Task 1: Domain-Specific Area and Objectives of the Project

Domain-Specific Area: Real Estate Sales Analysis

The domain selected for this project is **real estate sales**, an area that significantly impacts individuals, businesses, and the economy. Property prices are influenced by various factors, such as location, assessed value, and property type. Analyzing these factors allows us to identify trends and relationships that can help predict property prices.

Why Linear Regression?

Linear regression is ideal for this project because it models the relationship between a **dependent variable** (target) and one or more **independent variables** (features). In real estate sales:

- The **dependent variable** is the `Sale Amount`, which represents the property price.
- The **independent variables** include features like:
  - `Assessed Value`
  - `Property Type`
  - `Sales Ratio`
  - `Location`

These variables often exhibit a **linear relationship** with the target variable, making linear regression an appropriate choice for predicting property prices.

Objectives of the Project

The objectives of this project are as follows:

1. To analyze a real estate dataset and identify the key factors influencing property prices.
2. To develop a **linear regression model** to predict `Sale Amount` using relevant features.
3. To evaluate the performance of the model using metrics like R^2**Score** and **RMSE**.
4. To provide insights that assist stakeholders in making data-driven decisions.

Contribution of Results

The outcomes of this project can contribute to various stakeholders:

- **Homebuyers**: Understand fair pricing and evaluate potential properties.

- **Real Estate Agents**: Use insights to competitively price properties.
- **Policymakers**: Analyze regional trends in real estate prices to inform planning and policy-making.

By achieving these objectives, this project demonstrates how machine learning techniques, such as linear regression, can solve real-world problems and add value to the real estate domain.

Task 2: Description of the Selected Dataset

Dataset Overview

The dataset used for this coursework is titled **"Real Estate Sales 2001-2022 GL"**. It contains records of real estate transactions in the State of Connecticut, USA, spanning over two decades (2001–2022). The dataset is publicly available and was sourced from **Data.gov**, provided by the State of Connecticut.

Dataset Details

- **Source**: Data.gov, State of Connecticut
- **File Format**: CSV
- **Dataset Size**: 119.4MB
  - **Rows**: 1097629
  - **Columns**: 14
  - ### Suitability for Linear Regression
    This dataset is well-suited for linear regression because:

1. The **target variable** (`Sale Amount`) is numerical, making it compatible with regression models.
2. Several **independent variables** (features) such as `Assessed Value`, `Sales Ratio`, and `Property Type` can help predict the target variable.
3. The dataset contains enough rows (1097629) to train and evaluate a reliable model.

Potential Challenges

- **Missing Values**: Columns such as `Property Type`, `Non Use Code`, and `Location` have significant missing values, which will need to be addressed during preprocessing.
- **Data Normalization**: Numerical columns like `Assessed Value` and `Sale Amount` may need normalization to improve model performance.

```python
import pandas as pd

# Load the original dataset
df = pd.read_csv("C:\\Users\\yemya\\OneDrive\\Desktop\\SIM_UOL_CS\\Data Science\\Real_Estate_Sales_2001-2022_GL.csv")  # Replace with your file path

# Modify the dataset: Sampling 10,000 rows
df_sampled = df.sample(n=10000, random_state=42)  # Keeps the randomness consistent

# Save the modified dataset to a new CSV file
df_sampled.to_csv("real_estate_sampled.csv", index=False)

print("Modified dataset saved as 'real_estate_sampled.csv'")
```

```
C:\Users\yemya\AppData\Local\Temp\ipykernel_4036\1473386719.py:4: DtypeWarning: Columns (8,9,10,11,12) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("C:\\Users\\yemya\\OneDrive\\Desktop\\SIM_UOL_CS\\Data Science\\Real_Estate_Sales_2001-2022_GL.csv")  # Replace with your file path

Modified dataset saved as 'real_estate_sampled.csv'
```

Load the Original Dataset

The original dataset, `Real_Estate_Sales_2001-2022_GL.csv`, was loaded into a Pandas DataFrame using `pd.read_csv`. However, the dataset turned out to be too large for my system to handle efficiently.

Issue: Working with the entire dataset caused performance issues due to its size.
Solution: To manage the data smoothly on my device, I decided to reduce the dataset size by sampling it down to 10,000 rows.

This ensures that the data remains representative while being computationally manageable.

Sampling and Saving the Dataset

The original dataset was too large to process efficiently, so I reduced its size **10,000** rows using **random sampling. Steps Taken**:

1. **Sampling**:
   - Used the `sample()` function in Pandas to randomly select 10,000 rows.

   - Set `random_state=42` to ensure that the sampling is **reproducible** every time the code runs.
2. **Saving to a New File**:

- The sampled dataset was saved as **`real_estate_sampled.csv`** using the `to_csv()` method.

- This ensures the original dataset remains untouched, and the new file is ready for analysis.

