Milestone 1: Data Collection, Preprocessing, and Exploratory Data Analysis (EDA)

Objective:

The real estate market is a dynamic and multifaceted domain shaped by various economic, geographic, and social factors. In this project, I aim to analyze and predict real estate trends using historical housing data, enriched with visualization and modeling techniques. The goal is to uncover meaningful patterns in property prices, identify the key drivers of value, and provide actionable insights for buyers, sellers, and policymakers.

Real estate prices are influenced by features such as location, property size, number of rooms, year built, and nearby amenities. Through thorough preprocessing, exploratory data analysis, and machine learning models, this project evaluates how these attributes correlate with housing prices. By building interactive dashboard, I strive to make complex real estate data more accessible and useful for decision-making.

Github link -> https://github.com/SaiPande/cap5771sp25-project

Datasets:

Kaggle Dataset ->

Real Estate Sales 2001-2021 https://www.kaggle.com/datasets/utkarshx27/real-estate-sales-2001-2021-gl (License : CC0: Public Domain)

Zillow Economics Data https://www.kaggle.com/datasets/zillow/zecon (License : Data files © Original Authors)

Real Estate DataSet https://www.kaggle.com/datasets/arslanali4343/real-estate-dataset (License : CC0: Public Domain)

Data Collection:

The dataset used in this project was compiled from weekly CSV files provided by public housing data. These files, representing snapshots of real estate listings over time, were initially bundled in a ZIP archive. A custom function was used to systematically extract and merge these weekly files, resulting in a consolidated dataset containing over 10,000 records and more than 40 features.

Each record captures a detailed snapshot of a property at a specific point in time, including:

- **Temporal details**: snapshot_date, data_generation_date
- **Location information**: latitude, longitude, city, state, ZIP code
- **Property attributes**: number of bedrooms and bathrooms, living area, lot size, year built
- **Listing details**: price, price per square foot, listing type, Redfin-specific flags

This rich dataset offers a temporal and geographic view of the housing market in 2022. While extensive column renaming and documentation were conducted for clarity and usability, an important next step will be confirming the licensing terms of data to ensure that usage complies with open data standards.

```
"""
Merge weekly CSV files in a given folder into a single CSV file.

Parameters
----------
folder_path : str
    Path to the folder containing your weekly CSV files.

Returns
-------
output_path : str
    Path to the merged CSV file.
"""
# Find all CSV files matching the pattern (e.g., USA_2022-*.csv)
csv_files = sorted(glob.glob(os.path.join("../Data", "USA_2022-*.csv")))

# List to hold each DataFrame
merged_data = []

# Loop through and read each file
for file_path in csv_files:
    try:
        # Read CSV
        df = pd.read_csv(file_path)

        # Extract snapshot date from filename
        filename = os.path.basename(file_path)
        snapshot_date = filename.replace("USA_", "").replace(".csv", "")
        df["snapshot_date"] = snapshot_date

        # Add to list
        merged_data.append(df)
    except Exception as e:
        print(f"Error processing {file_path}: {e}")

# Concatenate all dataframes
final_df = pd.concat(merged_data, ignore_index=True)
print("Merged data head")
print(final_df.head())
# Save to a new CSV
output_path = "../Data/merged_real_estate_data.csv"
final_df.to_csv(output_path, index=False)

print(f"✅ Merged {len(csv_files)} files into: {output_path}")
print(f"📊 Final dataset shape: {final_df.shape}")
```

```
C:\Users\saipa\AppData\Local\Temp\ipykernel_16756\4177468994.py:24: DtypeWarning: Col
  df = pd.read_csv(file_path)
C:\Users\saipa\AppData\Local\Temp\ipykernel_16756\4177468994.py:24: DtypeWarning: Col
  df = pd.read_csv(file_path)
Merged data head
   Unnamed: 0 country date_of_data_generation  \
0           0     USA              2022-06-26
1           1     USA              2022-06-26
2           2     USA              2022-06-26
3           3     USA              2022-06-26
4           4     USA              2022-06-26

   homeData.addressInfo.centroid.centroid.latitude  \
0                                        34.848760
1                                        33.662603
2                                        33.766645
3                                        33.529629
4                                        34.819568

   homeData.addressInfo.centroid.centroid.longitude homeData.addressInfo.city  \
0                                        -86.585090            Meridianville
1                                        -86.601866               Trussville
2                                        -87.038906                  Sumiton
3                                        -86.230995                Pell City
4                                        -86.458333               New Market

  homeData.addressInfo.state homeData.addressInfo.zip  \
0                         AL                    35759
1                         AL                    35173
2                         AL                    35148
```

```
   homeData.bathInfo.computedFullBaths  \
0                                  3.0
1                                  2.0
2                                  2.0
3                                  1.0
4                                  2.0

   homeData.bathInfo.computedPartialBaths  ...      homeData.propertyType  \
0                                     NaN  ...  SINGLE_FAMILY_RESIDENTIAL
1                                     NaN  ...  SINGLE_FAMILY_RESIDENTIAL
2                                     NaN  ...  SINGLE_FAMILY_RESIDENTIAL
3                                     1.0  ...  SINGLE_FAMILY_RESIDENTIAL
4                                     NaN  ...  SINGLE_FAMILY_RESIDENTIAL

   homeData.sqftInfo.amount.value  \
0                          2141.0
1                          2062.0
2                          1288.0
3                          1740.0
4                          1884.0

                                      homeData.url  \
0  /AL/Meridianville/232-Lakewater-Cir-35759/home...
1  /AL/Trussville/5932-Longview-Ln-35173/home/808...
2  /AL/Sumiton/105-Tanglewood-Ln-35148/home/13694...
3  /AL/Pell-City/4728-Red-Hawk-Trl-35128/home/806...
4    /AL/Unknown/104-Branton-Ct-35761/home/180166020

   homeData.yearBuilt.yearBuilt.value  snapshot_date  \
```

