Exploratory Data Analysis of Berlin Housing Market

Insights into Pricing, Area, and Space Relationships

Sima Faramarzi

Simafaramarzi780@gmail.com

Contents:

Introduction

In this analysis, I have focused on Python, for processing data and using statistical models and exploratory data analysis ([1]EDA) to analyze the Berlin housing dataset and study the characteristics affecting housing prices in this metropolitan. Factors, such as Physical and qualitative conditions, time construction, and renovation conditions of the houses are investigated. In addition, important factors such as the total square foot, which has been claimed to have a straight impact on price are surveyed. Meanwhile, the correlation between living area and living square feet is studied. The purpose of these analyses is gaining an analytical view of data frames and cognition the affective relationships between price and other factors.

Features Exploration:

Price: The most important dependent variable that its relationships is supposed to be examined in this study.

sqft_totall: a continuous variable with Float data type that points to the total square foot of houses

Sqft_living: a continuous variable with Float data type that points to the square living of houses

sqft_area_living: a continuous variable with Float data type that points to the square area living of houses

Bedrooms, Bathrooms: independent variables with Float data type that refer to the number of rooms and bathrooms

floor: an independent variable with Float data type that points to the number of house floor

condition: an independent variable with Float data type that refers to the grade of appearance condition of the house

grade: an independent variable with the integer data type that is the grade of house qualification.

Built: an independent variable with Float data type that refers to the year of the house construction

Renovated: an independent variable with the integer data type that refers to the year of the house renovation.

Upload and Preliminary review

Step 1: Importting Libraries:

They are libraries that contain a set of different functions, variables, and classes that allow the programmer to achieve their goals without writing long code and just by calling libraries and their subset functions. It should be noted that the biggest reason for the popularity and power of Python is for these libraries. (VanderPlac,2016)

[1] The first step is to analyze the data, which leads to understanding the data structure, discovering the relationships between variables and identifying patterns.

```
#liebraries
```

```
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt
import Numpy as np
```

Figure 1- python codes for importing libraries

Pandas:" provides high-level data structures and functions designed to make working with structured or tabular data fast, easy, and expressive. Since its emergence in 2010, it has helped make Python a powerful and productive data analysis environment" (McKinney,2017, p. 4).

Seaborn: It offers tools for visualizing designed data. Drawing advanced and correlation graphs.

Statsmodel.api: investigates probabilities in hypothetical tests, statistical analyses analyze relationships between variables.

Matplotlib.pyplot draw linear and columnar charts, and control and edit the appearance of the diagrams.

Statsmodels.api: used for statistical analysis, including linear and advanced regression, analysis of variance, etc

Numpy: for performing mathematical operations, working with numerical arrays and matrices, and processing data.

Step 2: Upload File:

```
# Upload & initial exploration the file
```

```
data = pd.read_csv('BerlinHousing4049.csv') # Uploading the csv file
```

Figure2 - Python code for uploading and accessing the file

first, by **pd**.**read_csv** method from pandas library we upload the data set from the root of the client's platform file and access it. then we assign Data Frame to the data variable.

Step 3: Data Exploration

```
# Data Exploration
```

```
print(data.shape) #Number of Rows & Columns
print(data.columns) #exhibit columns names
print(data.info()) # showing the data type, missing values, used memory
print(data.describe()) #Display a statistical summary of the data frame
```

Figure3- Python codes for Data Exploration

At this step, we do an overview of data frame data. First, we obtain information about the number of rows and columns, shape () method. then we give information such as the data type of variables and missed and repeated values by info (), and finally, we extract the statistical information of the data set, describe () method.

Note: all above functions are from pandas library, and for info () and describe () methods, abnormal values are not considered.

```
(9999, 11)
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_total', 'floors',
       'condition', 'grade', 'built', 'renovated', 'living_area_sqft'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9999 entries, 0 to 9998
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             9990 non-null   float64
 1   bedrooms          9988 non-null   float64
 2   bathrooms         9990 non-null   float64
 3   sqft_living       9992 non-null   float64
 4   sqft_total        9994 non-null   float64
 5   floors            9999 non-null   float64
 6   condition         9998 non-null   float64
 7   grade             9999 non-null   int64
 8   built             9996 non-null   float64
 9   renovated         9999 non-null   int64
 10  living_area_sqft  9988 non-null   float64
dtypes: float64(9), int64(2)
memory usage: 859.4 KB
None
```

Figure4- Preview of Data Frame information

	price	bedrooms	bathrooms	sqft_living	sqft_total	\
count	9.988000e+03	9986.000000	9988.000000	9990.000000	9.992000e+03	
mean	5.333549e+05	3.364711	2.063301	2053.169870	1.604302e+04	
std	3.769158e+05	0.910943	0.765345	911.453717	4.505838e+04	
min	7.500000e+04	1.000000	0.500000	380.000000	5.720000e+02	
25%	3.150000e+05	3.000000	1.500000	1410.000000	5.426500e+03	
50%	4.458190e+05	3.000000	2.000000	1890.000000	7.916500e+03	
75%	6.399250e+05	4.000000	2.500000	2500.000000	1.118000e+04	
max	7.700000e+06	11.000000	8.000000	12050.000000	1.651359e+06	

	floors	condition	grade	built	renovated	\
count	9997.000000	9996.000000	9997.000000	9994.00000	9997.000000	
mean	1.431930	3.446379	7.594278	1967.26636	91.462639	
std	0.511857	0.666369	1.165926	27.98690	417.444121	
min	1.000000	1.000000	3.000000	1900.00000	0.000000	
25%	1.000000	3.000000	7.000000	1950.00000	0.000000	
50%	1.000000	3.000000	7.000000	1969.00000	0.000000	
75%	2.000000	4.000000	8.000000	1990.00000	0.000000	
max	3.500000	5.000000	13.000000	2015.00000	2015.000000	

	living_area_sqft
count	9986.000000
mean	1976.015021
std	672.388027
min	620.000000
25%	1490.000000
50%	1830.000000
75%	2340.000000
max	5790.000000

Figure 5 - Preview of Data Frame Description

A data frame of housing information that includes 11columns of float, int, and object data types. Occupies about 1015.4+ memory and contains almost 10,000 data records.

Preprocessing Process

The data framework should be modified through standardization, data type modification, and integration processes. In this analysis, we decided to use the ITU-T process.

The **ITU-T** process is a data preparation and standardization process that is significant in data analysis Science because it increases the quality of data and leads to standard modeling and analysis.

Step 4: Type Conversion

The values entered in the Build and Renovation fields refer to the year of construction and renovation of the houses. According to the type of input data, which is the object, the datetime data type is more logical for these factors.

```
#Type Conversion & Correction

data['built'] = pd.to_datetime(data['built'], format='%Y', errors='coerce') # pd.Datat type(Data Frame)
data['renovated'] = pd.to_datetime(data['renovated'], format='%Y', errors='coerce')

#format='%Y': When the input data is only one year.  The year 1990 is entered, considered as 01.01.1990.
#format= '%Y/%m/%d': If the input data includes month and day.
# errors='coerce': Converts invalid values to NAN value
data.to_csv("BerlinHousing4049.csv", index=False)
data.to_csv("BerlinHousing4049_clean_sckwedness.csv", index=False)# Creating a new data frame for sckwedness
data.info(5) # Check for change datatype
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9999 entries, 0 to 9998
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             9990 non-null   float64
 1   bedrooms          9988 non-null   float64
 2   bathrooms         9990 non-null   float64
 3   sqft_living       9992 non-null   float64
 4   sqft_total        9994 non-null   float64
 5   floors            9999 non-null   float64
 6   condition         9998 non-null   float64
 7   grade             9999 non-null   int64
 8   built             9996 non-null   datetime64[ns]
 9   renovated         459 non-null    datetime64[ns]
 10  living_area_sqft  9988 non-null   float64
dtypes: datetime64[ns](2), float64(8), int64(1)
memory usage: 859.4 KB
```

Figure 6- Python code and Preview of converting Data types

By to_datetime method for pandas library, we convert variables built and renovated data type to date&time. The format argument indicates that our date format only includes the year, and we set the errors parameter to ‘coerce’ to fill invalid values with NAN and NONE values.

BerlinHousing4049_sckwedness.csv : In the continuation of the data frame analysis, we will proceed to draw the distribution diagram of the Price variable. Applying the Log function on the data along with normalization causes asymmetry in the shape of the graph, so we need a data frame that has data empty of missing and duplicated values but not normalized.

Step 5: Removing Duplicate Values

Integration greatly increases the quality, and speed of data analysis and also reduces the percentage of analysis errors.

```
#Removing the Duplicates values
dupli_count = data.duplicated().sum() # collecting the duplicated rows
data_cle = data.drop_duplicates() # Removing
print(f"\nNumber of Duplicate Rows Removed: {dupli_count}")
data_cle.to_csv("BerlinHousing4049.csv", index=False) # apply the changes to file
data.to_csv("BerlinHousing4049_clean_sckwedness.csv", index=False)
```

```
Number of Duplicate Rows Removed: 2
```

Figure 7- Python code for exploratory and eliminating duplicate rows

First, we search for duplicate rows and collect their quantity with the data.duplicated() and sum() commands o the pandas library, then we delete them with the data.drop_duplicates (). Finally, we save the changes in the original file to the .to_csv method from the pandas library and determine that the index does not present, index=False.

