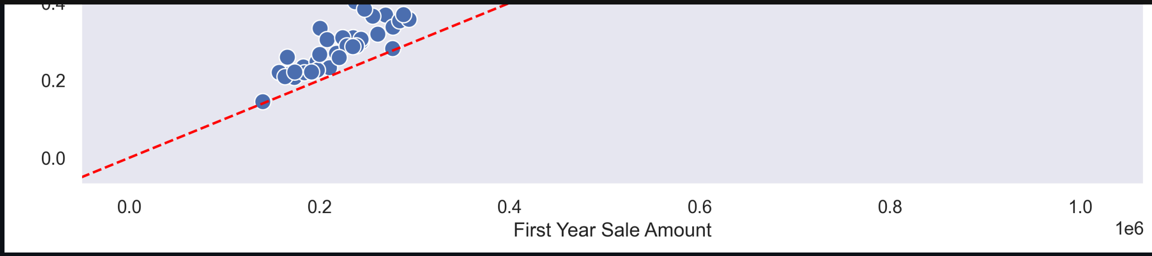
In [70]:

```
grouped_data = merged_data.groupby('Town').agg({
    'First_Year_Sale_Amount': 'mean',
    'Last_Year_Sale_Amount': 'mean'
}).reset_index()
```

The mean difference between first year listing sale amount and last year listing sale amount by town

In [71]:

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='First_Year_Sale_Amount', y='Last_Year_Sale_Amount', data=g

plt.title('First vs Last Year Sale Amounts')
plt.xlabel('First Year Sale Amount')
plt.ylabel('Last Year Sale Amount')
plt.axline((0, 0), slope=1, color='red', linestyle='--')
plt.grid()
plt.tight_layout()
plt.show()
```



In [72]:

```
import plotly.express as px

# Assuming 'grouped_data' is your DataFrame with columns: 'First_Year_Sale_Am
fig = px.scatter(grouped_data,
                 x='First_Year_Sale_Amount',
                 y='Last_Year_Sale_Amount',
                 hover_name='Town',  # This will show the town name when you 
                 title='First vs Last Year Sale Amounts',
                 labels={'First_Year_Sale_Amount': 'First Year Sale Amount',
                         'Last_Year_Sale_Amount': 'Last Year Sale Amount'},
                 template='plotly_dark')

# Add a diagonal line (slope = 1)
fig.add_shape(
    type='line',
    x0=0, y0=0,
    x1=max(grouped_data['First_Year_Sale_Amount'].max(), grouped_data['Last_Y
    y1=max(grouped_data['First_Year_Sale_Amount'].max(), grouped_data['Last_Y
    line=dict(color='red', dash='dash')
)

# Show the plot
fig.update_traces(marker=dict(size=10))  # You can adjust marker size here
fig.show()
```

In [73]:

```
grouped_data['distance'] = np.abs(grouped_data['First_Year_Sale_Amount'] - gr

outlier_row = grouped_data.loc[grouped_data['distance'].idxmax()]

outlier_town = outlier_row['Town']
print(f"The town corresponding to the outlier is: {outlier_town}")
print(outlier_row)
```

```
The town corresponding to the outlier is: New Canaan
Town                          New Canaan
First_Year_Sale_Amount    1016234.792431
Last_Year_Sale_Amount     1361788.681193
distance                   345553.888761
Name: 89, dtype: object
```

In [74]:

```
grouped_data.describe().T
```

Out[74]:

	count	mean	std	min	
First_Year_Sale_Amount	169.0	310909.008473	97335.007219	140223.412698	289283.1
Last_Year_Sale_Amount	169.0	483797.015248	143530.632380	145813.460317	466270.3
distance	169.0	172888.006775	66555.239987	5590.047619	136721.0

Filtering by Year: Should we create subsets of the data for specific years (e.g., analyzing post-2020 sales)?

In [75]:

```
data['Year'] = pd.to_datetime(data['Date_Recorded']).dt.year
filtered_df = data[data['Year'] >= 2020]
filtered_df.head()
```

Out[75]:

	Serial_Number	List_Year	Date_Recorded	Town	Address	Assessed_Value	Sal
0	220008	2022	2023-01-30	Andover	618 ROUTE 6	139020.0	
1	2020348	2020	2021-09-13	Ansonia	230 WAKELEE AVE	150500.0	
2	20002	2020	2020-10-02	Ashford	390 TURNPIKE RD	253000.0	
3	210317	2021	2022-07-05	Avon	53 COTSWOLD WAY	329730.0	
4	200212	2020	2021-03-09	Avon	5 CHESTNUT DRIVE	130400.0	

In [76]:

