Screenshots

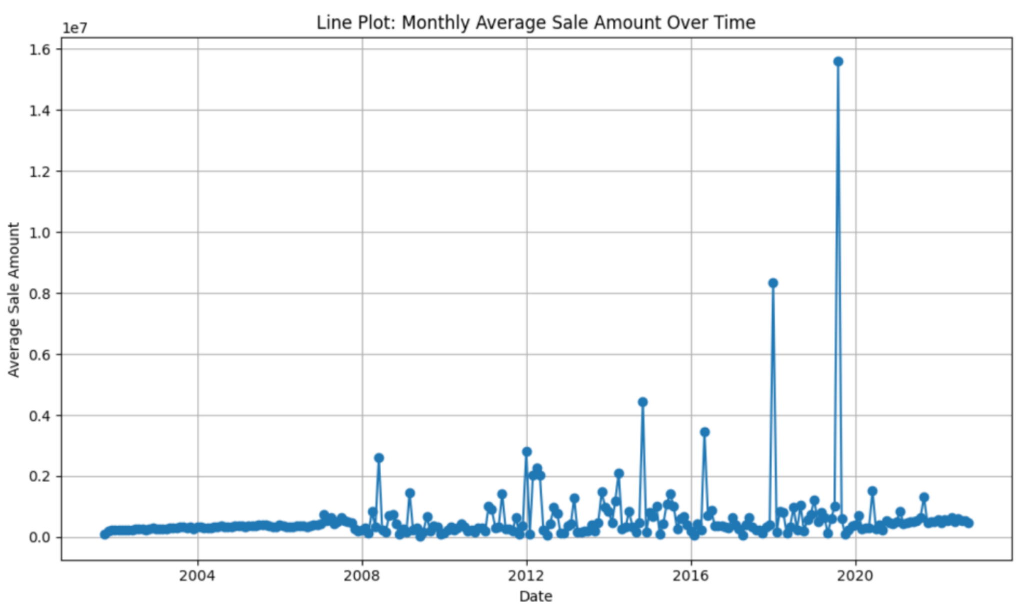
[6]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sales Ratio	Property Type	Residential Type
0	2020177	2020	4/14/2021	Ansonia	323 BEAVER ST	133000	248400.0	0.5354	Residential	Single Family
1	2020225	2020	5/26/2021	Ansonia	152 JACKSON ST	110500	239900.0	0.4606	Residential	Three Family
2	2020348	2020	9/13/2021	Ansonia	230 WAKELEE AVE	150500	325000.0	0.4630	Commercial	NaN
3	2020090	2020	12/14/2020	Ansonia	57 PLATT ST	127400	202500.0	0.6291	Residential	Two Family
4	210288	2021	6/20/2022	Avon	12 BYRON DRIVE	179990	362500.0	0.4965	Residential	Condo

Display the first five rows of your dataset using .head()

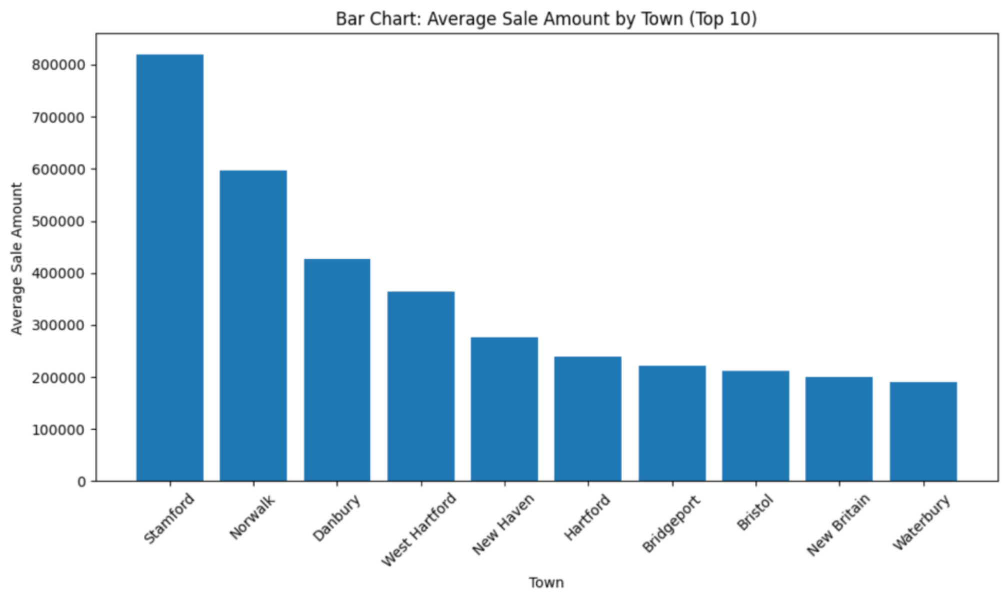
[10]:

```
# Line Plot: Monthly Average Sale Amount Over Time (without seaborn)
plt.figure(figsize=(10, 6))
plt.plot(df_line_monthly["Date Recorded"], df_line_monthly["Sale Amount"], marker='o')
plt.title("Line Plot: Monthly Average Sale Amount Over Time")
plt.xlabel("Date")
plt.ylabel("Average Sale Amount")
plt.grid(True)
plt.tight_layout()
plt.show()
```



