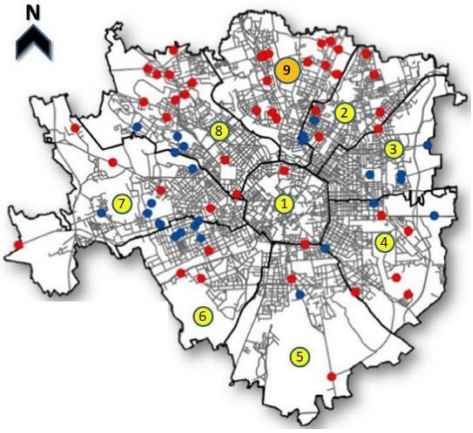
Milan House Prices

Amir Hemati **02.02.2023**

https://amir-hemati.github.io/portfolio/

https://www.linkedin.com/in/amir-hemati/



Chapter 1:

Introduction

About Milan:

Milan, the capital city of the Lombardy region in northern Italy, is a vibrant and cosmopolitan metropolis. It is located in the north-central part of the country, serving as the cultural, economic, and financial center of Italy. With its rich history and modern allure, Milan has established itself as one of the most prominent cities in Europe.

Dating back to ancient times, Milan has evolved from a small settlement to a thriving urban hub. The city is renowned for its contributions to art, fashion, design, and commerce. It is home to numerous architectural landmarks, including the famous Duomo di Milano, a magnificent cathedral that stands as an iconic symbol of the city.

As one of the most populous cities in Italy, Milan attracts residents and visitors alike with its bustling energy and diverse population. The city spans an area of approximately 70 square miles (181 square km). According to population data from 2016, Milan had a population of 1.35 million people within the city limits and over 3.2 million in the wider metropolitan area.

Milan's significance extends beyond its borders, as it plays a pivotal role in various sectors, such as fashion, finance, design, and manufacturing. It hosts international events like Milan Fashion Week and the Milan Furniture Fair, attracting global attention and showcasing the city's creative and innovative spirit.

With its blend of historical heritage, modern infrastructure, cultural attractions, and thriving industries, Milan continues to be a vibrant and influential city in Italy and Europe as a whole.

About Data:

The dataset used in this study was acquired by employing a web scraping technique on the Immobiliare website, a prominent platform specializing in house announcements. The primary objective of compiling this dataset is to leverage the available variables and establish a framework for predicting house prices.

Furthermore, the project encompasses an associated GitHub repository, which provides a comprehensive set of scripts. These scripts are specifically designed to automate the process of re-scraping the Immobiliare website. By employing these scripts, the dataset can be regularly updated to incorporate newly posted house announcements, ensuring that the information remains current and reflective of the evolving real estate market.

With the dataset being subject to periodic updates, researchers and analysts can explore a wide range of housing trends and dynamics. By meticulously analyzing the dataset, valuable insights can be unearthed, shedding light on the factors that influence house prices and the patterns that underlie their fluctuations. These insights pave the way for the development of robust predictive models, capable of capturing the intricate relationships and tendencies observed in the housing market.

By harnessing the power of this regularly updated dataset, researchers have the opportunity to delve into various facets of housing trends, such as spatial variations, temporal patterns, and the impact of different variables on house prices. This enables a comprehensive exploration of the housing market, driving informed decision-making and contributing to the broader understanding of real estate dynamics.

Explanation of Columns in the Dataset:

During the analysis of the dataset, several columns are utilized to gain valuable insights and facilitate the prediction of house prices. These columns provide relevant information and contribute to a comprehensive understanding of the properties:

1. Number of Rooms: This column indicates the number of rooms in each house, offering insights into the size and accommodation capacity of the property. It serves as an essential feature for understanding the layout and potential functionality of the house.

2. M^2: The M^2 column represents the area or size of each house in square meters. It provides a quantitative measure of the property's space, enabling analysis of the relationship between size and price. This information assists in assessing the value and competitiveness of different properties.

3. Number of Bathrooms: This column captures the count of bathrooms in each house, which is an important factor in determining the property's convenience and potential desirability. It helps prospective buyers understand the level of comfort and functionality provided by the bathrooms.

4. Floor: The Floor column indicates the floor level of each apartment, ranging from basement (S) and ground floor (T) to higher floor numbers. This feature aids in evaluating the location, accessibility, and views associated with different floors. It can also offer insights into potential noise levels and privacy.

5. Condominium Expenses: The Condominium Expenses column contains information about the monthly fees associated with shared amenities and maintenance in apartment buildings. It provides valuable insights into the additional costs associated with owning or renting a property, enabling potential buyers to consider the overall financial commitment.

6. Date: The Date column represents the date when each house announcement was uploaded online. This temporal information allows for the analysis of market trends, seasonality, and changes in housing demand over time. It helps identify patterns and fluctuations in prices and market activity.

7. Typology: The Typology column denotes the type of property, providing information about the specific category or classification of each house. It includes details such as "Apartment," "Villa," "Townhouse," or other property types. This column assists in understanding the diversity of properties in the dataset and allows for tailored analyses based on property types.

8. Availability: This column indicates the availability status of a house, with values such as "libero" (available) or null (unavailable). It helps determine the current availability of properties for sale or rent, allowing users to filter and analyze properties based on their availability in the market.

9. Price: The Price column represents the cost of each house in the dataset and serves as the target variable for the prediction task. Analyzing this column involves understanding the distribution of prices, identifying outliers, and exploring the relationship between price and other variables.

10. Year of Build: This column captures the year in which each building was constructed. It offers insights into the age of the property, which is a crucial factor for buyers considering potential maintenance and renovation requirements. Understanding the age of the property helps assess its condition and potential long-term value.

11. Condition: The Condition column provides information about the overall condition of the house or property. It includes categories such as "New," "Very Good," "Good," and "To Be Restructured." This feature allows potential buyers to assess the need for any renovations or repairs, making informed decisions based on the property's condition.

12. Air Conditioning: The Air Conditioning column indicates the presence or absence of air conditioning in the house. It is represented as a Boolean value, with True indicating the presence of air conditioning and False for its absence. This information helps buyers evaluate the level of comfort and convenience provided by the property, especially in regions with extreme temperatures.

13. Energy Efficiency: This column provides information about the energy efficiency class and consumption of each property. It includes a representation of the energy class (e.g., "D

Here are some initial **exploratory data analysis (EDA)** questions that can be asked about the Milan House Data dataset:

1. **What is the size of the dataset?** - This question aims to determine the total number of rows or records in the dataset, providing an understanding of the dataset's overall scale.
2. **What are the different variables present in the dataset?** - By exploring the variables, you can identify the specific attributes or characteristics that are recorded for each house sale. This helps in understanding the available information.
3. **Are there any missing values in the dataset?** - Identifying missing values is crucial as it may affect the quality and reliability of the analysis. Assessing the presence and extent of missing values helps in deciding how to handle them appropriately.
4. **What is the range of values for each variable?** - Understanding the range of values provides insights into the spread and diversity of the data. It helps identify any extreme or unusual values that may need further investigation.
5. **Are there any outliers or unusual values in the dataset?** - Outliers or unusual values can significantly impact data analysis results. Identifying and analyzing them helps in understanding if they are valid data points or require further investigation and treatment.
6. **What is the distribution of the variables?** - Analyzing the distribution of variables provides insights into their central tendency, spread, and shape. It helps in understanding the overall patterns and characteristics of the data.
7. **Are there any correlations between variables?** - Exploring the correlations between variables helps in understanding the relationships and dependencies among different attributes. It can uncover interesting insights and guide further analysis.

These questions serve as a starting point for exploring and understanding the Milan House Sales Data dataset, enabling deeper insights into the patterns and characteristics of the data.

Importing libraries and data set:

In [1]:
```
import imp
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


import plotly.graph_objs as go
from plotly.offline import init_notebook_mode,iplot
import plotly.express as px
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
%matplotlib inline
```

```
C:\Users\Amir Hbs\AppData\Local\Temp\ipykernel_11964\334813879.py:1: DeprecationWarning: the imp module is deprecated in favour
of importlib and slated for removal in Python 3.12; see the module's documentation for alternative uses
  import imp
```

In [2]:
```
data= pd.read_excel(r"C:\Users\Amir Hbs\Desktop\Projects\Python\Milan Housing\milano_housing_02_2_23.xlsx")
df=pd.DataFrame(data)
df
```

Out[2]:

	index	rooms	m2	bathrooms	floor	condominium_expenses	date	contract	typology	total_floors	...	energy_certification	co2_emissions	elev.
0	0	3	140.0	2	4	535	08/02/2023	sale	apartment | entire property | stately property...	6 floors	...	NaN	NaN	
1	0	4	121.0	2	3	200	17/02/2023	sale	apartment | entire property | stately property...	7 floors	...	NaN	NaN	
2	0	2	55.0	1	4	133	20/02/2023	sale	attic | entire property | stately property class	4 floors	...	NaN	NaN	
3	0	2	60.0	1	5	333	11/01/2023	sale	attic | bare ownership | medium property class	5 floors	...	NaN	NaN	
4	0	4	220.0	3	4	NaN	23/01/2023	sale	apartment | entire property | stately property...	7 floors	...	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2125	0	2	70.0	1	10	135	26/02/2023	sale	apartment | entire property | stately property...	11 floors	...	NaN	NaN	
2126	0	3	90.0	2	R	183	19/02/2023	sale	apartment | entire property | medium property ...	7 floors	...	NaN	NaN	
2127	0	3	130.0	2	T	200	26/02/2023	sale	apartment | entire property | stately property...	4 floors	...	NaN	NaN	
2128	0	2	51.0	1	1	NaN	26/02/2023	sale	apartment | entire property | stately property...	5 floors	...	NaN	NaN	
2129	0	2	72.0	1	NaN	250	26/02/2023	sale	apartment | entire property | medium property ...	5 floors	...	NaN	NaN	

2130 rows × 34 columns

Exploratory data analysis (EDA):

Q1: size of dataset:

```
In [44]: df.shape
Out[44]: (2112, 34)
```

Q2: data types:

Q3: missing values(nulls):

In [4]:
```
check(df)
```

Out[4]:

	column	dtypes	nunique	sum_null
0	index	int64	1	0
1	rooms	object	29	14
2	m2	float64	276	17
3	bathrooms	object	29	25
4	floor	object	34	281
5	condominium_expenses	object	239	460
6	date	object	134	14
7	contract	object	6	14
8	typology	object	44	14
9	total_floors	object	25	47
10	availability	object	1	441
11	other_features	object	1796	36
12	price	float64	567	143
13	year_of_build	float64	105	237
14	condition	object	4	113
15	air_conditioning	object	13	519
16	energy_efficiency	object	812	154
17	city	object	1	14
18	neighborhood	object	145	14
19	car_parking	object	40	1542
20	energy_performance_building	object	3	1897
21	housing units	float64	55	2048
22	start_end_works	object	82	2045
23	current_building_use	object	4	2121
24	energy_certification	object	3	1969
25	co2_emissions	object	1	2129
26	elevator	int64	2	0
27	floor_level	object	13	101
28	heating_centralized	object	2	55
29	heating_radiator	object	7	176
30	heating_gas	object	9	378
31	air_conditiong_centralized	object	5	519
32	air_conditioning_heat	object	3	776
33	renewable_energy_performance_index_KWh/m2	float64	71	2013

Q4: Range of variables:

```
In [5]: temp = df.describe()
        temp.style.background_gradient(cmap='Oranges')
```

Out[5]:

	index	m2	price	year_of_build	housing units	elevator	renewable_energy_performance_index_KWh/m2
count	2130.000000	2113.000000	1987.000000	1893.000000	82.000000	2130.000000	117.000000
mean	0.000000	709.609087	731128.407146	1961.295827	223.585366	0.785446	2814.145299
std	0.000000	27190.783179	661979.895398	52.262219	778.123462	0.410609	5576.682527
min	0.000000	1.000000	20000.000000	1100.000000	4.000000	0.000000	1.000000
25%	0.000000	70.000000	329500.000000	1940.000000	17.000000	1.000000	351.000000
50%	0.000000	100.000000	520000.000000	1960.000000	33.000000	1.000000	351.000000
75%	0.000000	140.000000	868000.000000	1990.000000	71.500000	1.000000	1193.000000
max	0.000000	1250000.000000	5300000.000000	2025.000000	5757.000000	1.000000	24993.000000

This information provides statistical measures for each column in the dataset:

- count: The number of non-null values in each column.

- mean: The average value of each column.

- std: The standard deviation, which measures the spread of values around the mean.

- min: The minimum value observed in each column.

- 25%: The 25th percentile, also known as the first quartile, indicating the value below which 25% of the data falls.

- 50%: The 50th percentile, also known as the median, representing the middle value in the data.

- 75%: The 75th percentile, indicating the value below which 75% of the data falls.

- max: The maximum value observed in each column.

Analyzing the information:

- The "m2" column represents the area or size of each property, ranging from 1 to 1,250,000 square meters. The mean size is approximately 709 square meters, with a standard deviation of 27,190 square meters. The minimum size is 1 square meter, and the maximum size is 1,250,000 square meters.

- The "price" column represents the cost of each property, ranging from 20,000 to 5,300,000. The mean price is approximately 731,128, with a standard deviation of 661,979. The minimum price is 20,000, and the maximum price is 5,300,000.

- The "year_of_build" column represents the year in which each property was constructed, ranging from 1100 to 2025. The mean year of build is approximately 1961, with a standard deviation of 52. The minimum year of build is 1100, and the maximum year of build is 2025.

- The "housing units" column represents the number of housing units in each property, ranging from 4 to 5,757. However, this column has a count of only 82, indicating that it has a significant number of missing values.

- The "elevator" column represents the presence or absence of an elevator in the building. It is represented as a binary value, with 1 indicating the presence of an elevator. The mean value is approximately 0.79, indicating that elevators are present in a significant portion of the properties.

- The "renewable_energy_performance_index_KWh/m2" column represents the energy performance index of each property in kilowatt-hours per square meter. It ranges from 1 to 24,993. The mean value is approximately 2,814, with a standard deviation of 5,577. This column also has a significant number of missing values, as indicated by the count of 117.