According to the result of the code execution, there were only 2 repeated rows in the data set, which were deleted, and the rest of the rows are unique.

Step 6: Handling Missing Values

```
# Explority for (NAN, missing, None) values
```

```
df= pd.DataFrame(data)
missed_data= df.isnull().sum() # counting the missed values and collecting their quantity
print("Number of Miseed Data:     ", missed_data)
df_fill=df.fillna(df.median(numeric_only=True), inplace=True) # filling the missed values with the median of their columns
data_cle.to_csv("BerlinHousing4049.csv", index=False) # apply the changes to file
# inplace= True method applies the changes directly to the data frame
data_cle.to_csv("BerlinHousing4049_clean_sckwedness.csv", index=False)
print(df_fill)
```

Figure 8- Python code for exploratory and Filling missed values

First, we search for missing values and collect their quantity with the .isnull () and sum() commands from the pandas library, then fill them with the median of their columns by the fillna () method.

Numeric_only: point to get median only from columns with number data types, and save the changes directly to the original file by setting inplace=True.

```
Number of Miseed Data:      price              9
bedrooms             11
bathrooms             9
sqft_living           7
sqft_total            5
floors                0
condition             1
grade                 0
built                 3
renovated             0
living_area_sqft     11
dtype: int64
None
```

Figure 9- Preview of The total missed values

The number of missing data for each variable is shown and all these missing values are filled with the median of the set of column values.

Step 7: Feature Engineering

In this step, we calculate the price per square meter based on the price of the whole house. Since the price of each house varies according to the location, construction quality, etc., this helps a lot to compare the prices better.

```
# Calculating the price based on the foot square of the houses
```

```
data['price_per_sqft'] =(data['price'] / data['sqft_total']).round(3) # Calculate the price of each square meter based on price and square footage
                                                                      # with three decimal places
data.head(50)
```

Figure 10- Python code for price of square meter

As the code, by dividing the price by the total square feet of the houses, the Price per square foot is obtained and stored in the new price_per_sqft column with three decimal digits.

	price	bedrooms	bathrooms	sqft_living	sqft_total	floors	condition	grade	built	renovated	living_area_sqft	build	renovate	price_per_sqft
0	221900.0	3.0	1.00	1180.0	5650.0	1.0	3.0	7	1955.0	0	1340.0	1955-01-01	NaT	39.274
1	538000.0	3.0	2.00	2570.0	7242.0	2.0	3.0	7	1951.0	1991	1690.0	1951-01-01	1991-01-01	74.289
2	180000.0	2.0	1.00	770.0	10000.0	1.0	3.0	6	1933.0	0	2720.0	1933-01-01	NaT	18.000
3	604000.0	4.0	3.00	1960.0	5000.0	1.0	5.0	7	1965.0	0	1360.0	1965-01-01	NaT	120.800
4	510000.0	3.0	2.00	1680.0	8080.0	1.0	3.0	8	1987.0	0	1830.0	1987-01-01	NaT	63.119
5	1225000.0	4.0	4.50	5420.0	101930.0	1.0	3.0	11	2001.0	0	4760.0	2001-01-01	NaT	12.018
6	257500.0	3.0	2.25	1715.0	6819.0	2.0	3.0	7	1995.0	0	2238.0	1995-01-01	NaT	37.762
7	291850.0	3.0	1.50	1060.0	9711.0	1.0	3.0	7	1963.0	0	1650.0	1963-01-01	NaT	30.054
8	229500.0	3.0	1.00	1780.0	7470.0	1.0	3.0	7	1960.0	0	1780.0	1960-01-01	NaT	30.723
9	323000.0	3.0	2.50	1890.0	7916.5	2.0	3.0	7	2003.0	0	2390.0	2003-01-01	NaT	40.801
10	662500.0	3.0	2.50	3560.0	9796.0	1.0	3.0	8	1965.0	0	2210.0	1965-01-01	NaT	67.630
11	468000.0	2.0	1.00	1160.0	6000.0	1.0	4.0	7	1942.0	0	1330.0	1942-01-01	NaT	78.000
12	310000.0	3.0	1.00	1430.0	19901.0	1.5	4.0	7	1927.0	0	1780.0	1927-01-01	NaT	15.577
13	400000.0	3.0	1.75	1370.0	9680.0	1.0	4.0	7	1977.0	0	1370.0	1977-01-01	NaT	41.322
14	445819.0	5.0	2.00	1810.0	4850.0	1.5	3.0	7	1900.0	0	1360.0	1900-01-01	NaT	91.921
15	650000.0	4.0	3.00	2950.0	5000.0	2.0	3.0	9	1979.0	0	2140.0	1979-01-01	NaT	130.000
16	395000.0	3.0	2.00	1890.0	14040.0	2.0	3.0	7	1994.0	0	1890.0	1994-01-01	NaT	28.134

Figure 11- Preview of Data Frame and price_per_sqft column

As the preview, a house with a price of 538000, its price per square meter is 74.289. Some prices are higher than 500,000 per square meter, which indicates luxury properties. A rare number of prices are above 2000 square meters, which are our outliers.

Step 8: Standardization

" Standardization is defined as a process that aims to reduce data redundancy and improve data integrity by organizing tables and relationships in a relational database" .(C.J.Date,2016)

The scale difference between the data can be seen, especially in important variables, such as price and Total square. While the scale of price is between thousands and millions scale, the Total square's scale is between ten and hundred square meters, this difference in scale affects the performance of the machine learning and reduces the influence of other variables on price. Meanwhile, the histogram of the distribution faces a challenge.

Note: StandardScaler method: use of Mean and Deviation for standardization.

```
# Standardzation
```

```
from sklearn.preprocessing import StandardScaler #Enter the class of StandardScaler normalization method
scaler = StandardScaler() # an object of the StandardScaler class named scaler is created

data[['price','sqft_total','sqft_living', 'living_area_sqft']] = scaler.fit_transform(data[['price','sqft_total','sqft_living',
                                                                                            'living_area_sqft']])# Applying Normalization
data.to_csv("BerlinHousing4049.csv", index=False)# save the Transforms in the orginal file

data.head(5)
```

Figure 12- Python code for Normalization data

First, import the sklearn library, from the preprocessing submodule. Then create an object from StandardScaler (), which is one of the given standardization methods. Normalizes the data with a mean of 1 and a standard deviation of 0. Second, use the scaler. Fit_transform() applies normalization to four variables. Finally, save the normalized columns to the file.

	price	bedrooms	bathrooms	sqft_living	sqft_total	floors	condition	grade	built	renovated	living_area_sqft
0	-0.826359	3.0	1.0	-0.958009	-0.230639	1.0	3.0	7	1955.0	0	-0.945876
1	0.012394	3.0	NaN	0.567156	-0.195302	2.0	3.0	7	1951.0	1991	-0.425329
2	-0.937538	2.0	1.0	-1.407878	-0.134083	1.0	3.0	6	1933.0	0	1.106565
3	0.187521	4.0	3.0	-0.102161	-0.245066	1.0	5.0	7	1965.0	0	-0.916130
4	-0.061903	3.0	2.0	-0.409388	-0.176701	1.0	3.0	8	1987.0	0	NaN

Figure 13 - Preview of normalized Data Frame

These 4 variables were chosen for normalization because they have the highest skewness. They scaled between 3- and 3, with a mean of 0 and a Standard deviation of 1.

Handling outliers

Outliers: Outlier data are abnormal data that are at an unusual distance from the rest of the data in a set, Usually more than 3 standard deviations from the mean. They are the values that are clearly defined in the pattern or graph of the set. Initially, we have an overview of the general situation of the Outliers, Medians, Box, and [2]Whiskers in the data frame.

[2] Whiskers: It refers to the tentacles on both sides of the data box

Step 9: Finding the Outliers in the Data Frame:

```
plt.figure(figsize=(8, 5)) #Adjust the size of the chart frame
sns.boxplot(x=data['price'], color='red', showfliers=True)# Drawing the box diagram for price in order to distribution data,
                                                          #Quartiles and statistical distribution

min_val = data['price'].min() # calculating the min of price column
max_val = data['price'].max() # calculating the max of price column
median_val = data['price'].median() # calculating the median of price column
mean_val = data['price'].mean() # calculating the mean of price column

#Drawing lines of statistical descriptions
plt.axvline(mean_val, color='orange', linestyle='-.', label=f'Mean: {mean_val:.2f}') #drawing an orange line to showing Mean
plt.axvline(median_val, color='green', linestyle='-', label=f'Median: {median_val:.2f}')#drawing an orange line to showing Median
plt.axvline(min_val, color='blue', linestyle='-', label=f'Min: {min_val:.2f}') #drawing an orange line to showing min
plt.axvline(max_val, color='purple', linestyle='-', label=f'Max: {max_val:.2f}') #drawing an orange line to showing max

plt.title('Boxplot of Price' , fontsize=10)
plt.xlabel('Price')
plt.legend(loc='upper right') # creating a legend box
plt.tight_layout()
plt.show()
```

Figure 14- Python code for searching outlier values in price and total square foot

First, using boxplot () from the seaborn library we draw a boxplot of the variable price. Because Seaborn limits the maximum and minimum quartiles (IQR). Thus, we set showfliers =True to show the min and max lines in the chart. After calculating the four values using the axvline method, from the Matplotlib library, we draw vertical lines, min, max, mean, and median lines. In the last part of the coding, after determining the title of the box and the horizontal axis, we create a guide box in the upper right corner using the plt.Legend() command.

