MACHINE LEARNING PROJECT

**REPORT AND ANALYSIS**

GROUP:

A.CHARITHA REDDY : SE22UCSE016

ABHIGNA KAPPAGANTULA : SE22UCSE003

K.RITHWIKA : SE22UCSE121

SRUTHI LIGAMPALLI : SE22UCSE150

We start by importing all the necessary libraries:

# importing libraries

import pandas as pd

import numpy as np

import matplotlib

import matplotlib.pyplot as plot

import seaborn as sns

from sklearn.preprocessing import OneHotEncoder

from scipy.stats import norm

from scipy.stats.mstats import winsorize

from sklearn.impute import SimpleImputer, KNNImputer

from scipy.stats import zscore

from scipy import stats

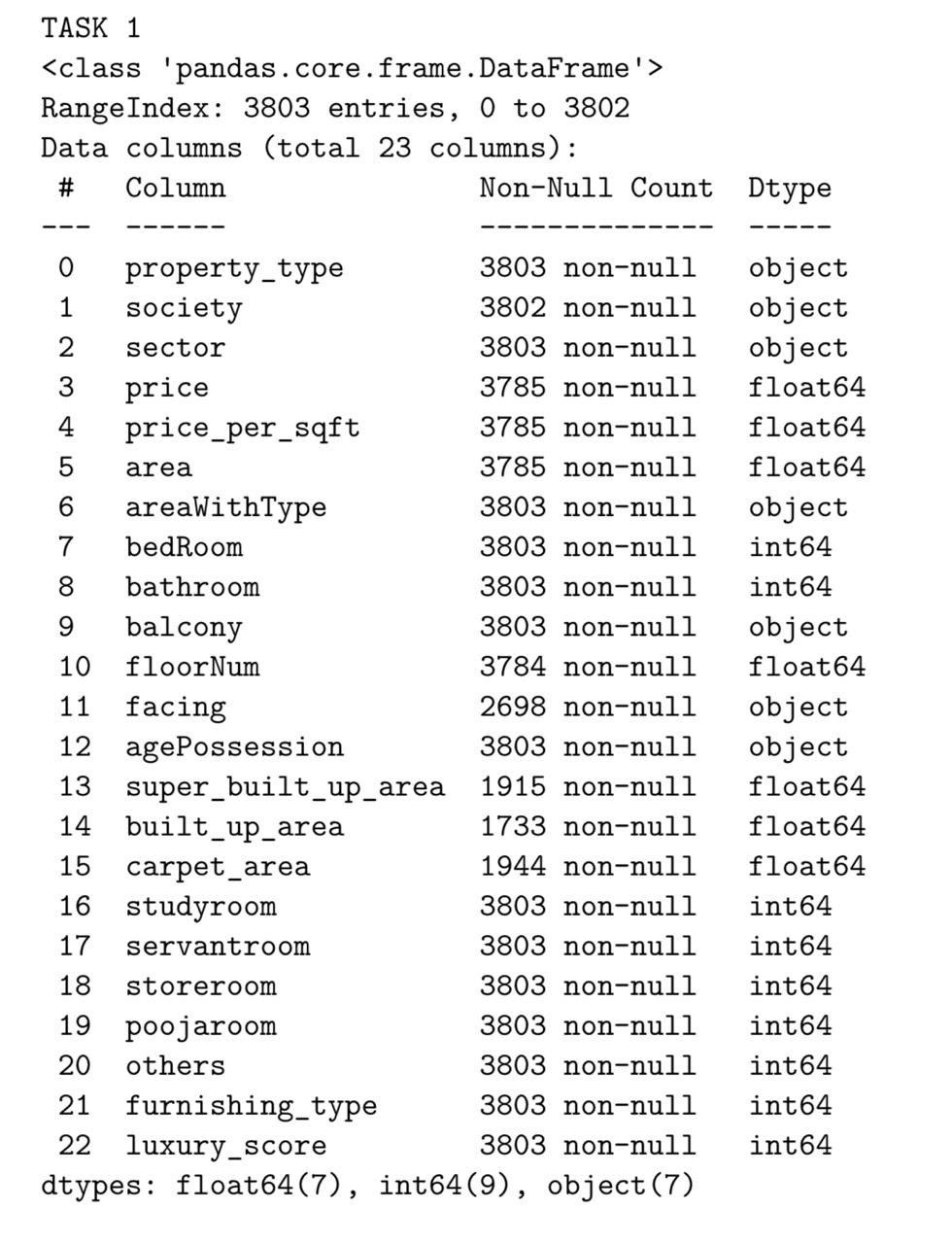
**TASK 1:**

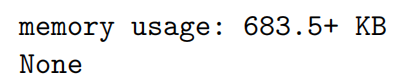
#loading dataset into python

abc=pd.read\_csv("C:\\Users\\bhara\\Downloads\\Gurgaon\_RealEstate.csv")

#to identify all features and datatypes

print(abc.info())





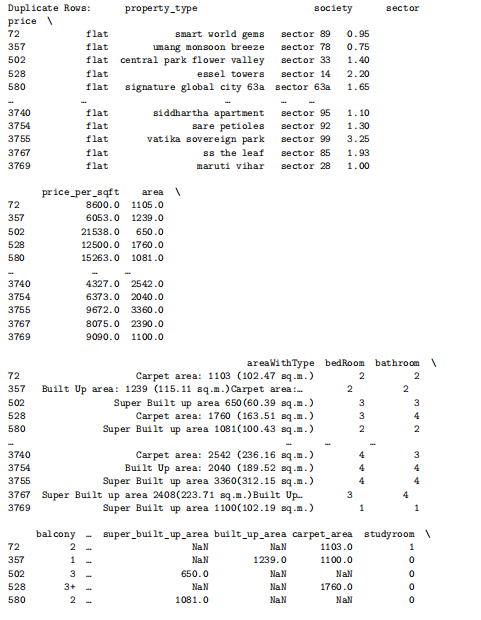
After loading all the necessary libraries, we check for duplicate rows:

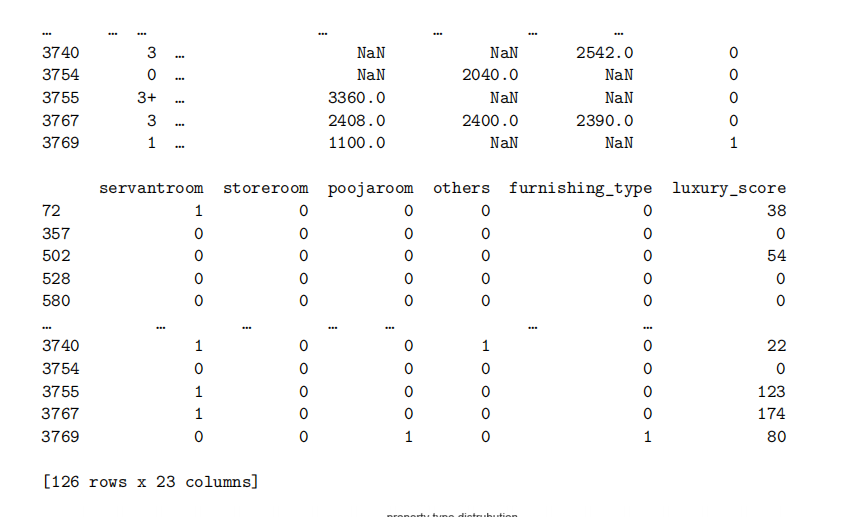
#to check if there is any duplicate row

DUPLICATEE\_ROWSS=abc[abc.duplicated()]

#to print duplicate rows.

print("Duplicate Rows:",DUPLICATEE\_ROWSS)





#to remove duplicate rows

abc=abc.drop\_duplicates()

we now start exploring each column of our dataset:

**EXPLORING property\_type COLUMN:**

#to explore property\_type column(flat or house)

plot.figure(figsize=(12,8))

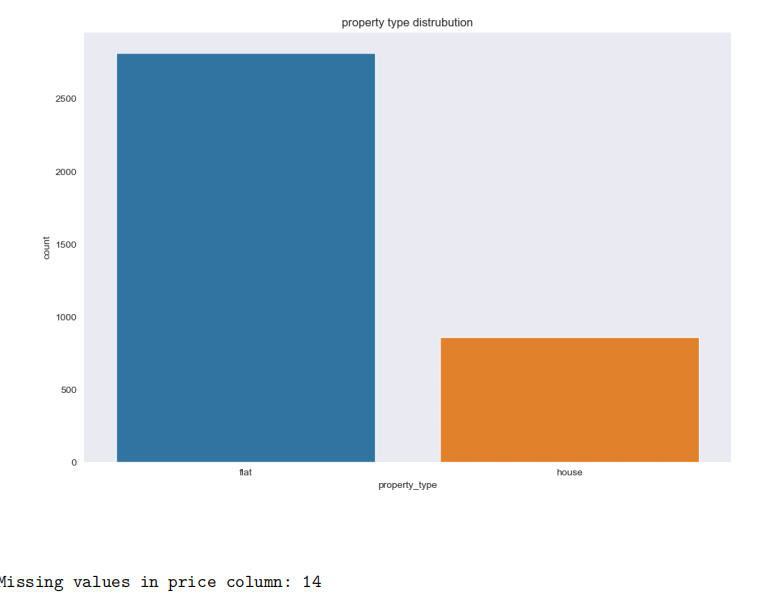
sns.countplot(x='property\_type',data=abc)

plot.title('property type distrubution')

plot.xlabel('property\_type')

plot.ylabel('count')

plot.show()



# to explore society column

society\_counts = abc['society'].value\_counts()

# Define a threshold for the minimum number of flats or houses per society

threshold = 6

valid\_societies = society\_counts[society\_counts >= threshold].index

abc = abc[abc['society'].isin(valid\_societies)]

**EXPLORING PRICE COLUMN :**

# to explore price column

print("Missing values in price column:", abc['price'].isnull().sum())

print("Descriptive statistics for price column:\n", abc['price'].describe())

plot.figure(figsize=(12,8))

sns.histplot(abc['price'], bins=20, kde=True)

plot.title('Price Distribution')

plot.xlabel('Price')

plot.ylabel('Frequency')

plot.show()

plot.figure(figsize=(12,8))

sns.boxplot(x=abc['price'])

plot.title('Boxplot of Price')

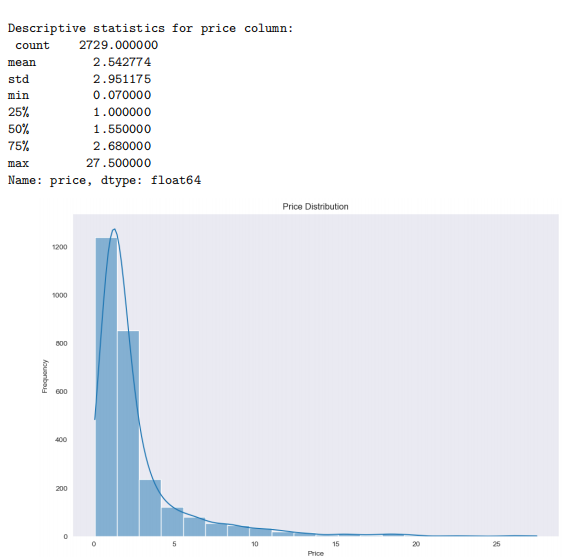
plot.xlabel('Price')