```
data['Year'] = pd.to_datetime(data['Date_Recorded']).dt.year
filtered_df2 = data[(data['Year'] >= 2010) & (data['Year'] < 2020)]
filtered_df2.head()
```

Out[76]:

	Serial_Number	List_Year	Date_Recorded	Town	Address	Assessed_Val
167746	900303	2009	2010-02-08	Bristol	BELGIAN CIR LOT 12-8	9443
167747	90068	2009	2010-03-04	Derby	2 HAWTHORNE PLACE	10850
167748	90245	2009	2010-07-01	Avon	15 HERITAGE DRIVE	14515
167749	90009	2009	2010-05-13	Chaplin	1 HAMPTON RD	17260
167750	900729	2009	2010-08-04	Bristol	681 BROAD ST	44849

In [77]:

```
data['Year'] = pd.to_datetime(data['Date_Recorded']).dt.year
filtered_df3 = data[(data['Year'] >= 2002) & (data['Year'] < 2010)]
filtered_df3.head()
```

Out[77]:

	Serial_Number	List_Year	Date_Recorded	Town	Address	Assessed_Value	Sa
69	11238	2001	2002-08-30	Bethel	50 FOURTH ST	76450.0	
70	10035	2001	2002-06-28	Chaplin	FEDERAL RD	19000.0	
71	10115	2001	2002-02-01	Clinton	42 PRATT RD	121900.0	
80	10041	2001	2002-07-17	Eastford	WESTFORD RD	180.0	
96	11665	2001	2002-07-30	Danbury	3 MORGAN AVE	94600.0	

Handling Sparse Columns: Columns like OPM Remarks and Assessor Remarks are mostly empty. Should they be dropped?

The columns 'OPM Remarks' and 'Assessor Remarks' have been removed due to the high number of missing values and their lack of contribution to the analysis.

In [78]:

```
data = data.drop(columns=['OPM_remarks', 'Assessor_Remarks'])
```

In [79]:

```
data.columns
```

Out[79]:

```
Index(['Serial_Number', 'List_Year', 'Date_Recorded', 'Town', 'Address',
       'Assessed_Value', 'Sale_Amount', 'Sales_Ratio', 'Property_Type',
       'Residential_Type', 'Non_Use_Code', 'Location', 'Latitude', 'Longitud
e',
       'Sale_Amount_log', 'Assessed_Value_log', 'Year'],
      dtype='object')
```

10 Data Visualization Questions

What is the distribution of Sale Amount and how does it vary across different Property Types?

In [80]:

```
df.Property_Type.unique()
```

Out[80]:

```
array(['Residential', 'Commercial', 'Vacant Land', 'Apartments',
       'Industrial', 'Single Family', 'Public Utility', 'Condo',
       'Two Family', 'Three Family', 'Four Family'], dtype=object)
```

In [81]:

```
df2 = data.copy()
```

In [82]:

```
df2['Property_Type'] = df['Property_Type'].replace({
```

```
    'Single Family': 'Residential',
    'Two Family': 'Residential',
    'Three Family': 'Residential',
    'Four Family': 'Residential',
    'Condo': 'Residential'
})
```

In [83]:

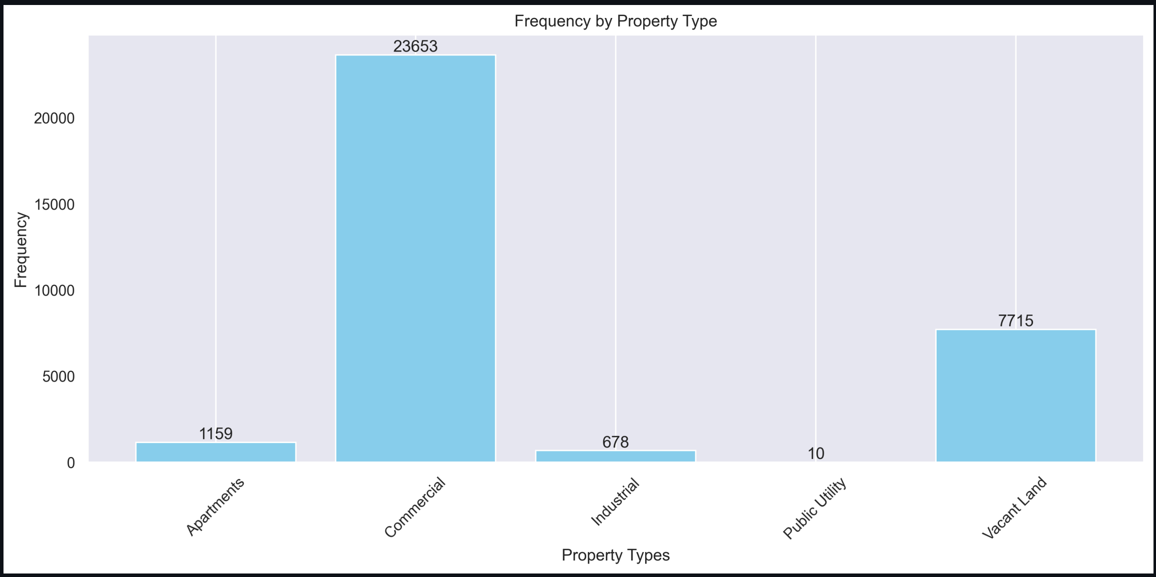
```
df2.Property_Type.unique()
```

Out[83]:

```
array(['Residential', 'Commercial', 'Vacant Land', 'Apartments',
       'Industrial', 'Public Utility'], dtype=object)
```

In [84]:

```
grouped_data = df2[df2['Property_Type'] != 'Residential'].groupby('Property_T

plt.figure(figsize=(12, 6))
plt.bar(grouped_data['Property_Type'], grouped_data['Sale_Amount'], color='sk

for index, value in enumerate(grouped_data['Sale_Amount']):
    plt.text(index, value, str(value), ha='center', va='bottom')


plt.title('Frequency by Property Type')
plt.xlabel('Property Types')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.grid(axis='y')

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



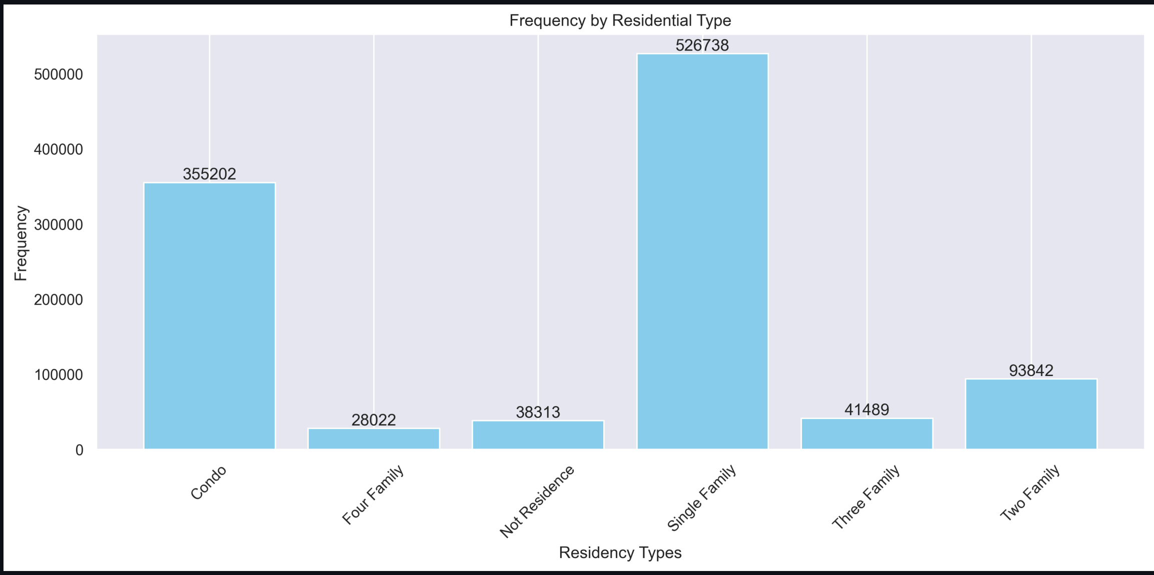
Residential is a outlier for Property type so it was removed from the graph for better understanding of other property types' distributions...

This graph was created based on the remaining property types. According to the graph, the highest sales frequency is for the commercial type.

In [85]:

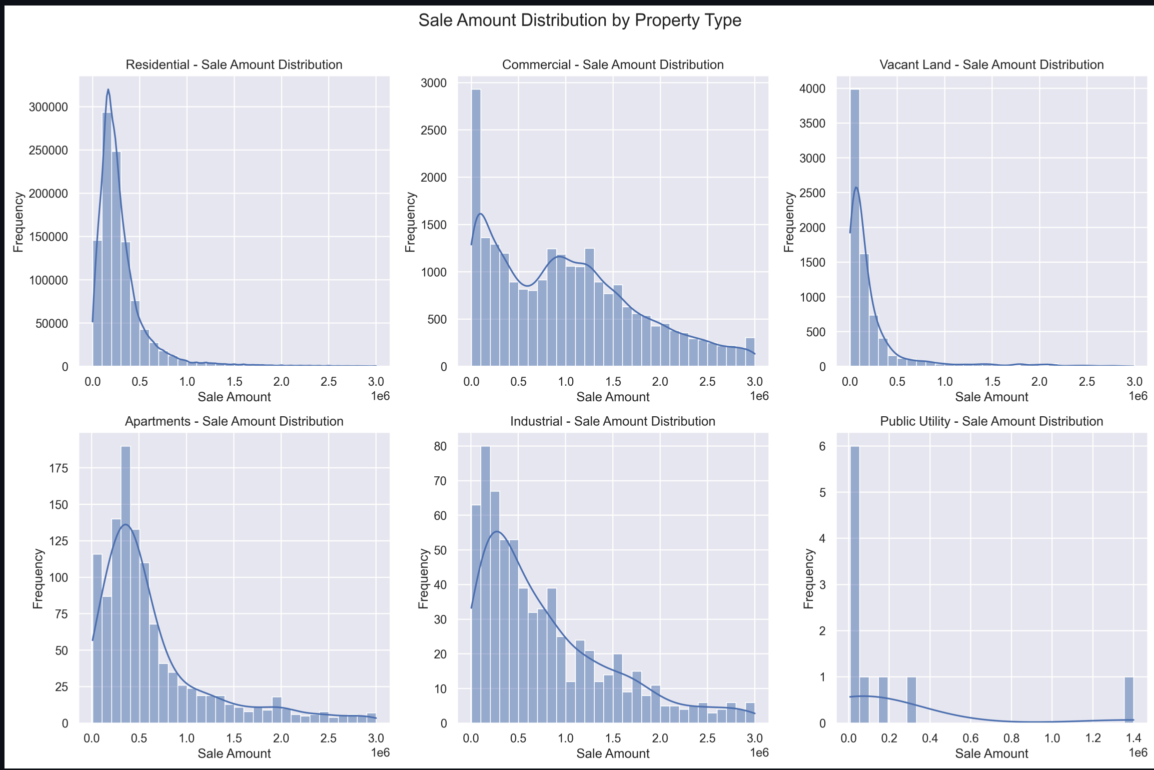
```
grouped_data = df2.groupby('Residential_Type').count().reset_index()

plt.figure(figsize=(12, 6))
plt.bar(grouped_data['Residential_Type'], grouped_data['Sale_Amount'], color=
```

```
for index, value in enumerate(grouped_data['Sale_Amount']):
    plt.text(index, value, str(value), ha='center', va='bottom')


plt.title('Frequency by Residential Type')
plt.xlabel('Residency Types')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.grid(axis='y')

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



Since the Residential Type data is highly dominant, we have analyzed this type separately.

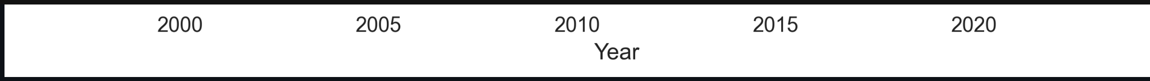
In [86]:
```
property_types = df2['Property_Type'].unique()

fig, axes = plt.subplots(2, 3, figsize=(15, 10))

axes = axes.flatten()


for i, property_type in enumerate(property_types):

    property_data = df2[df2['Property_Type'] == property_type]['Sale_Amount']


    sns.histplot(property_data, kde=True, bins=30, ax=axes[i])


    axes[i].set_title(f'{property_type} - Sale Amount Distribution')
    axes[i].set_xlabel('Sale Amount')
    axes[i].set_ylabel('Frequency')


plt.suptitle('Sale Amount Distribution by Property Type', fontsize=16)
plt.tight_layout()
plt.subplots_adjust(top=0.9)

plt.show()
```



Can we visualize the trend of total Sale Amount over the years (List Year)?

In [87]:

```python
data_grouped = data.groupby('Year')['Sale_Amount'].sum().reset_index()

plt.figure(figsize=(10, 6))
plt.plot(data_grouped['Year'], data_grouped['Sale_Amount'], marker='o', lines

plt.title('Total Sales Amount by Year', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Total Sales Amount', fontsize=12)

plt.grid(True)
plt.show()
```



This graph shows the change in total sales over the years.

In [88]:

```
data_grouped = data.groupby('Year')['Sale_Amount'].count().reset_index()

plt.figure(figsize=(10, 6))
plt.plot(data_grouped['Year'], data_grouped['Sale_Amount'], marker='o', lines

plt.title('Total Sales Amount by Year', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Total Sales Amount', fontsize=12)