These statistics provide a basic understanding of the distribution and range of values in each column, allowing for initial insights into the characteristics of the dataset. However, further analysis and exploration are necessary to gain a deeper understanding and uncover any underlying patterns or relationships between the variables.

```
In [6]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2130 entries, 0 to 2129
Data columns (total 34 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   index                                     2130 non-null   int64
 1   rooms                                     2116 non-null   object
 2   m2                                        2113 non-null   float64
 3   bathrooms                                 2105 non-null   object
 4   floor                                     1849 non-null   object
 5   condominium_expenses                      1670 non-null   object
 6   date                                      2116 non-null   object
 7   contract                                  2116 non-null   object
 8   typology                                  2116 non-null   object
 9   total_floors                              2083 non-null   object
 10  availability                              1689 non-null   object
 11  other_features                            2094 non-null   object
 12  price                                     1987 non-null   float64
 13  year_of_build                             1893 non-null   float64
 14  condition                                 2017 non-null   object
 15  air_conditioning                          1611 non-null   object
 16  energy_efficiency                         1976 non-null   object
 17  city                                      2116 non-null   object
 18  neighborhood                              2116 non-null   object
 19  car_parking                               588 non-null    object
 20  energy_performance_building               233 non-null    object
 21  housing units                             82 non-null     float64
 22  start_end_works                           85 non-null     object
 23  current_building_use                      9 non-null      object
 24  energy_certification                      161 non-null    object
 25  co2_emissions                             1 non-null      object
 26  elevator                                  2130 non-null   int64
 27  floor_level                               2029 non-null   object
 28  heating_centralized                       2075 non-null   object
 29  heating_radiator                          1954 non-null   object
 30  heating_gas                               1752 non-null   object
 31  air_conditiong_centralized                1611 non-null   object
 32  air_conditioning_heat                     1354 non-null   object
 33  renewable_energy_performance_index_KWh/m2  117 non-null    float64
dtypes: float64(5), int64(2), object(27)
memory usage: 565.9+ KB
```

Q6: The distribution of variables:

Chapter 2:

Data Cleaning

As part of the initial data cleaning process, the dataset will undergo duplicate value removal in the first step. This crucial step involves identifying and eliminating any duplicate entries present in the dataset

By removing duplicate values, we ensure that each observation within the dataset is unique, preventing any potential biases or inaccuracies that may arise from redundant data. Duplicate values can distort analysis and lead to misleading results, making it essential to address them early in the data cleaning process.

To remove duplicate values, various techniques and methods can be employed, such as utilizing built-in functions like "drop_duplicates()" in Python or applying custom logic based on specific criteria. The goal is to identify and remove duplicate records while preserving the integrity and quality of the remaining data.

By removing duplicate values, the dataset becomes more reliable and suitable for subsequent analysis, modeling, and decision-making. It helps in obtaining accurate insights, drawing reliable conclusions, and making informed data-driven decisions based on the cleaned dataset.

```
In [7]: """
        In the first step the duplicate values will be removed from dataset.
        """

        df = df.drop_duplicates()
```

```
In [8]: # looking at the data frame info, we can see that there were 28 duplicate rows in the dataset.
        df.info()

        <class 'pandas.core.frame.DataFrame'>
        Int64Index: 2112 entries, 0 to 2129
        Data columns (total 34 columns):
```

As the data cleaning process continues, the next step involves converting the data types of certain columns to numeric types. This conversion is necessary to ensure that the data is in a suitable format for analysis and modeling.

Converting data types to numeric allows for mathematical operations, statistical calculations, and other numerical manipulations to be performed accurately. It enables meaningful comparisons, aggregations, and computations on the data, providing valuable insights and facilitating the development of robust models.

During this step, columns that contain numerical information but are currently stored as non-numeric data types, such as strings or objects, will be converted to numeric types. This typically involves using functions or methods available in programming languages like Python to parse and transform the data appropriately.

By converting data types to numeric, the dataset becomes more consistent and compatible with numerical operations and statistical techniques. It helps eliminate potential errors or inconsistencies that may arise from incompatible data types and ensures the accuracy and integrity of subsequent analyses and modeling processes.

```
In [10]: # in the next step the data types will be converted to numeric types.

In [11]: def numeric_converter(df, column):
             """
            A new function has been developed to facilitate the conversion of non-numeric values into their numeric counterparts.
            This function takes two parameters: the data frame and the column to be converted.
            The result of this conversion is a column with numeric values
             """
             return pd.to_numeric(df[column].astype(str), errors='coerce')

In [33]:

In [12]: df['rooms']               = numeric_converter(df, "rooms")
         df['m2']                  = numeric_converter(df, "m2")
         df['bathrooms']           = numeric_converter(df, "bathrooms")
         df['condominium_expenses'] = numeric_converter(df, "condominium_expenses")
```

The next step focuses on cleaning the columns that contain redundant values. Redundant values refer to data entries that are duplicated or unnecessary, adding no additional information or insight to the dataset. Cleaning these columns involves identifying and removing such redundant values, streamlining the dataset and improving its quality.

By eliminating redundant values, the dataset becomes more concise and efficient, reducing data storage requirements and improving computational efficiency. It also enhances the accuracy and reliability of subsequent data analyses and modeling tasks.

During this step, various techniques can be employed to clean the columns with redundant values. These techniques may include:

1. Identifying and removing exact duplicates: This involves finding and removing rows that are exact copies of each other, keeping only one instance of each unique data entry.

2. Handling missing or null values: Redundant values may include missing or null data entries. Strategies such as imputation (replacing missing values with estimated values) or deletion (removing rows or columns with missing values) can be used to handle these cases.

3. Removing irrelevant or unused columns: Columns that do not contribute meaningful information to the analysis or modeling task can be removed to simplify the dataset and reduce noise.

4. Addressing inconsistent or conflicting values: Redundant values can also arise from inconsistencies or conflicts in the data. These can be resolved by standardizing data formats, correcting errors, or reconciling conflicting entries.

By cleaning the columns with redundant values, the dataset becomes more reliable and accurate, leading to more meaningful and reliable insights from subsequent data analyses and modeling.

In [15]:
```
df['energy_efficiency']
```

Out[15]:
```
0       d248.59 kwh/m² year
1       e≥ 3,51 kwh/m² year
2       e183.89 kwh/m² year
3        f138.8 kwh/m² year
4       g≥ 3,51 kwh/m² year
              ...
2125    b≥ 3,51 kwh/m² year
2126       f174 kwh/m² year
2127     f≥ 175 kwh/m² year
2128    f193.44 kwh/m² year
2129    f284.99 kwh/m² year
Name: energy_efficiency, Length: 2112, dtype: object
```

In [16]:
```
df['energy_efficiency']=df['energy_efficiency'].str.extract(r'(\d+[\.,]?\d*)')
df['energy_efficiency']=df['energy_efficiency'].str.replace(',','.').astype(float)
```

In [17]:
```
df['energy_efficiency']
```

Out[17]:
```
0       248.59
1         3.51
2       183.89
3       138.80
4         3.51
          ...
2125      3.51
2126    174.00
2127    175.00
2128    193.44
2129    284.99
Name: energy_efficiency, Length: 2112, dtype: float64
```

In [18]:
```
df['floor']
```

Out[18]:
```
0         4
1         3
2         4
3         5
4         4
        ...
2125     10
2126      R
2127      T
2128      1
2129    NaN
Name: floor, Length: 2112, dtype: object
```

In [20]:
```
"""
The dataframe has some codes for Basement(seminterrato) ,
Grand floor (piano terra) and low-ceilinged intermediate floor(Mezzanine)
For the porpuse of analysis of this column it is needed to change this codes to numeric values.
"""
floor_numeric_codes = {
    'S': -1,  # Basement
    'S-S': -1,
    'S-T' :-1,

    'T': 0,   # Ground floor
    'T-R': 0
    'R': 0,

    'M': 0.5  # Mezzanine
}
```

In [21]:
```
df['floor']=pd.to_numeric(
    df['floor'].map(floor_numeric_codes).fillna(df['floor']),
    errors = 'coerce')
```

```
In [22]: df['floor']
```

```
Out[22]: 0        4.0
         1        3.0
         2        4.0
         3        5.0
         4        4.0
                 ...
         2125    10.0
         2126     0.0
         2127     0.0
         2128     1.0
         2129     NaN
         Name: floor, Length: 2112, dtype: float64
```

```
In [23]: df['condition']
```

```
Out[23]: 0       very good / refurbished
         1       very good / refurbished
         2              good / habitable
         3       very good / refurbished
         4              good / habitable
                        ...
         2125    very good / refurbished
         2126    very good / refurbished
         2127    very good / refurbished
         2128    very good / refurbished
         2129         to be restructured
         Name: condition, Length: 2112, dtype: object
```

```
In [24]: """
         To enhance usability, the condition column requires encoding for improved readability. The following mappings are applied:

         'new / under construction' is coded as 'New'
         'very good / refurbished' is coded as 'G+'
         'good / habitable' is coded as 'G'
         'to be restructured' is coded as 'Rest'
         """


         condition_categories = ['new / under construction', 'very good / refurbished', 'good / habitable', 'to be restructured']
         condition_codes = ['NEW', 'G+', 'G', 'Rest']
         ordinal_dtype = pd.CategoricalDtype(categories = condition_categories, ordered = True)
```

```
In [25]: df['condition'] = pd.Categorical(df['condition'],categories = condition_categories,
                                          ordered = True).rename_categories(condition_codes)
```

```
In [26]: df['condition']
```

```
Out[26]: 0         G+
         1         G+
         2          G
         3         G+
         4          G
                  ...
         2125      G+
         2126      G+
         2127      G+
         2128      G+
         2129    Rest
         Name: condition, Length: 2112, dtype: category
         Categories (4, object): ['NEW' < 'G+' < 'G' < 'Rest']
```

The columns related to features of the houses will be converted to boolean type using a specific method. In this method, empty rows will be assigned a value of False, indicating the absence or non-existence of the corresponding feature. On the other hand, rows with values will be assigned a value of True, indicating the presence or existence of the specific feature.

By converting these feature columns to boolean type, we can effectively represent the presence or absence of certain attributes in a concise and standardized format. This conversion simplifies the data representation and facilitates further analysis or modeling tasks that rely on boolean variables.

Additionally, by converting the feature columns to boolean type, it becomes easier to perform operations and comparisons on these variables. For example, filtering or selecting houses based on specific features becomes straightforward using boolean operations.

Overall, this conversion enhances the clarity, usability, and interpretability of the dataset, allowing for more effective analysis and modeling of the house features.

```
In [ ]: """ The Columns related to features of the houses will convert to boolean type.
        in this method empty rows become False and Value columns become True."""
```

```
In [29]: df['air_conditioning']
```

```
Out[29]: 0       autonomous, cold/heat
         1          system preparation
         2       autonomous, cold/heat
         3            autonomous, cold
         4       autonomous, cold/heat
                        ...
         2125    autonomous, cold/heat
         2126         autonomous, cold
         2127    autonomous, cold/heat
         2128       system preparation
         2129       system preparation
         Name: air_conditioning, Length: 2112, dtype: object
```

```
In [30]: df['air_conditioning'] = df['air_conditioning'].fillna(False).astype(bool)
```

```
In [31]: df['air_conditioning']
```

```
Out[31]: 0       True
         1       True
         2       True
         3       True
         4       True
                 ...
         2125    True
         2126    True
         2127    True
         2128    True
         2129    True
         Name: air_conditioning, Length: 2112, dtype: bool
```

```
In [ ]:
```

```
In [32]: df['availability']
```

```
Out[32]: 0       libero
         1       libero
         2       libero
         3       libero
         4       libero
                  ...
         2125    libero
         2126    libero
         2127    libero
         2128    libero
         2129    libero
         Name: availability, Length: 2112, dtype: object
```

```
In [33]: df['availability']= df['availability'].fillna(False).astype(bool)
```

In [34]:
```
df['availability']
```

Out[34]:
```
0       True
1       True
2       True
3       True
4       True
        ...
2125    True
2126    True
2127    True
2128    True
2129    True
Name: availability, Length: 2112, dtype: bool
```

In [35]:
```
df['car_parking']
```

Out[35]:
```
0                  NaN
1                  NaN
2       1 in garage/box
3                  NaN
4       1 in garage/box
             ...
2125               NaN
2126               NaN
2127               NaN
2128               NaN
2129    1 in garage/box
Name: car_parking, Length: 2112, dtype: object
```

In [36]:
```
df['car_parking']= df['car_parking'].fillna(False).astype(bool)
```

In [37]:
```
df['car_parking']
```

Out[37]:
```
0       False
1       False
2        True
3       False
4        True
        ...
2125    False
2126    False
2127    False
2128    False
2129     True
Name: car_parking, Length: 2112, dtype: bool
```

In [38]:
```
df['elevator']
```

Out[38]:
```
0       1
1       1
2       1
3       1
4       1
       ..
2125    1
2126    1
2127    0
2128    1
2129    1
Name: elevator, Length: 2112, dtype: int64
```

In [39]:
```
df['elevator'] = df['elevator'].fillna(False).astype(bool)
```

In [40]:
```
df['elevator']
```

Out[40]:
```
0        True
1        True
2        True
3        True
4        True
        ...
2125     True
2126     True
2127    False
2128     True
2129     True
Name: elevator, Length: 2112, dtype: bool
```

In [43]:
```
df['heating_centralized']= df['heating_centralized'].fillna(False).astype(bool)
df['heating_radiator']= df['heating_radiator'].fillna(False).astype(bool)
df['heating_gas']= df['heating_gas'].fillna(False).astype(bool)
df['air_conditiong_centralized']= df['air_conditiong_centralized'].fillna(False).astype(bool)
df['air_conditioning_heat']= df['air_conditioning_heat'].fillna(False).astype(bool)
df['floor_level']= df['floor_level'].fillna(False).astype(bool)
df['energy_certification']= df['energy_certification'].fillna(False).astype(bool)
```

To streamline the DataFrame and improve clarity, certain columns will be removed due to having a limited number of rows. These columns and their corresponding row counts are as follows:

- 'co2_emissions': 1 row
- 'current_building_use': 9 rows
- 'renewable_energy_performance_index_KWh/m2': 71 rows

These columns will be dropped from the DataFrame as they contain insufficient data to provide meaningful insights or analysis. Removing these columns will simplify the data representation and eliminate potential confusion or misinterpretation caused by the limited number of available values.

By removing these columns, the DataFrame will become more focused and concise, allowing for easier exploration and analysis of the remaining relevant features. This data refinement process ensures that only the most informative and reliable columns are retained, resulting in a more accurate and comprehensive representation of the dataset.