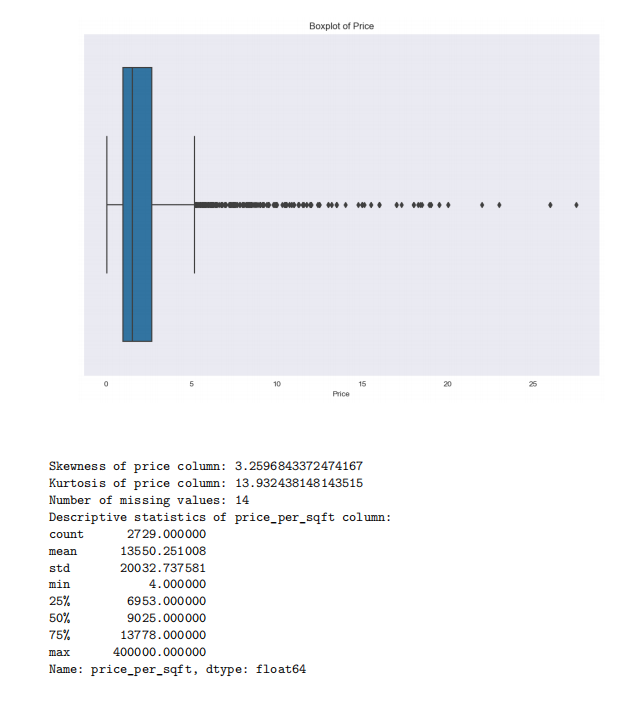
plot.show()

print("Skewness of price column:", abc['price'].skew())

print("Kurtosis of price column:", abc['price'].kurt())







**EXPLORING PRICE\_PER\_SQFT COLUMN :**

# to explore price\_per\_sqft

missing\_values = abc['price\_per\_sqft'].isnull().sum()

print("Number of missing values:", missing\_values)

print("Descriptive statistics of price\_per\_sqft column:")

print(abc['price\_per\_sqft'].describe())

#hostogram for price\_per\_sqft

plot.figure(figsize=(12,8))

sns.histplot(abc['price\_per\_sqft'], bins=20, kde=True, color='blue', edgecolor='black')

plot.xlabel('Price per Square Foot')

plot.ylabel('Frequency')

plot.title('Histogram of Price per Square Foot')

plot.show()

#boxplot for outliers of price\_per\_sqft

plot.figure(figsize=(12,8))

sns.boxplot(x=abc['price\_per\_sqft'])

plot.xlabel('Price per Square Foot')

plot.title('Box Plot of Price per Square Foot')

plot.show()

#to check skewness and kurtosis

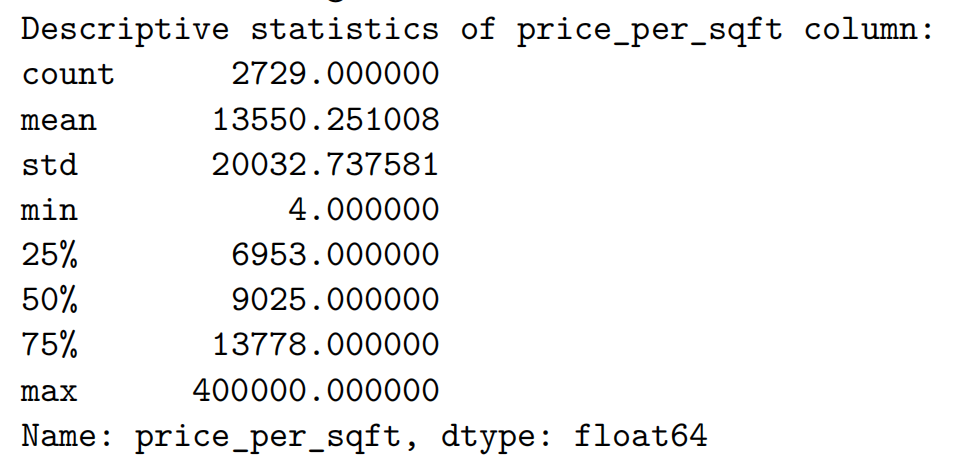
skewness = abc['price\_per\_sqft'].skew()

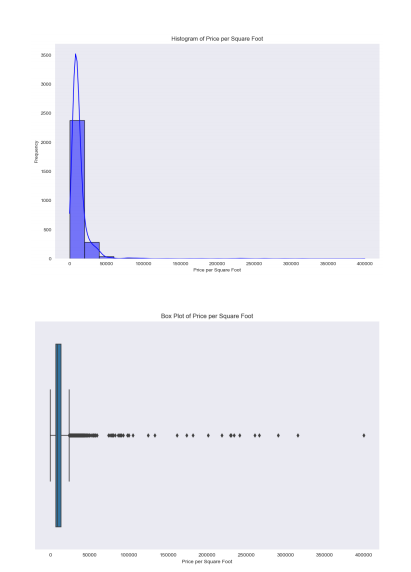
kurtosis = abc['price\_per\_sqft'].kurtosis()

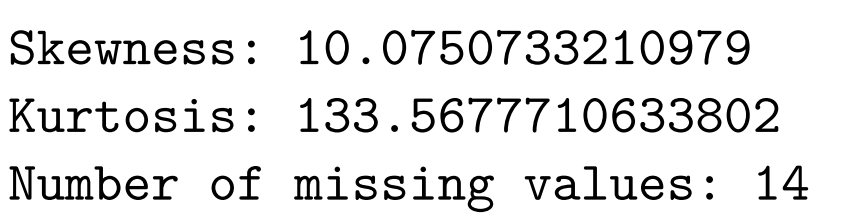
print("Skewness:",skewness)

print("Kurtosis:",kurtosis)









**EXPLORING AREA COLUMN:**

# to explore area column

missing\_values = abc['area'].isnull().sum()

print("Number of missing values:", missing\_values)

print("Descriptive statistics of area column:")

print(abc['area'].describe())

#hostogram for area

plot.figure(figsize=(12,8))

sns.histplot(abc['area'], bins=20, kde=True, color='blue', edgecolor='black')

plot.xlabel('area')

plot.ylabel('Frequency')

plot.title('Histogram of area')

plot.show()

#boxplot for outliers of area

plot.figure(figsize=(12,8))

sns.boxplot(x=abc['area'])

plot.xlabel('area')

plot.title('Box Plot of area')

plot.show()

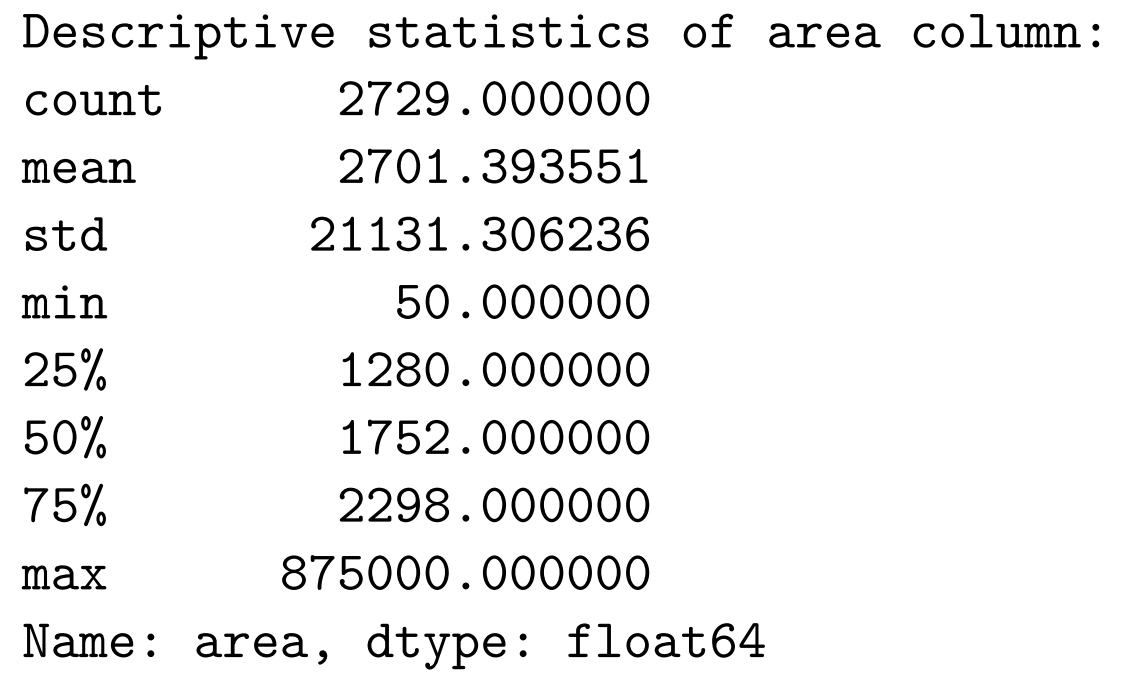
#to check skewness and kurtosis

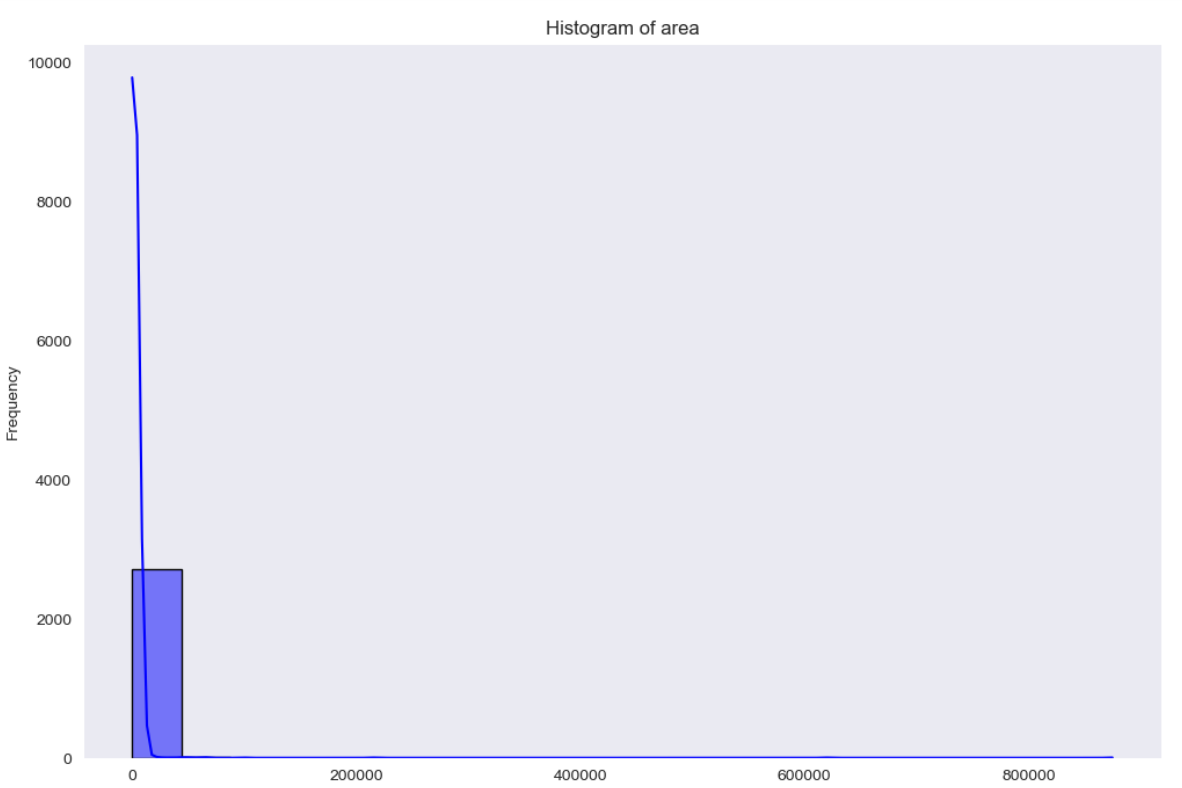
skewness = abc['area'].skew()