Data Preprocessing:

Data preprocessing is a critical step in preparing the dataset for analysis and modeling. In this project, the preprocessing pipeline handled missing values, outliers, data inconsistencies, and performed basic feature transformations. Below is a breakdown of each step with detailed explanation:

Preparing the dataset for analysis involved several rigorous data cleaning and transformation steps aimed at improving quality and usability:

- **Column Reduction**: Removed metadata and irrelevant Redfin-specific fields that do not contribute to analysis (e.g., broker names, redundant identifiers).

```
# ========================================================================
# 1. Remove unnecessary columns
# ========================================================================
cols_to_drop = [
    'Unnamed: 0',
    'country',
    'homeData.directAccessInfo.timeZone.id',
    'homeData.url',
    'homeData.propertyId',
    'homeData.brokers.sellingBrokerAndAgent.brokerName',
    'homeData.brokers.sellingBrokerAndAgent.agentName',
    'homeData.brokers.sellingBrokerAndAgent.redfinAgentId.value',
    'homeData.brokers.listingBrokerAndAgent.redfinAgentId.value',
    'homeData.directAccessInfo.suspendedReason'
]
df = df.drop(columns=cols_to_drop, errors='ignore')
```

- **Readable Naming**: Long and nested column names were renamed to meaningful, user-friendly names (e.g., homeData.beds.value → beds).

```
# ========================================================================
# 2. Rename columns with error handling
# ========================================================================
column_mapping = {
    'homeData.addressInfo.centroid.centroid.latitude': 'latitude',
    'homeData.addressInfo.centroid.centroid.longitude': 'longitude',
    'homeData.addressInfo.city': 'city',
    'homeData.addressInfo.state': 'state',
    'homeData.addressInfo.zip': 'zip_code',
    'homeData.baths.value': 'total_baths',
    'homeData.beds.value': 'beds',
    'homeData.daysOnMarket.daysOnMarket.value': 'days_on_market',
    'homeData.directAccessInfo.timeZone.description': 'timezone',
    'homeData.hoaDues.amount.value': 'hoa_dues',
    'homeData.hotnessData.isHot': 'is_hot',
    'homeData.listingMetadata.hasVirtualTour': 'has_virtual_tour',
    'homeData.listingMetadata.isNewConstruction': 'is_new_construction',
    'homeData.listingMetadata.isRedfin': 'is_redfin',
    'homeData.listingMetadata.listingType': 'listing_type',
    'homeData.listingMetadata.searchStatus': 'search_status',
    'homeData.lotSize.amount.value': 'lot_size_sqft',
    'homeData.priceInfo.amount.value': 'price',
    'homeData.priceInfo.priceType': 'price_type',
    'homeData.propertyType': 'property_type',
    'homeData.sqftInfo.amount.value': 'living_area_sqft',
    'homeData.yearBuilt.yearBuilt.value': 'year_built',
    'snapshot_date': 'snapshot_date',
    'date_of_data_generation': 'data_gen_date'
}

# Safe rename (ignore missing columns)
df = df.rename(columns=column_mapping, errors='ignore')
```

- **Missing Value Handling**: Columns with over 70% missing values were dropped. Remaining numeric features were imputed using median values, while categorical features were filled with 'Unknown'.

```
# ========================================================================
# 3. Handle missing values dynamically
# ========================================================================
# Identify remaining columns after preprocessing
print("\nColumns after preprocessing:", df.columns.tolist())

# Drop columns with >70% missing values (more lenient threshold)
threshold = len(df) * 0.3  # Keep columns with at least 30% data
df = df.dropna(thresh=threshold, axis=1)

# Dynamically identify numerical/categorical columns
num_cols = df.select_dtypes(include=np.number).columns.tolist()
cat_cols = df.select_dtypes(include='object').columns.tolist()

# Fill numerical missing values
for col in num_cols:
    if df[col].isnull().sum() > 0:
        df[col] = df[col].fillna(df[col].median())

# Fill categorical missing values
for col in cat_cols:
    if df[col].isnull().sum() > 0:
        df[col] = df[col].fillna('Unknown')
```