In [45]:
```
"""
To simplify the DataFrame and enhance clarity, certain columns will be removed due to having a limited number of rows.
The affected columns and their respective row counts are as follows:

'co2_emissions': 1 row
'current_building_use': 9 rows
'renewable_energy_performance_index_KWh/m2': 71 rows
These columns will be dropped from the DataFrame to streamline the data representation.
"""

columns_to_drop = ['co2_emissions', 'current_building_use', 'renewable_energy_performance_index_KWh/m2']
df = df.drop(columns_to_drop, axis=1)
```

```
In [46]: df.dropna(subset=['price'], inplace=True)

In [47]: df.dropna(subset=['m2'], inplace=True)

In [48]: df.dropna(subset=['other_features'], inplace=True)

In [49]: mode_value = df['rooms'].mode()[0]
         df['rooms'].fillna(mode_value, inplace=True)

In [50]: mode_value = df['bathrooms'].mode()[0]
         df['bathrooms'].fillna(mode_value, inplace=True)

In [51]: mode_value = df['condition'].mode()[0]
         df['condition'].fillna(mode_value, inplace=True)

In [52]: mode_value = df['total_floors'].mode()[0]
         df['total_floors'].fillna(mode_value, inplace=True)

In [53]: mode_value = df['energy_efficiency'].mean()
         df['energy_efficiency'].fillna(mode_value, inplace=True)

In [54]: mode_value = df['condominium_expenses'].mean()
         df['condominium_expenses'].fillna(mode_value, inplace=True)

In [55]: mode_value = df['year_of_build'].mean()
         df['year_of_build'].fillna(mode_value, inplace=True)

In [56]: mode_value = df['floor'].mode()[0]
         df['floor'].fillna(mode_value, inplace=True)

In [57]: mode_value = df['energy_performance_building'].mode()[0]
         df['energy_performance_building'].fillna(mode_value, inplace=True)

In [58]: df.dropna(subset=['condition'], inplace=True)
```

In the subsequent step of the data cleaning process, the null values within the DataFrame will be addressed and organized. Firstly, the null values in the 'price', 'm2', and 'other_features' columns will be eliminated by dropping the respective rows from the DataFrame. This ensures that only complete and meaningful data remains for further analysis.

For the remaining columns that contain null values, a different approach will be applied. The null values will be replaced using appropriate strategies depending on the data type of each column. Categorical columns will have their null values replaced with the mode value, which represents the most frequently occurring value in the column. On the other hand, numerical columns will have their null values replaced with the mean value, representing the average value across the column.

By employing these techniques, the data will be cleansed and prepared for subsequent analysis, ensuring a more comprehensive and reliable dataset.

cleaned data frame:

```
In [59]: check(df)
```

Out[59]:

	column	dtypes	nunique	sum_null
0	index	int64	1	0
1	rooms	float64	5	0
2	m2	float64	265	0
3	bathrooms	float64	5	0
4	floor	float64	19	0
5	condominium_expenses	float64	229	0
6	date	object	129	0
7	contract	object	6	0
8	typology	object	42	0
9	total_floors	object	25	0
10	availability	bool	2	0
11	other_features	object	1738	0
12	price	float64	565	0
13	year_of_build	float64	105	0
14	condition	category	4	0
15	air_conditioning	bool	2	0
16	energy_efficiency	float64	722	0
17	city	object	1	0
18	neighborhood	object	143	0
19	car_parking	bool	2	0
20	energy_performance_building	object	3	0
21	energy_certification	bool	2	0
22	elevator	bool	2	0
23	floor_level	bool	2	0
24	heating_centralized	bool	2	0
25	heating_radiator	bool	2	0
26	heating_gas	bool	2	0
27	air_conditiong_centralized	bool	2	0
28	air_conditioning_heat	bool	2	0

Outliers:

In the subsequent step of the data analysis process, the presence of outliers within the dataset will be addressed and examined. Outliers are data points that significantly deviate from the majority of the data, either in terms of their magnitude or their distribution.

Firstly, the outliers will be detected using appropriate statistical techniques such as the Z-score, Tukey's fences, or the interquartile range (IQR). These methods allow for the identification of data points that fall outside a certain threshold or range of acceptable values.

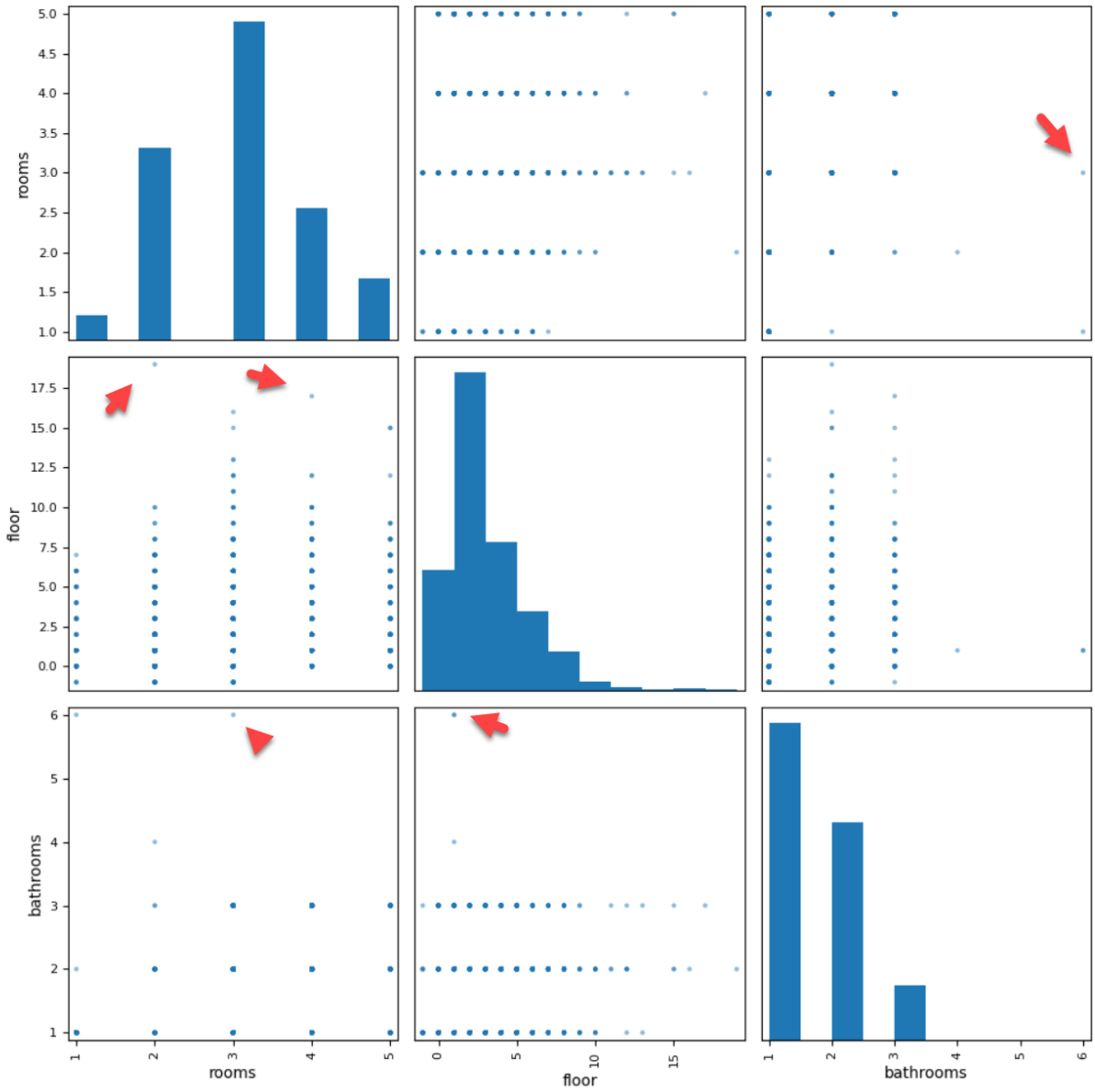
Once the outliers are identified, a thorough analysis will be conducted to determine their nature and potential causes. This analysis aims to differentiate between two possibilities: whether the outliers are indicative of errors in the data collection or recording process, or if they represent genuine and meaningful observations.

In cases where the outliers are likely to be errors, further investigation will be conducted to identify and rectify any data issues. This may involve cross-referencing with external sources, consulting domain experts, or examining the data collection methods.

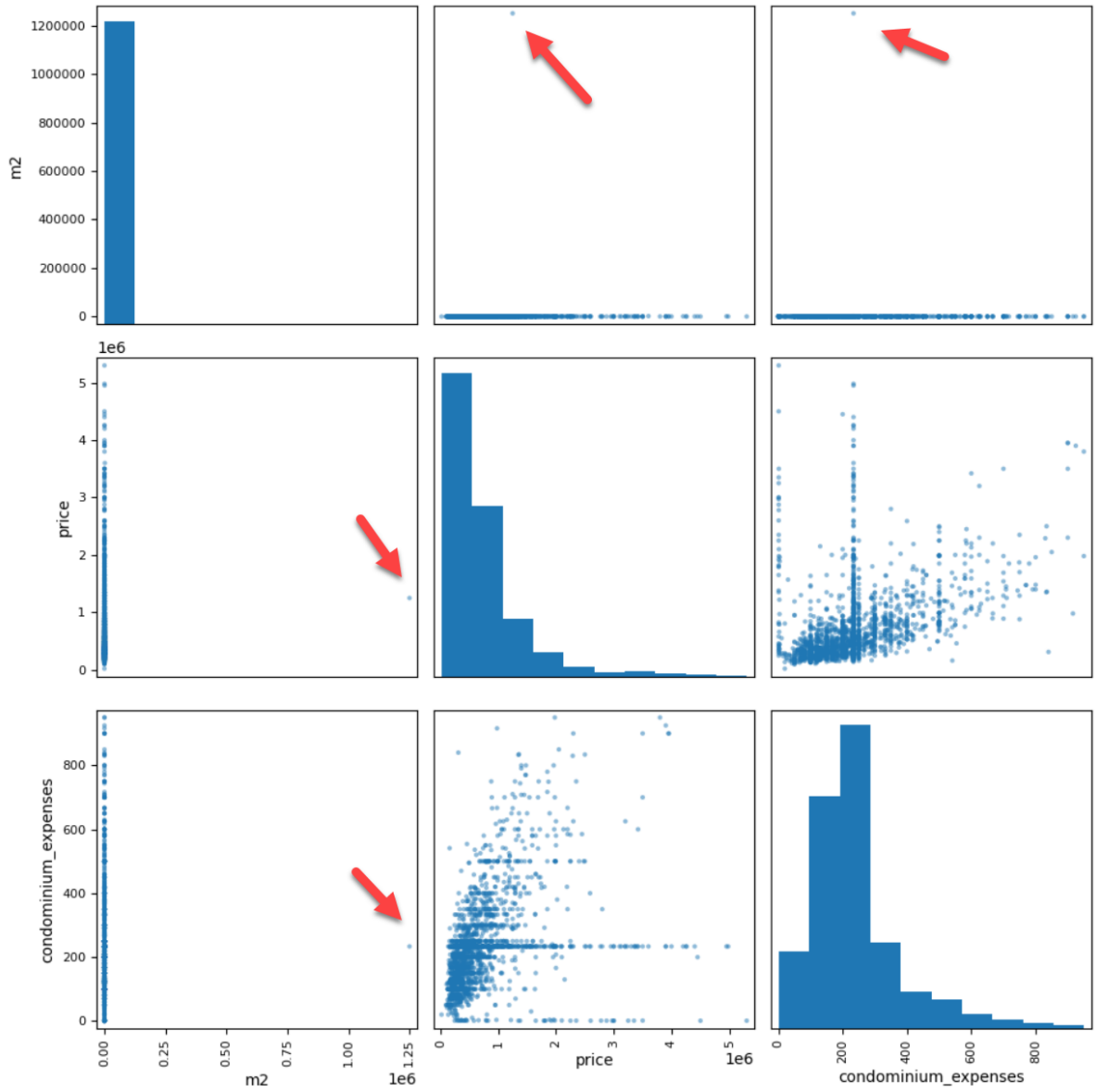
Alternatively, if the outliers are found to be understandable and valid observations, they will be retained in the dataset. These outliers may hold valuable information, such as extreme values that reflect rare events, exceptional cases, or significant variations in the underlying phenomenon being studied.

By detecting and analyzing outliers in this manner, the data integrity and reliability of the dataset will be enhanced, ensuring that subsequent analysis and modeling efforts are based on accurate and meaningful infomation.

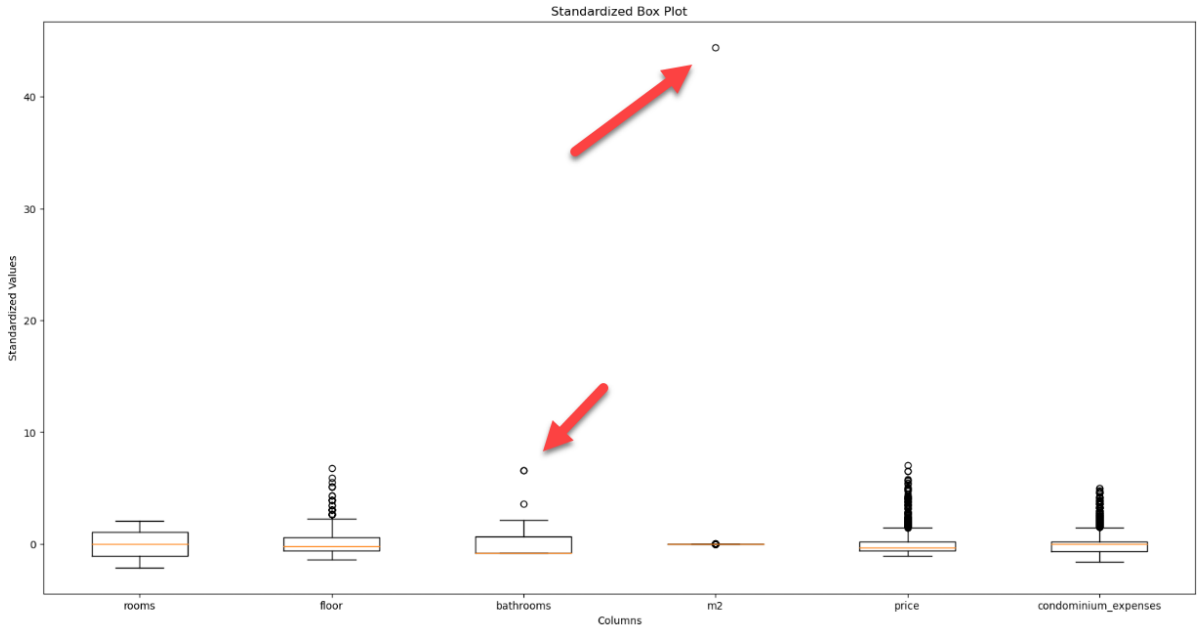
In [65]:
```
#To analyze the outliers from the dataframe the scatter matrix will be used
columns_to_plot = ['rooms','floor', 'bathrooms']

pd.plotting.scatter_matrix(df[columns_to_plot], figsize=(10, 10))

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



In [69]:
```
#To analyze the outliers from the dataframe the scatter matrix will be used
columns_to_plot = ['m2','price', 'condominium_expenses']

pd.plotting.scatter_matrix(df[columns_to_plot], figsize=(10, 10))