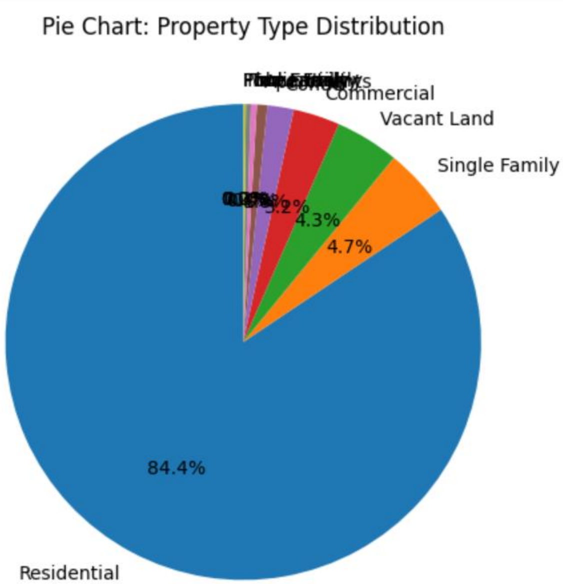
Line Plots: Line Plot: Monthly Average Sale Amount Over Time

```
[11]: # Bar Chart: Average Sale Amount by Town (Top 10)
plt.figure(figsize=(10, 6))
plt.bar(avg_sale_by_town.index, avg_sale_by_town.values)
plt.title("Bar Chart: Average Sale Amount by Town (Top 10)")
plt.xlabel("Town")
plt.ylabel("Average Sale Amount")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Bar Chart: Average Sale Amount by Town (Top 10)

```
[13]: # Pie Chart: Property Type Distribution
plt.figure(figsize=(5, 5))
plt.pie(property_type_counts.values, labels=property_type_counts.index, autopct='%1.1f%%', startangle=90)
plt.title("Pie Chart: Property Type Distribution")
plt.tight_layout()
plt.show()
```



Pie Chart: Property Type Distribution

```python
# Count missing values in each column before cleaning
missing_before = df.isnull().sum()

# Fill numeric columns with mean, categorical with mode
df_cleaned = df.copy()
for col in df_cleaned.columns:
    if df_cleaned[col].dtype in ['float64', 'int64']:
        df_cleaned[col] = df_cleaned[col].fillna(df_cleaned[col].mean())
    else:
        df_cleaned[col] = df_cleaned[col].fillna(df_cleaned[col].mode()[0])

# Count missing values after cleaning
missing_after = df_cleaned.isnull().sum()
df_cleaned.head()

# Show the changes in missing values
missing_before, missing_after
```

[16]:
```
(Serial Number            0
 List Year                0
 Date Recorded            2
 Town                     0
 Address                 51
 Assessed Value           0
 Sale Amount              0
 Sales Ratio              0
 Property Type       382062
 Residential Type    393500
 dtype: int64,
 Serial Number       0
 List Year           0
 Date Recorded       0
 Town                0
 Address             0
 Assessed Value      0
 Sale Amount         0
 Sales Ratio         0
 Property Type       0
 Residential Type    0
 dtype: int64)
```

Display the dataset before and after handling missing values