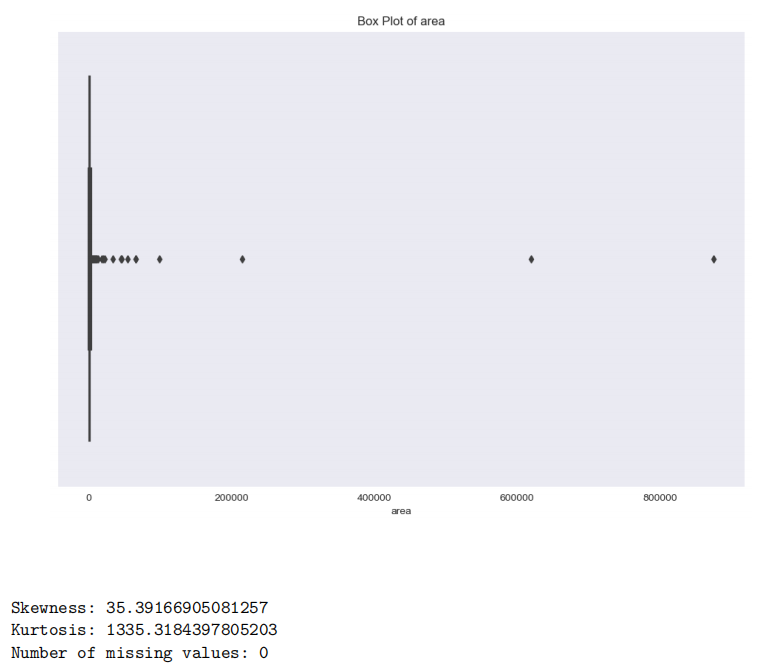
kurtosis = abc['area'].kurtosis()

print("Skewness:",skewness)

print("Kurtosis:",kurtosis)







**EXPLORING BEDROOM COLUMN:**

# to explore bedroom

missing\_values = abc['bedRoom'].isnull().sum()

print("Number of missing values:", missing\_values)

print("Descriptive statistics of bedRoom column:")

print(abc['bedRoom'].describe())

#hostogram for bedroom

plot.figure(figsize=(12,8))

sns.histplot(abc['bedRoom'], bins=20, kde=True, color='blue', edgecolor='black')

plot.xlabel('bedRoom')

plot.ylabel('Frequency')

plot.title('Histogram of bedRoom')

plot.show()

#boxplot for outliers of bedroom

plot.figure(figsize=(12,8))

sns.boxplot(x=abc['bedRoom'])

plot.xlabel('bedRoom')

plot.title('Box Plot of bedRoom')

plot.show()

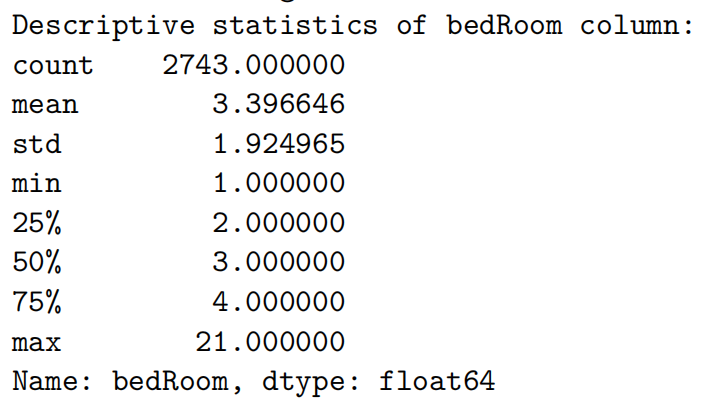
#to check skewness and kurtosis

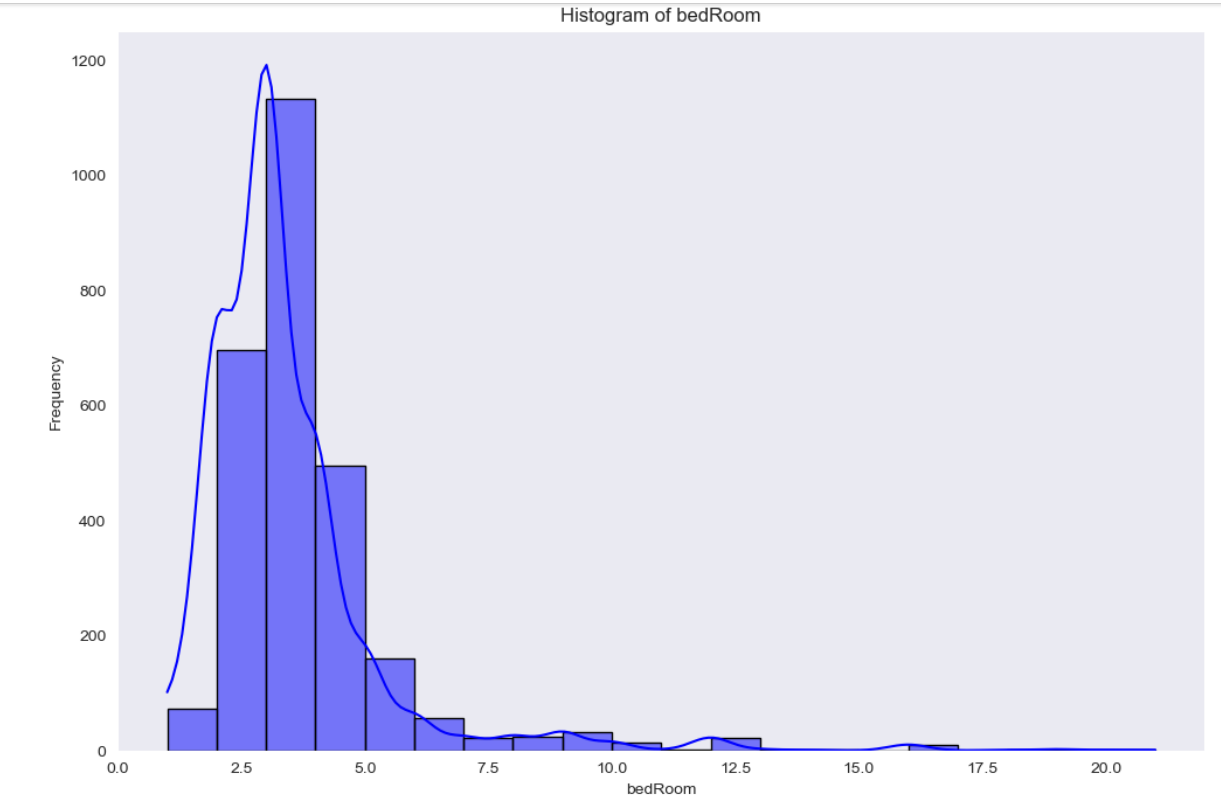
skewness = abc['bedRoom'].skew()

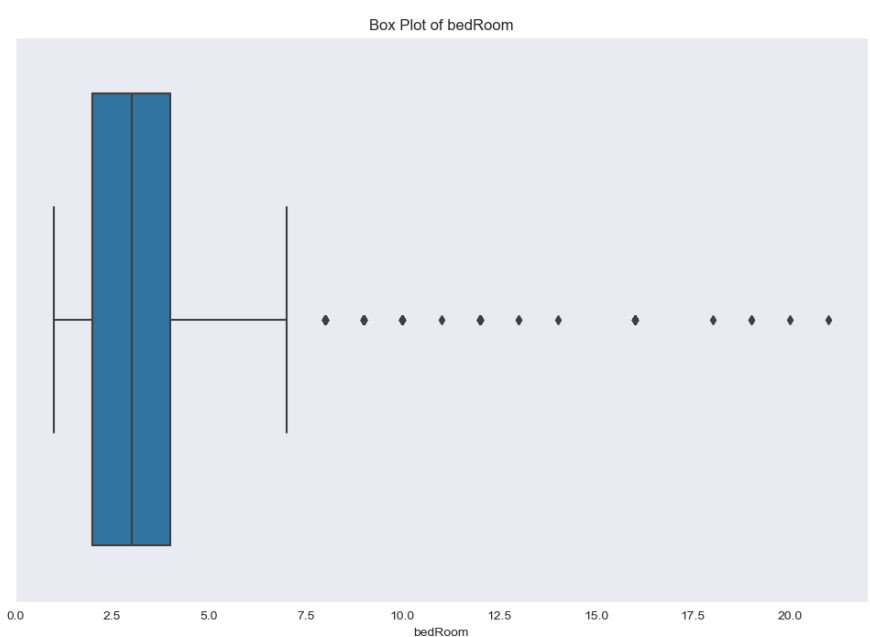
kurtosis = abc['bedRoom'].kurtosis()

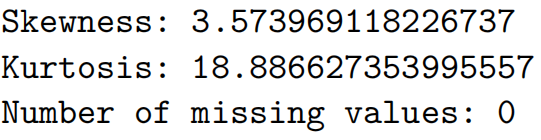
print("Skewness:",skewness)

print("Kurtosis:",kurtosis)









EXPLORING BATHROOM COLUMN:

# to explore bathroom

missing\_values = abc['bathroom'].isnull().sum()

print("Number of missing values:", missing\_values)

print("Descriptive statistics of bathroom column:")

print(abc['bathroom'].describe())

#hostogram for bathroom

plot.figure(figsize=(12,8))

sns.histplot(abc['bathroom'], bins=20, kde=True, color='blue', edgecolor='black')

plot.xlabel('bathroom')

plot.ylabel('Frequency')

plot.title('Histogram of bathroom')

plot.show()

#boxplot for outliers of bathroom

plot.figure(figsize=(12,8))

sns.boxplot(x=abc['bathroom'])

plot.xlabel('bathroom')

plot.title('Box Plot of bathroom')

plot.show()