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



In [98]:
```
columns_to_plot = ['rooms','floor', 'bathrooms','m2','price', 'condominium_expenses']

# Standardizing the selected column(s) using z-scores
df_standardized = (df[columns_to_plot] - df[columns_to_plot].mean()) / df[columns_to_plot].std()

plt.figure(figsize=(20, 10))


plt.boxplot(df_standardized.values)


plt.title('Standardized Box Plot')
plt.xlabel('Columns')
plt.ylabel('Standardized Values')


plt.xticks(range(1, len(columns_to_plot) + 1), columns_to_plot)

plt.show()
```



In [71]:
```
df.describe()
```

Out[71]:

	index	rooms	m2	bathrooms	floor	condominium_expenses	price	year_of_build	energy_efficiency
count	1976.0	1976.000000	1.976000e+03	1976.000000	1976.000000	1976.000000	1.976000e+03	1976.000000	1976.000000
mean	0.0	3.018725	7.494747e+02	1.562753	2.494939	233.520433	7.270203e+05	1961.722252	149.786935
std	0.0	0.958455	2.811757e+04	0.675884	2.444726	144.857352	6.506547e+05	49.964902	84.757945
min	0.0	1.000000	1.000000e+00	1.000000	-1.000000	1.000000	2.000000e+04	1100.000000	1.000000
25%	0.0	2.000000	7.075000e+01	1.000000	1.000000	140.000000	3.290000e+05	1940.000000	123.505000
50%	0.0	3.000000	1.000000e+02	1.000000	2.000000	230.000000	5.200000e+05	1960.000000	175.000000
75%	0.0	4.000000	1.400000e+02	2.000000	4.000000	260.000000	8.615000e+05	1974.000000	175.000000
max	0.0	5.000000	1.250000e+06	6.000000	19.000000	950.000000	5.300000e+06	2025.000000	1167.100000

To standardize the given Data-frame, the z-scores will be calculated for each column. This involves measuring how many standard deviations each individual data point deviates from the mean of its corresponding column. By applying standardization, the data will be transformed into a standardized representation, facilitating meaningful comparisons and the identification of outliers across different columns. The z-scores provide a common scale, allowing for a more accurate assessment of the relative

position of each data point within its respective column. This process is valuable in detecting extreme values that significantly differ from the average behavior of the dataset.

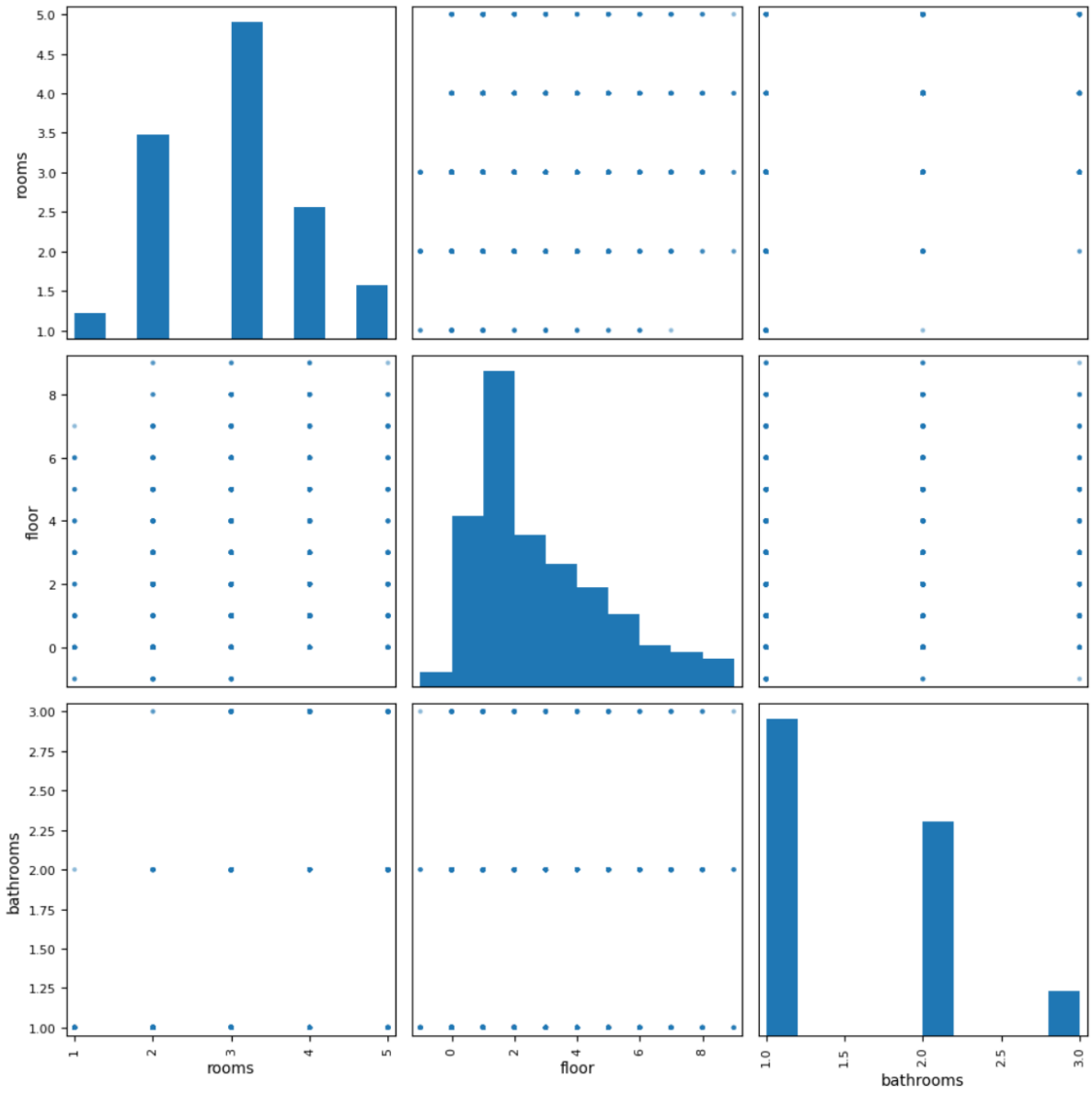
In [72]:
```
"""
Compute the z-scores for each column in the given DataFrame by utilizing the concept of standardization.
Z-scores provide a measure of how many standard deviations an individual data point is away from the mean of its respective colum
This process allows for a standardized representation of the data,
enabling effective comparison and identification of outliers across different columns.
"""
numeric_columns = df.select_dtypes(include=['int', 'float'])
z_scores = np.abs((numeric_columns - numeric_columns.mean()) / numeric_columns.std())

threshold = 3
outlier_mask = z_scores > threshold

df_cl = df[~outlier_mask.any(axis=1)]

df_cl
```

In [74]:
```
df_cl.describe()
```

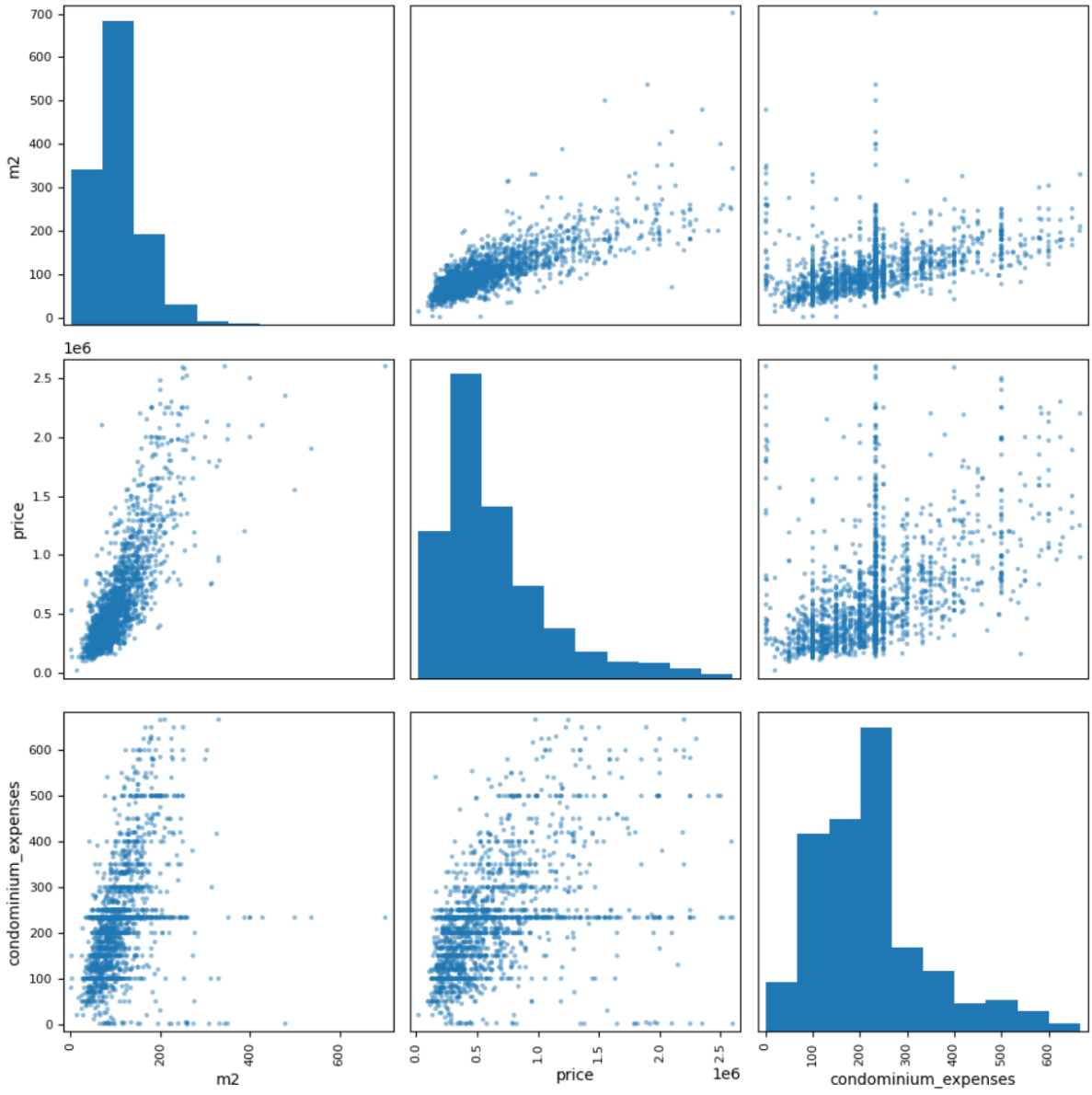
Out[74]:

	index	rooms	m2	bathrooms	floor	condominium_expenses	price	year_of_build	energy_efficiency
count	1835.0	1835.000000	1835.000000	1835.000000	1835.000000	1835.000000	1.835000e+03	1835.000000	1835.000000
mean	0.0	2.977657	107.965123	1.525886	2.338965	223.624250	6.339990e+05	1964.360492	146.856449
std	0.0	0.941612	56.420729	0.635002	2.141805	122.146774	4.494217e+05	33.998288	72.769487
min	0.0	1.000000	1.000000	1.000000	-1.000000	1.000000	2.000000e+04	1850.000000	1.000000
25%	0.0	2.000000	70.000000	1.000000	1.000000	135.000000	3.200000e+05	1950.000000	123.885000
50%	0.0	3.000000	96.000000	1.000000	2.000000	220.000000	4.950000e+05	1960.000000	175.000000
75%	0.0	4.000000	130.000000	2.000000	4.000000	250.000000	7.980000e+05	1971.500000	175.000000
max	0.0	5.000000	702.000000	3.000000	9.000000	667.000000	2.600000e+06	2025.000000	400.110000

In [75]:
```
columns_to_plot = ['rooms','floor', 'bathrooms']

pd.plotting.scatter_matrix(df_cl[columns_to_plot], figsize=(10, 10))

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



In [76]:
```
columns_to_plot = ['m2','price', 'condominium_expenses']

pd.plotting.scatter_matrix(df_cl[columns_to_plot], figsize=(10, 10))

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



In the subsequent step, we will perform data transformation to create a new DataFrame named `df_ncl` by filtering out non-numeric values from the existing DataFrame `df_cl`. This process will involve leveraging the IQR (Interquartile Range) method to identify and handle outliers within the dataset.