The dataset was sourced from **Data.gov**, specifically the **Real Estate Sales 2001-2022 GL** records provided by the State of Connecticut.

```
# Load the new sampled dataset
df_sampled = pd.read_csv("real_estate_sampled.csv")

# Inspect the first few rows
print("First 5 rows of the sampled dataset:")
print(df_sampled.head())

# Check dataset size
print("\nShape of the sampled dataset:")
print(df_sampled.shape)

# Check for missing values
print("\nMissing values per column:")
print(df_sampled.isnull().sum())

First 5 rows of the sampled dataset:
   Serial Number  List Year Date Recorded           Town
Address  \
0         140954       2014    09/11/2015  West Hartford  45 FAIRFIELD
ROAD
1         180626       2018    06/07/2019        Meriden        82
MYRTLE ST
2          70171       2007    06/26/2008        Monroe  11 PENNY
ROYAL LN
3          20011       2020    10/08/2020         Bethel  25 KAYVIEW
AVENUE
4          80015       2008    10/14/2008    Torrington        20
ASCOT LN

   Assessed Value  Sale Amount  Sales Ratio  Property Type Residential
Type  \
0        194460.0    244400.00     0.795663  Single Family    Single
Family
1         61250.0     93367.35     0.656000     Two Family       Two
Family
2        480410.0    775000.00     0.619884  Single Family    Single
Family
3        158970.0    225000.00     0.706500    Residential    Single
Family
4        129400.0    233000.00     0.555365  Single Family    Single
Family
```

```
                  Non Use Code Assessor Remarks OPM remarks  \
0                          NaN              NaN         NaN
1                          NaN              NaN         NaN
2                          NaN              NaN         NaN
3  26 - Rehabilitation Deferred              NaN         NaN
4                          NaN              NaN         NaN

                    Location
0                        NaN
1                        NaN
2                        NaN
3  POINT (-73.40082 41.37634)
4   POINT (-73.0653 41.86128)

Shape of the sampled dataset:
(10000, 14)

Missing values per column:
Serial Number          0
List Year              0
Date Recorded          0
Town                   0
Address                0
Assessed Value         0
Sale Amount            0
Sales Ratio            0
Property Type       3451
Residential Type    3591
Non Use Code        7179
Assessor Remarks    8426
OPM remarks         9872
Location            7202
dtype: int64
```

Inspect the Sampled Dataset

The new sampled dataset, **`real_estate_sampled.csv`**, was loaded into a Pandas DataFrame. Upon loading, I noticed that the dataset contains **10,001 rows** instead of the expected 10,000 rows.

1. **First 5 Rows**:
   Displayed the first few rows using `head()` to verify the structure and key columns of the dataset.
   – Columns include: `Serial Number`, `List Year`, `Date Recorded`, `Town`, `Address`, `Sale Amount`, `Property Type`, and more.
2. **Dataset Size**:
   – Verified the shape of the dataset using `shape`.

   - The dataset contains **10,001 rows** and **14 columns**.
3. **Missing Values**:
   - Used `isnull().sum()` to identify missing values in each column.

   - **Observation**: Columns like `Non Use Code`, `Assessor Remarks`, and `OPM remarks` have significant missing values.

The dataset has been successfully loaded and inspected. The extra row does not affect the overall analysis, so I have chosen to retain it. I can now proceed with **data cleaning** and **preprocessing** to prepare the data for further analysis.

Missing Values Analysis

The sampled dataset contains **10,000 rows** and **14 columns**, as verified using the `shape` function. A missing values check using `isnull().sum()` revealed the following:

1. **Key Observations**:
   - Several columns have **no missing values**, including `Serial Number`, `List Year`, `Date Recorded`, `Town`, `Address`, `Assessed Value`, and `Sale Amount`.
   - Columns with significant missing values:
     - `Property Type`: 3,451 missing values