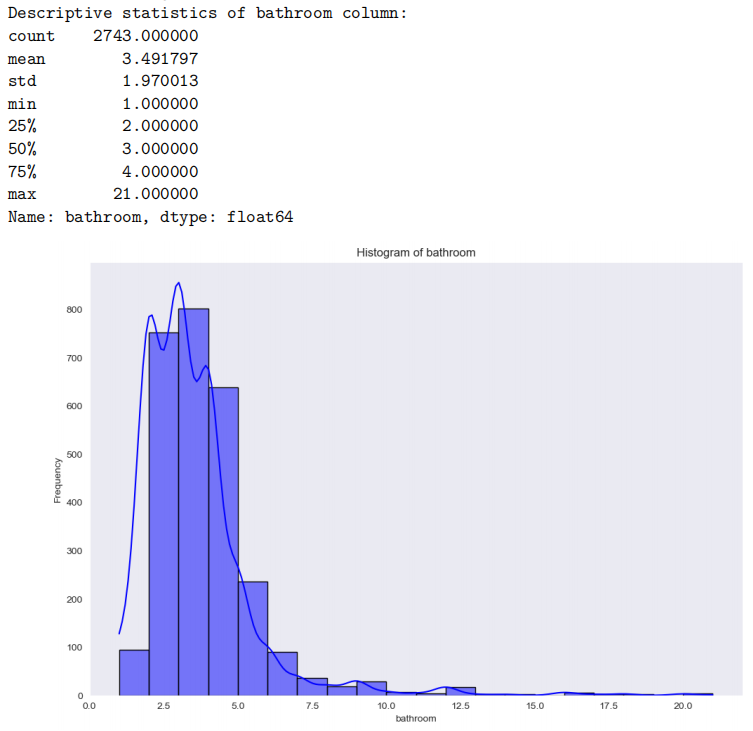
#to check skewness and kurtosis

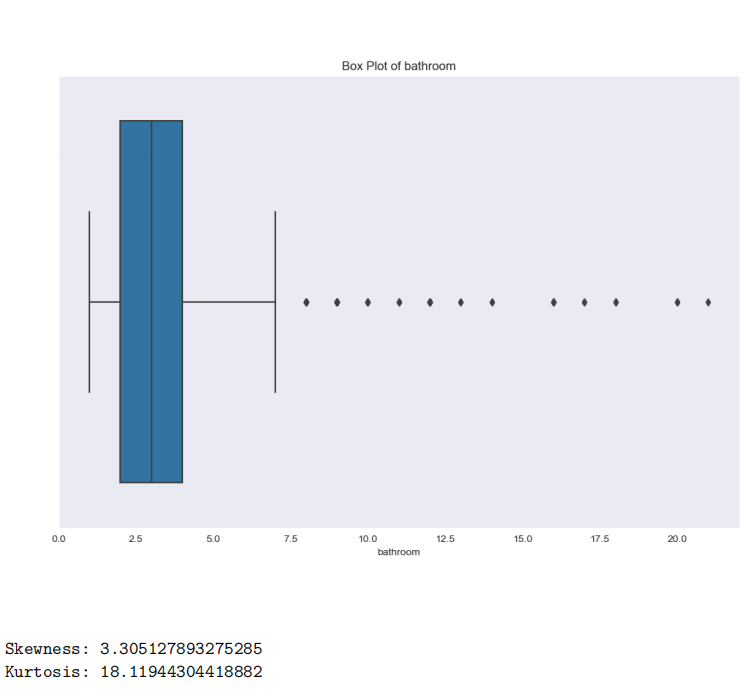
skewness = abc['bathroom'].skew()

kurtosis = abc['bathroom'].kurtosis()

print("Skewness:",skewness)

print("Kurtosis:",kurtosis)





MULTIVARIATE ANALYSIS OF ALL COLUMNS VS TARGET COLUMN :

**PROPERTY TYPE VS PRICE :**

# Property type vs price

plot.figure(figsize=(12,8))

sns.boxplot(x='property\_type', y='price', data=abc)

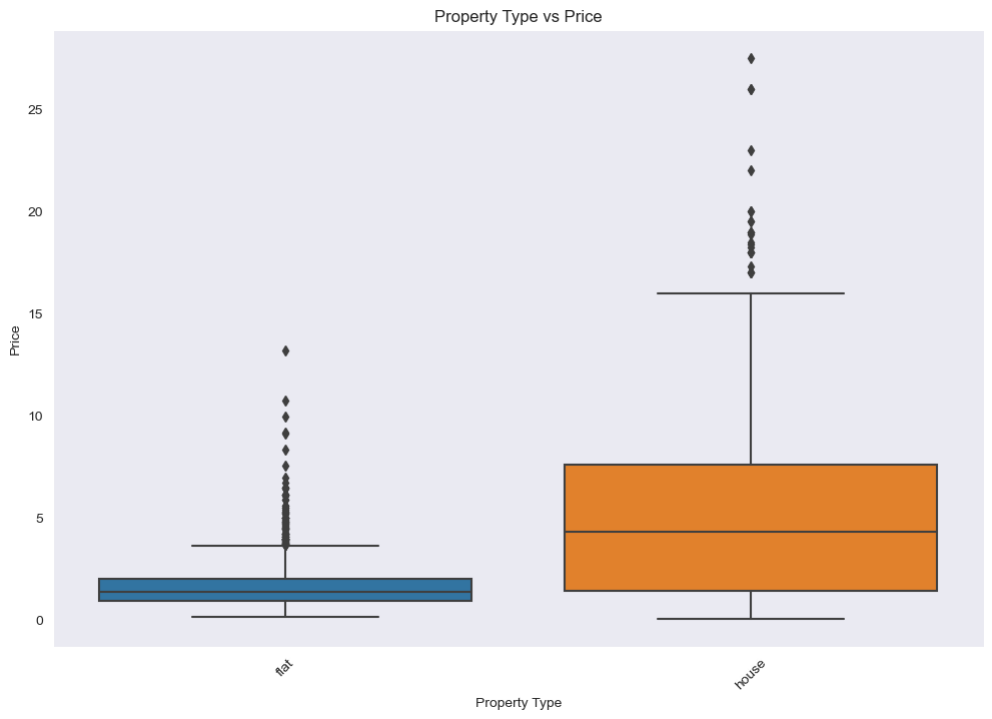
plot.title('Property Type vs Price')

plot.xlabel('Property Type')

plot.ylabel('Price')

plot.xticks(rotation=45)

plot.show()



PRICE vs AREA:

# Scatter plot between price and area

plot.figure(figsize=(12,8))

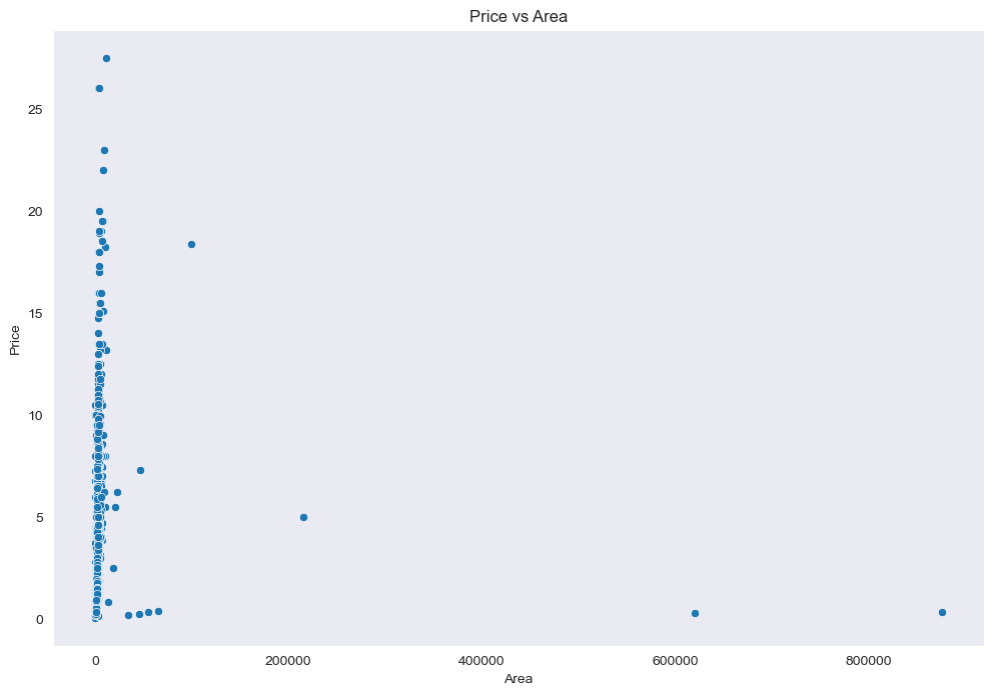
sns.scatterplot(x='area', y='price', data=abc)

plot.title('Price vs Area')

plot.xlabel('Area')

plot.ylabel('Price')

plot.show()



TASK 2:

MISSING VALUES HANDLING:

#MISSING VALUE HANDLING

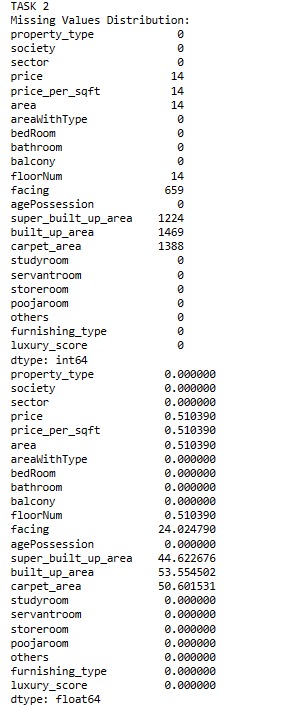
missval = abc.isnull().sum()

missdist = (missval / len(abc)) \* 100

print("Missing Values Distribution:")

print(missval)

print(missdist)



# FILLING NULL VALUES WITH MEAN OF A COLUMN

abcd=abc.fillna(value=abc['carpet\_area'].mean())

abcd=abc.fillna(value=abc['built\_up\_area'].mean())

abcd=abc.fillna(value=abc['super\_built\_up\_area'].mean())

abcd=abc.fillna(value=abc['price'].mean())

abcd=abc.fillna(value=abc['price\_per\_sqft'].mean())

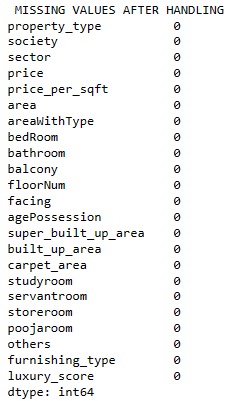
#HANDLING MISSING VALUES OF CATEGORICAL COLUMNS USING MODE:

abcd["society"].replace(np.NaN, abcd["society"].mode()[0], inplace=True)

abcd["facing"].replace(np.NaN, abcd["facing"].mode()[0], inplace=True)

print("\n MISSING VALUES AFTER HANDLING:")

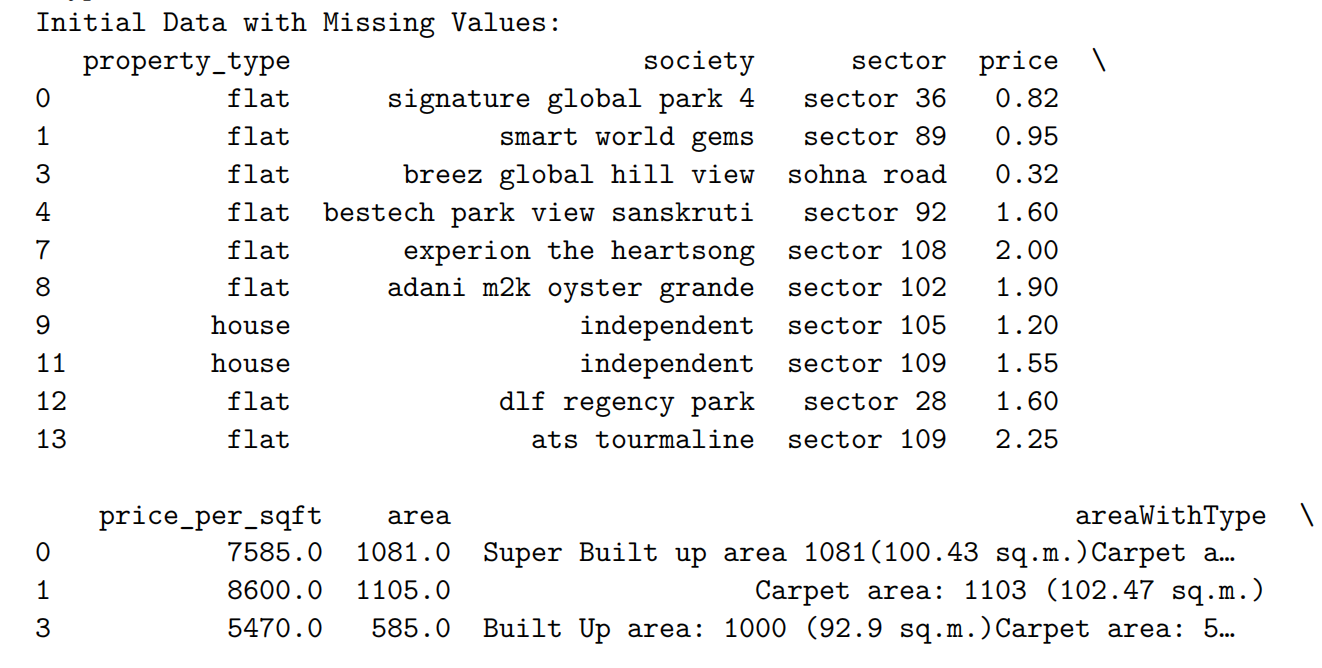
print(abcd.isnull().sum())