```python
# Re-cleaning missing values
df_cleaned = df.copy()
for col in df_cleaned.columns:
    if df_cleaned[col].dtype in ['float64', 'int64']:
        df_cleaned[col] = df_cleaned[col].fillna(df_cleaned[col].mean())
    else:
        df_cleaned[col] = df_cleaned[col].fillna(df_cleaned[col].mode()[0])

# Outlier detection on 'Sale Amount'
Q1 = df_cleaned['Sale Amount'].quantile(0.25)
Q3 = df_cleaned['Sale Amount'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = df_cleaned[(df_cleaned['Sale Amount'] < lower_bound) | (df_cleaned['Sale Amount'] > upper_bound)]
df_no_outliers = df_cleaned[(df_cleaned['Sale Amount'] >= lower_bound) & (df_cleaned['Sale Amount'] <= upper_bound)]

# Output summary
{
    "Q1": Q1,
    "Q3": Q3,
    "IQR": IQR,
    "Lower Bound": lower_bound,
    "Upper Bound": upper_bound,
    "Number of Outliers": len(outliers),
    "Dataset Size Before": len(df_cleaned),
    "Dataset Size After": len(df_no_outliers)
}
```

```
{'Q1': np.float64(142000.0),
 'Q3': np.float64(370000.0),
 'IQR': np.float64(228000.0),
 'Lower Bound': np.float64(-200000.0),
 'Upper Bound': np.float64(712000.0),
 'Number of Outliers': 88347,
 'Dataset Size Before': 1048575,
 'Dataset Size After': 960228}
```

Display the IQR calculation, identified outliers, and the dataset after outlier handling

```python
# Convert numeric columns to proper types
numeric_columns = ['Assessed Value', 'Sale Amount', 'Sales Ratio']
for col in numeric_columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Fill missing values correctly
df_cleaned = df.copy()
for col in df_cleaned.columns:
    if df_cleaned[col].dtype in ['float64', 'int64']:
        df_cleaned[col] = df_cleaned[col].fillna(df_cleaned[col].mean())
    else:
        df_cleaned[col] = df_cleaned[col].fillna(df_cleaned[col].mode()[0])

# Remove outliers from 'Sale Amount' using IQR
Q1 = df_cleaned['Sale Amount'].quantile(0.25)
Q3 = df_cleaned['Sale Amount'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_no_outliers = df_cleaned[(df_cleaned['Sale Amount'] >= lower_bound) & (df_cleaned['Sale Amount'] <= upper_bound)]

# Step 3.3: Apply Data Reduction
df_sampled = df_no_outliers.sample(frac=0.1, random_state=42)
columns_to_drop = ['Serial Number', 'Address']
df_reduced = df_sampled.drop(columns=columns_to_drop)

# Display shape before and after reduction
df_no_outliers.shape, df_reduced.shape
```

```
((960228, 10), (96023, 8))
```

Display the dataset before and after applying data reduction techniques

```python
# Fill missing values
df['Assessed Value'] = df['Assessed Value'].fillna(df['Assessed Value'].mean())
df['Sale Amount'] = df['Sale Amount'].fillna(df['Sale Amount'].mean())

# IQR filtering (repeat to get df_no_outliers)
Q1 = df['Sale Amount'].quantile(0.25)
Q3 = df['Sale Amount'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR
df_no_outliers = df[(df['Sale Amount'] >= lower) & (df['Sale Amount'] <= upper)]

# Sample and drop columns
df_reduced = df_no_outliers.sample(frac=0.1, random_state=42).drop(columns=["Serial Number", "Address"], errors='ignore')

# Manual Min-Max Scaling
sale_min, sale_max = df_reduced['Sale Amount'].min(), df_reduced['Sale Amount'].max()
df_reduced['Sale_MinMax'] = (df_reduced['Sale Amount'] - sale_min) / (sale_max - sale_min)