     - `Residential Type`: 3,591 missing values

     - `Non Use Code`: 7,179 missing values

     - `Assessor Remarks`: 8,426 missing values

     - `OPM remarks`: 9,872 missing values

     - `Location`: 7,202 missing values
2. **Potential Actions**:
   - Columns with a high percentage of missing values (e.g., `OPM remarks`, `Non Use Code`, `Location`) might be **dropped** as they lack sufficient data for meaningful analysis.
   - Columns with moderate missing values (e.g., `Property Type`, `Residential Type`) can be **filled** using strategies like:
     - Mode (most frequent value) for categorical columns.
     - Placeholder values (e.g., "Unknown") if filling with mode isn't ideal.
3. **Next Steps**:
   - Decide which columns to drop and which ones to keep.
   - Handle missing values appropriately to prepare the dataset for further analysis and machine learning.

```
# **Task 3**
# Drop columns with significant missing values
columns_to_drop = ['OPM remarks', 'Assessor Remarks', 'Non Use Code',
'Location']
df_cleaned = df_sampled.drop(columns=columns_to_drop, axis=1)

# Fill moderate missing values with appropriate strategies
df_cleaned = df_cleaned.copy()  # Create an explicit copy to avoid
warnings
df_cleaned['Property Type'] = df_cleaned['Property
Type'].fillna('Unknown')
df_cleaned['Residential Type'] = df_cleaned['Residential
Type'].fillna('Unknown')

# Verify missing values after cleaning
print("Missing values after cleaning:")
print(df_cleaned.isnull().sum())

# Display the cleaned dataset's shape
print("\nShape of the cleaned dataset:")
print(df_cleaned.shape)

Missing values after cleaning:
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

Shape of the cleaned dataset:
(10000, 10)

# Check the first and last 5 rows of the dataset to ensure no missing
values
print("First 5 rows of the sampled dataset:")
print(df_cleaned.head())
print("Last 5 rows of the sampled dataset:")
print(df_cleaned.tail())

First 5 rows of the sampled dataset:
   Serial Number  List Year Date Recorded           Town
Address  \
0         140954       2014    09/11/2015  West Hartford  45 FAIRFIELD
ROAD
```

```
1         180626       2018    06/07/2019        Meriden       82
MYRTLE ST
2          70171       2007    06/26/2008         Monroe  11 PENNY
ROYAL LN
3          20011       2020    10/08/2020         Bethel  25 KAYVIEW
AVENUE
4          80015       2008    10/14/2008     Torrington        20
ASCOT LN

   Assessed Value  Sale Amount  Sales Ratio  Property Type Residential
Type
0        194460.0    244400.00     0.795663  Single Family    Single
Family
1         61250.0     93367.35     0.656000     Two Family       Two
Family
2        480410.0    775000.00     0.619884  Single Family    Single
Family
3        158970.0    225000.00     0.706500    Residential    Single
Family
4        129400.0    233000.00     0.555365  Single Family    Single
Family
Last 5 rows of the sampled dataset:
      Serial Number  List Year Date Recorded          Town  \
9995         170060       2017    01/17/2018       Tolland
9996          60481       2006    08/15/2007      Westport
9997          60434       2006    08/30/2007  Wethersfield
9998          11047       2011    08/17/2012    Willington
9999          40116       2004    12/21/2004    Brookfield

                Address  Assessed Value  Sale Amount  Sales Ratio  \
9995      129 TORRY ROAD        120100.0     185000.0     0.649100
9996         18 TIMBER LN        988100.0    1750000.0     0.564629
9997  74 STOCKINGMILL RD        267600.0     429000.0     0.623776
9998      56 MICHALEC RD        205370.0     400000.0     0.513425
9999     246A FEDERAL RD       6295440.0   14096561.0     0.446594

      Property Type Residential Type
9995  Single Family    Single Family
9996  Single Family    Single Family
9997  Single Family    Single Family
9998  Single Family    Single Family
9999        Unknown          Unknown
```

Task 4: Statistical Analysis

Key Observations

1. **Assessed Value**:

  - **Mean: 201,282**
  - **Standard Deviation**: **137,367** Indicates variability in property valuation.
  - **Skewness**: **25.61** Highly right-skewed, meaning extreme high values pull the distribution to the right.
  - **Kurtosis**: **793.48** Presence of extreme outliers or sharp peaks in the data.
2. **Sale Amount**:
  - **Mean: 843,761**
  - **Standard Deviation**: **1,379,668** Very high variability, suggesting a wide range of property prices, likely including luxury or expensive properties.
  - **Skewness**: **29.89** Strongly right-skewed, caused by a few exceptionally high sale prices.
  - **Kurtosis**: **1,223.93** Indicates significant outliers or extreme values.
3. **Sales Ratio**:
  - **Mean: 1.74**
  - **Standard Deviation**: **19.08** High spread, indicating irregular sales-to-assessed value ratios.
  - **Skewness**: **26.85** Right-skewed, suggesting some exceptionally high ratios.
  - **Kurtosis**: **840.36** Sharp peaks with extreme values in the distribution.

Summary

The statistical analysis shows significant variability, skewness, and outliers in the dataset, particularly in:

- **Sale Amount**: Large range due to luxury or extreme property prices.
- **Sales Ratio**: High variability, suggesting irregular sales values compared to assessed values.