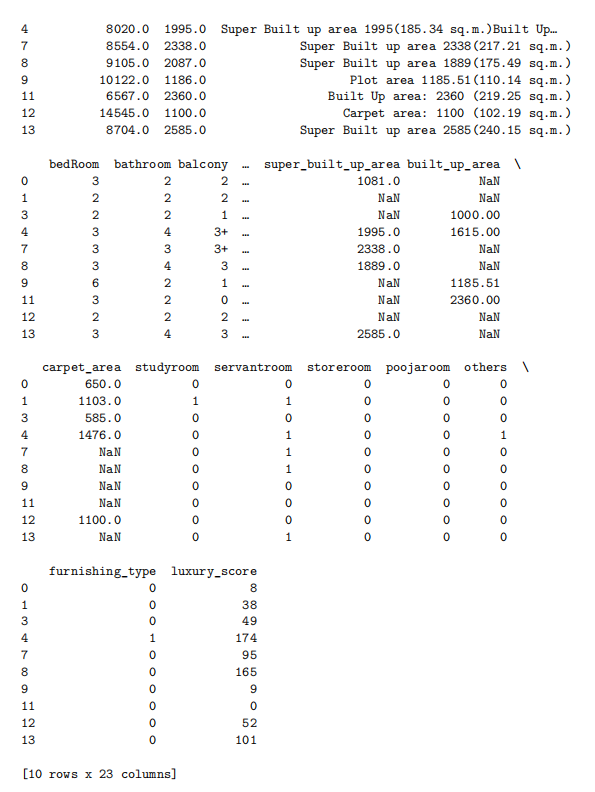


# Display initial data with missing values

print("Initial Data with Missing Values:")

print(abc.head(10))



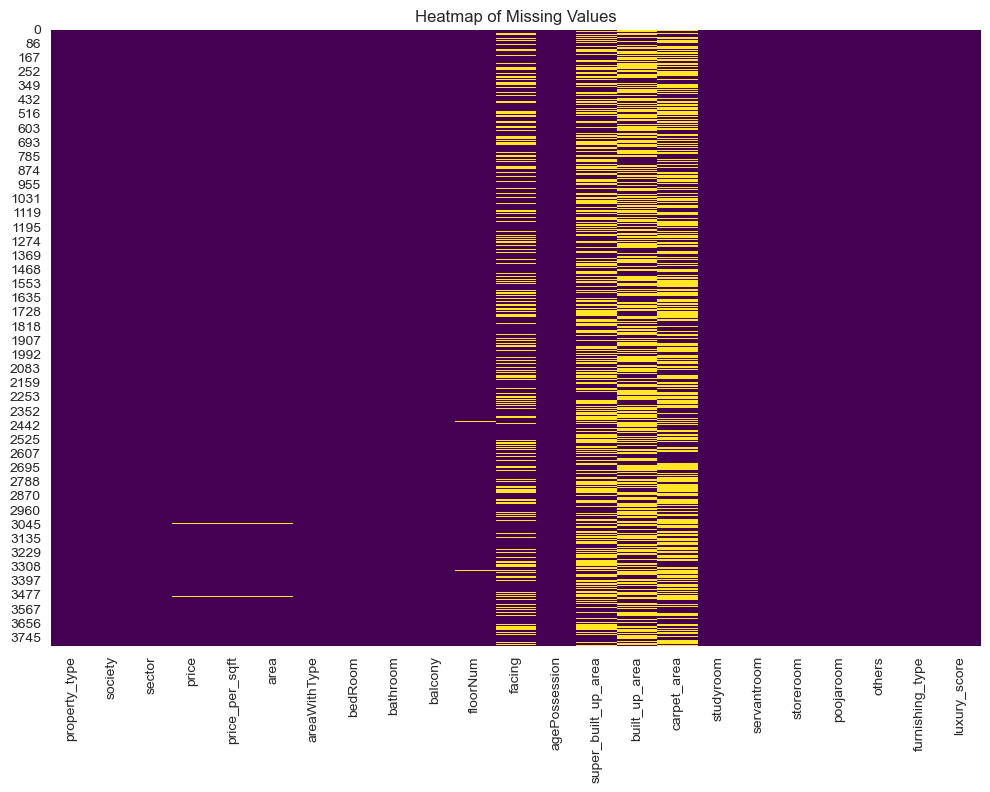


# Visualize missing data

sns.heatmap(abc.isnull(), cbar=False, cmap="viridis")

plot.title('Heatmap of Missing Values')

plot.show()



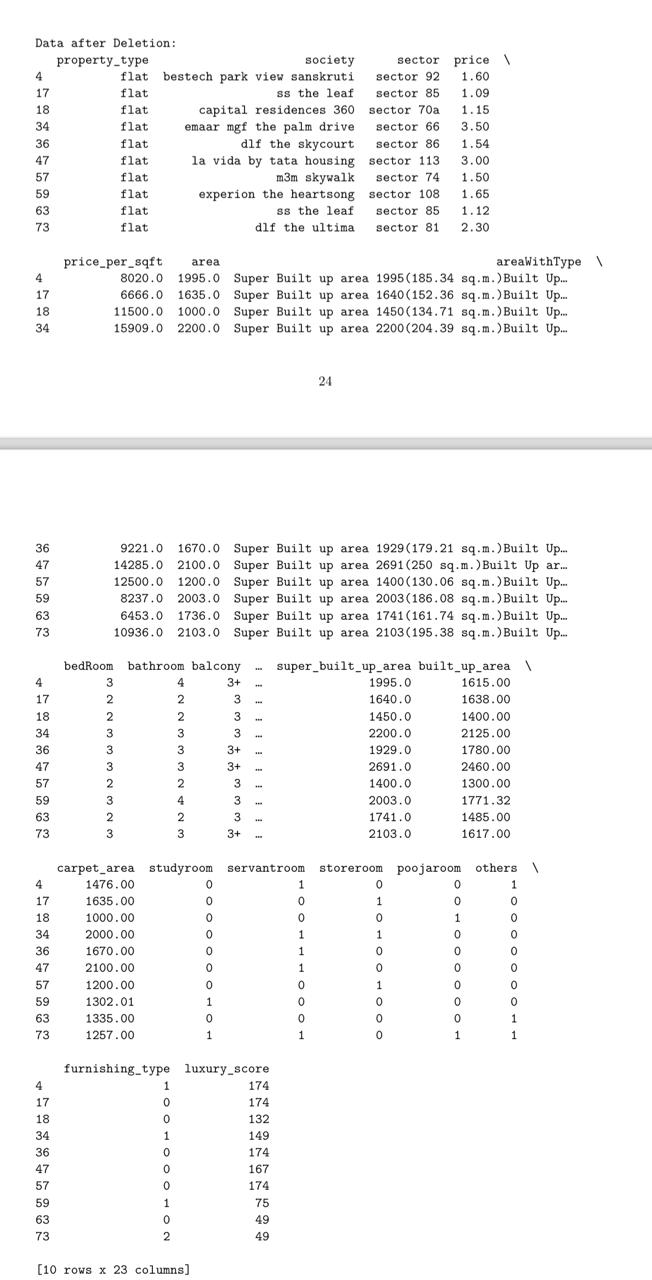
The strategies that we used for handling missing values are – deletion, mean, mode and we tried one hot encoding (to convert categorical data to numerical).

#Deletion

abc\_deletion = abc.dropna()

print("\nData after Deletion:")

print(abc\_deletion.head(10))



We initially tried OneHotEncoding to convert categorical data to numerical data :

#ONEHOT ENCODING

# from sklearn.preprocessing import OneHotEncoder

# abcd.tail()

# abcd.dtypes

# abcd["society"].unique()

# abcd["facing"].unique()

# ohe = OneHotEncoder()

# ohe.fit\_transform(abcd[["society","facing"]]).toarray()

# featurearr = ohe.fit\_transform(abcd[["society","facing"]]).toarray()

# featlabels = ohe.get\_feature\_names\_out(["society","facing"])

# np.array(featlabels).ravel()

# featlabels = np.array(featlabels).ravel()

# print(featlabels)

# pd.dataframe(featurearr, columns = featlabels)

# features = pd.dataframe(featurearr, columns = featlabels)

# print(features)

# pd.concat([abc, features], axis=1)

# abcnew = pd.concat([abc, features], axis=1)

**TASK 3:**

OUTLIERS DETECTION:

We used IQR method and Z-Score method based on the distribution of data of that particular column, z- score if the distribution is normal and IQR if otherwise.

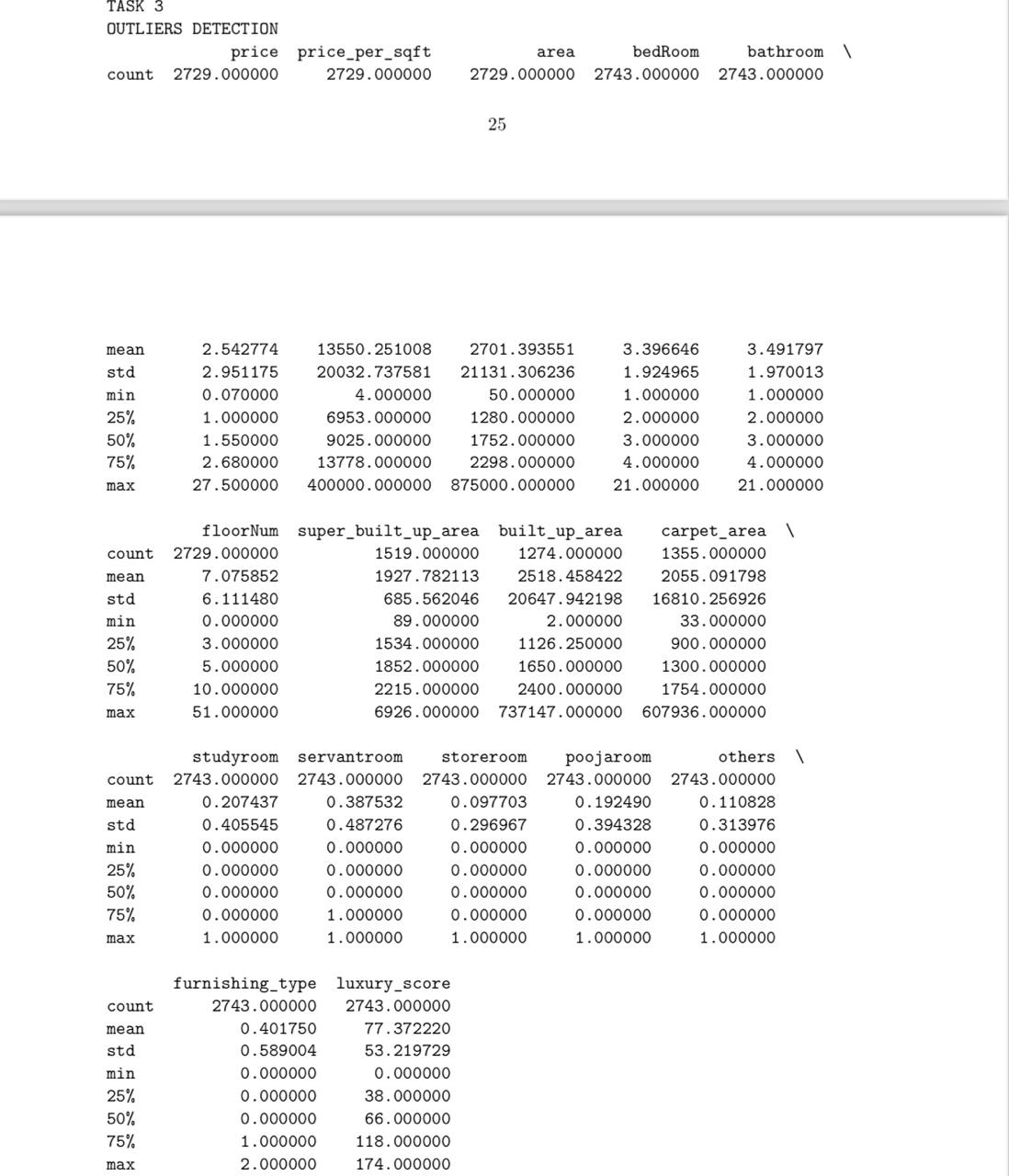
print("OUTLIERS DETECTION")