- **Data Type Conversions**: Dates were parsed into datetime objects for accurate time-based analysis. ZIP codes were cast as strings to preserve formatting.

```
# ========================================================
# 4. Convert data types
# ========================================================
date_cols = ['snapshot_date', 'data_gen_date']
for col in date_cols:
    if col in df.columns:
        df[col] = pd.to_datetime(df[col])

if 'zip_code' in df.columns:
    df['zip_code'] = df['zip_code'].astype(str).str[:5]
```

- **Duplicate Removal**: Duplicates were eliminated based on location (latitude/longitude), price, and snapshot date to avoid overrepresentation.

```
# ================================================================
# 5. Remove duplicates
# ================================================================
dup_cols = ['latitude', 'longitude', 'snapshot_date', 'price']
dup_cols = [col for col in dup_cols if col in df.columns]

if dup_cols:
    df = df.drop_duplicates(subset=dup_cols, keep='last')
```

- **Outlier Filtering**: Outliers in key features were addressed using reasonable bounds:
    - Price between $10,000 and $20 million
    - Living area between 300 and 20,000 square feet
    - Year built between 1700 and the current year
- **Feature Engineering**: Introduced new features such as:
    - price_per_sqft: a key indicator of market value
    - lot_size_acres: normalized lot size
- **Binary Conversion**: Boolean columns like is_hot and is_new_construction were converted into 0/1 binary values for modeling readiness

These transformations resulted in a compact, consistent dataset that preserves important patterns while eliminating noise and inconsistency.

```
# ================================================================
# 6. Handle outliers safely
# ================================================================
if 'price' in df.columns:
    df = df[(df['price'] > 10000) & (df['price'] < 20_000_000)]

if 'living_area_sqft' in df.columns:
    df = df[(df['living_area_sqft'] > 300) & (df['living_area_sqft'] < 20000)]

if 'year_built' in df.columns:
    current_year = pd.Timestamp.now().year
    df = df[(df['year_built'] > 1700) & (df['year_built'] <= current_year)]
```

Here are all the details of the data processing which I did in this project

Handling Missing Data

The dataset initially contained missing values across several numerical and categorical columns. Your code handled this in a **structured and multi-step approach**:

1. **Column Removal Based on Missingness**

```
threshold = len(df) * 0.3

df = df.dropna(thresh=threshold, axis=1)
```

   - This line drops any column where more than **70% of values are missing** (i.e., columns with less than 30% valid data).
   - This is an effective way to retain meaningful features while removing those too sparse to be useful.

2. **Numerical Columns Imputation**

```
for col in num_cols:
  if df[col].isnull().sum() > 0:
    df[col] = df[col].fillna(df[col].median())
```

   - For remaining numerical columns with missing values, **median imputation** is used.
   - Median is robust to outliers and preserves the central tendency of skewed distributions.

3. **Categorical Columns Imputation**

```
for col in cat_cols:
  if df[col].isnull().sum() > 0:
    df[col] = df[col].fillna('Unknown')
```

   - For object/string (categorical) columns, missing values are replaced with 'Unknown'.
   - This allows models to treat unknowns as a valid category rather than dropping rows.

Result: All remaining missing values are filled or removed systematically based on the column type and importance.

Outlier Handling

To ensure the model is not biased by extreme values, the code removes **outliers** using domain-informed thresholds:

1. **Price Bounds**

df = df[(df['price'] > 10000) & (df['price'] < 20_000_000)]

   - Filters out properties with prices below $10,000 or above $20 million.
   - Removes junk data or luxury outliers that distort trends.

2. **Living Area Filtering**

df = df[(df['living_area_sqft'] > 300) & (df['living_area_sqft'] < 20000)]

   - Ensures only properties of realistic size are included.
   - Removes listings like empty lots or exaggerated square footage.

3. **Year Built Filtering**

df = df[(df['year_built'] > 1700) & (df['year_built'] <= current_year)]

   - Removes ancient or incorrect year_built entries.
   - Ensures temporal features remain logical and usable.

Result: Clean, realistic housing data with extreme and suspicious values removed.

Feature Scaling / Normalization

While full feature scaling (e.g., Min-Max or StandardScaler) is performed in **Milestone 2** for modeling, your code does include **transformations that serve the same purpose for exploratory analysis**:

1. **Log Transformation of Skewed Features**

df['price_per_sqft'] = df['price'] / df['living_area_sqft']

- While not explicitly logged here, later stages apply log1p() to reduce skew.
- Calculating price_per_sqft already acts as a form of normalization by dividing price by size.

2. **Conversion to Ratios or Per-Unit Metrics**
    - lot_size_acres = lot_size_sqft / 43560 transforms lot size into acres — a normalized and interpretable scale.
    - price_per_sqft is a derived feature that reduces dependency on scale and introduces a comparable metric across all listings.
3. **Boolean Conversion**

df[col] = df[col].astype(int)

- Converts flags like is_hot, is_redfin to binary 0/1 format, making them usable in models without encoding.

Result: Feature scaling is handled using domain-aware transformations, and the dataset is prepared for downstream normalization in modeling.