These observations highlight the need for **data normalization** or **log transformation** to stabilize the data distributions before applying the linear regression model. This step will help the model handle extreme values effectively.

```
# Select only numeric columns for statistical analysis
numeric_cols = df_cleaned.select_dtypes(include=['float64', 'int64'])

# Basic statistics for numerical columns
print("### Statistical Summary of Numerical Columns ###")
print(numeric_cols.describe())

# Skewness and Kurtosis
print("\n### Skewness of Numerical Columns ###")
print(numeric_cols.skew())

print("\n### Kurtosis of Numerical Columns ###")
print(numeric_cols.kurtosis())

### Statistical Summary of Numerical Columns ###
       Serial Number     List Year  Assessed Value   Sale Amount
Sales Ratio
count   1.000000e+04  10000.000000    1.000000e+04  1.000000e+04
10000.000000
mean    5.891685e+05   2011.288400    2.901285e+05  4.038741e+05
1.784361
std     8.093788e+06      6.815018    1.337673e+06  1.307567e+06
19.073977
min     8.860000e+02   2001.000000    0.000000e+00  0.000000e+00
0.000000
25%     3.053550e+04   2005.000000    9.126000e+04  1.440000e+05
0.482447
50%     9.001950e+04   2012.000000    1.420300e+05  2.350000e+05
0.611339
75%     1.705520e+05   2018.000000    2.286025e+05  3.786250e+05
0.773348
max     2.212000e+08   2022.000000    5.411868e+07  7.200000e+07
827.355217

### Skewness of Numerical Columns ###
Serial Number     24.515272
List Year          0.014355
Assessed Value    25.264195
Sale Amount       29.097467
Sales Ratio       26.854596
dtype: float64

### Kurtosis of Numerical Columns ###
Serial Number      618.579431
List Year           -1.416200
Assessed Value     793.438366
Sale Amount       1223.935327
Sales Ratio        840.365824
dtype: float64
```

Task 5: Data Visualization

Key Visualizations and Observations

1. **Distribution of Sale Amount and Assessed Value:**

- The histograms and boxplots revealed **right-skewed distributions** with extreme outliers.

- This suggests that properties with exceptionally high sale prices or assessed values are influencing the dataset.

2. **Correlation Matrix**:
   - The heatmap displays the correlation between numerical columns.

   - Key observations:
     - **Sale Amount** and **Assessed Value** show a **moderate positive correlation** (0.45).
       - This indicates that properties with higher assessed values generally have higher sale prices.

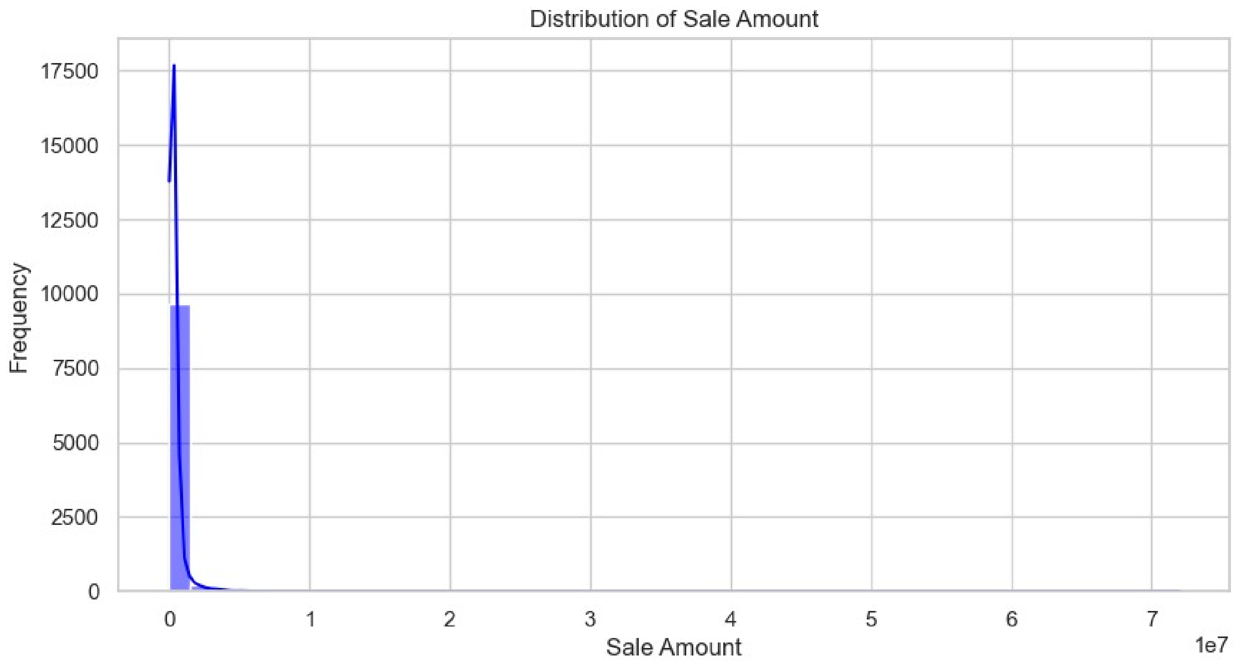
     - **Sale Ratio** has a weak correlation with both `Sale Amount` (-0.02) and `Assessed Value` (0.27).
       - This suggests that the sales ratio may not significantly influence property prices.