# Select only numerical columns

numerical\_abc = abc.select\_dtypes(include=[np.number])

# Display descriptive statistics

print(numerical\_abc.describe())



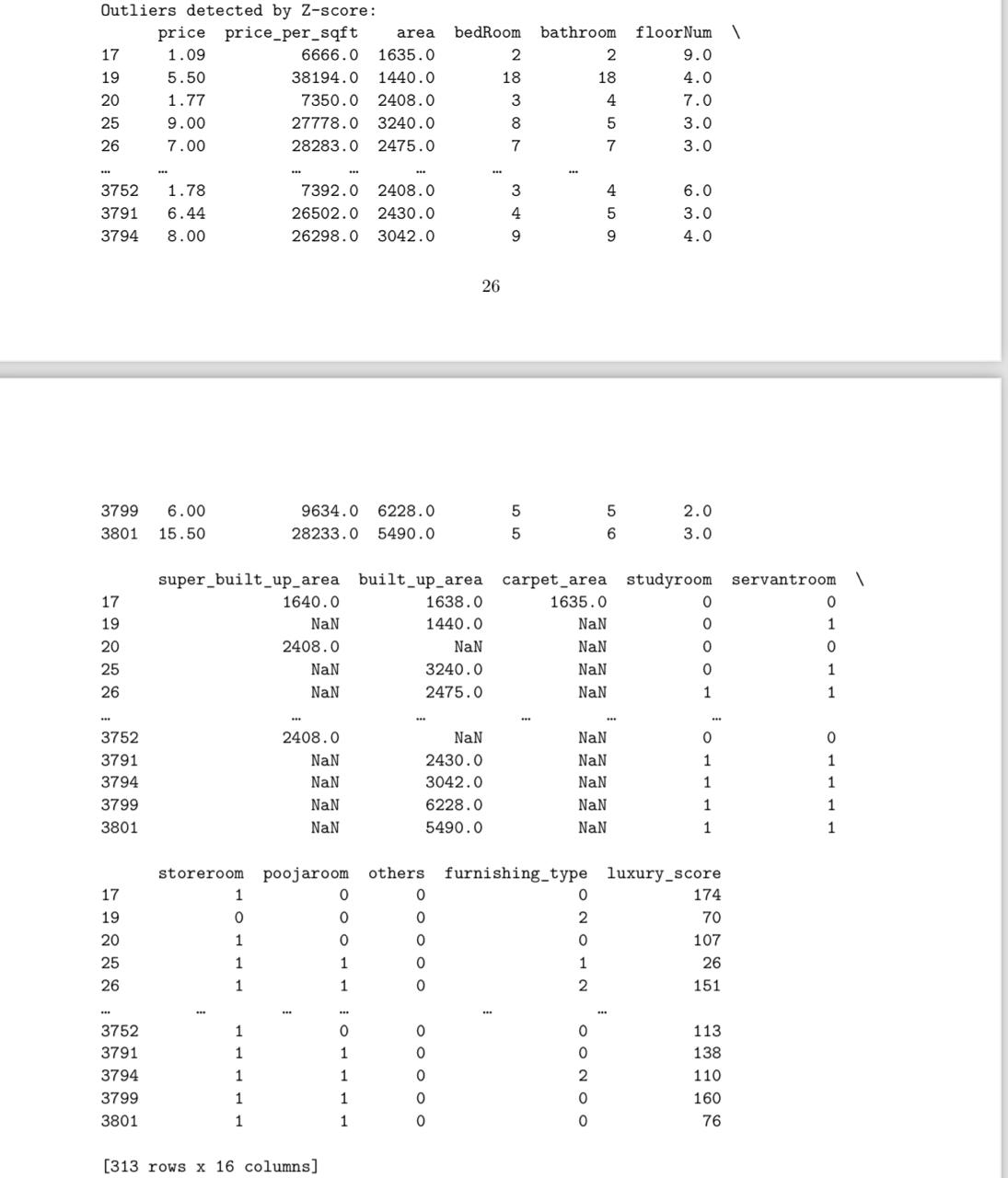
Z-Score Method:

# Z-score method

z\_score = np.abs(stats.zscore(numerical\_abc))

outlier\_z = (z\_score > 3).any(axis=1)

print(f"Outliers detected by Z-score:\n{numerical\_abc[outlier\_z]}")



IQR Method:

# IQR method

Q1 = numerical\_abc.quantile(0.25)

Q3 = numerical\_abc.quantile(0.75)

IQR = Q3 - Q1

outlier\_iqr = ((numerical\_abc < (Q1 - 1.5 \* IQR)) | (numerical\_abc > (Q3 + 1.5 \* IQR))).any(axis=1)

print(f"Outliers detected by IQR:\n{numerical\_abc[outlier\_iqr]}")

# Visualization of distributions and outliers

fig, ax = plot.subplots(len(numerical\_abc.columns), 2, figsize=(16, 4 \* len(numerical\_abc.columns)))

for i, feature in enumerate(numerical\_abc.columns):

 sns.histplot(numerical\_abc[feature], kde=True, ax=ax[i, 0])

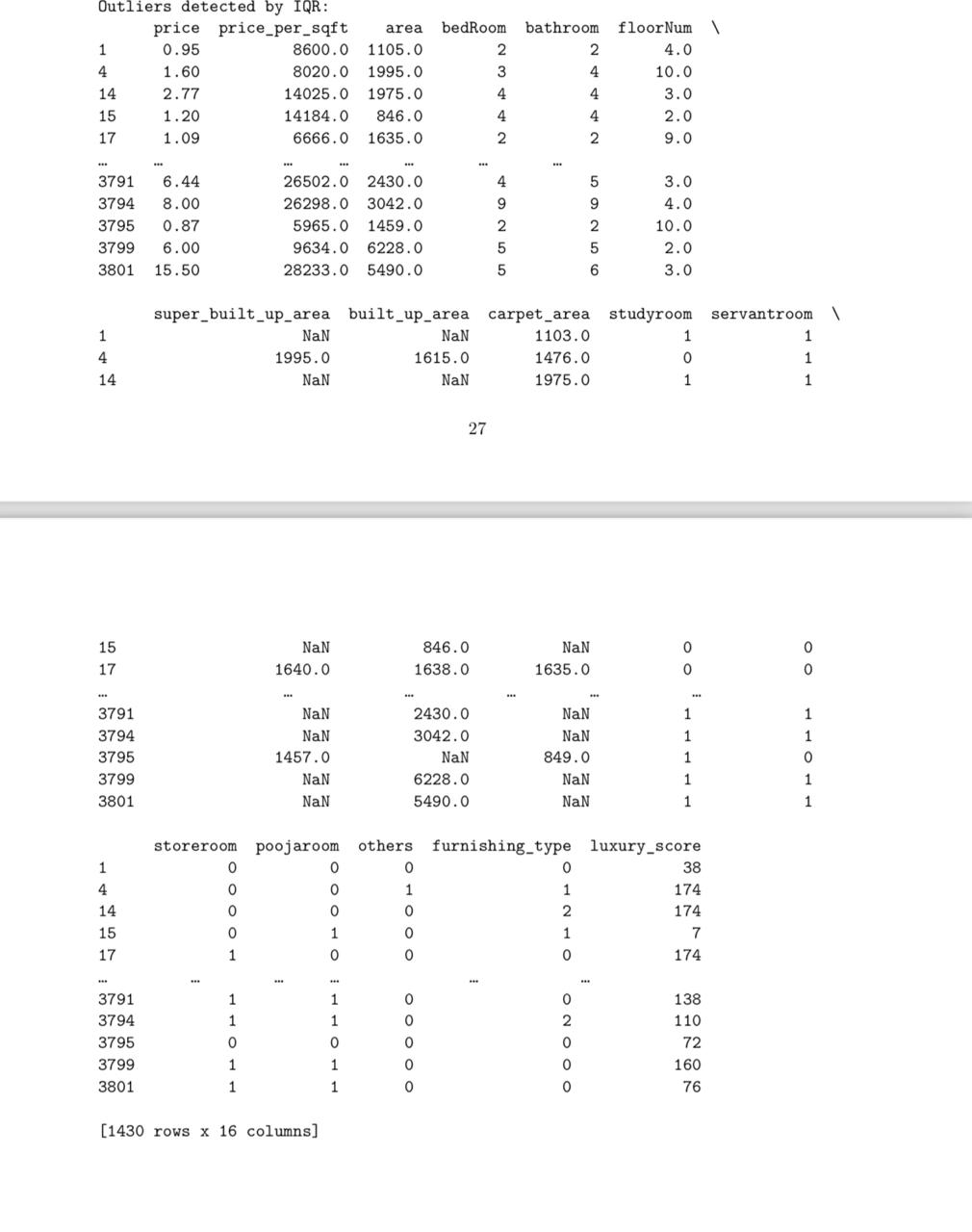
 ax[i, 0].set\_title(f'Histogram of {feature}')

 sns.boxplot(x=numerical\_abc[feature], ax=ax[i, 1])

 ax[i, 1].set\_title(f'Box plot of {feature}')

plot.tight\_layout()

plot.show()



# Visualization of distributions and outliers

fig, ax = plot.subplots(len(numerical\_abc.columns), 2, figsize=(16, 4 \* len(numerical\_abc.columns)))

for i, feature in enumerate(numerical\_abc.columns):

 sns.histplot(numerical\_abc[feature], kde=True, ax=ax[i, 0])

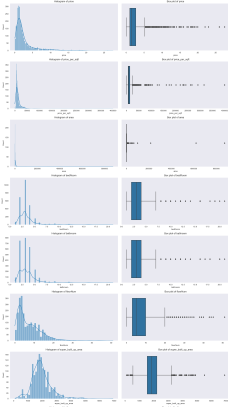
 ax[i, 0].set\_title(f'Histogram of {feature}')