# Manual Z-Score Normalization
sale_mean = df_reduced['Sale Amount'].mean()
sale_std = df_reduced['Sale Amount'].std()
df_reduced['Sale_ZScore'] = (df_reduced['Sale Amount'] - sale_mean) / sale_std

# Discretization into quartile bins
df_reduced['Sale_Category'] = pd.qcut(df_reduced['Sale Amount'], 4, labels=["Low", "Medium", "High", "Very High"])

# Display results
df_reduced[['Sale Amount', 'Sale_MinMax', 'Sale_ZScore', 'Sale_Category']].head()
```

!0]:

	Sale Amount	Sale_MinMax	Sale_ZScore	Sale_Category
1030756	275000.0	0.386236	0.232212	High
510031	565000.0	0.793539	2.184374	Very High
820530	110000.0	0.154494	-0.878500	Low
279758	130000.0	0.182584	-0.743869	Low
850756	25000.0	0.035112	-1.450686	Low

Display the dataset after scaling or discretization

```python
# Convert numeric columns
df['Assessed Value'] = pd.to_numeric(df['Assessed Value'], errors='coerce')
df['Sale Amount'] = pd.to_numeric(df['Sale Amount'], errors='coerce')

# Drop rows with missing values in relevant columns
df = df.dropna(subset=['Assessed Value', 'Sale Amount'])

# Step 4.1: General Overview
info_str = df.info()
description = df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 10 columns):
 #   Column            Non-Null Count    Dtype
---  ------            --------------    -----
 0   Serial Number     1048575 non-null  int64
 1   List Year         1048575 non-null  int64
 2   Date Recorded     1048573 non-null  object
 3   Town              1048575 non-null  object
 4   Address           1048524 non-null  object
 5   Assessed Value    1048575 non-null  int64
 6   Sale Amount       1048575 non-null  float64
 7   Sales Ratio       1048575 non-null  float64
 8   Property Type     666513 non-null   object
 9   Residential Type  655075 non-null   object
dtypes: float64(2), int64(3), object(5)
memory usage: 60.0+ MB
```

General Overview of Data

```python
[24]: # Step 4.2: Central Tendency
central_tendency = {
    "Minimum": df["Sale Amount"].min(),
    "Maximum": df["Sale Amount"].max(),
    "Mean": df["Sale Amount"].mean(),
    "Median": df["Sale Amount"].median(),
    "Mode": df["Sale Amount"].mode().iloc[0]
}

central_tendency
```

```
[24]: {'Minimum': np.float64(0.0),
 'Maximum': np.float64(5000000000.0),
 'Mean': np.float64(398617.9438311422),
 'Median': np.float64(230000.0),
 'Mode': np.float64(150000.0)}
```

Display results of central tendency calculations

```python
[26]: # Step 4.3: Dispersion
data_range = maximum - minimum
q1 = df['Sale Amount'].quantile(0.25)
q3 = df['Sale Amount'].quantile(0.75)
iqr = q3 - q1
variance = df['Sale Amount'].var()
std_dev = df['Sale Amount'].std()
{
    "Dispersion Measures": {
        "Range": data_range,
        "Q1": q1,
        "Q3": q3,
        "IQR": iqr,
        "Variance": variance,
        "Standard Deviation": std_dev
    }
}
```

```
[26]: {'Dispersion Measures': {'Range': np.float64(5000000000.0),
  'Q1': np.float64(142000.0),
  'Q3': np.float64(370000.0),
  'IQR': np.float64(228000.0),
  'Variance': np.float64(27413255581910.01),
  'Standard Deviation': np.float64(5235766.9525973)}}
```

Display results of dispersion calculations

'Standard Deviation': np.float64(5235766.9525973)}}

```python
[27]: # Step 4.4: Correlation Matrix
correlation_matrix = df[['Assessed Value', 'Sale Amount']].corr()
{
    "Correlation Matrix": correlation_matrix
}
```

```
[27]: {'Correlation Matrix':                  Assessed Value  Sale Amount
 Assessed Value         1.00000      0.11778
 Sale Amount            0.11778      1.00000}
```

Display the correlation matrix output