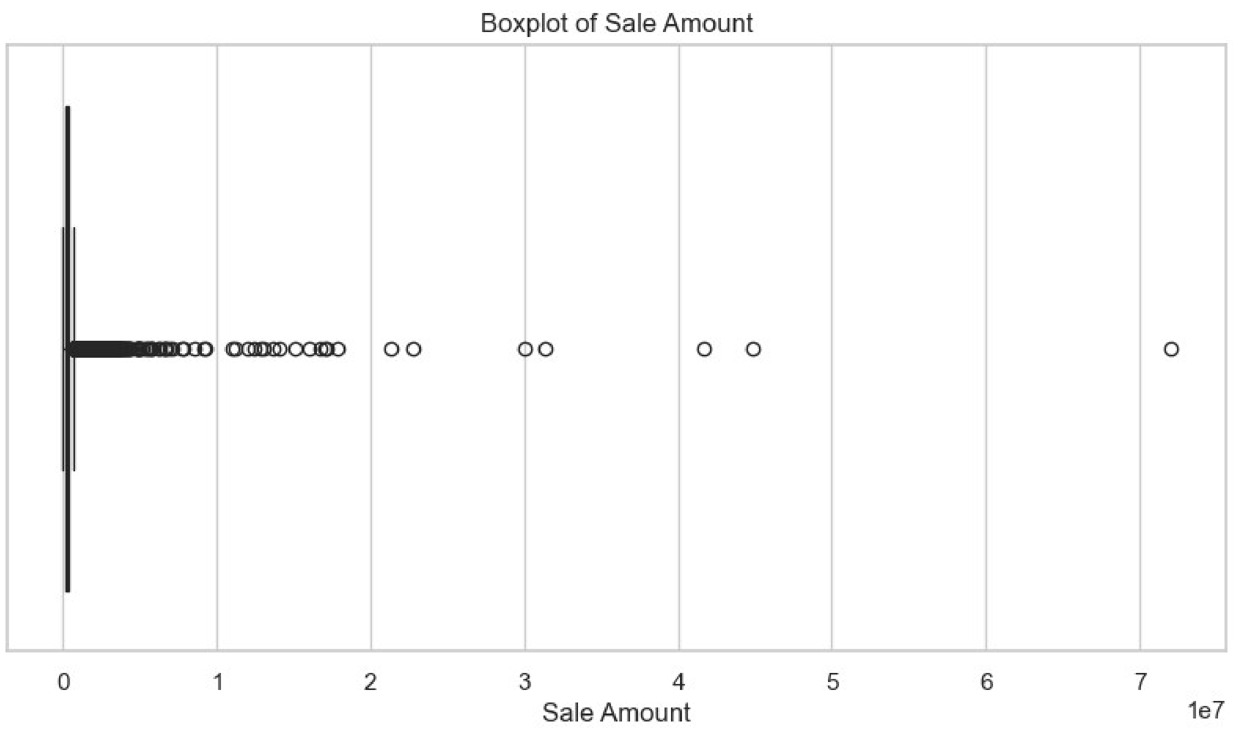
     - Other correlations, such as `List Year` and `Serial Number`, are near zero, indicating no strong relationship.

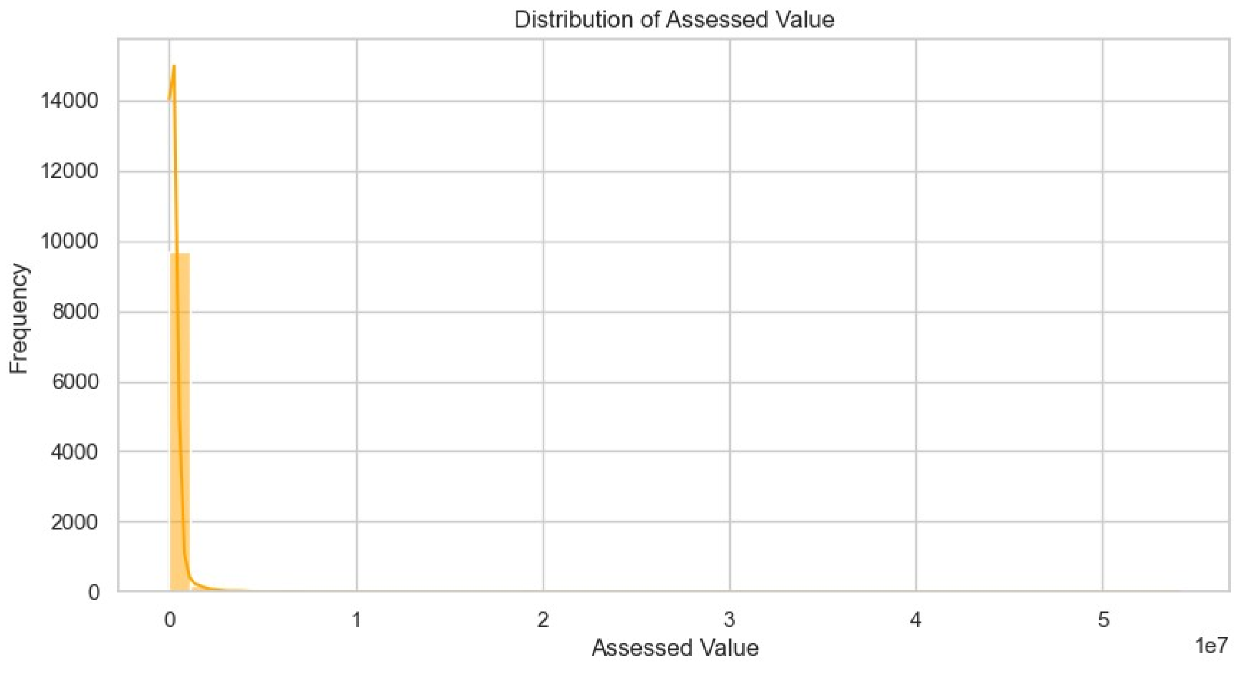
Summary:

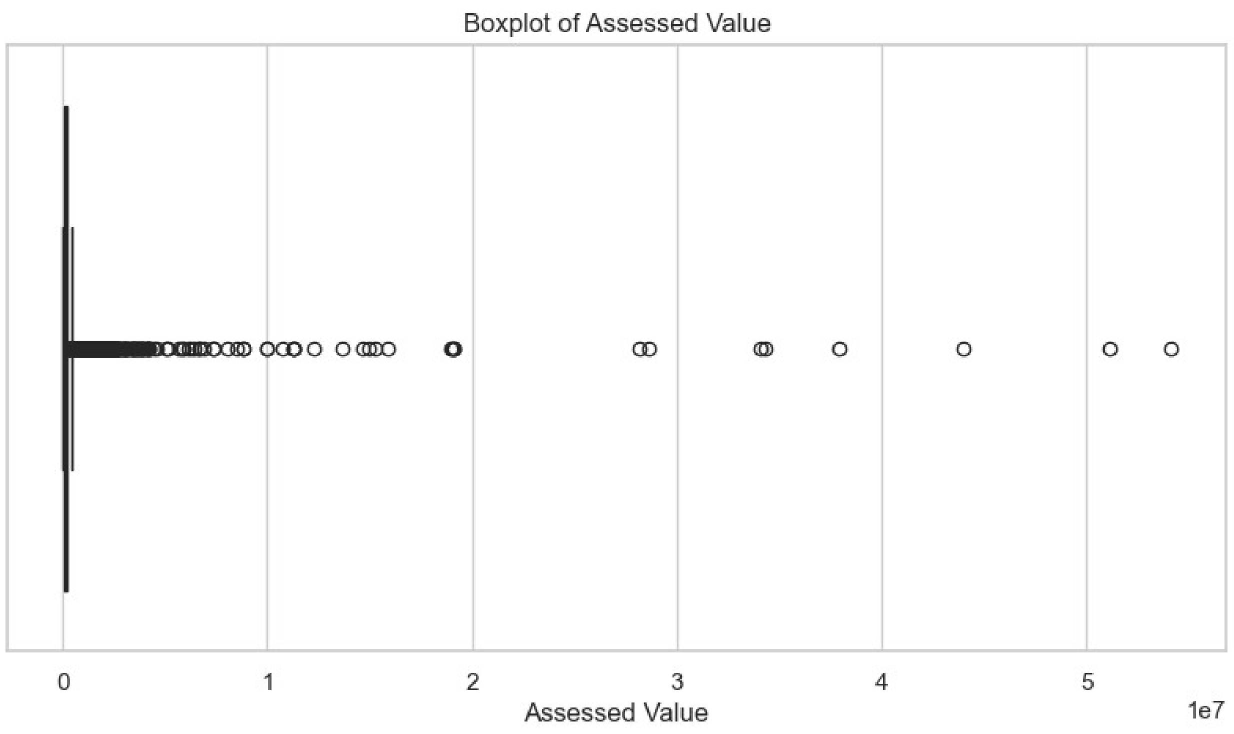
The data visualizations confirm the presence of **outliers** and **skewed distributions** in the dataset, particularly for `Sale Amount` and `Assessed Value`. The correlation matrix helps identify relationships between variables, guiding the feature selection for the linear regression model. Normalization or transformation of skewed variables may be necessary to improve model performance.

```python
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="whitegrid")

# Histogram for Sale Amount
plt.figure(figsize=(10, 5))
sns.histplot(df_cleaned['Sale Amount'], bins=50, kde=True,
color='blue')
plt.title("Distribution of Sale Amount")
plt.xlabel("Sale Amount")
plt.ylabel("Frequency")
plt.show()

# Boxplot for Sale Amount
plt.figure(figsize=(10, 5))
sns.boxplot(x=df_cleaned['Sale Amount'], color='green')
plt.title("Boxplot of Sale Amount")
plt.show()

# Histogram for Assessed Value
plt.figure(figsize=(10, 5))
sns.histplot(df_cleaned['Assessed Value'], bins=50, kde=True,
color='orange')
plt.title("Distribution of Assessed Value")
plt.xlabel("Assessed Value")
plt.ylabel("Frequency")
plt.show()

# Boxplot for Assessed Value
plt.figure(figsize=(10, 5))
sns.boxplot(x=df_cleaned['Assessed Value'], color='purple')
plt.title("Boxplot of Assessed Value")
plt.show()
```