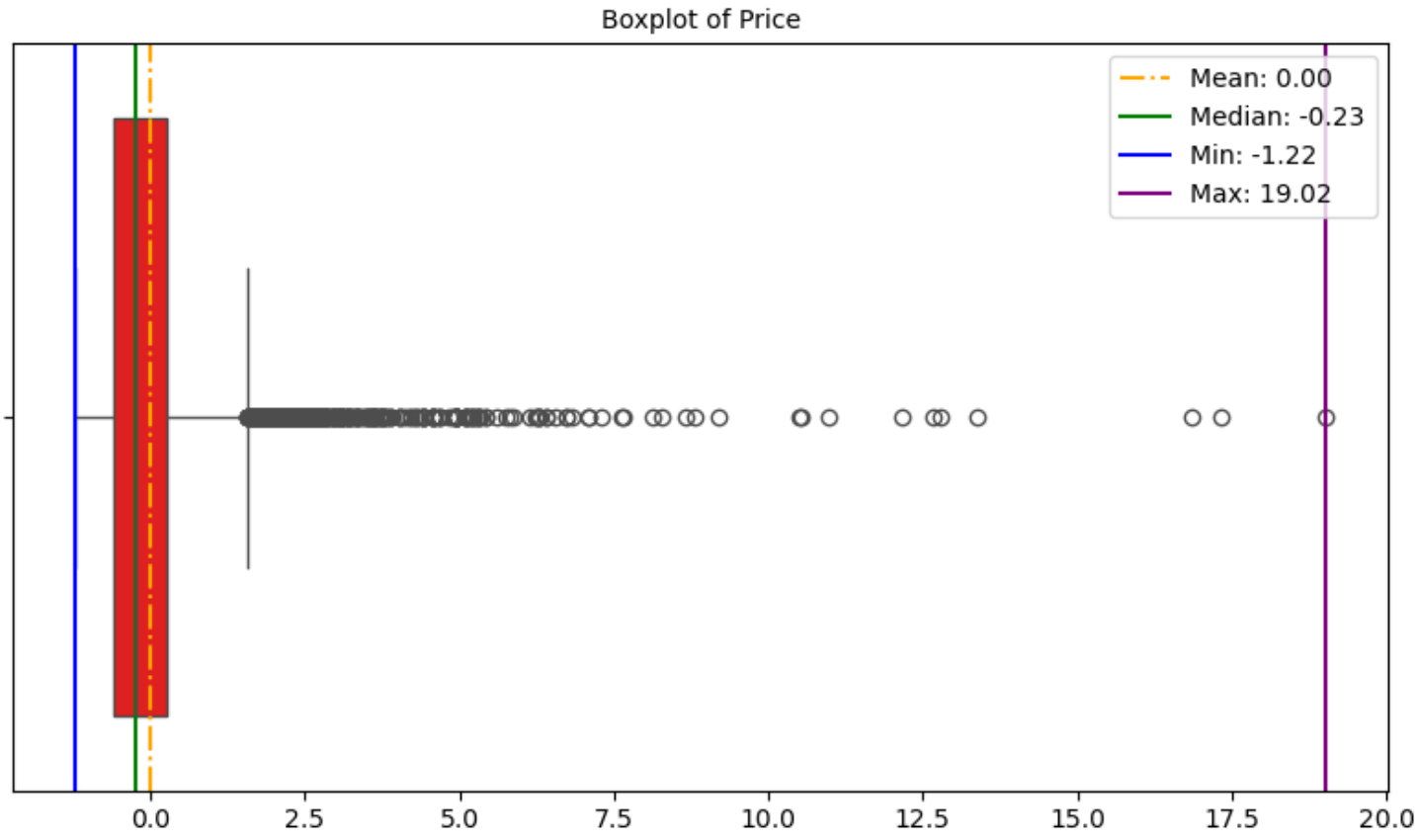

Diagram1- a preview of outliers in price and total square feet

Based on the distance between Min (-1.22) and Max (19.02) and the distance between Mean (-0.00) and Max (19.02), it is clear that this distribution is skewed to the right. A large percentage of the data is compact and located in the lower range of the graph (low-price houses). Outliers (high-value houses) are mostly seen in the faraway right points from the mean and median. Meanwhile, few houses have very high prices (luxurious houses).

Step 10: Searching and Removing Outliers IQR:

```
Q1_price = data['price'].quantile(0.25)  # Assign 25% of  price column data to Q1_price
Q3_price = data['price'].quantile(0.75) #Assign 75% of price column data to Q3_price
IQR_price = Q3_price - Q1_price         #Calculate the distance between Q1 and Q2 to get the distance between these two quadrants
lower_bound_price = Q1_price - 1.5 * IQR_price  #Calculation of  Low quadrants
upper_bound_price = Q3_price + 1.5 * IQR_price   #Calculation of High  quadrants
data = data[(data['price'] >= lower_bound_price) & (data['price'] <= upper_bound_price)] # Remove the price column outliers
print("Number of rows after removing outliers:", len(data)) # show the number of removed rows
plt.figure(figsize=(8, 5)) #Adjust the size of the chart frame
sns.boxplot(x=data['price'], color='red', showfliers=True)# Drawing the box diagram for price in order to distribution data,
                                                          #Quartiles and statistical distribution
min_val = data['price'].min() # calculating the min of price column
max_val = data['price'].max() # calculating the max of price column
median_val = data['price'].median() # calculating the median of price column
mean_val = data['price'].mean() # calculating the mean of price column
#Drawing lines of statistical descriptions
plt.axvline(mean_val, color='orange', linestyle='-.', label=f'Mean: {mean_val:.2f}') #drawing an orange line to showing Mean
plt.axvline(median_val, color='green', linestyle='-', label=f'Median: {median_val:.2f}')#drawing an orange line to showing Median
plt.axvline(min_val, color='blue', linestyle='-', label=f'Min: {min_val:.2f}') #drawing an orange line to showing min
plt.axvline(max_val, color='purple', linestyle='-', label=f'Max: {max_val:.2f}') #drawing an orange line to showing max
plt.title('Boxplot of Price' , fontsize=10)
plt.xlabel('Price')
plt.legend(loc='upper right') # creating a legend box
plt.tight_layout()
data_cle.to_csv("BerlinHousing4049.csv", index=False)
data_cle.to_csv("BerlinHousing4049_clean_sckwedness.csv", index=False)
plt.show()
```

```
Number of rows after removing outliers: 9498
```

Figure 15- Python code for Removing outlier values in price and total square feet

R Language Programming: “R is a language and environment for statistical computing and graphics, similar to the language originally developed at Bell Labs. It’s an open-source solution to data analysis that’s supported by a large and active worldwide research community”. (Kabacoff, 2011)

IQR: one of the most practical strategies for finding and removing outliers. First, we specify the lowest (25%) and highest (75%) data ranges using the quantile () function, this function is used in R language to calculate quartiles and other quarters. Then by subtracting them from each other, we get the distance between the two limits(IQR_sqft).

In the next step, using the formula Q1_sqft- 1.5 * IQR, identify the outlier in the left side, and by calculating Q3_sqft- 1.5 * IQR, we find the outliers in the right of the bound.

Data= 30,27,25,21,18,14,12,8,7,5

Q1 (25%)= $\frac{n+1}{4} = \frac{10+1}{4}$ = 2.75 this value is between the second(7) and third(8) element.

Q1= 7 + 0.75(8-7) = 7 + 0.75 = 7.75

Q3 (75%)= $\frac{(n+1)3}{4} = \frac{(10+1)3}{4}$ = 8.25 this value is between the eighth and ninth elements.

Q3 = 25+0.25 (27-25) = 25 + 0.5 = 25.5

IQR= Q3 –Q1

IQR= 7.75 – 25.5 = 17.75

[IQR * 1.5 + IQR, Q3*1.5-Q1]

[(17.75 * 1.5) + 25.5, (17.75 * 1.5) – 17.75]