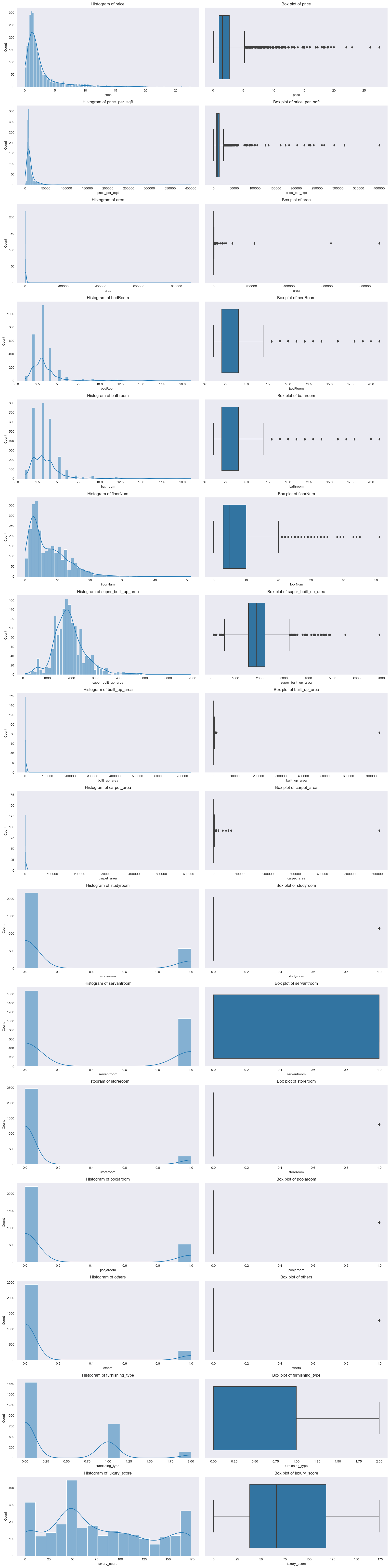
 sns.boxplot(x=numerical\_abc[feature], ax=ax[i, 1])

 ax[i, 1].set\_title(f'Box plot of {feature}')

plot.tight\_layout()

plot.show()





Our initial code for z-score:

#OUTLIERS DETECTION FOR NORMALLY DISTRUBUTED COLUMN THROUGH ZSCORE

#%matplotlib inline

#matplotlib.rcParams['figure.figsize'] = (12,8)

#bedroom

#plot.hist(abcd.bedRoom, bins=20, rwidth=0.8)

#plot.xlabel('bedRoom')

#plot.ylabel('count')

#plot.show()

#rngg = np.arange(abcd.bedRoom.min(), abcd.bedRoom.max(), 0.1)

#plot.plot(rngg, norm.pdf(rngg, abcd.bedRoom.mean(), abcd.bedRoom.std()))

#upperlim = abcd.bedRoom.mean() + 3\*abcd.bedRoom.std()

#lowerlim = abcd.bedRoom.mean() - 3\*abcd.bedRoom.std()

#abcd[(abcd.bedRoom>upperlim)|(abcd.bedRoom<lowerlim)]

#abcde=abcd[(abcd.bedRoom>upperlim)|(abcd.bedRoom<lowerlim)]

#abcd['zscore'] = (abcd.bedRoom - abcd.bedRoom.mean())/abcd.bedRoom.std()

#bathroom

#plot.hist(abcd.bathroom, bins=20, rwidth=0.8)

#plot.xlabel('bathroom')

#plot.ylabel('count')

#plot.show()

#rngg = np.arange(abcd.bathroom.min(), abcd.bathroom.max(), 0.1)

#plot.plot(rngg, norm.pdf(rngg, abcd.bathroom.mean(), abcd.bathroom.std()))

#upperlim = abcd.bathroom.mean() + 3\*abcd.bathroom.std()

#lowerlim = abcd.bathroom.mean() - 3\*abcd.bathroom.std()

#abcd[(abcd.bathroom>upperlim)|(abcd.bathroom<lowerlim)]

#abcde=abcd[(abcd.bathroom>upperlim)|(abcd.bathroom<lowerlim)]

#abcd['zscore'] = (abcd.bathroom - abcd.bathroom.mean())/abcd.bathroom.std()

# #age possession

# plot.hist(abcd.agePossession, bins=20, rwidth=0.8)

# plot.xlabel('agePossession')

# plot.ylabel('count')

# plot.show()

# rngg = np.arange(abcd.agePossession.min(), abcd.agePossession.max(), 0.1)

# plot.plot(rngg, norm.pdf(rngg, abcd.agePossession.mean(), abcd.agePossession.std()))

# upperlim = abcd.agePossession.mean() + 3\*abcd.agePossession.std()

# lowerlim = abcd.agePossession.mean() - 3\*abcd.bedRoom.std()

# abcd[(abcd.agePossession>upperlim)|(abcd.agePossession<lowerlim)]

# abcde=abcd[(abcd.agePossession>upperlim)|(abcd.agePossession<lowerlim)]

# abcd['zscore'] = (abcd.agePossession - abcd.agePossession.mean())/abcd.agePossession.std()

**TASK 4:**

**OUTLIERS HANDLING:**

print("OUTLIERS HANDLING")

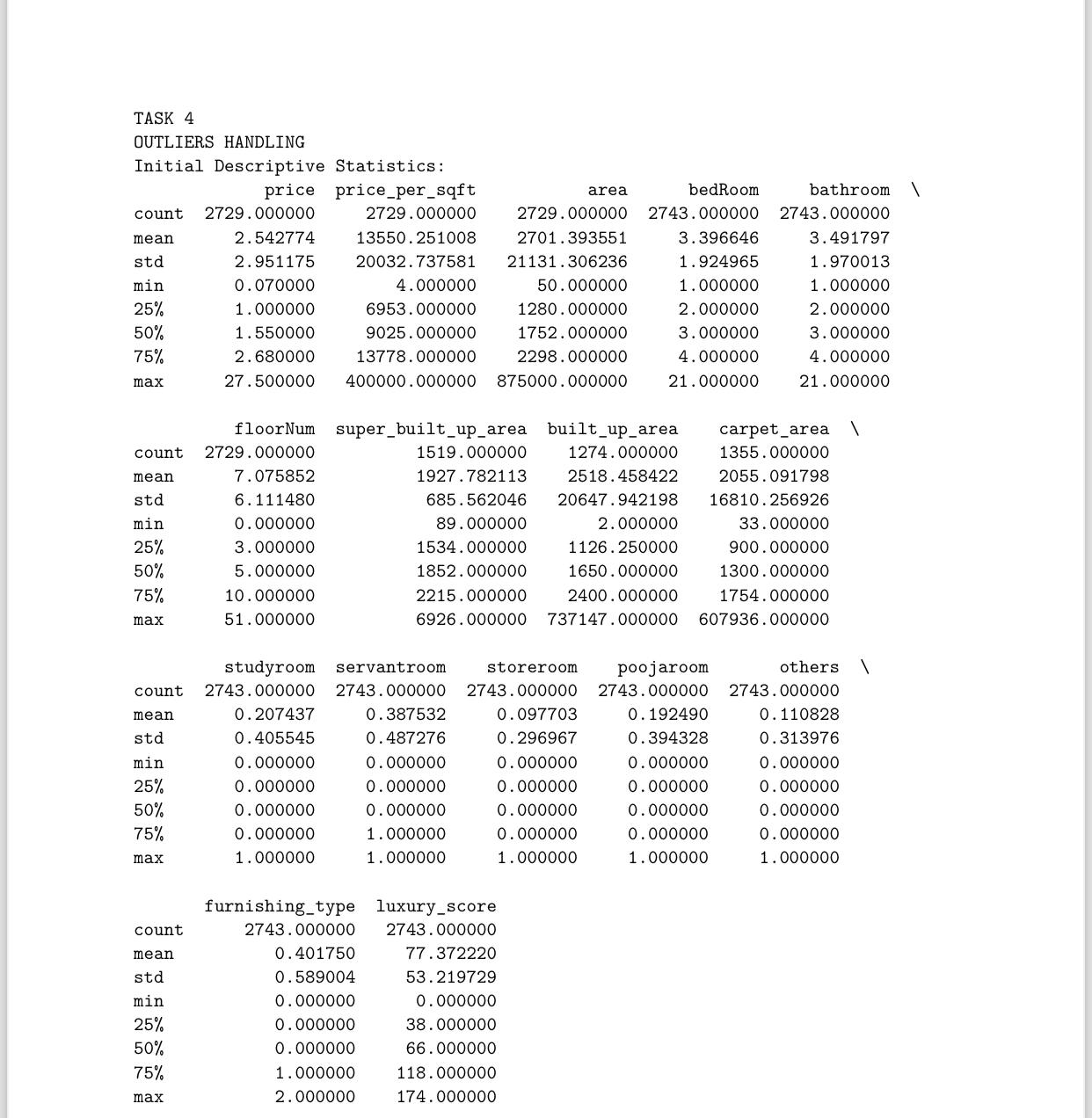
# Select only numerical columns

numerical\_abc = abc.select\_dtypes(include=[np.number])

# Display initial descriptive statistics

print("Initial Descriptive Statistics:")

print(numerical\_abc.describe())

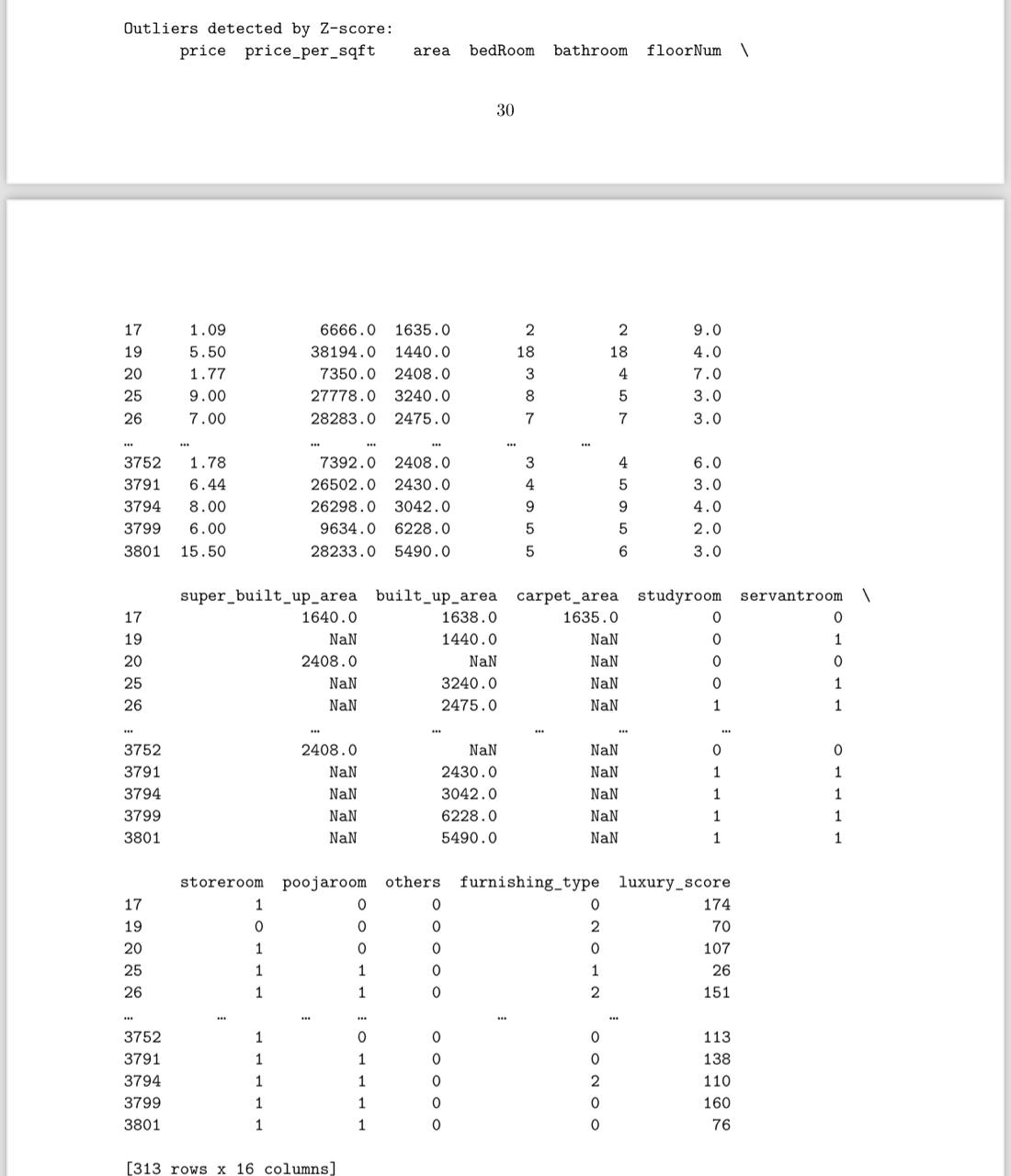


# Z-score method to identify outliers

z\_score = np.abs(stats.zscore(numerical\_abc))

outlier\_z = (z\_score > 3).any(axis=1)

print(f"\nOutliers detected by Z-score:\n{numerical\_abc[outlier\_z]}")