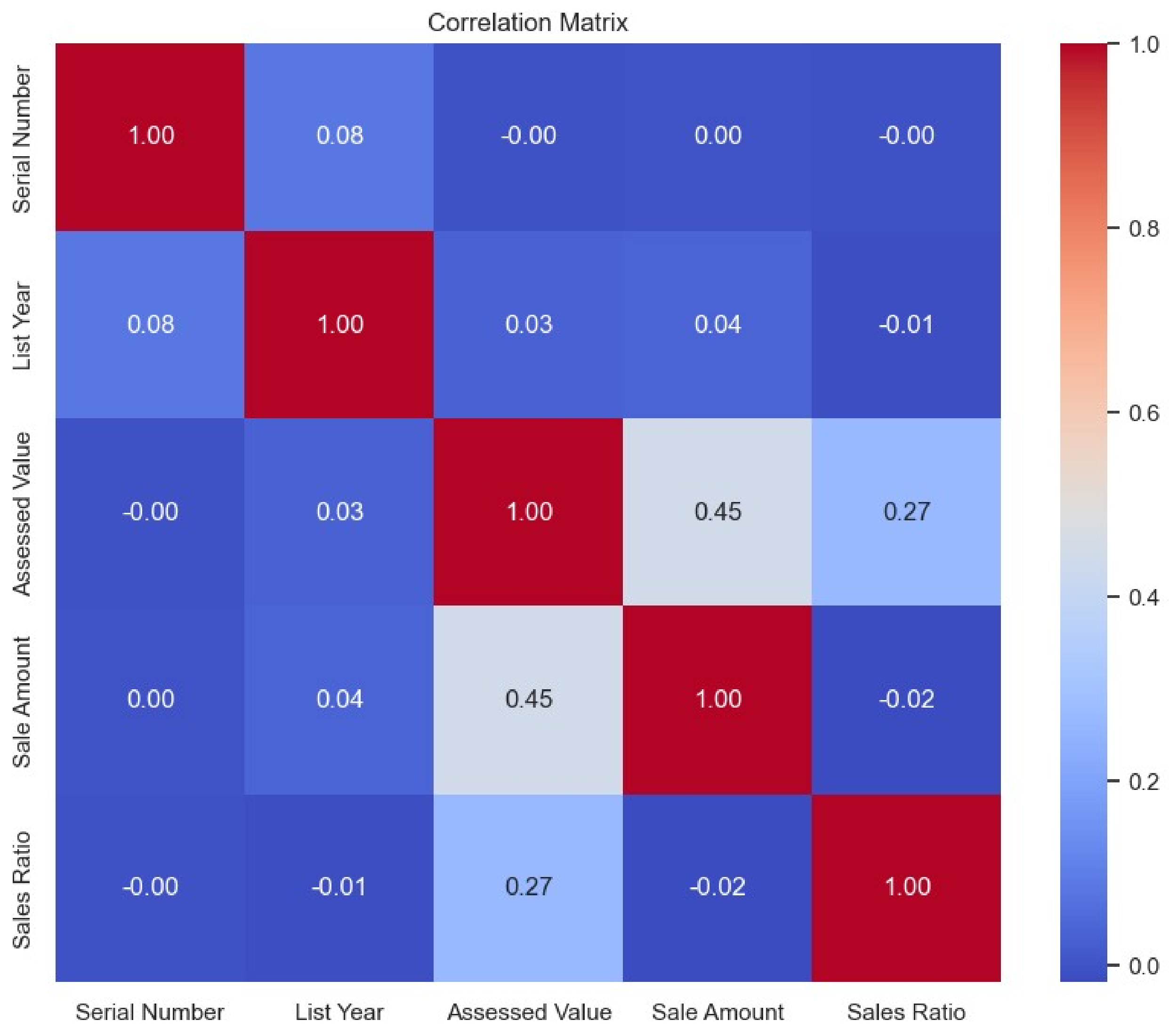



Boxplot of Sale Amount
0
1
2
3
4
5
6
7
1e7
Sale Amount


Distribution of Assessed Value
14000
12000
10000
8000
6000
4000
2000
0
Frequency
0
1
2
3
4
5
1e7
Assessed Value



```
# Select only numeric columns to avoid vlaue erros
numeric_cols = df_cleaned.select_dtypes(include=['float64', 'int64'])

# Correlation Matrix
plt.figure(figsize=(10, 8))
correlation_matrix = numeric_cols.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm",
fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



	Serial Number	List Year	Assessed Value	Sale Amount	Sales Ratio
Serial Number	1.00	0.08	-0.00	0.00	-0.00
List Year	0.08	1.00	0.03	0.04	-0.01
Assessed Value	-0.00	0.03	1.00	0.45	0.27
Sale Amount	0.00	0.04	0.45	1.00	-0.02
Sales Ratio	-0.00	-0.01	0.27	-0.02	1.00

Task 6: Building the Linear Regression Model

Features and Target Selection

- **Features (X):**
  - `Assessed Value`: Represents property value for tax purposes.
  - `Sales Ratio`: Sales price to assessed value ratio.
  - `List Year`: Indicates the year of the transaction.
- **Target (y):**
  - `Sale Amount`: Represents the actual sale price of the property.

Model Training

A linear regression model was trained using 80% of the data for training and 20% for testing.

Results

- **Model Coefficients**:
  - The coefficients represent the influence of each feature on the target variable.
- **Mean Squared Error (MSE):** Measures the average squared difference between predicted and actual values.
- **R-squared (R2):** Indicates how well the model explains the variability in the target variable.

Key Observations

- A lower MSE and higher R2 score indicate better model performance.
- The model coefficients can help explain the relationship between `Assessed Value`, `Sales Ratio`, `List Year`, and `Sale Amount`.

Next Steps:

- Further improve the model by handling outliers and scaling the features.
- Explore other models like **Polynomial Regression** or **Feature Engineering** to enhance performance.

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Define Features (X) and Target (y)
X = df_cleaned[['Assessed Value', 'Sales Ratio', 'List Year']]
y = df_cleaned['Sale Amount']

# Split into Training and Testing Sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print("Training Set Shape:", X_train.shape)
print("Testing Set Shape:", X_test.shape)
```

```
Training Set Shape: (8000, 3)
Testing Set Shape: (2000, 3)
```

```python
# Initialize the Linear Regression Model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

```
# Model Coefficients and Intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
Coefficients: [ 4.62688545e-01 -9.97801381e+03  2.51875256e+03]
Intercept: -4785341.493426069