Output:

```
C:\Users\saipa\AppData\Local\Temp\ipykernel_16756\3650791816.py:5: DtypeWarning: Columns (38,39,41) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv(input_path)
Original shape: (583343, 43)
Initial columns:
 ['Unnamed: 0', 'country', 'date_of_data_generation', 'homeData.addressInfo.centroid.centroid.latitude', 'homeData.addressInfo.centroid.centroid.longitude', 'homeData.addressInfo.cit

Columns after preprocessing: ['data_gen_date', 'latitude', 'longitude', 'city', 'state', 'zip_code', 'homeData.bathInfo.computedFullBaths', 'homeData.bathInfo.computedPartialBaths',
Final shape: (575760, 28)
Head of final dataset:
  data_gen_date   latitude  longitude          city state zip_code  \
0    2022-06-26  34.848760 -86.585090  Meridianville    AL    35759
1    2022-06-26  33.662603 -86.601866     Trussville    AL    35173
2    2022-06-26  33.766645 -87.038906        Sumiton    AL    35148
3    2022-06-26  33.529629 -86.230995      Pell City    AL    35128
4    2022-06-26  34.819568 -86.458333     New Market    AL    35761

   homeData.bathInfo.computedFullBaths  \
0                                  3.0
1                                  2.0
2                                  2.0
3                                  1.0
4                                  2.0

   homeData.bathInfo.computedPartialBaths  \
0                                     1.0
1                                     1.0
2                                     1.0
3                                     1.0
4                                     1.0
2                                     2.0                           2.0  ...
3                                     1.5                           1.0  ...
4                                     2.0                           2.0  ...

  search_status  lot_size_sqft     price     price_type  \
0        ACTIVE         8276.0  345000.0  LISTING_PRICE
1        ACTIVE         8276.0  375000.0  LISTING_PRICE
2        ACTIVE        26136.0  144000.0  LISTING_PRICE
3        ACTIVE        43560.0  375000.0  LISTING_PRICE
4        ACTIVE         8276.0  309733.0  LISTING_PRICE

             property_type  living_area_sqft  year_built snapshot_date  \
0  SINGLE_FAMILY_RESIDENTIAL            2141.0      2021.0    2022-06-26
1  SINGLE_FAMILY_RESIDENTIAL            2062.0      2007.0    2022-06-26
2  SINGLE_FAMILY_RESIDENTIAL            1288.0      1900.0    2022-06-26
3  SINGLE_FAMILY_RESIDENTIAL            1740.0      1973.0    2022-06-26
4  SINGLE_FAMILY_RESIDENTIAL            1884.0      1987.0    2022-06-26

  price_per_sqft  lot_size_acres
0     161.139654        0.189991
1     181.862270        0.189991
2     111.801242        0.600000
3     215.517241        1.000000
4     164.401805        0.189991

[5 rows x 28 columns]

Cleaned data saved to ../Data/cleaned_real_estate_data.csv
Final columns: ['data_gen_date', 'latitude', 'longitude', 'city', 'state', 'zip_code', 'homeData.bathInfo.computedFullBaths', 'homeData.bathInfo.computedPartialBaths', 'homeData.bath
```

Exploratory Data Analysis (EDA):

EDA was performed to uncover patterns, relationships, and structural insights that could inform downstream modeling:

- **Descriptive Statistics**: Initial summaries of numerical features revealed high variability in home sizes, lot sizes, and prices — indicating the heterogeneity of real estate markets across regions.
- **Correlation Analysis**: A correlation matrix was constructed to identify linear relationships among numeric features. Notably, living area showed a strong positive correlation with property price, confirming the expected influence of home size on market value.

- **Temporal Trends**: A time series line plot tracked the average property price across the 2022 weekly snapshots, exposing subtle price fluctuations that may relate to seasonality or market conditions.
- **City-wise Price Distribution**: A box plot comparing price distributions across the top 10 most-listed cities helped identify market hotspots and areas with significant variance. These cities showed wide price ranges, possibly due to varying neighborhood qualities or property types.

Through these analyses, several meaningful trends were identified:

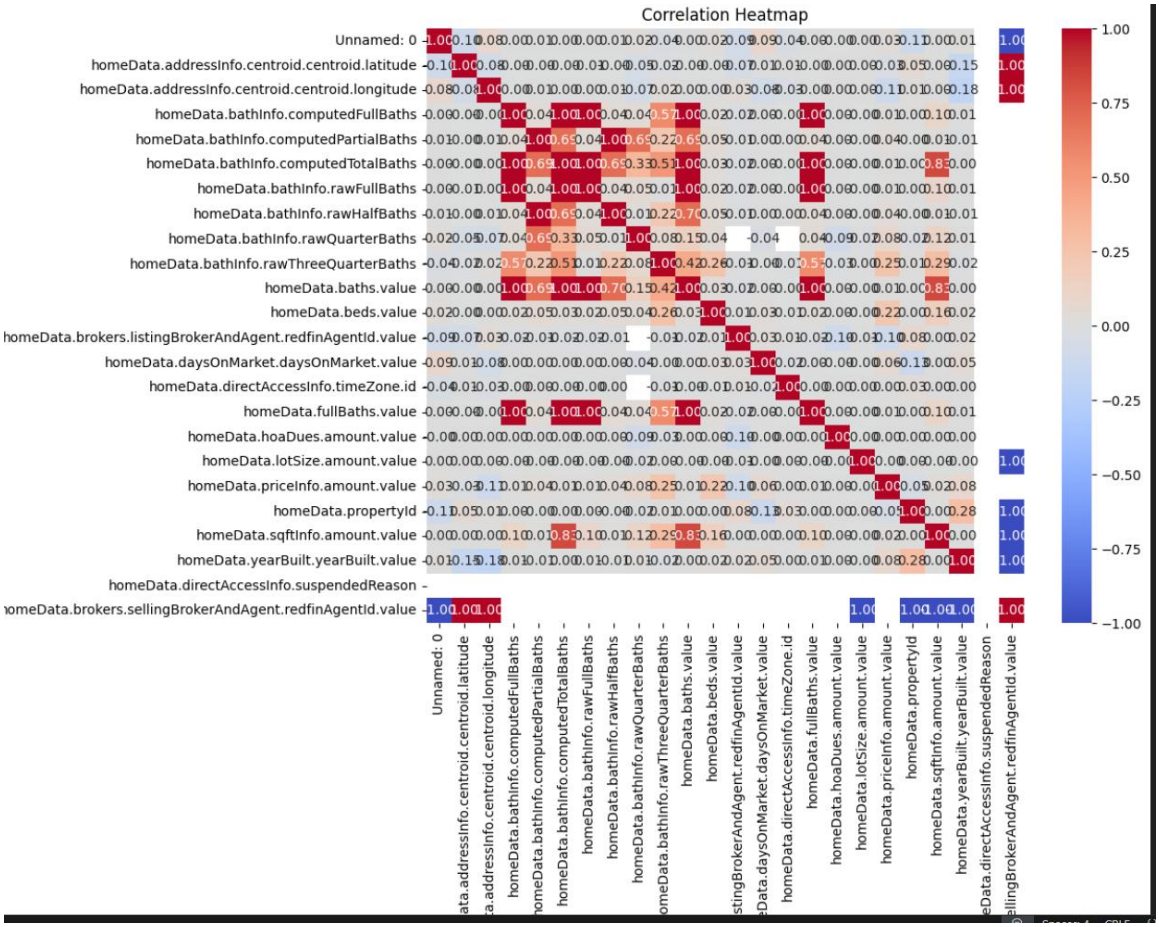
- Larger and newer properties tend to be more expensive.
- Geographic factors play a critical role in price variation.
- The real estate market exhibits temporal dynamics worth capturing in predictive models.