plt.grid(True)
plt.show()
```

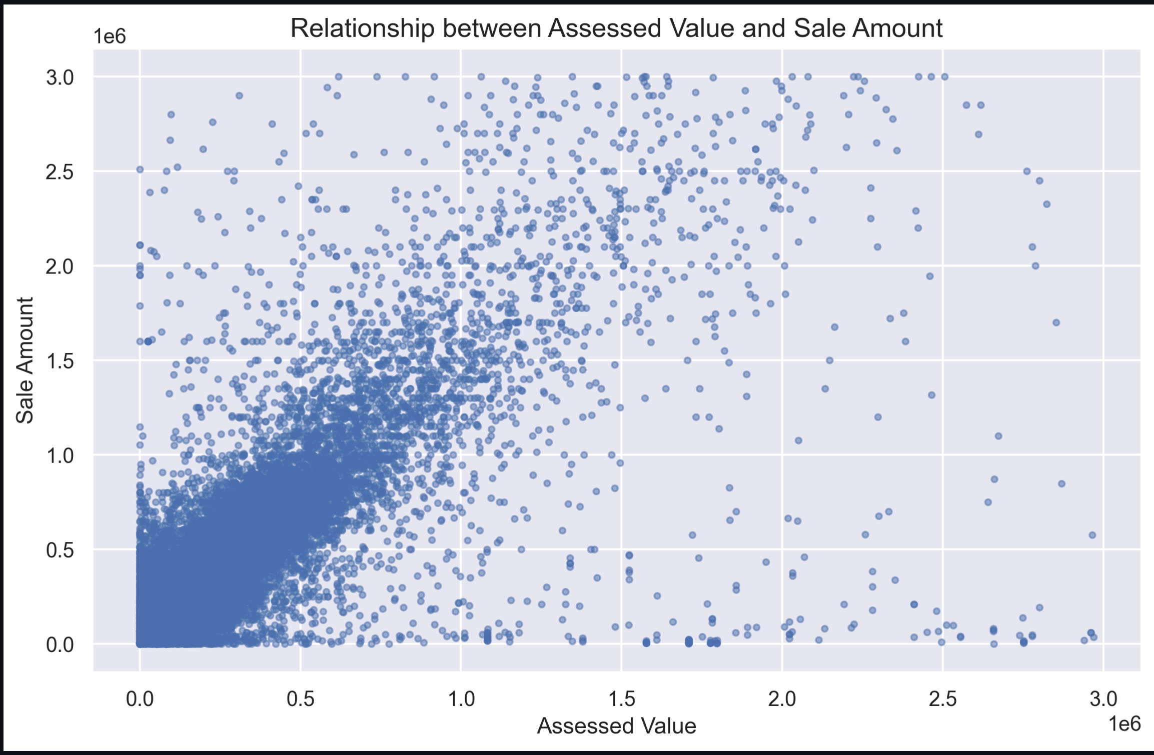
Looking at the graph above, the impact of societal factors (such as economic crises, pandemics, etc.) on property values is clearly evident. These external factors have caused fluctuations in the market, leading to significant changes in property values over time. The graph illustrates how these events have shaped the real estate market, influencing both supply and demand dynamics.

How do Assessed Value and Sale Amount correlate, and can this be visualized as a scatter plot?

In [89]:

```
np.random.seed(42)

plt.figure(figsize=(10, 6))
plt.scatter(x='Assessed_Value', y='Sale_Amount', data=data.sample(50000, rand

plt.title('Relationship between Assessed Value and Sale Amount', fontsize=14)
plt.xlabel('Assessed Value', fontsize=12)
plt.ylabel('Sale Amount', fontsize=12)
```

```
plt.grid(True)
plt.show()
```

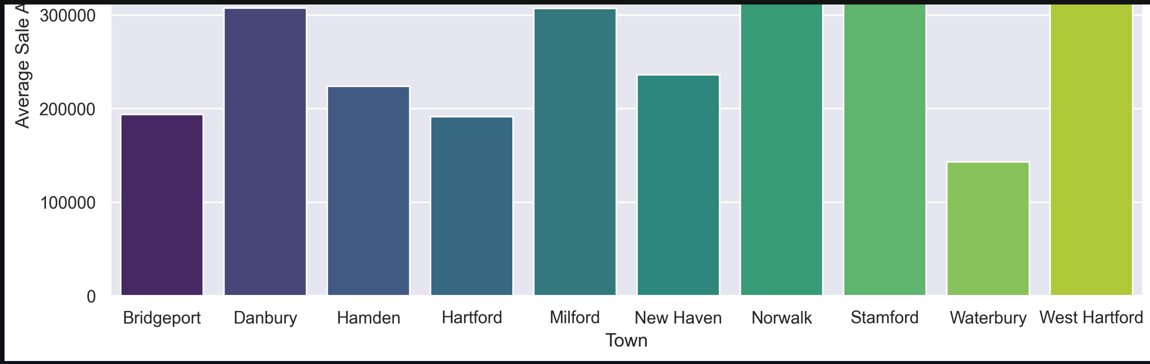


The scatter plot above visualizes the relationship between Sale Amount and Assessed Values for a random sample of 50,000 observations from the dataset. As seen in the graph, a noticeable correlation between the two columns is evident. Due to the large size of the dataset, we used a random sample of 50,000 records for this analysis.

Are there significant differences in the average Sale Amount across different towns?

In [90]:

```
town_avg_sale = data.groupby('Town')['Sale_Amount'].mean().reset_index()

town_counts = data['Town'].value_counts().head(10).index
top_towns = town_avg_sale[town_avg_sale['Town'].isin(town_counts)]

plt.figure(figsize=(12, 6))
sns.barplot(y='Sale_Amount', x='Town', data=top_towns, palette='viridis')

plt.title('Average Sale Amount for the Top 10 Most Popular Towns', fontsize=1
plt.xlabel('Town', fontsize=12)
plt.ylabel('Average Sale Amount', fontsize=12)

plt.show()
```

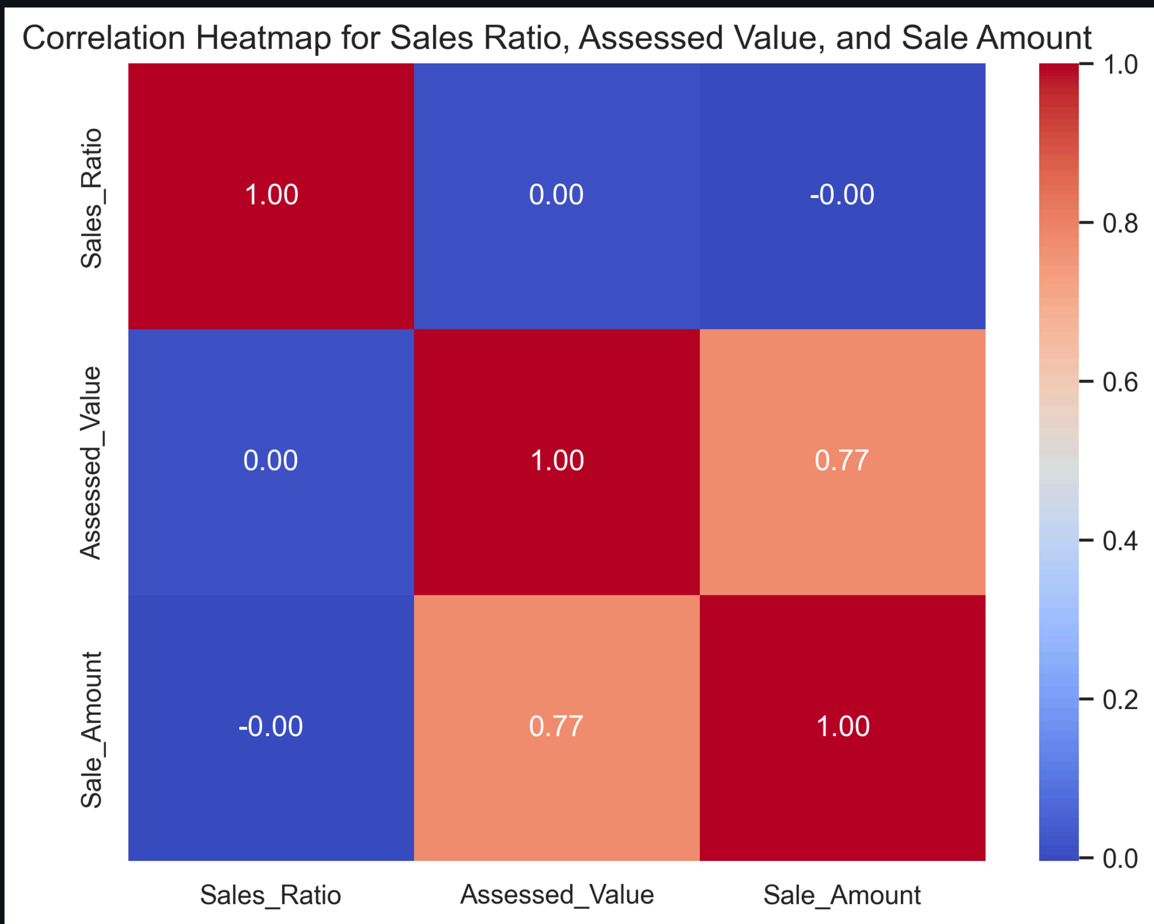


This graph shows the average sale prices in the top 10 most preferred towns.

Can we create a heatmap to show correlations between numeric variables like Sales Ratio, Assessed Value, and Sale Amount?

In [91]:

```
correlation_matrix = data[['Sales_Ratio', 'Assessed_Value', 'Sale_Amount']].c

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', cbar=

plt.title('Correlation Heatmap for Sales Ratio, Assessed Value, and Sale Amou
plt.show()
```



The above graph is a correlation matrix showing the Pearson correlation coefficients for all numeric columns. The group of variables with the highest correlation is between Sale Amount and Assessed Value, while there is no significant correlation found among other variable combinations.

Can we map the sales distribution using the Location column, if geospatial data is extracted?

In [92]:

```
import folium
from folium.plugins import MarkerCluster
from IPython.display import display

town_info = data.groupby('Town').agg(
    avg_sale_amount=('Sale_Amount', 'mean'),
    most_common_property_type=('Property_Type', lambda x: x.mode()[0]),
    most_common_residential_type=('Residential_Type', lambda x: x.mode()[0])
).reset_index()

map_center = [data['Latitude'].mean(), data['Longitude'].mean()]  # Set the m
m = folium.Map(location=map_center, zoom_start=10)

marker_cluster = MarkerCluster().add_to(m)

for _, row in town_info.iterrows():
    town = row['Town']
    avg_sale_amount = row['avg_sale_amount']
    property_type = row['most_common_property_type']
    residential_type = row['most_common_residential_type']

    town_data = data[data['Town'] == town].iloc[0]
    latitude = town_data['Latitude']
    longitude = town_data['Longitude']

    popup_text = f"<strong>{town}</strong><br>"
    popup_text += f"Average Sale Amount: ${avg_sale_amount:,.2f}<br>"
    popup_text += f"Most Common Property Type: {property_type}<br>"
    popup_text += f"Most Common Residential Type: {residential_type}<br>"

    folium.Marker(
        location=[latitude, longitude],
        popup=popup_text,
        icon=folium.Icon(color='blue', icon='info-sign')
    ).add_to(marker_cluster)

display(m)
```

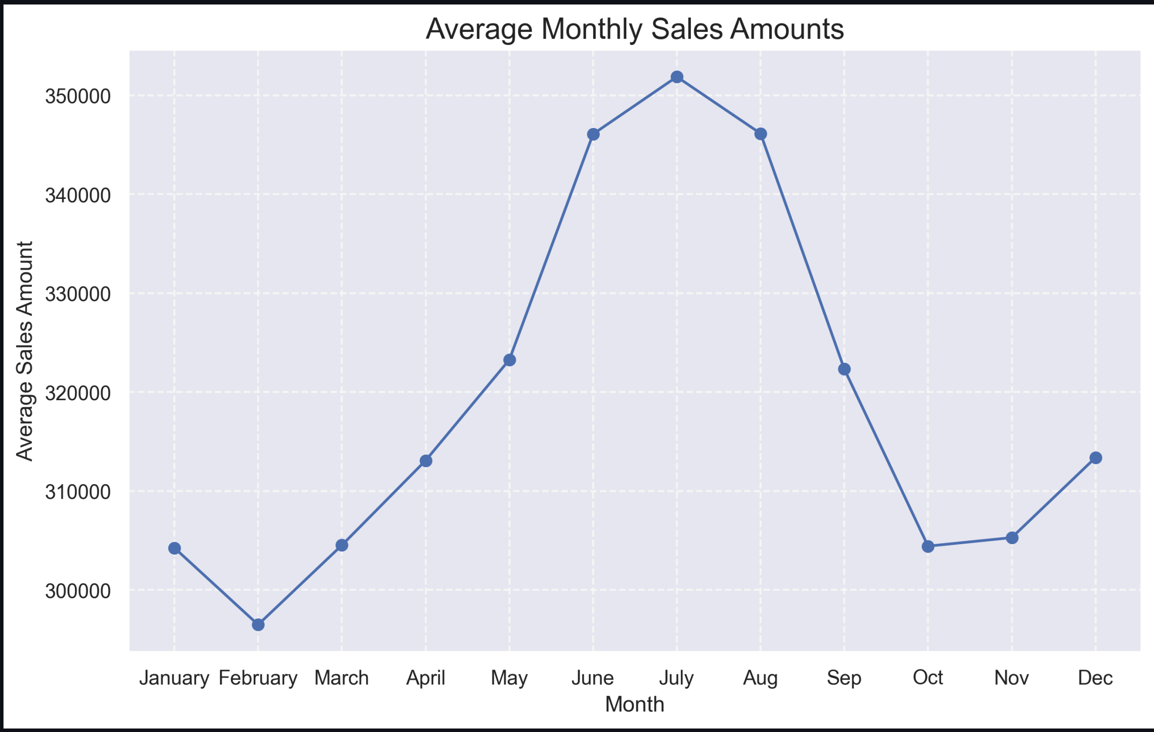
Make this Notebook Trusted to load map: File -> Trust Notebook

The above map was created using the Folium library. It displays the average sale amounts in towns, as well as the most popular property and residential types.

Are there seasonal patterns in sales, based on the Date Recorded column?

In [93]:

```
data['Month'] = data['Date_Recorded'].dt.month

monthly_sales = data.groupby('Month')['Sale_Amount'].mean()

plt.figure(figsize=(10, 6))
plt.plot(monthly_sales.index, monthly_sales.values, marker='o', color='b', li
plt.title('Average Monthly Sales Amounts', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Average Sales Amount', fontsize=12)
plt.xticks(range(1, 13), ['January', 'February', 'March', 'April', 'May', 'Ju
                          'July', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

