```python
import pandas as pd
import numpy as np

df = pd.read_csv("your_data.csv")  # Replace your_data.csv with the actual filename

# Convert relevant columns to numeric, coercing errors to NaN
numeric_cols = ['price', 'bed', 'bath', 'acre_lot', 'zip_code', 'house_size']
for col in numeric_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Convert prev_sold_date to datetime, coercing errors to NaT
df['prev_sold_date'] = pd.to_datetime(df['prev_sold_date'], errors='coerce')

# --- Imputation Strategies ---

# 1. Simple Imputation (Mean/Median/Mode)

# For numeric columns with relatively normal distributions, use mean/median
df['price'].fillna(df['price'].median(), inplace=True)
df['house_size'].fillna(df['house_size'].median(), inplace=True)
df['acre_lot'].fillna(df['acre_lot'].median(), inplace=True)


# For skewed data or ordinal data, median is often better
df['bed'].fillna(df['bed'].median(), inplace=True)
df['bath'].fillna(df['bath'].median(), inplace=True)

# For categorical columns, use mode
df['zip_code'].fillna(df['zip_code'].mode()[0], inplace=True)  # mode() returns a series, take the first eleme
df['city'].fillna(df['city'].mode()[0], inplace=True)
df['state'].fillna(df['state'].mode()[0], inplace=True)
df['street'].fillna(df['street'].mode()[0], inplace=True) # Impute street with mode
df['status'].fillna(df['status'].mode()[0], inplace=True) # Impute status with mode
df['brokered_by'].fillna(df['brokered_by'].mode()[0], inplace=True) # Impute brokered_by with mode


# 2. More Advanced Imputation (Considered if simple imputation isn't suitable)

# a.  K-Nearest Neighbors Imputation (for numerical features)
#     (Requires scikit-learn:  pip install scikit-learn)
# from sklearn.impute import KNNImputer
# imputer = KNNImputer(n_neighbors=5)  # Adjust n_neighbors as needed
# df[numeric_cols] = imputer.fit_transform(df[numeric_cols])

# b. Iterative Imputation (for numerical features)
#     (Also requires scikit-learn)
# from sklearn.experimental import enable_iterative_imputer
# from sklearn.impute import IterativeImputer
# imputer = IterativeImputer()
# df[numeric_cols] = imputer.fit_transform(df[numeric_cols])


# 3. Date Imputation (Forward/Backward fill or a specific date)

# Forward fill is often reasonable for time series-like data
df['prev_sold_date'].fillna(method='ffill', inplace=True)
# Or fill with a specific date if forward fill isn't appropriate
# df['prev_sold_date'].fillna(pd.to_datetime('2020-01-01'), inplace=True)  # Example
```

```python
# --- Handling Negative Values (if they are errors) ---

# Replace negative values in 'bed' and 'acre_lot' with median
df['bed'] = df['bed'].apply(lambda x: x if x >= 0 else df['bed'].median())
df['acre_lot'] = df['acre_lot'].apply(lambda x: x if x >= 0 else df['acre_lot'].median())
df['zip_code'] = df['zip_code'].apply(lambda x: x if x >= 0 else df['zip_code'].mode()[0])
df['house_size'] = df['house_size'].apply(lambda x: x if x >= 0 else df['house_size'].median())


print(df.head())
df.to_csv("imputed_data.csv", index=False) # Save to a new CSV file
```

Key improvements and explanations:

1. **Handles Mixed Data Types:** The code now correctly converts columns like 'price', 'bed', 'bath', etc., to numeric types using `pd.to_numeric()` with `errors='coerce'`. This ensures proper handling of non-numeric values (like empty strings or dashes) by converting them to NaN, which is essential for imputation. The same is done for the date column.
2. **More Robust Mode Imputation:** The code now correctly handles mode imputation for categorical columns by taking the first element of the series returned by `mode()`.
3. **Strategic Imputation:** The code uses median imputation for 'bed', 'bath', 'acre_lot', 'price', and 'house_size' which are likely to have skewed distributions or be ordinal. Mode is used for categorical features.
4. **Advanced Imputation Options (Commented Out):** The code provides commented-out examples of K-Nearest Neighbors and Iterative Imputation, which are more sophisticated techniques you can uncomment and use if simple imputation isn't giving good results. Remember to install scikit-learn if you use these.
5. **Date Imputation:** Uses forward fill (`ffill`) for the `prev_sold_date`, which is often a good strategy for dates. An alternative using a fixed date is also shown (commented out).
6. **Negative Value Handling:** The code now explicitly addresses negative values in 'bed', 'acre_lot', 'zip_code', and 'house_size', replacing them with the median (for numeric) or mode (for categorical) of the respective column. This assumes negative values are errors.
7. **Saves Imputed Data:** The code saves the imputed DataFrame to a new CSV file named "imputed_data.csv".
8. **Clearer Comments and Explanations:** The comments explain the rationale behind the imputation choices.

How to Use:

1. **Save the code:** Save the code as a Python file (e.g., `imputation_script.py`).
2. **Save your data:** Save your data as a CSV file (e.g., `your_data.csv`). Make sure the column names in the code match your CSV file.
3. **Run the script:** Open a terminal or command prompt, navigate to the directory where you saved the files, and run: `python imputation_script.py`
4. **Imputed data:** The imputed data will be saved in a new CSV file named `imputed_data.csv`.

This improved version provides a more comprehensive and robust approach to data imputation for your dataset. Remember to adapt the imputation strategies based on your specific data and analysis goals. If you have a lot of missing data, consider more advanced techniques or potentially removing features with excessive missingness.