If any data is out of this range, it is an Outlier **[52.125, -18.875]**

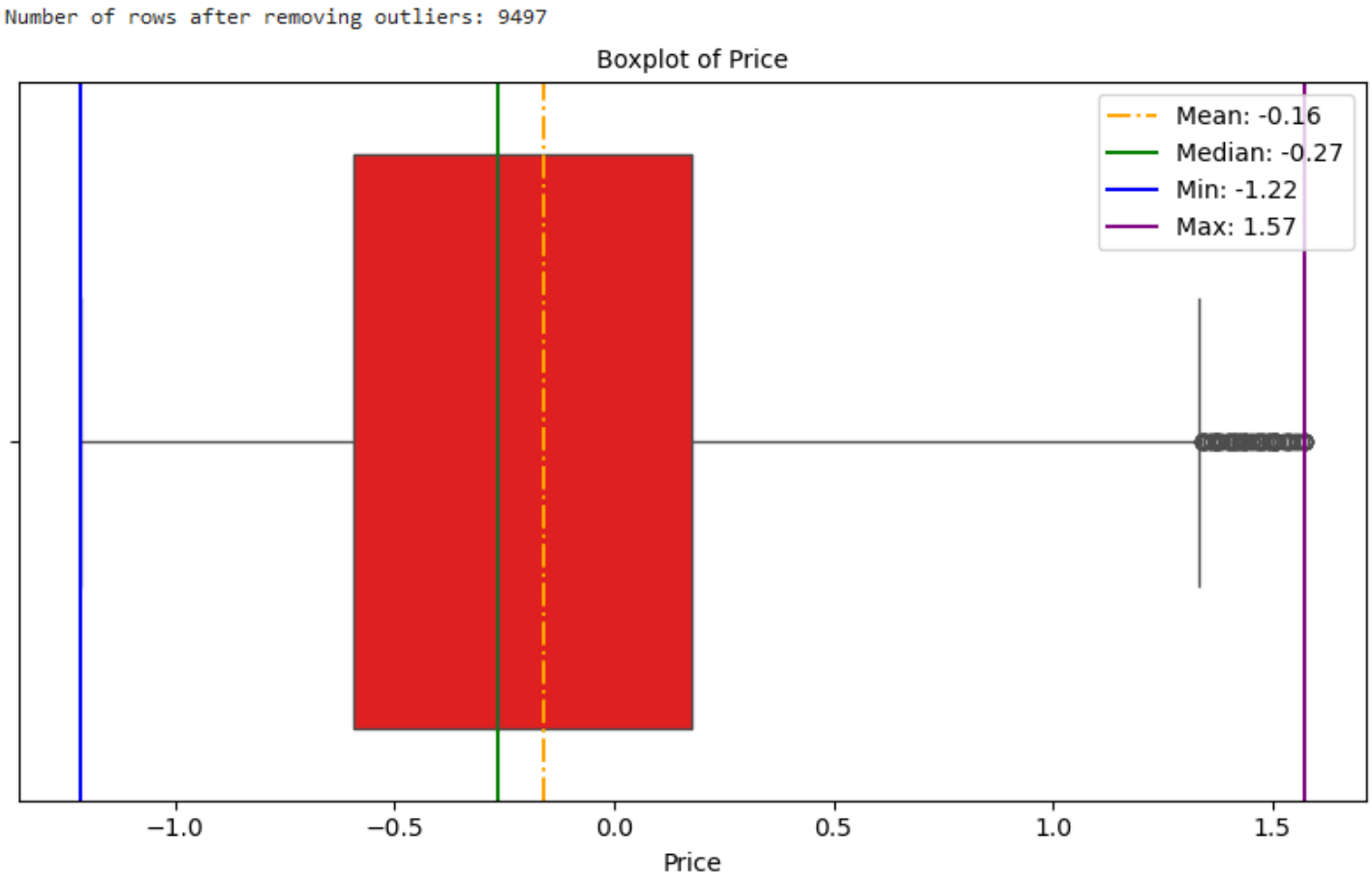

Diagram 2- a preview of the Data Frame after removing the outliers

The number of 9498 rows of outlier data was removed, and as the graph shows, the values of Min (-1.22) and Max (1.57) are spread on both sides of the Mean (-0.16), which has partially caused the symmetrical distribution of the data. But there is still a slight skew to the right.

Step 11: Statistical Description

Analytical report of the data frame after pre-processing and preparation for analysis .By performing standardization on the data, the values are displayed on a standard scale, but the distribution graph is still preserved

```
#Descriptive statistics
```

```
# Statistic Report
print("Statistical report of numerical data:")
print(data.describe(include='all'))# bcause of using all, All numeric properties display all numeric and non-numeric columns
```

Figure 16- Python code of data frame descriptions

```
Statistical report of numerical data:
              price     bedrooms    bathrooms  sqft_living   sqft_total  \
count  9490.000000  9480.000000  9482.000000  9483.000000  9485.000000
mean     -0.161558     3.326899     2.003691    -0.108026    -0.010630
min      -1.216150     1.000000     0.500000    -1.835802    -0.343353
25%      -0.595344     3.000000     1.500000    -0.727588    -0.237031
50%      -0.267014     3.000000     2.000000    -0.233830    -0.182916
75%       0.176907     4.000000     2.500000     0.380625    -0.116326
max       1.572617    11.000000     7.500000     5.954608    36.298504
std       0.553746     0.896356     0.708559     0.844799     0.965668

             floors    condition        grade        built    renovated  \
count  9490.000000  9489.000000  9490.000000  9487.000000  9490.000000
mean      1.412908     3.443145     7.473551  1967.123643    80.125290
min       1.000000     1.000000     3.000000  1900.000000     0.000000
25%       1.000000     3.000000     7.000000  1950.000000     0.000000
50%       1.000000     3.000000     7.000000  1969.000000     0.000000
75%       2.000000     4.000000     8.000000  1990.000000     0.000000
max       3.500000     5.000000    12.000000  2015.000000  2015.000000
std       0.505661     0.663711     1.030255    27.736870   391.814908
```

```
       living_area_sqft                        build  \
count       9480.000000                         9487
mean          -0.094466  1967-02-15 06:49:13.009381248
min           -2.016715            1900-01-01 00:00:00
25%           -0.752530            1950-01-01 00:00:00
50%           -0.261729            1969-01-01 00:00:00
75%            0.422418            1990-01-01 00:00:00
max            5.062719            2015-01-01 00:00:00
std            0.895215                            NaN

                           renovate
count                           381
mean   1995-10-10 00:22:40.629921280
min              1940-01-01 00:00:00
25%              1986-01-01 00:00:00
50%              2000-01-01 00:00:00
75%              2010-01-01 00:00:00
max              2015-01-01 00:00:00
std                              NaN
```

Figure 17- Preview of all numeric and non-numeric data

Visualization

In the visualization, the **matplotlib** and **seaborn** libraries are the most practical libraries.

Step 12: categorization

The purpose is to calculate in which order most houses are in the price range and which variables affect the price of houses.

```
#Data grouping and average Catigurization
grouped_data = data.groupby(['price']).mean() #Grouping and average
plt.figure(figsize=(8, 2))
colors = ['skyblue', 'lightgreen', 'salmon', 'orange', 'MediumPurple'] # for making a barchart colorfull
print(grouped_data.head(20)) # Display 20 rows of data frame data
data['price'].value_counts().head(20).plot(kind='bar', figsize=(6,4), color=colors , rot=85) # Create a bar chart of (20) frequently for more readability
# rot=85:Create a 45 degree angle to insert price mean on x axis to avoid overlapping
plt.title('Price grouping bar chart',fontsize=10)
plt.ylabel('frequently')
plt.show()
```

Figure 18- Python code for categorization

First, we group the price column using the groupby () function from pandas. Then for each group, we obtain the average data of each group, mean (). Then, by comparing the average of each group with other factors, we find out the relationship between them. As a visualization, using the plot () command, we will draw the grouped price column which is normalized.

price	bedrooms	bathrooms	sqft_living	sqft_total	floors	condition
-1.216697	1.0	2.063263	-1.518133	0.606924	1.0	3.000000
-1.203424	1.0	0.750000	-1.781563	-0.244018	1.0	2.000000
-1.198115	3.0	1.000000	-1.309584	-0.124659	1.0	3.000000
-1.196787	2.0	1.000000	-1.682777	0.139724	1.0	2.000000
-1.192805	2.0	1.000000	-1.485204	0.090791	1.0	3.000000
-1.179532	3.0	1.000000	-1.265679	-0.250678	1.0	4.000000
-1.177010	1.0	1.000000	-1.627896	-0.265554	1.0	3.000000
-1.176878	1.0	1.000000	-1.397394	-0.267330	1.0	3.000000
-1.163605	1.5	0.875000	-1.397394	-0.197438	1.0	3.500000
-1.159623	3.0	1.000000	-1.331537	-0.087692	1.0	3.000000
-1.152986	2.0	1.000000	-1.199822	-0.173282	1.0	2.000000
-1.150331	2.5	1.000000	-1.262935	-0.167043	1.0	2.750000
-1.131749	2.0	1.000000	-1.518133	-0.251344	1.0	4.000000
-1.123785	2.0	1.000000	-1.382759	-0.203787	1.0	2.666667
-1.120334	2.0	1.000000	-1.090059	-0.222926	1.0	4.000000
-1.110578	2.0	1.000000	-1.441299	-0.217375	1.0	3.000000
-1.110512	2.0	1.000000	-1.155917	0.010730	1.0	3.000000
-1.102548	1.0	1.000000	-1.518133	-0.179476	1.0	4.000000
-1.102216	2.0	1.000000	-1.386418	-0.197326	1.0	4.000000
-1.098566	3.0	1.000000	-0.969321	-0.111915	1.0	2.000000

price	grade	built	renovated	living_area_sqft	build
-1.216697	3.000000	1966.00	0.0	-1.214254	1966-01-01 00:00:00
-1.203424	4.000000	1912.00	0.0	-1.154730	1912-01-01 00:00:00
-1.198115	6.000000	1954.00	0.0	-1.244016	1954-01-01 00:00:00
-1.196787	5.000000	1951.00	0.0	-0.601159	1951-01-01 00:00:00
-1.192805	6.000000	1949.00	0.0	-0.723182	1949-01-01 00:00:00
-1.179532	6.000000	1969.00	0.0	-1.601158	1969-01-01 00:00:00
-1.177010	5.000000	1942.00	0.0	-1.616039	1942-01-01 00:00:00
-1.176878	5.000000	1905.00	0.0	-1.229135	1905-01-01 00:00:00
-1.163605	6.000000	1931.00	0.0	-1.110087	1931-01-01 00:00:00
-1.159623	6.000000	1959.00	0.0	-1.690444	1959-01-01 00:00:00
-1.152986	6.000000	1948.00	0.0	-1.065444	1948-01-01 00:00:00
-1.150331	6.000000	1950.75	0.0	-0.976159	1950-10-01 18:00:00
-1.131749	6.000000	1948.00	0.0	-0.738063	1948-01-01 00:00:00
-1.123785	5.666667	1940.00	0.0	-1.333301	1940-01-01 08:00:00
-1.120334	5.000000	1908.00	0.0	-1.541635	1908-01-01 00:00:00
-1.110578	6.000000	1942.00	0.0	-1.482111	1942-01-01 00:00:00
-1.110512	5.000000	1913.00	0.0	-0.931516	1913-01-01 00:00:00
-1.102548	6.000000	1978.00	0.0	-0.559492	1978-01-01 00:00:00
-1.102216	6.000000	1944.00	0.0	-1.735087	1944-01-01 00:00:00
-1.098566	6.000000	1980.00	0.0	-0.574373	1980-01-01 00:00:00

Figure 19- normalize price column categorization

The Square living factor significantly impacts price changes. Meanwhile, factors such as build, renovation, and condition have a relative impact on housing market changes. On the other hand, the remaining features, such as the number of bedrooms and bathrooms, do not have a notable effect on the price.