In [81]:
```
df_ncl = df_cl.select_dtypes(include='number')
df_ncl
```

Out[81]:

	index	rooms	m2	bathrooms	floor	condominium_expenses	price	year_of_build	energy_efficiency
0	0	3.0	140.0	2.0	4.0	535.000000	880000.0	1900.000000	248.59
1	0	4.0	121.0	2.0	3.0	200.000000	649000.0	1960.000000	3.51
2	0	2.0	55.0	1.0	4.0	133.000000	300000.0	2006.000000	183.89
3	0	2.0	60.0	1.0	5.0	333.000000	250000.0	1960.000000	138.80
4	0	4.0	220.0	3.0	4.0	233.520433	2250000.0	1970.000000	3.51
...	...	...	...	...	...	...	...	...	...
2124	0	2.0	51.0	1.0	2.0	100.000000	375000.0	1960.000000	3.51
2126	0	3.0	90.0	2.0	0.0	183.000000	280000.0	1960.000000	174.00
2127	0	3.0	130.0	2.0	0.0	200.000000	625000.0	1940.000000	175.00
2128	0	2.0	51.0	1.0	1.0	233.520433	440000.0	1961.722252	193.44
2129	0	2.0	72.0	1.0	1.0	250.000000	355000.0	1969.000000	284.99

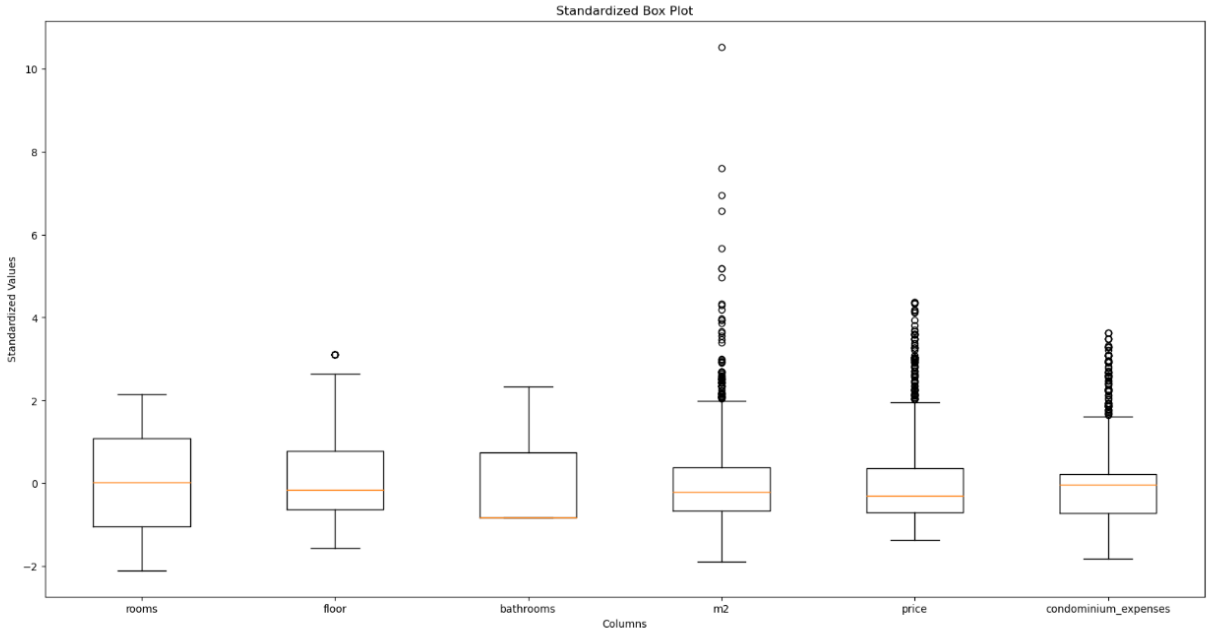
1835 rows × 9 columns

In [82]:
```
def detect_outliers_iqr(df_ncl, multiplier=1.5):
    """
    Detects outliers in a DataFrame using the IQR method.
    Returns a boolean mask indicating the rows containing outliers.
    """
    Q1 = df_ncl.quantile(0.25)
    Q3 = df_ncl.quantile(0.75)
    IQR = Q3 - Q1

    outlier_mask = np.logical_xor(df_ncl < Q1 - multiplier * IQR, df_ncl > Q3 + multiplier * IQR)

    return outlier_mask

# Assuming you have a DataFrame called 'df'
outlier_mask = detect_outliers_iqr(df_ncl)

# Print rows containing outliers
outliers = df_ncl[outlier_mask.any(axis=1)]
print(outliers)
```

```
      index  rooms     m2  bathrooms  floor  condominium_expenses      price  \
0         0    3.0  140.0        2.0    4.0           535.000000   880000.0
1         0    4.0  121.0        2.0    3.0           200.000000   649000.0
2         0    2.0   55.0        1.0    4.0           133.000000   300000.0
4         0    4.0  220.0        3.0    4.0           233.520433  2250000.0
6         0    4.0  150.0        2.0    0.0           300.000000   950000.0
...     ...    ...    ...        ...    ...                  ...        ...
2119      0    2.0   89.0        1.0    0.0           350.000000   519000.0
2121      0    3.0   95.0        2.0    0.0           270.000000   545000.0
2123      0    2.0   59.0        1.0    2.0           110.000000   398000.0
2124      0    2.0   51.0        1.0    2.0           100.000000   375000.0
2129      0    2.0   72.0        1.0    1.0           250.000000   355000.0

      year_of_build  energy_efficiency
0            1900.0             248.59
1            1960.0               3.51
2            2006.0             183.89
4            1970.0               3.51
6            1960.0             272.10
...             ...                ...
2119         1900.0             183.79
2121         1920.0               3.51
2123         1960.0               3.51
2124         1960.0               3.51
2129         1969.0             284.99

[840 rows x 9 columns]
```

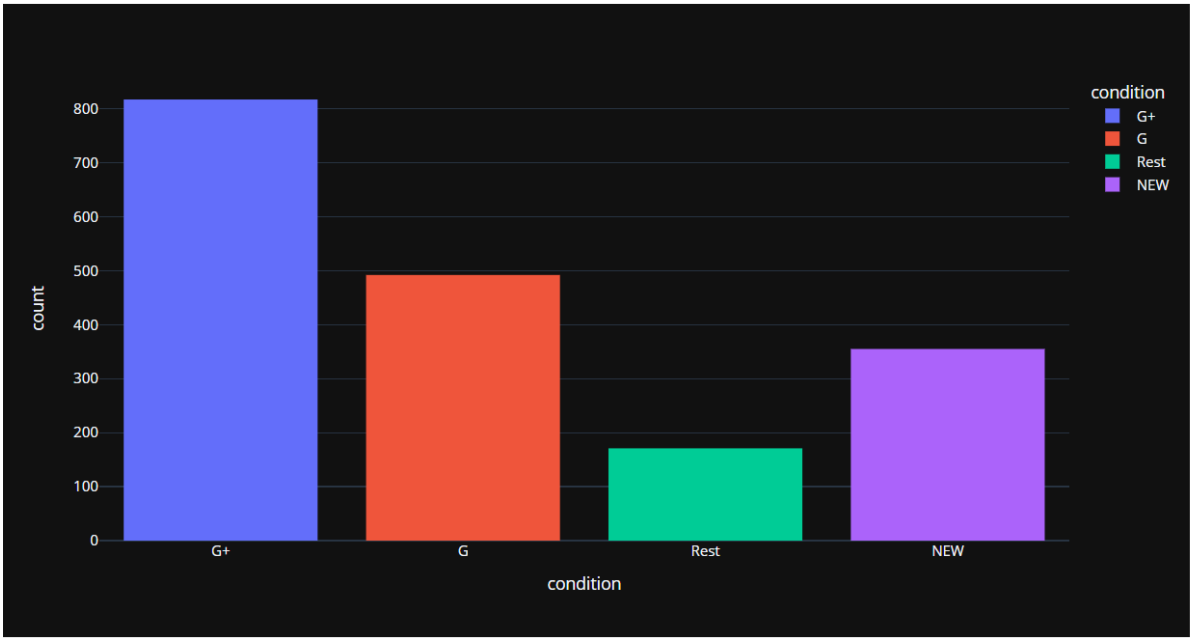
In the subsequent step, we will generate standardized box plots for each column in order to assess the distribution and identify any remaining outliers. This will provide us with a visual representation of the data range within each column, allowing us to detect and visualize any data points that lie significantly outside the typical range. By standardizing the box plots, we can compare the distribution of different columns on a consistent scale, facilitating a comprehensive analysis of the remaining outliers present in the dataset.

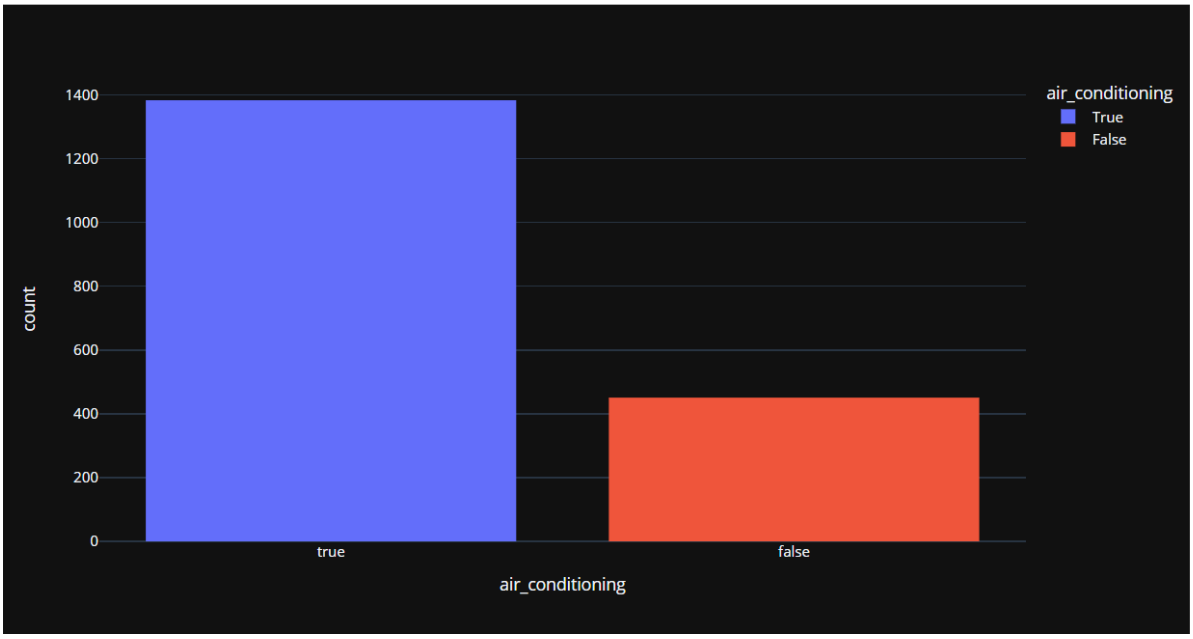
In [97]:
```
columns_to_plot = ['rooms','floor', 'bathrooms','m2','price', 'condominium_expenses']

# Standardizing the selected column(s) using z-scores
df_standardized = (df_ncl[columns_to_plot] - df_ncl[columns_to_plot].mean()) / df_ncl[columns_to_plot].std()

plt.figure(figsize=(20, 10))


plt.boxplot(df_standardized.values)


plt.title('Standardized Box Plot')
plt.xlabel('Columns')
plt.ylabel('Standardized Values')


plt.xticks(range(1, len(columns_to_plot) + 1), columns_to_plot)

plt.show()
```



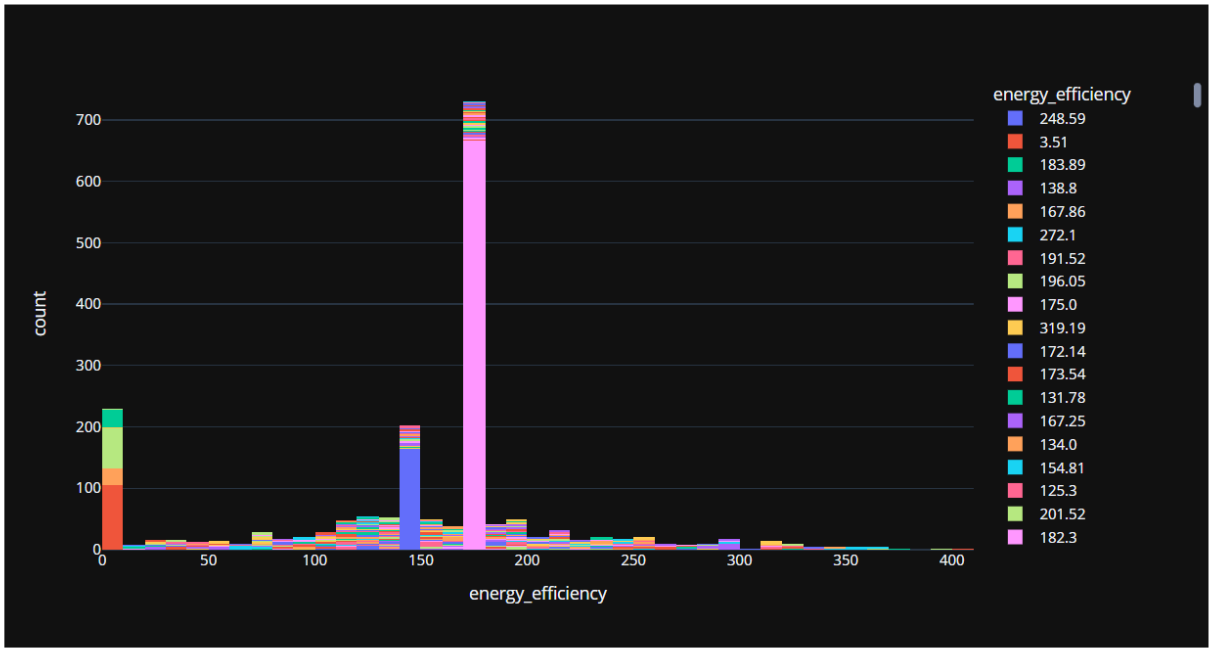
In the next step of the analysis, individual plots will be generated to examine the distribution of data in each column. These plots will provide visual representations of the data distribution, allowing for a better understanding of the patterns, trends, and outliers within each column. By creating specific plots for each column, we can gain insights into the data's distributional characteristics and make informed observations about the underlying data patterns.