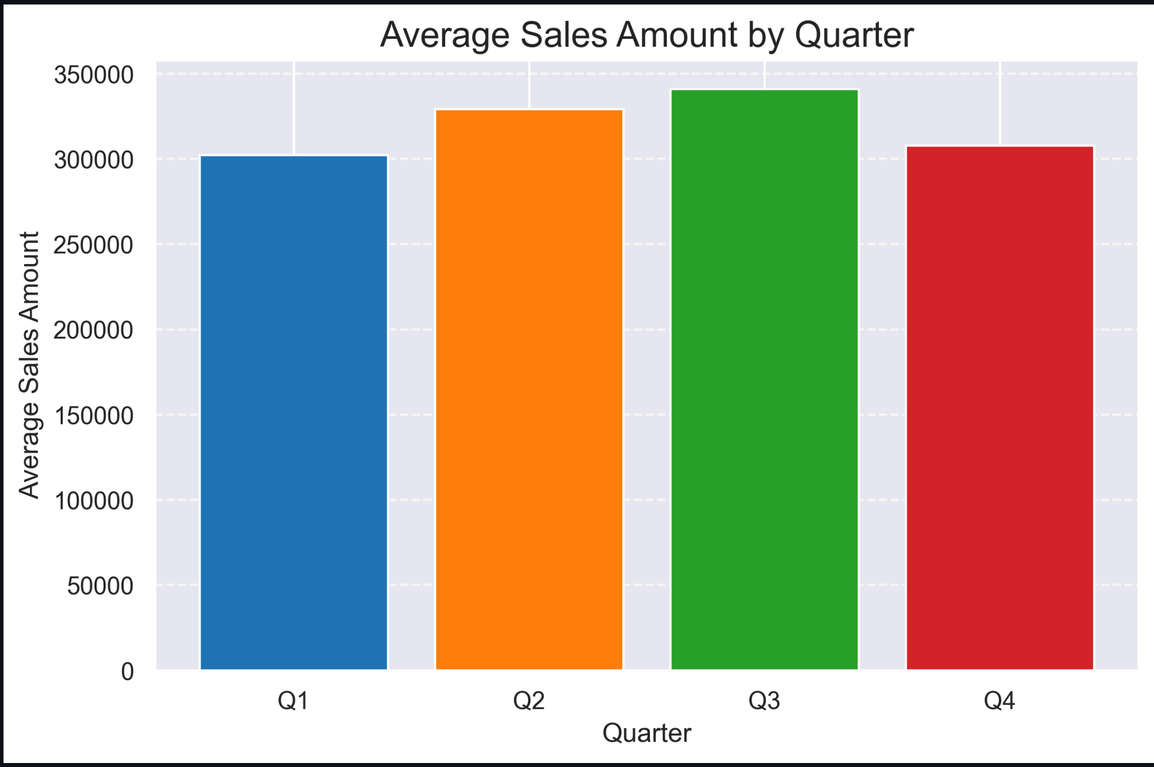


This graph shows the average sales by month.

In [94]:

```
data['Quarter'] = data['Date_Recorded'].dt.quarter

quarterly_sales = data.groupby('Quarter')['Sale_Amount'].mean()

plt.figure(figsize=(8, 5))
plt.bar(quarterly_sales.index, quarterly_sales.values, color=['#1f77b4', '#ff
plt.title('Average Sales Amount by Quarter', fontsize=16)
plt.xlabel('Quarter', fontsize=12)
plt.ylabel('Average Sales Amount', fontsize=12)
plt.xticks([1, 2, 3, 4], ['Q1', 'Q2', 'Q3', 'Q4'])
plt.grid(axis='y', linestyle='--', alpha=0.6)
```

```
plt.show()
```



This graph displays the average sales by dividing the years into quarterly periods. By looking at this graph, sales trends in each quarter can be analyzed.

About Analysis by Dataset

As a result of the data cleaning process, missing data and outliers were addressed. A significant portion of the missing values were filled appropriately (with the suggested method) and the data set was made more consistent.

- 🎯 Data cleaning has been successfully performed, resulting in a more consistent and reliable dataset, ensuring that the information is accurate and ready for further analysis.
- 🎯 Data distributions and relationships between variables have been examined, and detailed visualizations and analyses were conducted, focusing on variables that show positive correlations.
- 🎯 By generating new data from the existing dataset, we have made the dataset more suitable for modeling.
- 🎯 This project has been a collaborative effort, with the team working in harmony, and through effective teamwork, valuable skills have been acquired that contributed to the overall success of the project.

This dataset was used for educational purposes.

- 📊 https://www.kaggle.com/zeripek