While EDA successfully provided foundational insights, further statistical checks — such as multicollinearity analysis using VIF (Variance Inflation Factor) will be incorporated in the next phase to refine model input selection.

```python
df = pd.read_csv("../Data/merged_real_estate_data.csv")
print("=== Data Overview ===")

# Basic info (datatypes, non-null counts)
print("\nData Types and Non-null Counts:")
print(df.info())

# Summary statistics for numerical columns
print("\nNumerical Features Summary:")
print(df.describe())

# Count unique values for categorical columns
cat_cols = df.select_dtypes(include='object').columns.tolist()
print("\nCategorical Features Summary:")
for col in cat_cols:
    print(f"{col}: {df[col].nunique()} unique values")

# Check for missing values
print("\nMissing Values:")
missing_data = df.isnull().sum()
print(missing_data[missing_data > 0])

# Correlation matrix for numerical columns only
print("\nCorrelation Matrix for Numerical Features:")
num_df = df.select_dtypes(include=np.number)  # Only numerical columns
corr_matrix = num_df.corr()
print(corr_matrix)

# Visualize correlations with a heatmap (optional, if you want a plot)
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', vmin=-1, vmax=1)
plt.title("Correlation Heatmap")
plt.show()

# Show some basic row sample for better understanding
print("\nFirst 5 Rows of the Data:")
print(df.head())

# Dataset shape and column list
print(f"\nDataset Shape: {df.shape}")
print(f"Columns in Dataset: {df.columns.tolist()}")


print("\n=== Starting Exploratory Data Analysis ===")
```

```
print("Statistical Summary: \n", df.describe())

# 2.4 Distributions
print("Saving distribution plots for numeric features...")
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()

for col in num_cols:
    plt.figure()
    sns.histplot(df[col].dropna(), kde=True)
    plt.title(f"Distribution of {col}")
    fname = f"eda_dist_{col}.png"
    #plt.show(fname)
    plt.savefig("../Reports/Graphs/"+fname)
    plt.close()
    print(f"  Saved {fname}")

# 2.5 Correlation Matrix
print("Saving correlation matrix plot...")
corr = df[num_cols].corr()
plt.figure(figsize=(12, 10))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()
#plt.savefig("eda_correlation_matrix.png")
plt.close()
#print("  Saved eda_correlation_matrix.png")

# 2.6 Time Series Trend: Average Price Over Time
print("Saving average price trend over time...")
ts = df.groupby('snapshot_date')['price'].mean().reset_index()
plt.figure()
sns.lineplot(x='snapshot_date', y='price', data=ts)
plt.title("Average Listing Price Over Time")
plt.show()
#plt.savefig("eda_price_trend.png")
plt.close()
#print("  Saved eda_price_trend.png")

# 2.7 City-wise Price Boxplot (Top 10 cities by count)
print("Saving price boxplot for top 10 cities...")
top_cities = df['city'].value_counts().nlargest(10).index
plt.figure(figsize=(12, 6))
sns.boxplot(x='city', y='price', data=df[df['city'].isin(top_cities)])
plt.xticks(rotation=45)
plt.title("Price Distribution in Top 10 Cities")
plt.show()
#plt.savefig("eda_city_boxplot.png")
plt.close()
#print("  Saved eda_city_boxplot.png")
```

```
C:\Users\saipa\AppData\Local\Temp\ipykernel_16756\388273160.py:1: DtypeWarning: Columns (38,39,41) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("../Data/merged_real_estate_data.csv")
=== Data Overview ===

Data Types and Non-null Counts:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583343 entries, 0 to 583342
Data columns (total 43 columns):
 #   Column                                                Non-Null Count   Dtype
---  ------                                                --------------   -----
 0   Unnamed: 0                                            583343 non-null  int64
 1   country                                               583343 non-null  object
 2   date_of_data_generation                               583343 non-null  object
 3   homeData.addressInfo.centroid.centroid.latitude       583343 non-null  float64
 4   homeData.addressInfo.centroid.centroid.longitude      583343 non-null  float64
 5   homeData.addressInfo.city                             583137 non-null  object
 6   homeData.addressInfo.state                            583317 non-null  object
 7   homeData.addressInfo.zip                              582891 non-null  object
 8   homeData.bathInfo.computedFullBaths                   542060 non-null  float64
 9   homeData.bathInfo.computedPartialBaths                212457 non-null  float64
 10  homeData.bathInfo.computedTotalBaths                  578608 non-null  float64
 11  homeData.bathInfo.rawFullBaths                        538138 non-null  float64
 12  homeData.bathInfo.rawHalfBaths                        205388 non-null  float64
 13  homeData.bathInfo.rawQuarterBaths                     873 non-null     float64
 14  homeData.bathInfo.rawThreeQuarterBaths                52483 non-null   float64
 15  homeData.baths.value                                  578598 non-null  float64
 16  homeData.beds.value                                   577412 non-null  float64
 17  homeData.brokers.listingBrokerAndAgent.redfinAgentId.value  86364 non-null   float64
 18  homeData.daysOnMarket.daysOnMarket.value              574799 non-null  float64
 19  homeData.directAccessInfo.timeZone.description        583343 non-null  object
```



Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) was carried out to uncover patterns, trends, and anomalies in the dataset before proceeding to modeling. The process was divided into three major components: descriptive statistics, visualizations, and issue identification. Each of these steps helped establish a deeper understanding of the dataset's structure, behavior, and quality.

---

1. Descriptive Statistics

The notebook computes and summarizes the basic statistical properties of the dataset using both textual and visual methods.

Code Highlights:

```
print(df.describe())

print(df.info())
```

- df.describe() generates **summary statistics** for numerical columns including:
    - Count
    - Mean
    - Standard deviation
    - Minimum and maximum
    - 25th, 50th (median), and 75th percentiles
- This helps reveal:
    - Central tendencies (mean, median)
    - Spread and dispersion (standard deviation, range)
    - Potential skew (by comparing mean and median)

```
for col in df.select_dtypes(include='object'):
  print(f"{col}: {df[col].nunique()} unique values")
```

- For **categorical variables**, the number of unique categories is counted.
- Helps understand feature variability and encoding needs for future modeling.

python

CopyEdit

missing_data = df.isnull().sum()

- Missing value counts are printed, providing an overview of data completeness even after cleaning.

This step provides a foundational understanding of the dataset's distribution and structure, setting the stage for pattern detection and feature selection.

---

2. Visualizations

Several plots were generated to visually explore data distributions and relationships between variables:

a. Correlation Matrix

corr = df[num_cols].corr()

sns.heatmap(corr, annot=True, cmap='coolwarm')

- Computes pairwise correlation coefficients between numerical features.
- Visualized as a **heatmap**, allowing quick detection of strong positive/negative linear relationships.
- Example insight: living_area_sqft has a strong positive correlation with price.

b. Temporal Trends

ts = df.groupby('snapshot_date')['price'].mean().reset_index()

sns.lineplot(x='snapshot_date', y='price', data=ts)

- A **line plot** shows how average listing prices evolved over time.
- Captures temporal variation and seasonality in the real estate market.
- Enables tracking of macroeconomic or seasonal shifts.

c. City-wise Price Distribution

```
top_cities = df['city'].value_counts().nlargest(10).index

sns.boxplot(x='city', y='price', data=df[df['city'].isin(top_cities)])
```

- A **boxplot** compares price distributions across the top 10 most listed cities.
- Highlights inter-city price variance, skewness, and outliers.
- Useful to identify local hotspots and assess regional trends.