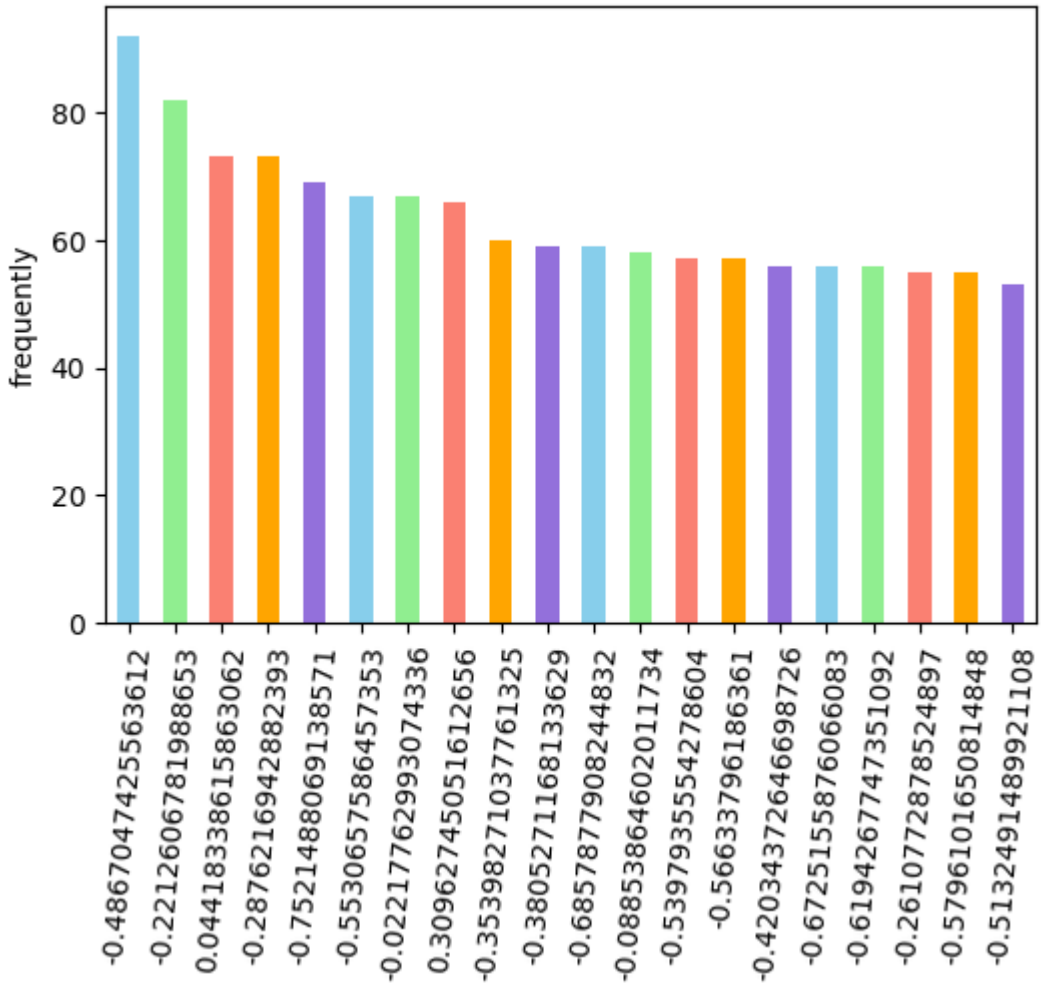

Diagram 3- a preview of the most 10 repeated values price grouping chart

The descending order shows the most common home prices from smaller square footage and average amenities and lower prices to astronomical prices with higher square footage for luxury homes.

Step 13: Distribution Diagram of price

```
#Distribution Histogram
```

```
# Distribution of 'price
df=pd.read_csv("BerlinHousing4049_clean_sckwedness.csv") # Refer to un-normalized Data Frame
df_clean =df['price'].dropna() # Remove zero values to avoid NAN error.
df_clean = df_clean[df_clean > 0]  # Remove invalid values

plt.figure(figsize=(10,6))  # Setting Chart dimensions
sns.histplot((df_clean), kde=True, bins=50, color="skyblue", edgecolor="black")# Drawing chart
# bins= 50 : The more it is, the more details are displayed from the histogram
# Kde=True: Display the density curve
plt.title('Transformed Price Distribution (Log)', fontsize=14) # setting chart title
plt.xlabel('log(Price)', fontsize=12)# setting horizental title
plt.ylabel('Frequency', fontsize=12)#setting vertical title

plt.tight_layout() # Setting intervalsn
plt.show()
```

Figure 20- Python code for Distribution Histogram

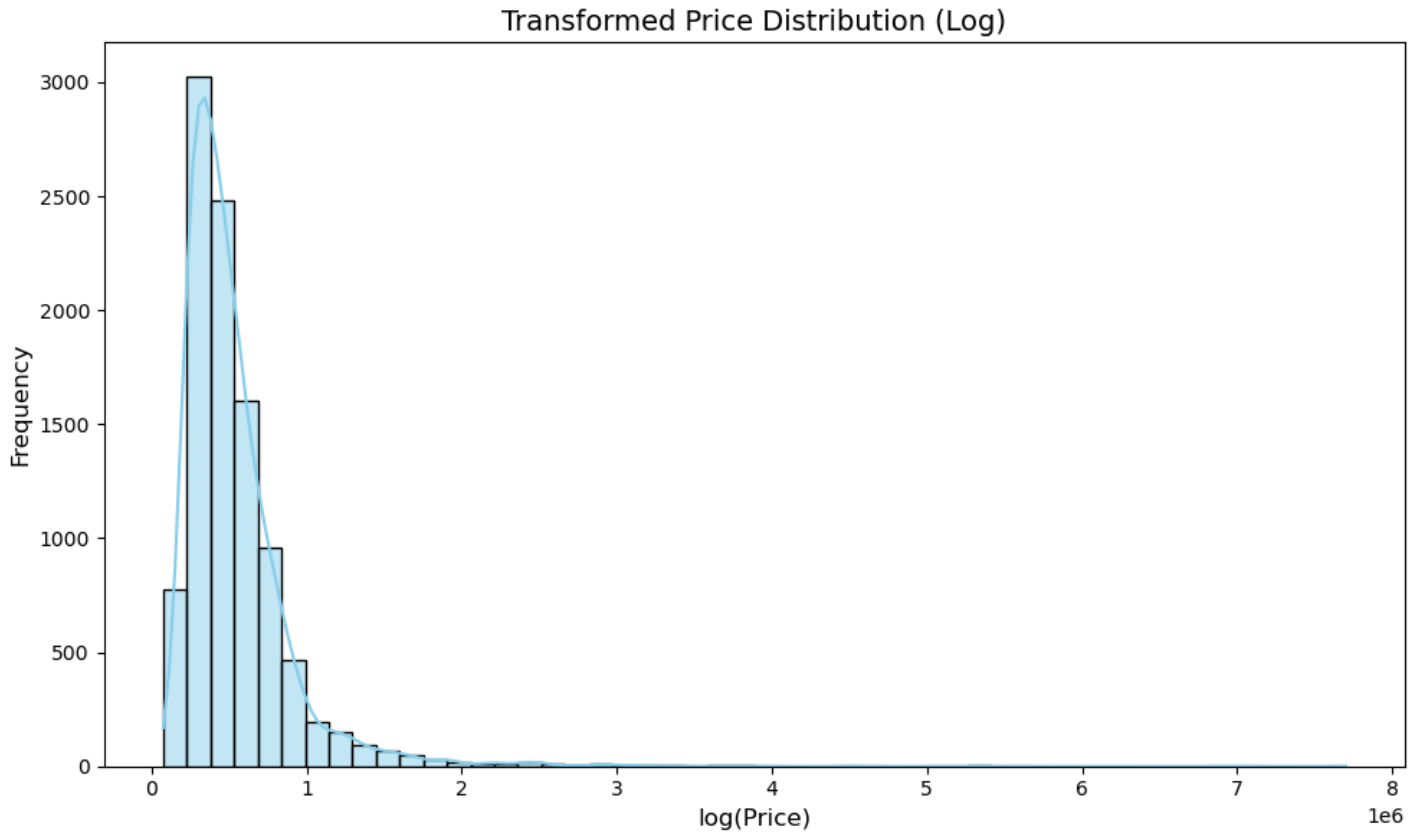

Diagram 5- a preview of the asymmetry distribution histogram

In the previous steps, we tried to reduce the outliers by removing them. In the first drawing of the distribution chart from the price column, we noticed that there is still a large amount of right - skewedness, which causes the asymmetry of the chart.

```
# Distribution of 'price
df=pd.read_csv("BerlinHousing4049_clean_sckwedness.csv")
df_clean = df['price'].dropna()
df_clean = df_clean[df_clean > 0]

plt.figure(figsize=(10,6))
sns.histplot(np.log(df_clean), kde=True, bins=50, color="skyblue", edgecolor="black") # Applying the log() for creat Symmetric distribution graph

plt.title('Transformed Price Distribution (Log)', fontsize=14)
plt.xlabel('log(Price)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)

plt.tight_layout() # Setting intervalsn
plt.show()
```