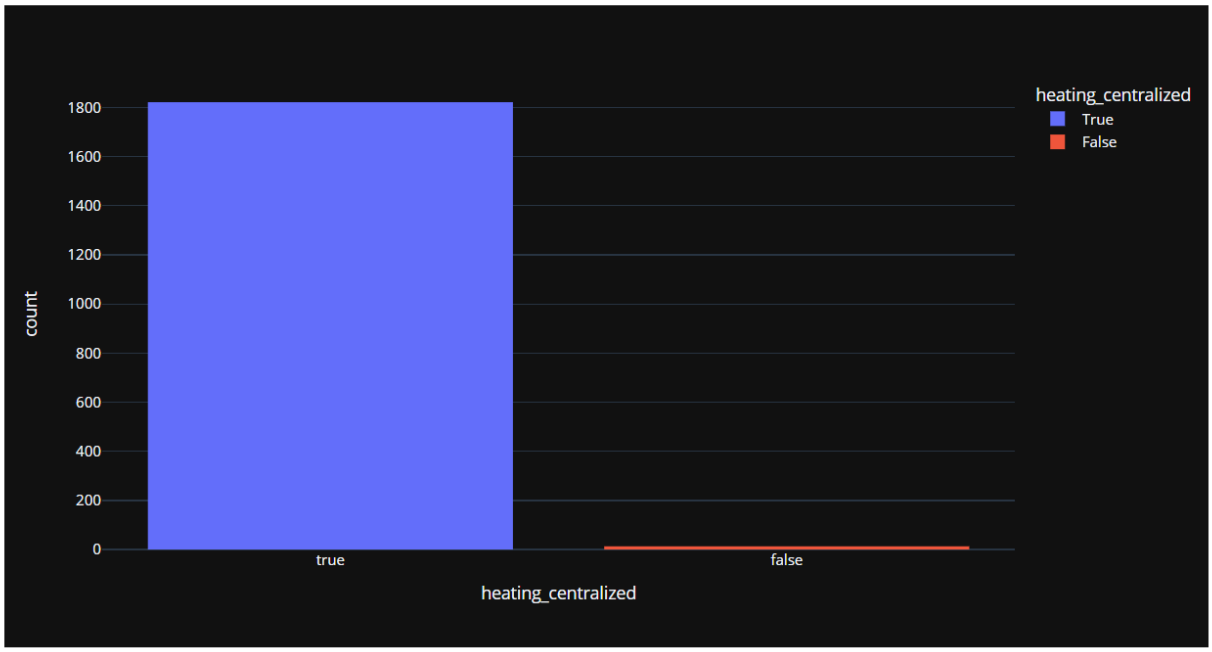
In [111]:
```
fig1 = px.histogram(df_cl,x='condition',color='condition',template='plotly_dark')
fig1.show()
fig2 = px.histogram(df_cl,x='air_conditioning',color='air_conditioning',template='plotly_dark')
fig2.show()
```





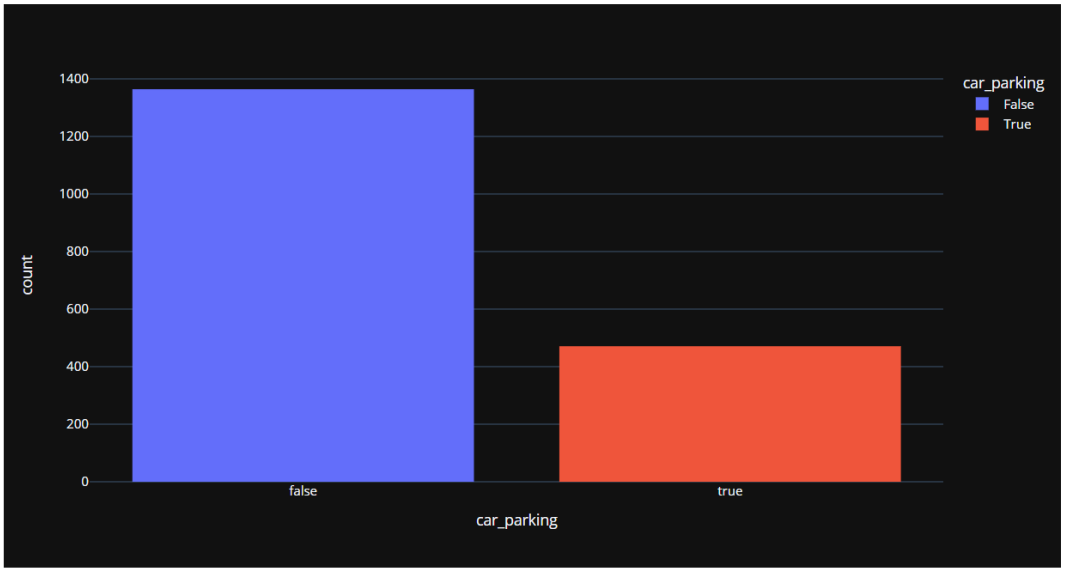
In [112]:
```
fig3 = px.histogram(df_cl,x='energy_efficiency',color='energy_efficiency',template='plotly_dark')
fig3.show()
fig4 = px.histogram(df_cl,x='heating_centralized',color='heating_centralized',template='plotly_dark')
fig4.show()
```

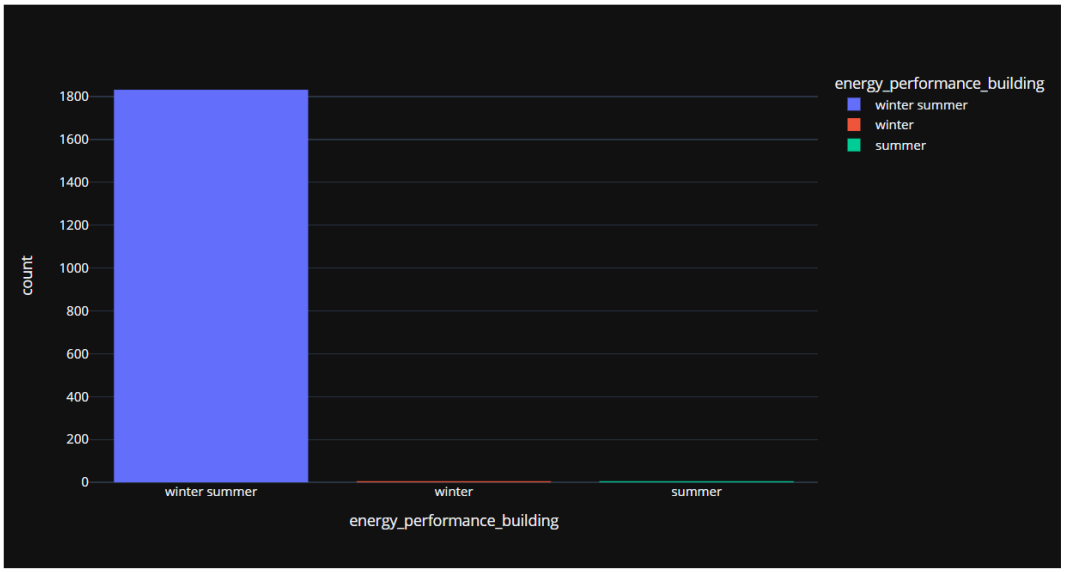


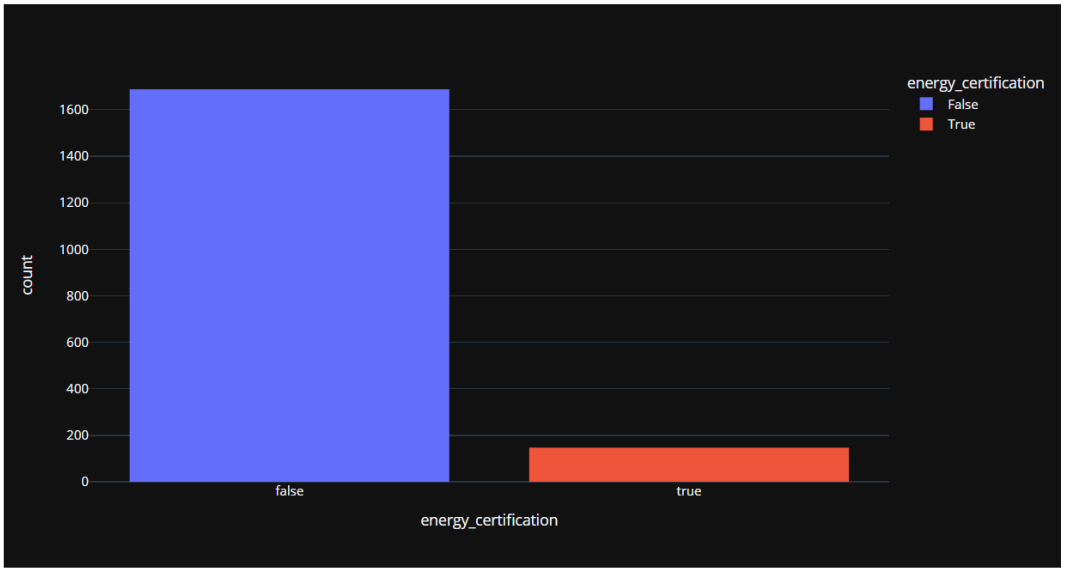


In [113]:
```
fig5 = px.histogram(df_cl,x='car_parking',color='car_parking',template='plotly_dark')
fig5.show()
```

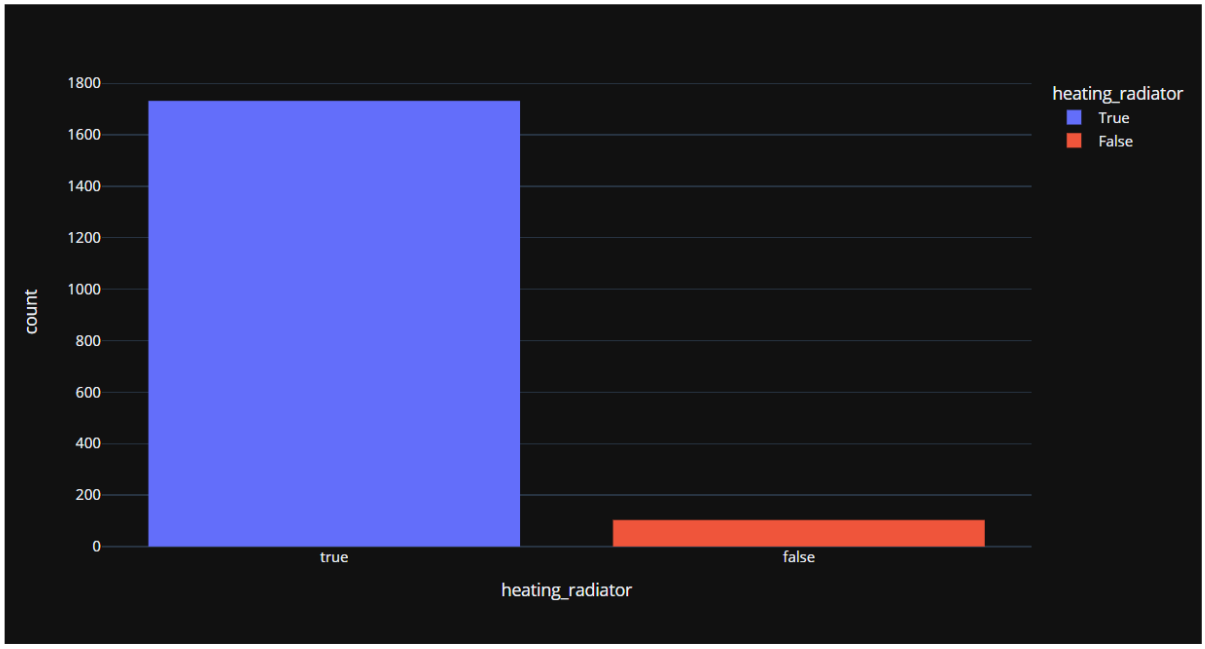
In [114]:
```
fig6 = px.histogram(df_cl,x='energy_performance_building',color='energy_performance_building',template='plotly_dark')
fig6.show()
fig7 = px.histogram(df_cl,x='energy_certification',color='energy_certification',template='plotly_dark')
fig7.show()
```