# IQR method to identify outliers

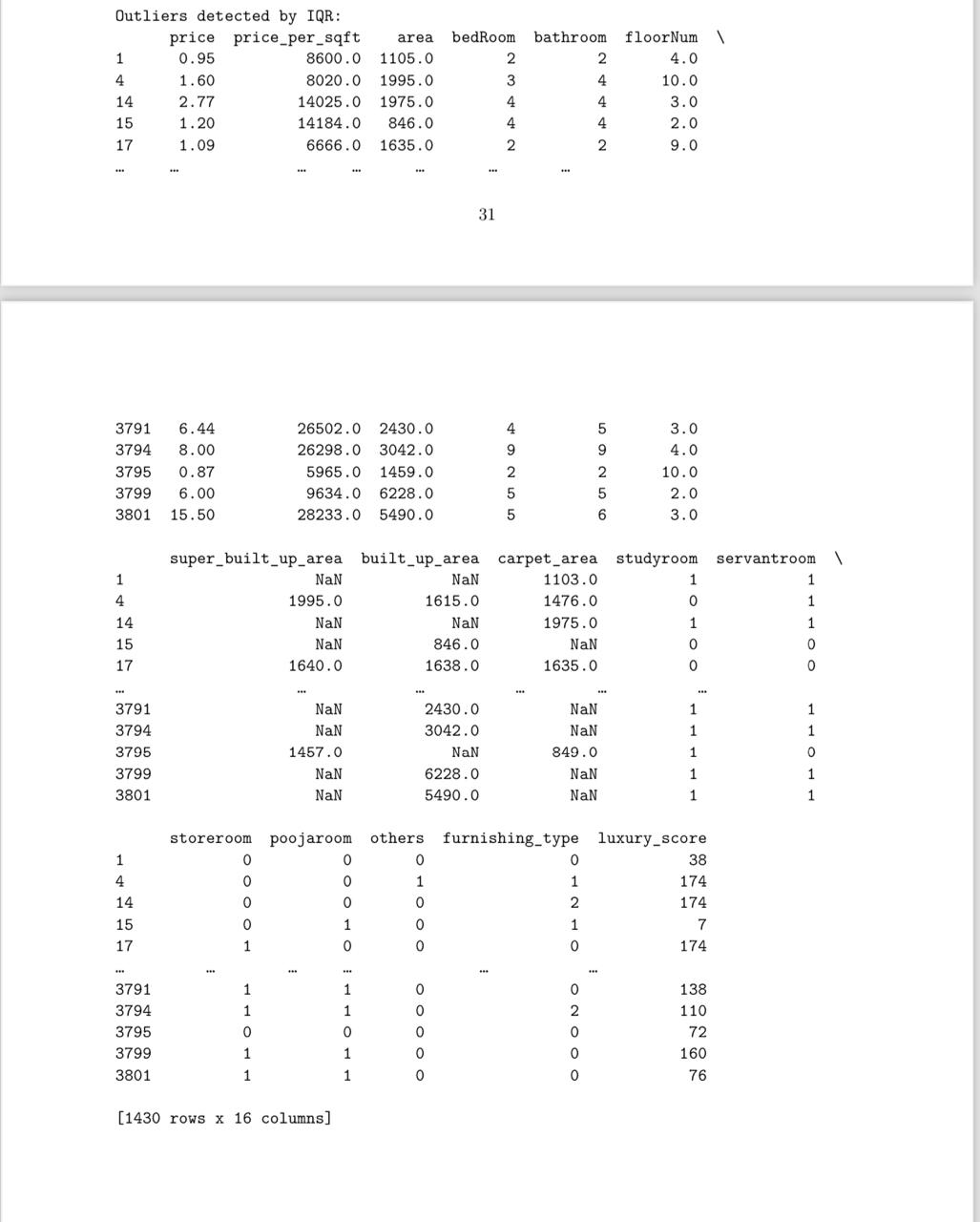
Q1 = numerical\_abc.quantile(0.25)

Q3 = numerical\_abc.quantile(0.75)

IQR = Q3 - Q1

outlier\_iqr = ((numerical\_abc < (Q1 - 1.5 \* IQR)) | (numerical\_abc > (Q3 + 1.5 \* IQR))).any(axis=1)

print(f"\nOutliers detected by IQR:\n{numerical\_abc[outlier\_iqr]}")



The three strategies that we used to handle outliers are – Winsorization, trimming, transformation.

#STRATERGY 1

winsorized\_abc = numerical\_abc.apply(lambda x: winsorize(x, limits=[0.05, 0.05]))

# Strategy 2: Trimming

trimmed\_abc = numerical\_abc[~outlier\_z]

# Strategy 3: Log Transformation (example for one feature, can be applied to all if needed)

transformed\_abc = numerical\_abc.apply(lambda x: np.log1p(x))

# Visualize the distributions and outliers before and after handling

fig, axs = plot.subplots(4, len(numerical\_abc.columns), figsize=(20, 16))

for i, feature in enumerate(numerical\_abc.columns):

 sns.histplot(numerical\_abc[feature], kde=True, ax=axs[0, i])

 axs[0, i].set\_title(f'Original {feature}')

 sns.histplot(winsorized\_abc[feature], kde=True, ax=axs[1, i])

 axs[1, i].set\_title(f'Winsorized {feature}')

 sns.histplot(trimmed\_abc[feature], kde=True, ax=axs[2, i])

 axs[2, i].set\_title(f'Trimmed {feature}')

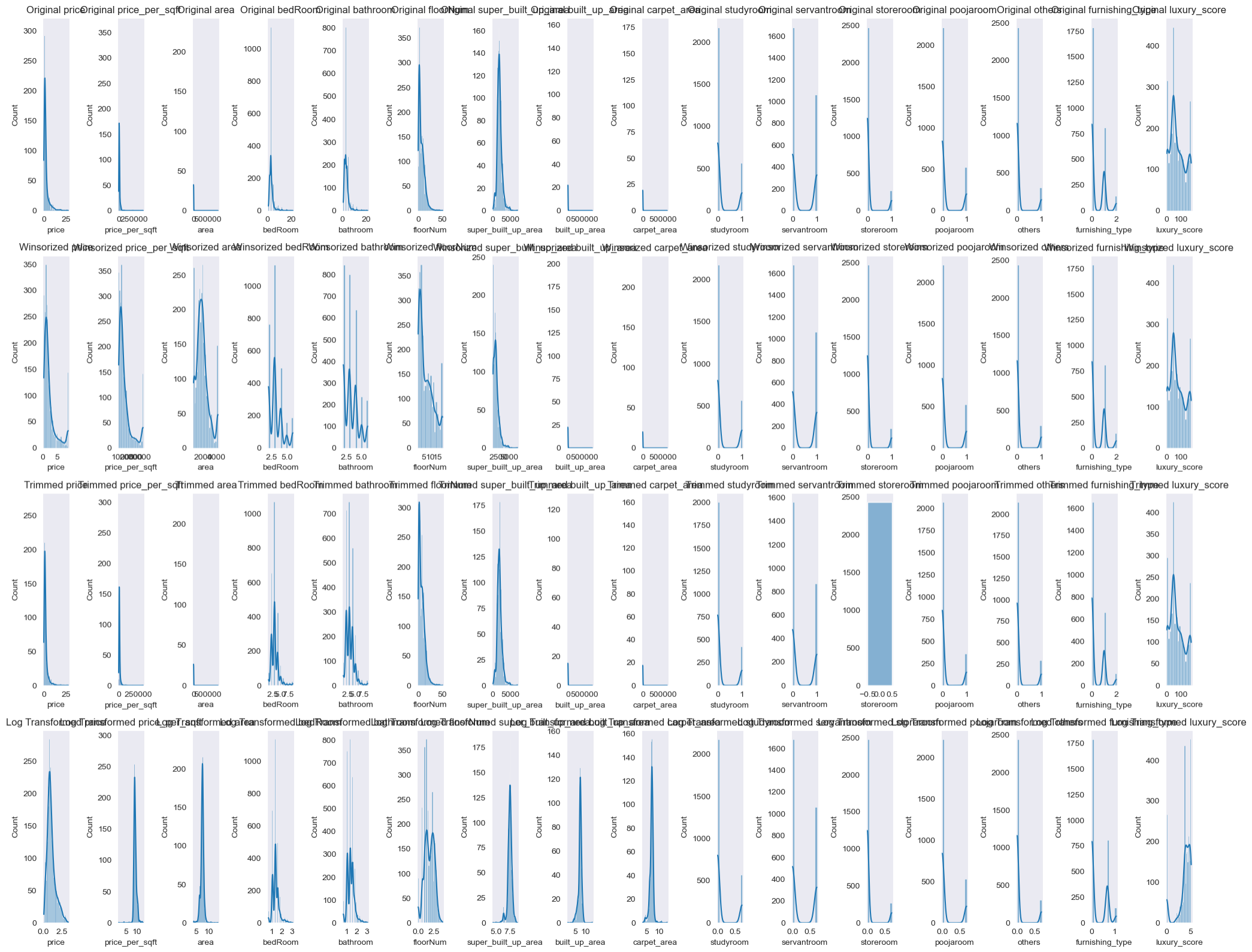
 sns.histplot(transformed\_abc[feature], kde=True, ax=axs[3, i])

 axs[3, i].set\_title(f'Log Transformed {feature}')

plot.tight\_layout()

plot.show()

**DISTRIBUTIONS AND OUTLIERS BEFORE AND AFTER HANDLING:**



# Function to calculate metrics for evaluating impact

def calculatemetrics(data, title):

 metrics = pd.DataFrame({

 'Mean': data.mean(),

 'Std Dev': data.std(),

 'Skewness': data.skew(),

 'Kurtosis': data.apply(lambda x: stats.kurtosis(x))

 })

 print(f'\n{title}')

 print(metrics)

# Evaluate the impact of outlier handling

print("\nMetrics Before Handling Outliers:")

calculatemetrics(numerical\_abc, 'Original Data Metrics')

print("\nMetrics After Winsorization:")

calculatemetrics(winsorized\_abc, 'Winsorized Data Metrics')

print("\nMetrics After Trimming:")

calculatemetrics(trimmed\_abc, 'Trimmed Data Metrics')

print("\nMetrics After Log Transformation:")

calculatemetrics(transformed\_abc, 'Log Transformed Data Metrics')