Figure21- Python code for making Symmetry

we decided to reduce this deviation to some extent by shrinking the data. For this reason, we decided to draw the graph by using the np. Log () function from numpy library, and take the logarithm from the data before drawing the graph. However, we must be sure about 2 issues. The removal of zero values, .dropna (), and the removal of invalid values, [df_clean>0], because these values cause disturbances in the application of the Log () function.

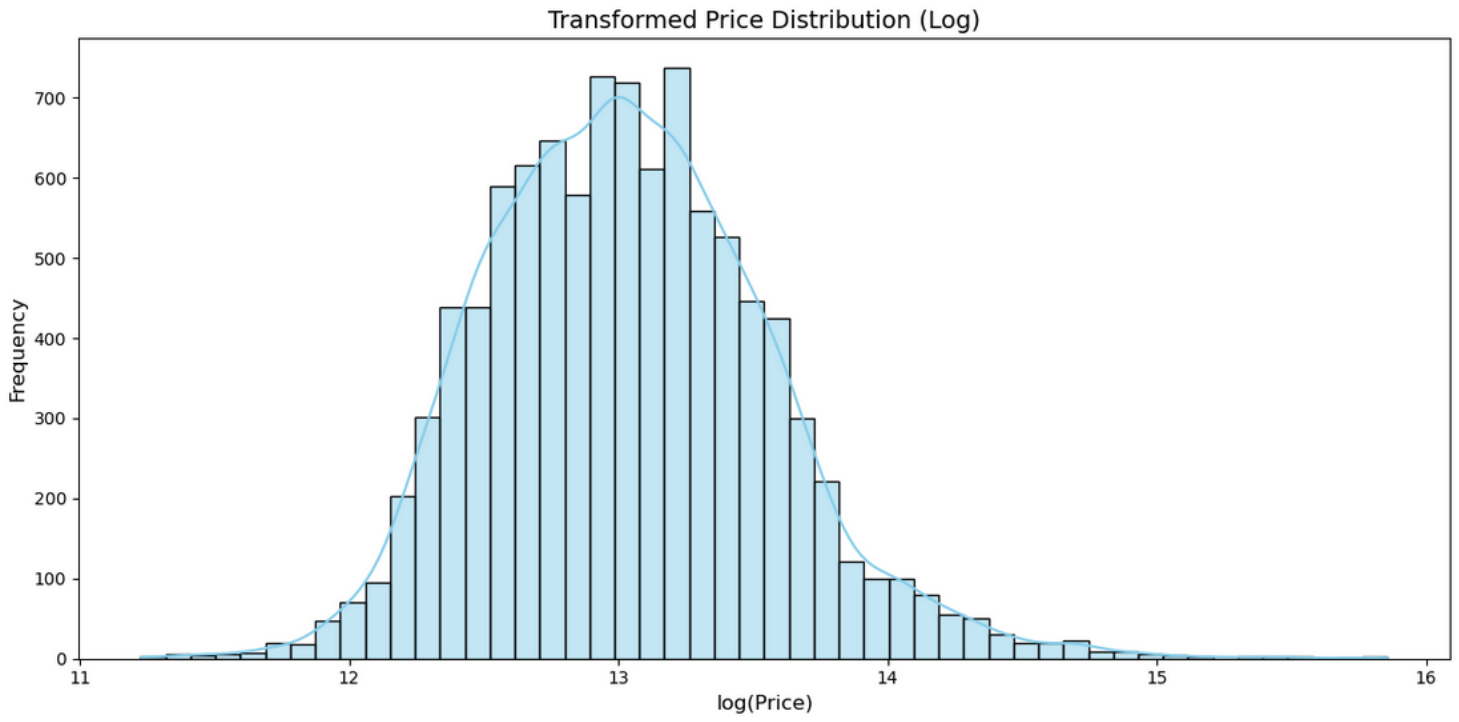

Diagram6- Symmetric distribution chart

This is a symmetrical plot of logarithmic values that still has slight skewness to the right. Most of the data is around the median point (13), and between Logarithmic values of 14 and 12. The tails on both sides represent some data outliers of houses with very high and low prices.

Note: In this graph, we used non-normalized data. Applying log () function on normalized data causes asymmetry and error in histogram drawing.

Step 14: Survey the relationship of Price and other Features:

```
fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(15, 15)) #Setting the size of the chart frame and the number of rows and columns for chart layout
axes = axes.flatten() #Creates a two-dimensional array for easier access to the axes of the graph


variables = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_total', 'floors',
             'condition', 'grade', 'built', 'renovated', 'living_area_sqft']


for i, col in enumerate(variables):  # Drawwing a scatter plot for each variable
    sns.scatterplot(data, x=col, y='price', ax=axes[i], color='red', alpha=0.5)
    axes[i].set_title(f' {col} vs price', fontsize=8)
    axes[i].set_xlabel(col, fontsize=8)
    axes[i].set_ylabel('Price', fontsize=8)


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

Figure22- Python code for drawing scatter Diagram of all variables

At first, a box is created with the size of 15x15, for arranging 12 images in the form of regular tiles. Then, using the axes.flatten, we create a two-dimensional array to access each tile or graph. After determining the variables, Start to draw any variable's chart by for loop. Parameter col in scatterplot function refers to the column name in a data frame that shows in x-axes. Ax=axes[i], Determines which axis the graph should be placed on, and alpha defines the represented data contrast.

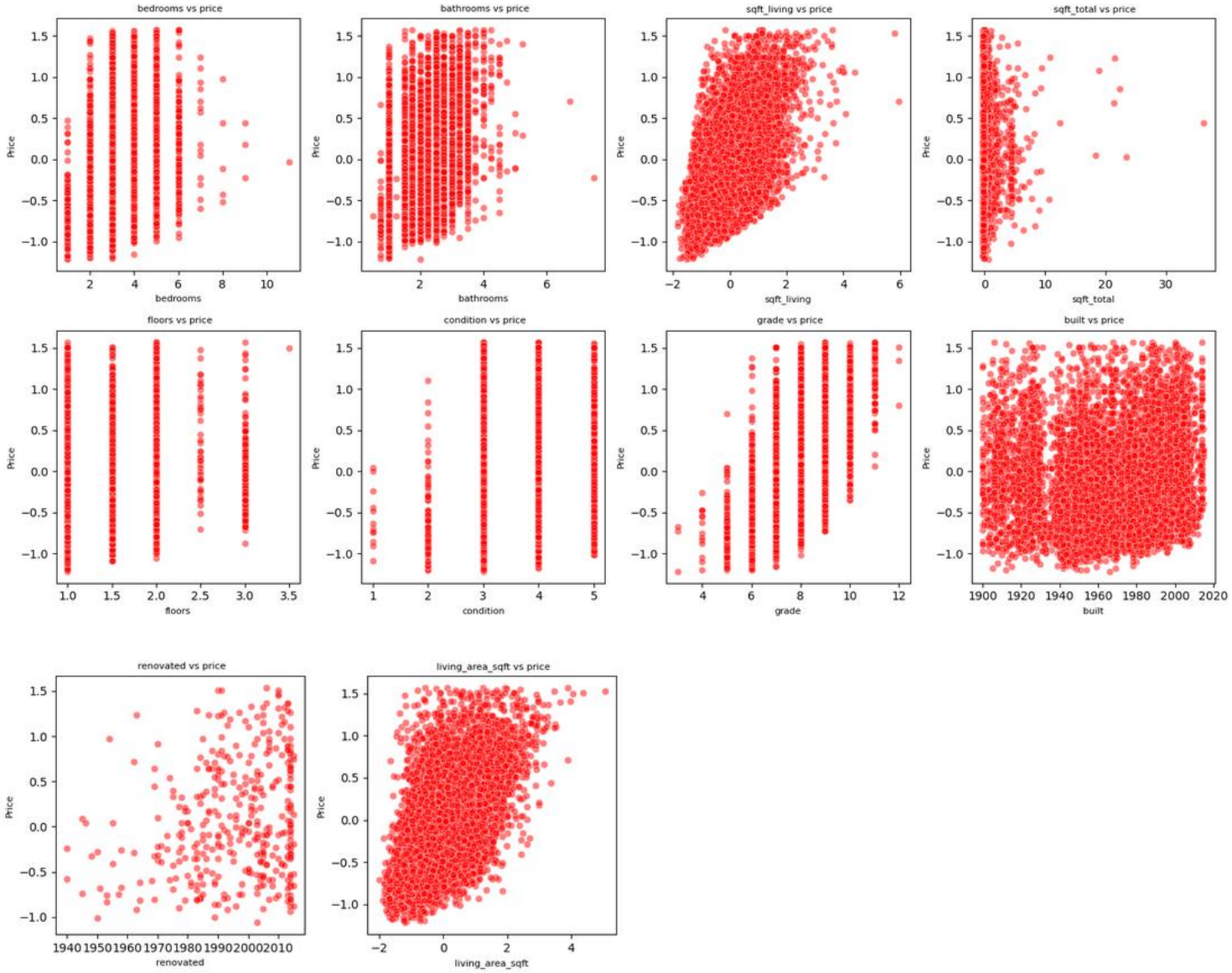

Diagram 7 - Preview of the scatterplot relationship of all Data Frame variable with Price

Price and total square foot relationship: The frequency of data is mostly between 0 and 0.25 points. It means that most of the houses have low square footage. According to the diagram, there is not even a fixed price pattern for small square footage houses and the prices are variable. While there is no linear correlation can be seen between these two variables, perhaps there are some non-linear relationships with the influence of other factors such as build, grade, and living area.

Price and grade relationship: There is a strong and positive relationship between the quality of construction and the price of the house. The higher the quality and standard construction, the higher the price.

price & square living: There is a positive relationship that the price increases as the living space increases. The extreme scatter in the data suggests that other factors also influence the price. In this chart, the outliers on the right represent large, luxurious homes with unpredictable prices. An outlier can be seen at the farthest point on the right side of the graph, which represents a house with very large dimensions and an unusual price.

price & square living area: There is a positive but very weak correlation between these two variables, which has a lot of noise. As the living space square increases, the price of houses increases, but due to the large dispersion that is observed, the number of outlier data also increases.

Step 15: Survey the relationship between square living and Square living area analysis:

```
#Square living and Square living area analysis
#Scatter plot for 'square living' vs 'Square living area'
plt.figure(figsize=(5, 4))
sns.scatterplot(x='sqft_living', y='living_area_sqft', data=data)# Drawing the scatter plot of two variables
plt.title('sqft_living vs living_area_sqft')
plt.xlabel('Square living')
plt.ylabel('square living area')
plt.show()
```

Figure 23- python code for drawing scatter diagram square living and Square living area

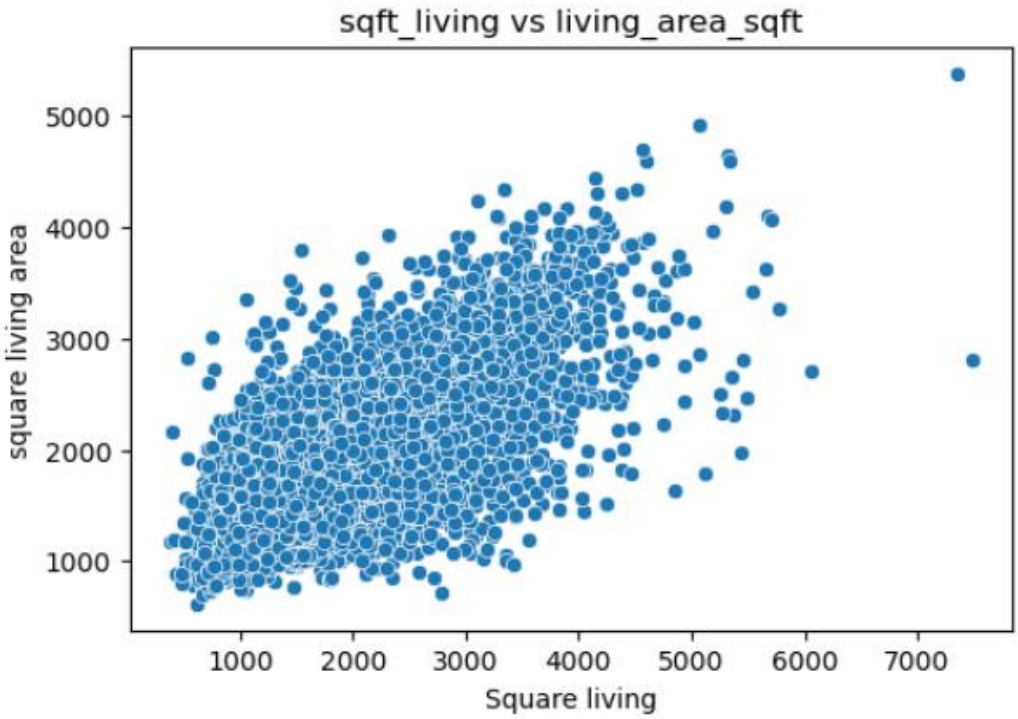

Diagram8- Scatter Histogram of square living and square living area

The scale of square living is between 500 and 7000, and the square living area is between 1000 and 5000 square feet. Most of the data is scattered between 1000 and 4000 in both variables. Meanwhile, the distribution of houses is more in the middle of the graph, which shows that most houses have an area between 2000 and 3000 square feet. On the other hand, outliers are also seen, which represent houses with big and unusual living spaces (luxury houses). According to the graph, there is a linear relationship and positive correlation between these two variables.

Step 16: Correlation

The statistical relationship and mutual effect of two factors is called correlation. So that the changes of one factor depend on the changes of another factor. Correlation is usually displayed with three numbers 0, 1, -1.

- 0: absence of correlation between two factors.
- 1: Whenever one factor increases with the increase of another factor and vice versa.
- -1: When two factors have an inverse relationship.

The price variable is a dependent variable on other factors, which has the highest correlation with sqft_total variable which refers to the total square area of the house's infrastructure. In other words, the biggest change in house price is affected by the square of the house.

```
#Attribute Correlation
```

```
# correlation matrix
correlation_matrix = data[['price','sqft_total','sqft_living', 'living_area_sqft','bedrooms', 'bathrooms','floors',
                           'condition', 'grade', 'built', 'renovated']].corr() #calculating the correlation matrix

# Visualize correlation matrix
plt.figure(figsize=(6,6)) #Adjust the size of the chart frame
sns.heatmap(correlation_matrix, annot=True, cmap='Spectral', fmt='.2f')#Drawing a correlation diagram as a heatmap
plt.title('Correlation Matrix')# adjust title for diagram
plt.show()# depicting diagram
```

Figure 24- Python code for calculating the correlation

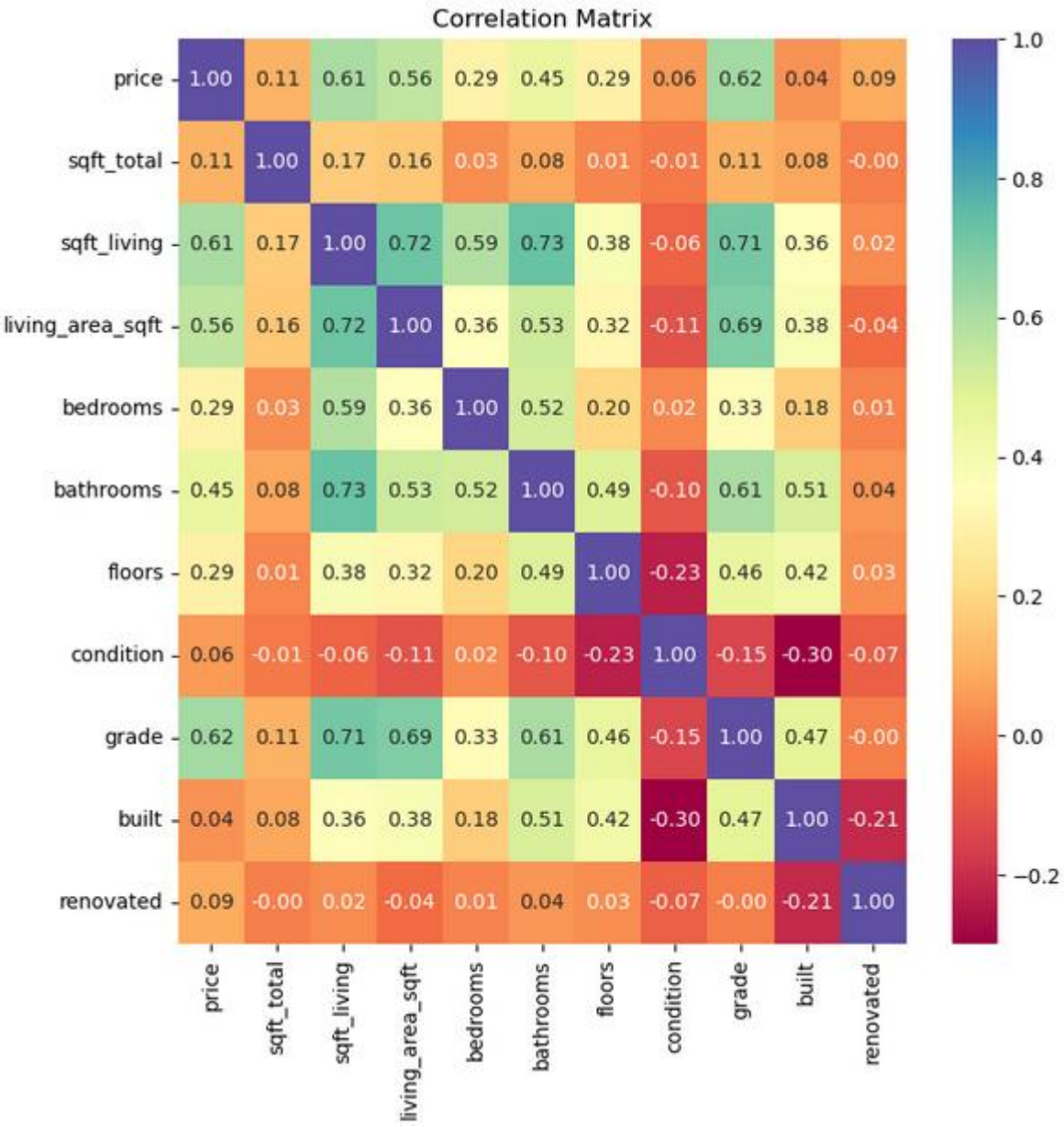

Diagram9- Heatmap diagram of the correlation between all variables

The highest correlation of price is with the four factors, grade (62%), sqft_living (61%), living_area_sqft (56%), and Bathrooms (45%), respectively. On the other hand, condition (6%), renovated (9%), and Build (4%) have the least correlation with housing prices. Meanwhile, Floor and bedroom have an equal effect on price at (0.29). On the other hand, the total square foot has a very small correlation with price. Intrestingly, sqft_living has a strong positive correlation with living_area_sqft (72%), bathroom t (73%), total square foot (71%) and grade (71%). Among all the variables, renovation has the highest number of zero correlations with variables. The most negative correlation is between the condition with build (-31%), and grade (-15%), respectively. Meanwhile, between renovated and built at(-0.21)

Step 17: Report

To conclude, analyzing Berlin housing data determines a considerable correlation and relationship between property value and other factors, such as living Area, Bedrooms, and grade, which shows welfare factors are effective. Meanwhile, a mutual and positive relevance has been detected between price and Living space, which represents the highlighted effect of The vastness of space on house price. Although three factors Build, Condition, and Renovation, have the least impact on the price of the house, buyers pay attention to these items when buying. Contrary to expectations, the Total Square factor does not affect the price of the house, because the appearance and liveable space are much more important for customers than the infrastructure of the house. Future analyses can exert more factors such as location, facilities, and amenities to predict the real estate market accurately.

Step 18: Implications and Recommendations

• More focus on the appearance and physical condition of properties to more accurately predict housing prices

• Review of geographic data such as access to the city center, public transportation, recreational and educational places, markets, etc.

• By removing the factors that do not affect housing prices, we can increase the accuracy and speed of the analysis and analyze more influencing factors.

References:

McKinney, W. (2017). *Python for data analysis* (2nd ed.). O'Reilly Media.

Date, C. J. (2003). *An introduction to database systems* (8th ed.). Pearson Education.

Kabacoff, R. I. (2011). *R in action*. Manning

Python Code:

```
import pandas as pd

import seaborn as sns

import statsmodels.api as sm

import matplotlib.pyplot as plt

import numpy as np

data = pd.read_csv('BerlinHousing4049.csv')

print(data.shape)

print(data.columns)

print(data.info())

print(data.describe())

data['built'] = pd.to_datetime(data['built'], format='%Y', errors='coerce') # pd.Datat type(Data Frame)

data['renovated'] = pd.to_datetime(data['renovated'], format='%Y', errors='coerce')

data.to_csv("BerlinHousing4049.csv", index=False)

data.to_csv("BerlinHousing4049_clean_sckwedness.csv", index=False)

data.info(5)

dupli_count = data.duplicated().sum()

data_cle = data.drop_duplicates()

print(f"\nNumber of Duplicate Rows Removed: {dupli_count}")

data_cle.to_csv("BerlinHousing4049.csv", index=False)

data.to_csv("BerlinHousing4049_clean_sckwedness.csv", index=False)

df= pd.DataFrame(data)

missed_data= df.isnull().sum()

print("Number of Miseed Data:     ", missed_data)

df_fill=df.fillna(df.median(numeric_only=True), inplace=True)

data_cle.to_csv("BerlinHousing4049.csv", index=False)

data_cle.to_csv("BerlinHousing4049_clean_sckwedness.csv", index=False)

print(df_fill)

data['price_per_sqft'] =(data['price'] / data['sqft_total']).round(3)

print(data.head())
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

data[['price','sqft_total','sqft_living', 'living_area_sqft']] = scaler.fit_transform(data[['price','sqft_total','sqft_living',
                                                                                            'living_area_sqft']])

data.to_csv("BerlinHousing4049csv", index=False)

print(data.head())

plt.figure(figsize=(8, 5)) #Adjust the size of the chart frame

sns.boxplot(x=data['price'], color='red', showfliers=True)

min_val = data['price'].min()

max_val = data['price'].max()

median_val = data['price'].median()

mean_val = data['price'].mean()

plt.axvline(mean_val, color='orange', linestyle='-.', label=f'Mean: {mean_val:.2f}')

plt.axvline(median_val, color='green', linestyle='-', label=f'Median: {median_val:.2f}')

plt.axvline(min_val, color='blue', linestyle='-', label=f'Min: {min_val:.2f}')

plt.axvline(max_val, color='purple', linestyle='-', label=f'Max: {max_val:.2f}')

plt.title('Boxplot of Price' , fontsize=10)

plt.xlabel('Price')

plt.legend(loc='upper right')

plt.tight_layout()

plt.show()

data.head(5)

Q1_price = data['price'].quantile(0.25)

Q3_price = data['price'].quantile(0.75)

IQR_price = Q3_price - Q1_price

lower_bound_price = Q1_price - 1.5 * IQR_price

upper_bound_price = Q3_price + 1.5 * IQR_price
```

```
data = data[(data['price'] >= lower_bound_price) & (data['price'] <= upper_bound_price)]
print("Number of rows after removing outliers:", len(data))
plt.figure(figsize=(8, 5))
sns.boxplot(x=data['price'], color='red', showfliers=True)
min_val = data['price'].min()
max_val = data['price'].max()
median_val = data['price'].median()
mean_val = data['price'].mean()
plt.axvline(mean_val, color='orange', linestyle='-.', label=f'Mean: {mean_val:.2f}')
plt.axvline(median_val, color='green', linestyle='-', label=f'Median: {median_val:.2f}')
plt.axvline(min_val, color='blue', linestyle='-', label=f'Min: {min_val:.2f}')
plt.axvline(max_val, color='purple', linestyle='-', label=f'Max: {max_val:.2f}')
plt.title('Boxplot of Price' , fontsize=10)
plt.xlabel('Price')
plt.legend(loc='upper right')
plt.tight_layout()
data_cle.to_csv("BerlinHousing4049.csv", index=False)
data_cle.to_csv("BerlinHousing4049_clean_sckwedness.csv", index=False)
plt.show()
print(data.describe(include='all'))
grouped_data = data.groupby(['price']).mean()
plt.figure(figsize=(8, 2))
colors = ['skyblue', 'lightgreen', 'salmon', 'orange', 'MediumPurple']
print(grouped_data.head(20))
data['price'].value_counts().head(20).plot(kind='bar', figsize=(6,4), color=colors , rot=85)
plt.title('Price grouping bar chart',fontsize=10)
plt.ylabel('frequently')
plt.show()
```

```
df=pd.read_csv("BerlinHousing4049_clean_sckwedness.csv")
df_clean =df['price'].dropna()
df_clean = df_clean[df_clean > 0]
plt.figure(figsize=(10,6))
sns.histplot((df_clean), kde=True, bins=50, color="skyblue", edgecolor="black")
plt.title('Transformed Price Distribution (Log)', fontsize=14)
plt.xlabel('log(Price)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.tight_layout()
plt.show()
df=pd.read_csv("BerlinHousing4049_clean_sckwedness.csv")
df_clean = df['price'].dropna()
df_clean = df_clean[df_clean > 0]
plt.figure(figsize=(10,6))
sns.histplot(np.log(df_clean), kde=True, bins=50, color="skyblue", edgecolor="black")
plt.title('Transformed Price Distribution (Log)', fontsize=14)
plt.xlabel('log(Price)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.tight_layout()
plt.show()
fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(15, 15))
axes = axes.flatten()
variables = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_total', 'floors',
        'condition', 'grade', 'built', 'renovated', 'living_area_sqft']
for i, col in enumerate(variables):
    sns.scatterplot(data, x=col, y='price', ax=axes[i], color='red', alpha=0.5)
    axes[i].set_title(f' {col} vs price', fontsize=8)
    axes[i].set_xlabel(col, fontsize=8)
    axes[i].set_ylabel('Price', fontsize=8)
plt.figure(figsize=(5, 4))
```

```
sns.scatterplot(x='sqft_living', y='living_area_sqft', data=data)
plt.title('sqft_living vs living_area_sqft')
plt.xlabel('Square living')
plt.ylabel('square living area')
plt.show()
plt.tight_layout()
plt.show()
correlation_matrix = data[['price','sqft_total','sqft_living', 'living_area_sqft','bedrooms', 'bathrooms','floors',
                    'condition', 'grade', 'built', 'renovated']].corr()
plt.figure(figsize=(8,8))
sns.heatmap(correlation_matrix, annot=True, cmap='Spectral', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```