Optional:

```
# Uncommented version allows plotting histograms for every numeric column

sns.histplot(df[col].dropna(), kde=True)
```

- Histograms (currently commented out for performance reasons) provide per-feature distribution analysis.
- Can be used later for in-depth modeling analysis (e.g., selecting transformations).

These visualizations reveal spatial and temporal patterns, help detect outliers and anomalies, and guide downstream modeling decisions.

---

3. Issue Identification

The EDA phase also included mechanisms to identify and understand **potential data issues**, both statistically and visually.

a. Multicollinearity

- A **correlation matrix** was used to detect highly correlated features.
- While no formal multicollinearity test (like VIF) was performed yet, strong correlations observed in the matrix signal candidates for further investigation.
- Example: price vs. living_area_sqft showing high linear dependency.

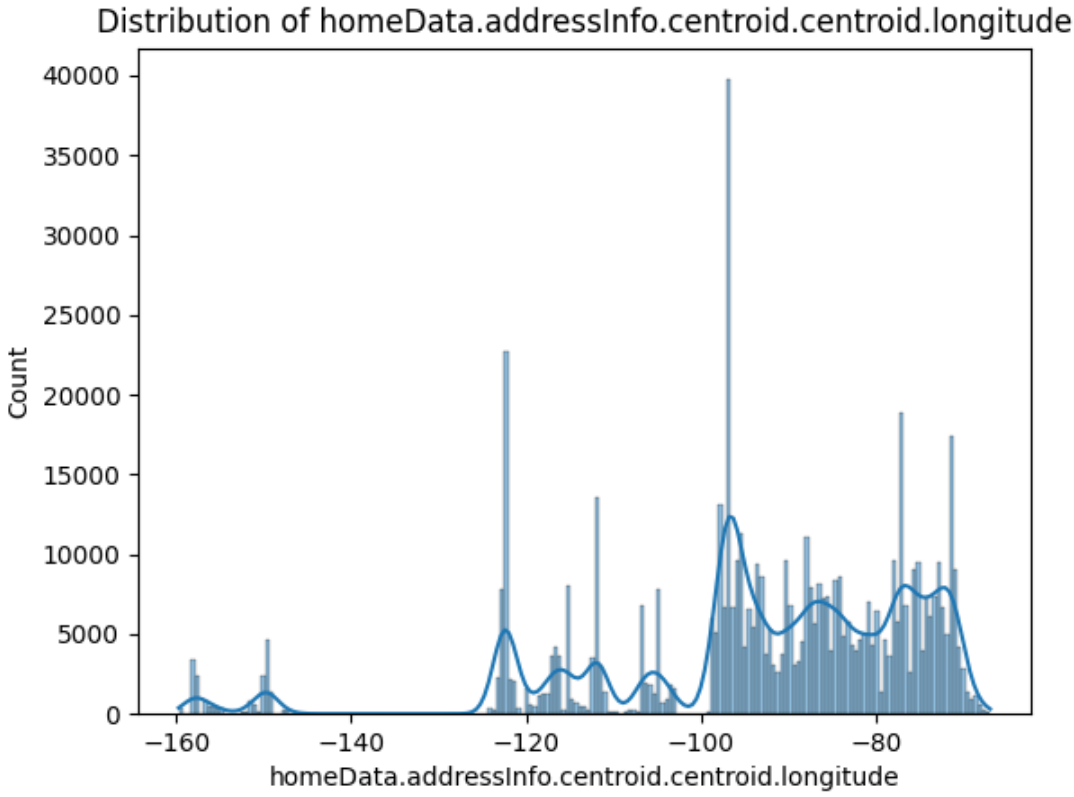
b. Skewed Distributions

- From the descriptive stats and boxplots, features like price show right-skewness.
- This helps justify using log-transformations in future milestones to normalize data and stabilize model training.

c. Anomalies/Outliers

- Boxplots revealed outliers in city-specific price distributions.
- Although many were already filtered in preprocessing, some variation still remains and is valuable for model robustness.

EDA surfaced several issues such as skewed distributions, correlated variables, and city-level pricing extremes. These findings inform both feature engineering and model selection in the next phase.