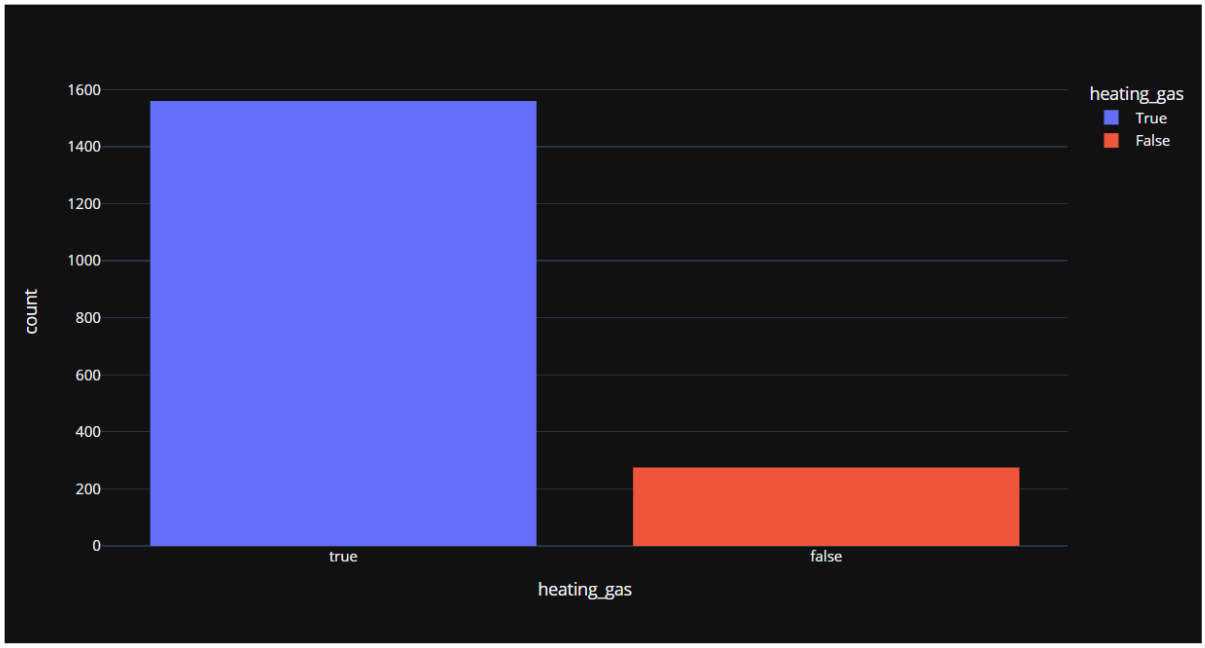




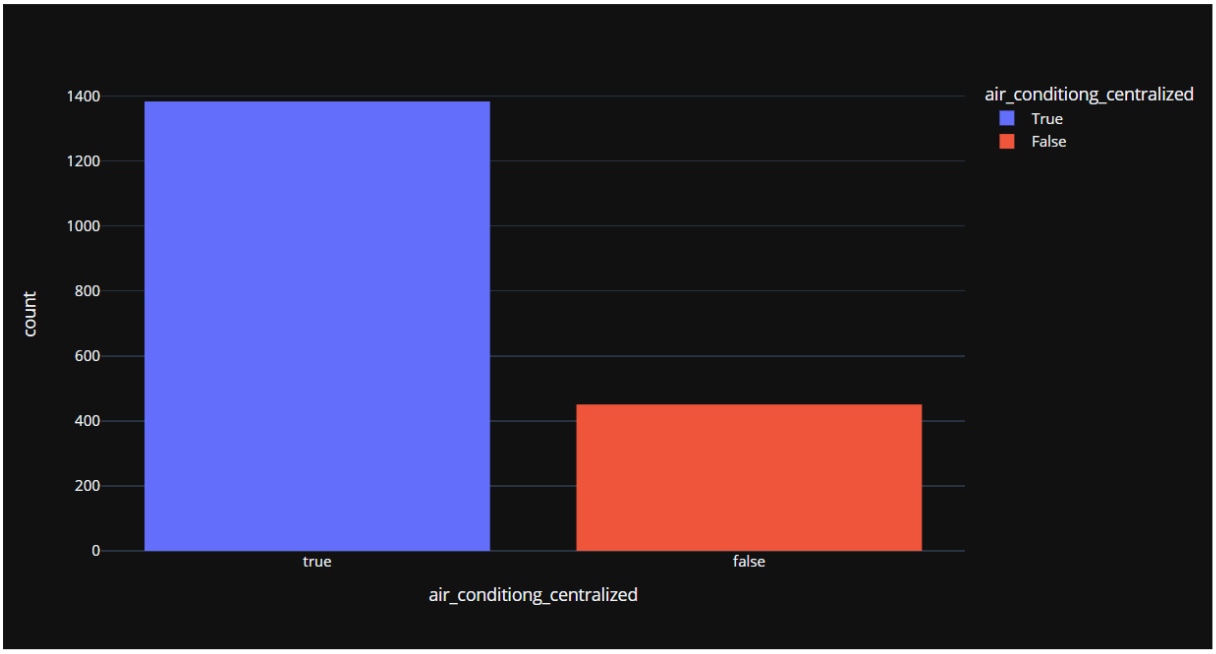


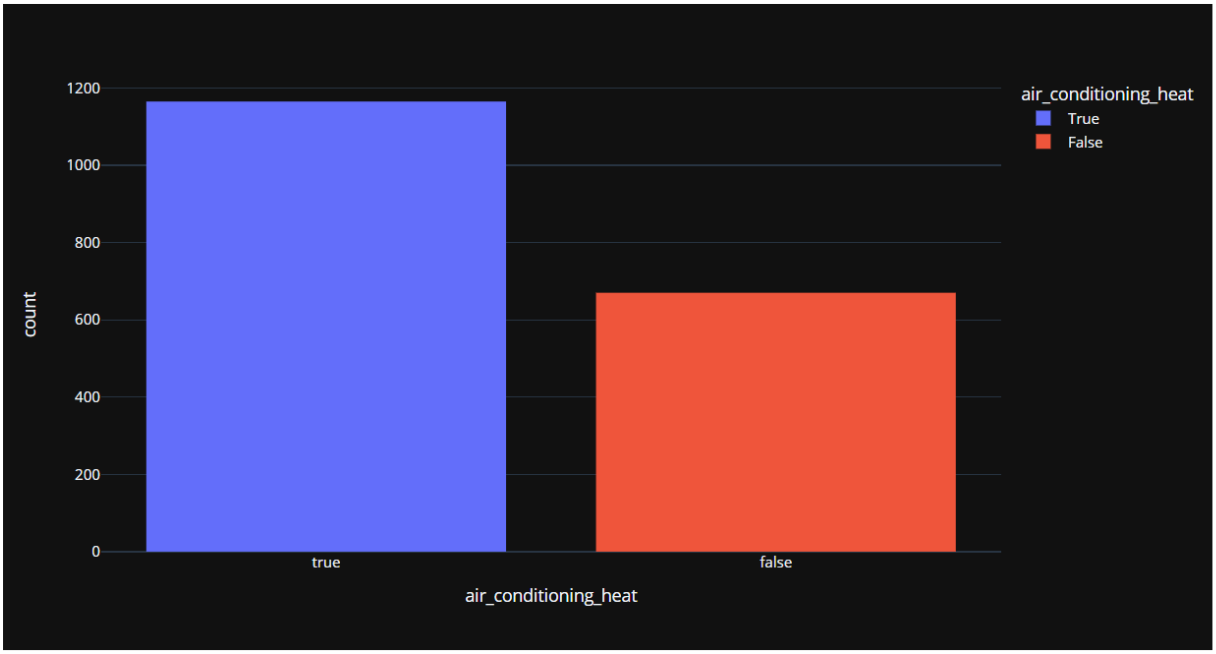
In [116]:
```
fig5 = px.histogram(df_cl,x='heating_radiator',color='heating_radiator',template='plotly_dark')
fig5.show()
fig6 = px.histogram(df_cl,x='heating_gas',color='heating_gas',template='plotly_dark')
fig6.show()
```



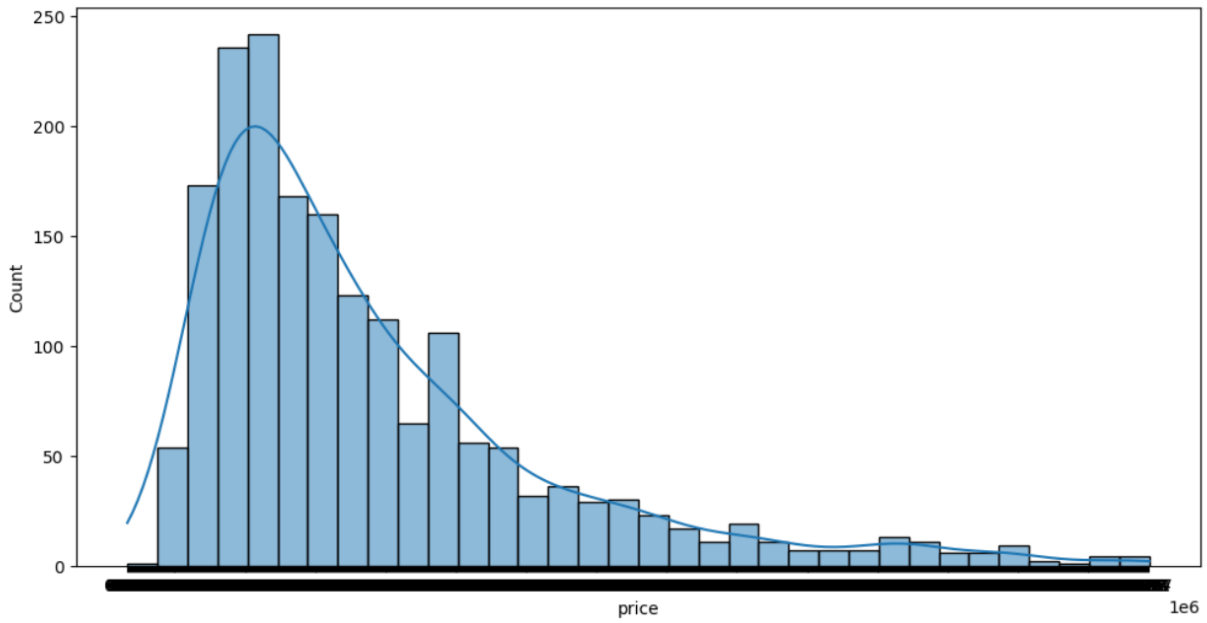


In [117]:
```
fig7 = px.histogram(df_cl,x='air_conditiong_centralized',color='air_conditiong_centralized',template='plotly_dark')
fig7.show()
fig8 = px.histogram(df_cl,x='air_conditioning_heat',color='air_conditioning_heat',template='plotly_dark')
fig8.show()
```

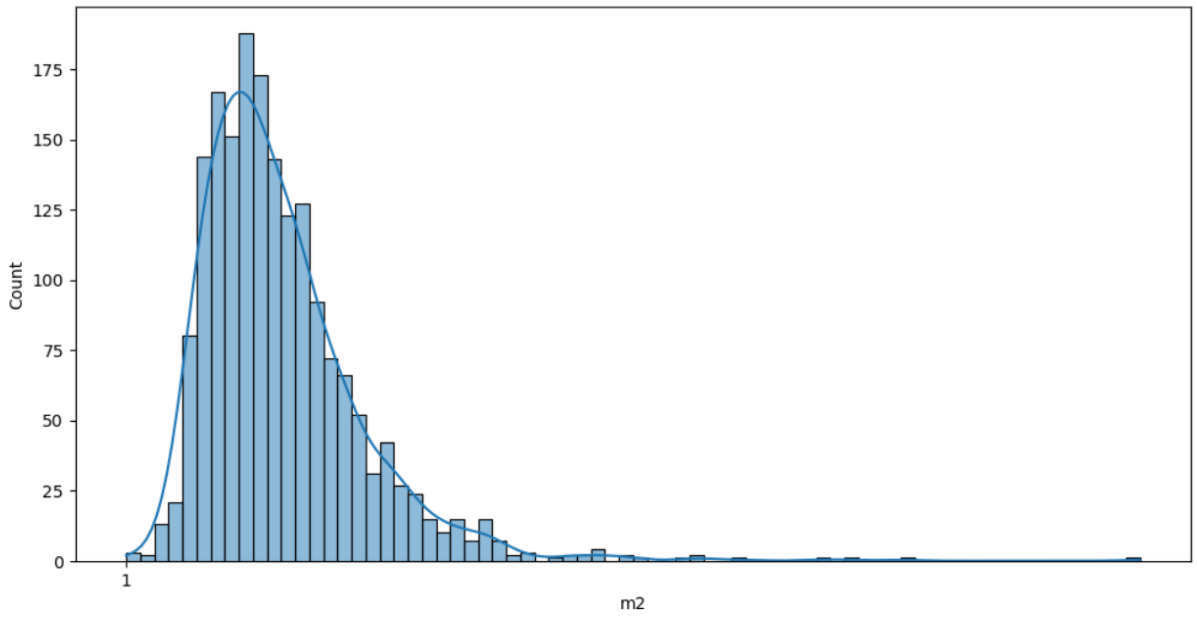




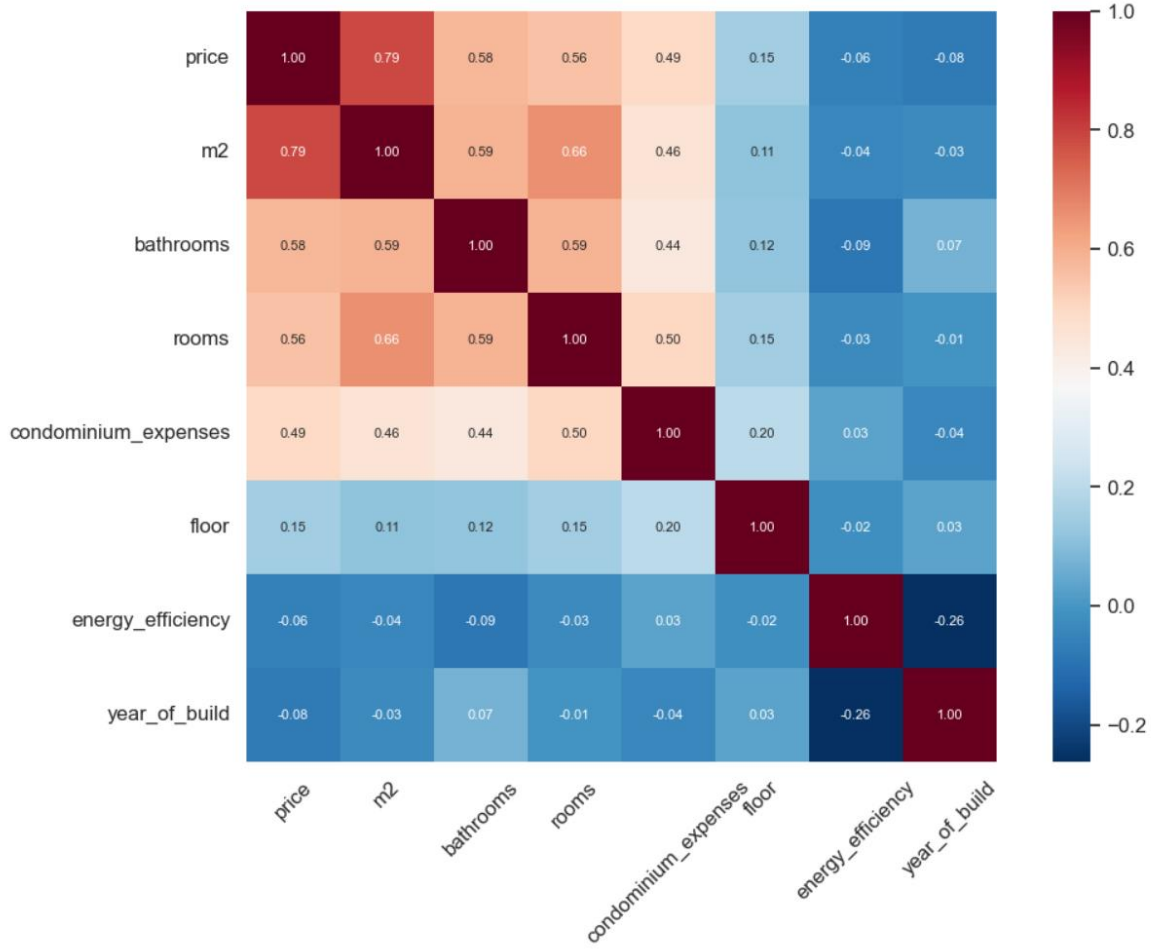
In [119]:
```
plt.figure(figsize=(12, 6))
sns.histplot(df_cl['price'], kde=True)
plt.xticks(np.arange(df_ncl['price'].min(), df_cl['price'].max(), step=3000))
plt.show()
```



In [121]:
```
plt.figure(figsize=(12, 6))
sns.histplot(df_cl['m2'], kde=True)
plt.xticks(np.arange(df_cl['m2'].min(), df_cl['m2'].max(), step=3000))
plt.show()
```



In [145]:
```
corr = df_ncl.corr()
plt.figure(figsize=(12, 8))
k = 9  # Number of variables for the heatmap
cols = corr.nlargest(k, 'price')['price'].index
cm = np.corrcoef(df_ncl[cols].values.T)
sns.set(font_scale=1.1)
hm = sns.heatmap(
    cm,
    cbar=True,
    annot=True,
    square=True,
    fmt='.2f',
    annot_kws={'size': 8},
    yticklabels=cols.values,
    xticklabels=cols.values,
    cmap='RdBu_r'  # Diverging color map
)
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.show()
```



Chapter 3:

Model Selection

In the subsequent stage of analysis, various models are employed to effectively capture the patterns and relationships within the data. These models serve as mathematical representations or algorithms that can learn from the provided dataset and make predictions or inferences based on the input variables. By utilizing different models, we aim to explore the data from multiple perspectives, compare their performance, and select the most suitable model(s) for the given task. This approach allows us to leverage the strengths of different modeling techniques and gain deeper insights into the underlying patterns and dynamics of the data. Additionally, deploying multiple models provides a means of validating and cross-checking the results, enhancing the reliability and robustness of the analysis.