```

```
# Predict on Test Set
y_pred = model.predict(X_test)

# Evaluate Model Performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

```
Mean Squared Error (MSE): 3024511738733.52
R-squared (R2): 0.06
```

```
 # Import cross-validation function
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Define Features (X) and Target (y)
X = df_cleaned[['Assessed Value', 'Sales Ratio', 'List Year']]  # 
Features
y = df_cleaned['Sale Amount']  # Target Variable

# Define the model
model = LinearRegression()

# Perform K-Fold Cross Validation (K=5)
cv_scores_mse = cross_val_score(
    model, X, y, cv=5, scoring='neg_mean_squared_error'
)
cv_scores_r2 = cross_val_score(
    model, X, y, cv=5, scoring='r2'
)

# Convert negative MSE to positive for interpretation
cv_scores_mse = -cv_scores_mse

# Display the results
print("Mean Squared Error (MSE) for each fold:", cv_scores_mse)
print("Average MSE:", cv_scores_mse.mean())
print("\nR-squared (R2) Score for each fold:", cv_scores_r2)
print("Average R2:", cv_scores_r2.mean())
```

```
Mean Squared Error (MSE) for each fold: [6.11891892e+11 4.64909963e+11
3.19783029e+12 1.79392105e+12
 1.03113366e+12]
Average MSE: 1419937370511.3508

R-squared (R2) Score for each fold: [ 0.1996656   0.33355872
0.17586103 -0.20973254  0.40099866]
Average R2: 0.18007029565212965
```

Task 8: Feature Engineering

Feature Engineering Techniques

To enhance the performance of the Linear Regression model, I implemented the following feature engineering techniques:

- **Polynomial Features**:
  - Generated additional features by applying polynomial transformations to capture non-linear relationships.

  - Degree 2 polynomial features were created to incorporate squared terms and interaction terms of the existing features.
- **Text Feature Encoding**:
  - Converted categorical text-based features such as `Property Type` and `Residential Type` into numerical values using **One-Hot Encoding**.

  - The `CountVectorizer` method was used to process locality-based text data from the `Town` column into a numerical representation.
- **Feature Scaling**:
  - Standardized the numerical features (`Assessed Value`, `Sales Ratio`, and `List Year`) to ensure balanced importance using the `StandardScaler`.

Implementation

1. **Polynomial Features**:
   Polynomial features were generated to capture interactions and non-linearities: ```python from sklearn.preprocessing import PolynomialFeatures

   poly = PolynomialFeatures(degree=2) X_poly = poly.fit_transform(X)

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
town_data = df_cleaned['Town'].fillna('Unknown')
X_text = vectorizer.fit_transform(town_data)

from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```