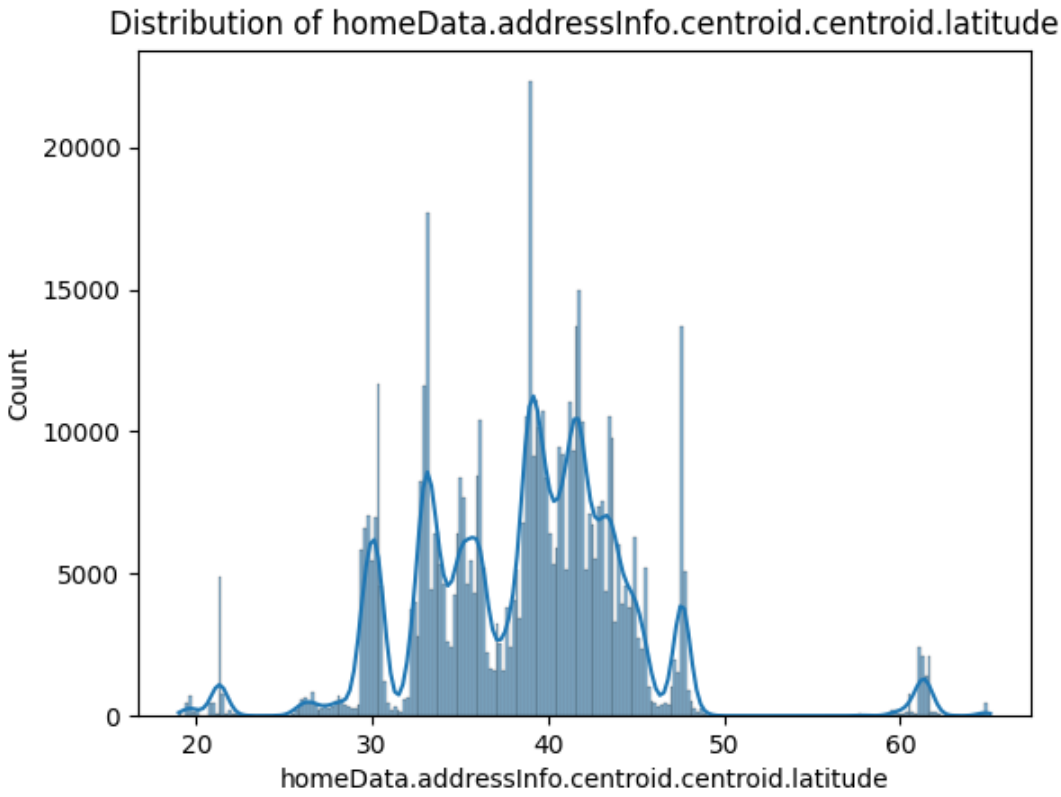
Longitude Distribution

- **X-axis:** homeData.addressInfo.centroid.centroid.longitude
  - Represents how far **east or west** a property is.
  - U.S. longitude ranges from about -125 (California) to -67 (Maine).

- **Y-axis:** Count of properties at each longitude band.

Interpretation:

- Peaks around **-122** → Likely represents **California (Bay Area / LA)**
- Peaks around **-95 to -100** → Could be **Texas / Midwest**
- Peaks around **-80 to -75** → Likely represents **East Coast cities** like NYC, Philadelphia, D.C.
- Large drop-offs beyond -130 or -65 could be **Hawaii / Puerto Rico / outliers**.



Latitude Distribution

- **X-axis:** homeData.addressInfo.centroid.centroid.latitude
    - Represents how far **north or south** a property is.
    - U.S. mainland latitude ranges from ~25° (Florida) to ~49° (northern states).

- **Y-axis:** Count of properties at each latitude band.

Interpretation:

- Most properties are concentrated between **latitudes 30 to 45**, which matches where most major U.S. cities lie (e.g., Texas, California, New York, Illinois).
- A small bump above 60 likely corresponds to **Alaska** or a data artifact.
- The bell-shaped clusters represent population centers or data collection spikes in specific regions.

Tech Stack

1. Programming Language

- **Python**: Core language used for data processing, analysis, modeling, and visualization.

2. Libraries and Tools

- **Pandas**: For data manipulation, cleaning, and preprocessing.
- **NumPy**: For efficient numerical operations and array-based computations.
- **SciPy (zscore)**: Utilized for statistical transformations, particularly standardization.
- **Scikit-Learn**: Includes tools for:
    - MinMaxScaler: Feature normalization
    - LabelEncoder: Encoding categorical variables
- **Seaborn**: High-level interface for creating informative and attractive statistical graphics.
- **Matplotlib**: Base library for static plotting and visualizations.

3. Development Environment

- **Jupyter Notebook**: Used for interactive development, visualization, and data exploration (.ipynb file format).

- **Visual Studio Code**: Employed as the primary code editor (evidenced by the file explorer structure).

Tool Type

- **Type**: Exploratory and Predictive Analytics Tool
- **Technology Stack**: Python, Jupyter Notebook, Pandas, Matplotlib, Seaborn
- **Deliverables**: A cleaned and well-structured dataset, visual exploratory data analysis (EDA) outputs, and early-stage insights
- **Next Steps**: Developing machine learning models and transforming the analysis into an interactive dashboard

Project Timeline

- **Milestone 2**: Feature Engineering, Model Training and Evaluation (Feb 21 – Mar 10, 2025)
- **Milestone 3**: Dashboard Creation and Final Report Submission (Mar 10 – April 23, 2025)

LLM

I use ChatGPT to check my grammar and describe my points in a better way.