Linear Regression:

Initially, we employ a linear regression model to analyze the data and make predictions specifically for the price column. Linear regression is a statistical modeling technique that aims to establish a linear relationship between the independent variables (features) and the dependent variable (price) by fitting a straight line to the data points. By estimating the coefficients of the linear equation, the model can make predictions for the price based on the given set of features.

The linear regression model assumes that there is a linear relationship between the features and the target variable, and it attempts to minimize the difference between the predicted values and the actual values of the price. This allows us to quantify the impact of each feature on the price and identify the most influential factors in determining the housing prices.

By modeling the data using linear regression, we can gain insights into the direction and magnitude of the relationships between the features and the price. Additionally, we can evaluate the performance of the model by assessing metrics such as the coefficient of determination (R-squared), root mean squared error (RMSE), or mean absolute error (MAE). These metrics help us understand how well the linear regression model fits the data and how accurately it predicts the prices.

By utilizing linear regression as an initial modeling approach, we can establish a baseline understanding of the relationships within the data and evaluate the predictive power of the selected features in determining the housing prices. This serves as a foundation for further analysis and potentially allows us to explore more advanced modeling techniques to enhance the accuracy and precision of the predictions.

```
In [142]: df2 = df_ncl.copy()
          y=df2.pop('price')
          X=df2

In [143]: X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)

In [144]: regmodel=LinearRegression()
          regmodel.fit(X_train,y_train)

Out[144]: LinearRegression
          LinearRegression()

In [145]: y_pred=regmodel.predict(X_test)

In [146]: print('r2_score: ',metrics.r2_score(y_test,y_pred))
          print('mean_squared_error: ',metrics.mean_squared_error(y_test,y_pred))
          print('mean_absolute_error: ',metrics.mean_absolute_error(y_test,y_pred))

          r2_score:  0.6310769990148919
          mean_squared_error:  74118446471.89688
          mean_absolute_error:  189387.8923823736
```

The analysis of the results shows the performance of the linear regression model in predicting the housing prices. Here are the key metrics:

1. R2 Score: The R2 score, also known as the coefficient of determination, measures the proportion of the variance in the target variable (price) that can be explained by the linear regression model. In this case, the R2 score is 0.6311, indicating that approximately 63.11% of the variance in the housing prices can be attributed to the features included in the model. A higher R2 score suggests a better fit of the model to the data.

2. Mean Squared Error (MSE): The MSE quantifies the average squared difference between the predicted prices and the actual prices. In this analysis, the MSE is 741,184,464,71.89688. The MSE gives an idea of the magnitude of the errors made by the model, with higher values indicating larger errors. It is important to note that the MSE is measured in squared units of the target variable, which in this case is the housing price.

3. Mean Absolute Error (MAE): The MAE measures the average absolute difference between the predicted prices and the actual prices. In this analysis, the MAE is 189,387.8923823736. The MAE provides an indication of the average magnitude of the errors made by the model, regardless of their direction. A lower MAE suggests better accuracy of the predictions.

Based on these results, we can conclude that the linear regression model has moderate predictive power for the housing prices, as indicated by the R2 score of 0.6311. The MSE and MAE values provide insights into the magnitude of the errors made by the model, with the MSE indicating larger errors compared to the MAE. Further analysis and improvement can be done to enhance the model's performance and reduce the errors in predicting the housing prices.

Polynomial Linear Regression:

In [147]:
```
#polynomial Linear Regression
```

In [148]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

In [149]:
```
poly=PolynomialFeatures(degree=2)
X_train_quad=poly.fit_transform(X_train)
X_test_quad=poly.fit_transform(X_test)
polymodel=LinearRegression()
```

In [150]:
```
polymodel.fit(X_train_quad,y_train)
```

Out[150]:
```
LinearRegression
LinearRegression()
```

In [151]:
```
y_pred_quad=polymodel.predict(X_test_quad)
y_pred2=y_pred_quad
```

In [152]:
```
print(y_pred_quad.shape)
print(y_test.shape)
```

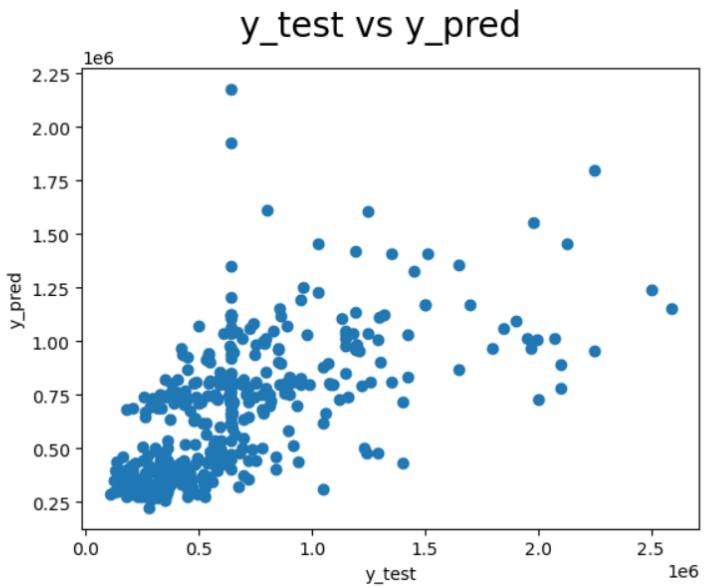
```
(367,)
(367,)
```

In [153]:
```
print('r2_score: ',metrics.r2_score(y_test,y_pred_quad))
print('explained_variance_score: ',metrics.explained_variance_score(y_test,y_pred_quad))
print('mean_squared_error: ',metrics.mean_squared_error(y_test,y_pred_quad))
print('mean_absolute_error: ',metrics.mean_absolute_error(y_test,y_pred_quad))
```

```
r2_score:  0.6835665832543322
explained_variance_score:  0.684035718593609
mean_squared_error:  62389761207.4448
mean_absolute_error:  177923.37962479383
```

In [304]:
```
fig = plt.figure()
plt.scatter(y_test,y_pred_quad)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test')                          # X-label
plt.ylabel('y_pred')
```

Out[304]:
```
Text(0, 0.5, 'y_pred')
```



The analysis of the results from the polynomial linear regression model shows its performance in predicting the housing prices. Here are the key metrics:

1. R2 Score: The R2 score, also known as the coefficient of determination, measures the proportion of the variance in the target variable (price) that can be explained by the polynomial regression model. In this case, the R2 score is 0.6836, indicating that approximately 68.36% of the variance in the housing prices can be attributed to the features included in the model. A higher R2 score suggests a better fit of the model to the data.

2. Explained Variance Score: The explained variance score measures the proportion of the variance in the target variable that can be explained by the model. In this analysis, the explained variance score is 0.6840, indicating that the model explains approximately 68.40% of the variance in the housing prices. A higher explained variance score suggests a better ability of the model to capture the underlying patterns in the data.

3. Mean Squared Error (MSE): The MSE quantifies the average squared difference between the predicted prices and the actual prices. In this analysis, the MSE is 62,389,761,207.4448. The MSE gives an idea of the magnitude of the errors made by the model, with higher values indicating larger errors. It is important to note that the MSE is measured in squared units of the target variable, which in this case is the housing price.

4. Mean Absolute Error (MAE): The MAE measures the average absolute difference between the predicted prices and the actual prices. In this analysis, the MAE is 177,923.37962479383. The MAE provides an indication of the average magnitude of the errors made by the model, regardless of their direction. A lower MAE suggests better accuracy of the predictions.

Based on these results, we can conclude that the polynomial linear regression model has a relatively better performance than the previous linear regression model, as indicated by the higher R2 score and explained variance score. The MSE and MAE values provide insights into the magnitude of the errors made by the model, with the MSE indicating larger errors compared to the MAE. Further analysis and improvement can be done to enhance the model's performance and reduce the errors in predicting the housing prices.

Regression Decision Trees:

In [154]:
```
#Regression Decision Trees
```

In [155]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=0)
```

In [156]:
```
regression_tree_houses = DecisionTreeRegressor()
regression_tree_houses.fit(X_train, y_train)
```

Out[156]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

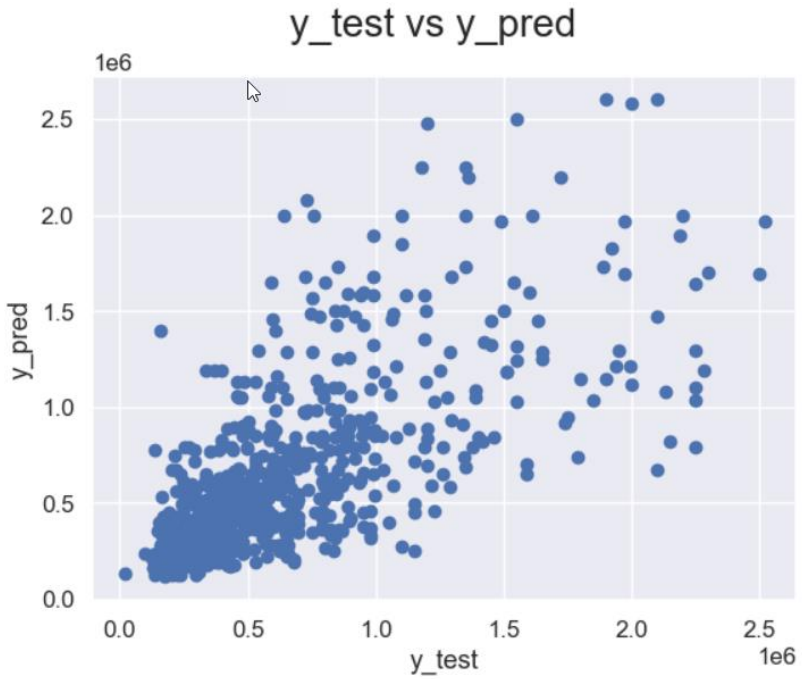
In [157]:
```
y_pred_tree = regression_tree_houses.predict(X_test)
```

In [158]:
```
print('r2_score: ',metrics.r2_score(y_test,y_pred_tree))
print('explained_variance_score: ',metrics.explained_variance_score(y_test,y_pred_tree))
print('mean_squared_error: ',metrics.mean_squared_error(y_test,y_pred_tree))
print('mean_absolute_error: ',metrics.mean_absolute_error(y_test,y_pred_tree))
```

```
r2_score:  0.3995023002014907
explained_variance_score:  0.3995856743210334
mean_squared_error:  123555393010.17847
mean_absolute_error:  236651.14305177113
```

In [159]:
```
fig = plt.figure()
plt.scatter(y_test,y_pred_tree)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test')                            # X-label
plt.ylabel('y_pred')
```

Out[159]:
```
Text(0, 0.5, 'y_pred')
```



The analysis of the results shows the performance of a model in predicting a target variable. Here are the key metrics for evaluation:

1. R2 Score: The R2 score, also known as the coefficient of determination, measures the proportion of the variance in the target variable that can be explained by the model. In this case, the R2 score is 0.3995, indicating that approximately 39.95% of the variance in the target variable can be attributed to the features included in the model. A higher R2 score indicates a better fit of the model to the data.

2. Explained Variance Score: The explained variance score measures the proportion of the variance in the target variable that is explained by the model. In this analysis, the explained variance score is 0.3996, indicating that the model explains approximately 39.96% of the variance in the target variable. A higher explained variance score suggests a better ability of the model to capture the underlying patterns in the data.

3. Mean Squared Error (MSE): The MSE quantifies the average squared difference between the predicted values and the actual values of the target variable. In this analysis, the MSE is 123,555,393,010.17847. The MSE gives an idea of the magnitude of the errors made by the model, with larger values indicating larger errors. It is important to note that the MSE is measured in squared units of the target variable.

4. Mean Absolute Error (MAE): The MAE measures the average absolute difference between the predicted values and the actual values of the target variable. In this analysis, the MAE is 236,651.14305177113. The MAE provides an indication of the average magnitude of the errors made by the model, regardless of their direction.

Based on these results, we can conclude that the model has a moderate performance in predicting the target variable. The R2 score and explained variance score indicate that around 40% of the variance in the target variable can be explained by the features in the model. However, the MSE and MAE values suggest that the model still has room for improvement, as the errors are relatively large. Further analysis and model refinement may be necessary to enhance the performance and accuracy of the predictions.

Next step:

To improve the model with the given results, several steps can be taken:

1. Feature Engineering: Review and analyze the features used in the model. Consider exploring additional relevant features that may have a stronger relationship with the target variable. Feature engineering techniques like creating interaction terms, polynomial features, or transforming variables can also be applied to capture more complex relationships.

2. Data Cleaning: Examine the dataset for any anomalies, inconsistencies, or missing values. Clean the data by handling missing values appropriately, addressing outliers, and ensuring the data quality is high. Removing or correcting any erroneous data points can help improve the model's performance.

3. Model Selection: Evaluate alternative regression models to determine if a different model type can better capture the underlying patterns and relationships in the data. Consider trying models such as random forests, support vector regression, or gradient boosting to see if they provide better performance compared to the current linear regression model.

4. Regularization Techniques: Apply regularization techniques, such as L1 or L2 regularization, to prevent overfitting and improve the model's generalization ability. Regularization can help control the complexity of the model and reduce the impact of irrelevant or noisy features.

5. Hyperparameter Tuning: Fine-tune the hyperparameters of the chosen model to optimize its performance. Grid search, random search, or Bayesian optimization techniques can be employed to find the best combination of hyperparameters that yield improved results.

6. Cross-Validation: Utilize cross-validation techniques to evaluate the model's performance on multiple subsets of the data. This helps ensure that the model's performance is robust and not dependent on a specific training/test split. Consider using k-fold cross-validation or stratified cross-validation to obtain more reliable performance estimates.

7. Ensemble Methods: Explore ensemble methods, such as bagging or boosting, to combine multiple models and improve prediction accuracy. Ensemble models can help reduce variance, improve generalization, and capture a wider range of patterns and relationships in the data.

8. Feature Selection: Conduct a thorough analysis of the features and their relevance to the target variable. Eliminate any features that are redundant, irrelevant, or have a weak relationship with the target. This can help simplify the model and improve its interpretability and performance.

9. Data Augmentation: If feasible, consider augmenting the dataset by collecting more data points or generating synthetic data to increase the diversity and quantity of the training data. More data can help the model better capture the underlying patterns and improve its generalization ability.

10. Domain Knowledge: Incorporate domain knowledge and expertise to gain insights into the problem and the data. This can guide the feature engineering process, help identify relevant variables, and inform decisions during model selection and interpretation.

Remember that model improvement is an iterative process, and it may require trying different combinations of the above steps and evaluating their impact on the model's performance using appropriate evaluation metrics.