Importing Libraries and loading Dataset

```
import numpy as np
import pandas as pd

org_df = pd.read_csv('/content/house_price.csv')

#taking copy of org_df for operations and keeping org_df as backup
df = org_df.copy()
```

Q1. Perform basic EDA

```
[34] #first 5 rows of dataframe
df.head()
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	4615
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
[35] #shape of dataframe
df.shape
```

```
(13200, 7)
```

```
#columns of dataframe
df.columns
```

```
Index(['location', 'size', 'total_sqft', 'bath', 'price', 'bhk',
       'price_per_sqft'],
      dtype='object')
```

```
#informations about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13200 entries, 0 to 13199
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   location        13200 non-null  object
 1   size            13200 non-null  object
 2   total_sqft      13200 non-null  float64
 3   bath            13200 non-null  float64
 4   price           13200 non-null  float64
 5   bhk             13200 non-null  int64
 6   price_per_sqft  13200 non-null  int64
dtypes: float64(3), int64(2), object(2)
memory usage: 722.0+ KB
```

```
#checking null values
df.isnull().sum()
```

	0
location	0
size	0
total_sqft	0
bath	0
price	0
bhk	0
price_per_sqft	0

[38]
```
#checking null values
df.isnull().sum()
```

	0
location	0
size	0
total_sqft	0
bath	0
price	0
bhk	0
price_per_sqft	0

dtype: int64

```
#examine price_per_sqft column to check min, max, mean and percentile values
df.price_per_sqft.describe()
```

	price_per_sqft
count	1.320000e+04
mean	7.920337e+03
std	1.067272e+05
min	2.670000e+02
25%	4.267000e+03
50%	5.438000e+03
75%	7.317000e+03
max	1.200000e+07

dtype: float64

```
#examine columns to check min, max, meand and percentile values
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
total_sqft	13200.0	1555.302783	1237.323445	1.0	1100.0	1275.00	1672.0	52272.0
bath	13200.0	2.691136	1.338915	1.0	2.0	2.00	3.0	40.0
price	13200.0	112.276178	149.175995	8.0	50.0	71.85	120.0	3600.0
bhk	13200.0	2.800833	1.292843	1.0	2.0	3.00	3.0	43.0
price_per_sqft	13200.0	7920.336742	106727.160328	267.0	4267.0	5438.00	7317.0	12000000.0

```
#Check unique values in bath and size columns
df['size'].unique()
```

```
array(['2 BHK', '4 Bedroom', '3 BHK', '4 BHK', '6 Bedroom', '3 Bedroom',
       '1 BHK', '1 RK', '1 Bedroom', '8 Bedroom', '2 Bedroom',
       '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',
       '9 BHK', '9 Bedroom', '27 BHK', '10 Bedroom', '11 Bedroom',
       '10 BHK', '19 BHK', '16 BHK', '43 Bedroom', '14 BHK', '8 BHK',
       '12 Bedroom', '13 BHK', '18 Bedroom'], dtype=object)
```

```
[11] df['bath'].unique()
```

```
array([ 2.,  5.,  3.,  4.,  6.,  1.,  9.,  8.,  7., 11., 10., 14., 27.,
       12., 16., 40., 15., 13., 18.])
```

Analsis after EDA

Findings other than price_per_sqft

1. bath column which represent bathroom has upto 40 which is not possible to happen.

2. size column which represent size or number of bedroom has upto 43 which is not possible to

Analsis after EDA

Findings other than price_per_sqft

1. bath column which represent bathroom has upto 40 which is not possible to happen.

2. size column which represent size or number of bedroom has upto 43 which is not possible to happen, and also the column has both integer and string values which is not in a standard form. String values are different too.

Q2. Detect the outliers using following methods and remove it using methods like trimming / capping/ imputation using mean or median

```
#a) Mean and Standard deviation

#limits
max_limit = df.price_per_sqft.mean() + 3 * df.price_per_sqft.std()
min_limit = df.price_per_sqft.mean() - 3 * df.price_per_sqft.std()

#outliers
outliers_mean_std = df.loc[(df['price_per_sqft'] > max_limit) | (df['price_per_sqft'] < min_limit)]

#1) trimming
trim_mean_std_df = df.loc[(df['price_per_sqft'] <= max_limit) & (df['price_per_sqft'] >= min_limit)]

#2) capping
cap_df_mean_std = df.copy()
cap_df_mean_std['price_per_sqft'] = cap_df_mean_std['price_per_sqft'].astype(float)
cap_df_mean_std.loc[cap_df_mean_std['price_per_sqft'] > max_limit, 'price_per_sqft'] = max_limit
cap_df_mean_std.loc[cap_df_mean_std['price_per_sqft'] < min_limit, 'price_per_sqft'] = min_limit

#3) imputation using median
```

[43]
```
#a) Mean and Standard deviation

#limits
max_limit = df.price_per_sqft.mean() + 3 * df.price_per_sqft.std()
min_limit = df.price_per_sqft.mean() - 3 * df.price_per_sqft.std()

#outliers
outliers_mean_std = df.loc[(df['price_per_sqft'] > max_limit) | (df['price_per_sqft'] < min_limit)]

#1) trimming
trim_mean_std_df = df.loc[(df['price_per_sqft'] <= max_limit) & (df['price_per_sqft'] >= min_limit)]

#2) capping
cap_df_mean_std = df.copy()
cap_df_mean_std['price_per_sqft'] = cap_df_mean_std['price_per_sqft'].astype(float)
cap_df_mean_std.loc[cap_df_mean_std['price_per_sqft'] > max_limit, 'price_per_sqft'] = max_limit
cap_df_mean_std.loc[cap_df_mean_std['price_per_sqft'] < min_limit, 'price_per_sqft'] = min_limit

#3) imputation using median
imp_median_std_df = df.copy()
imp_median_std_df['price_per_sqft'] = imp_median_std_df['price_per_sqft'].astype(float)
median = imp_median_std_df['price_per_sqft'].median()
imp_median_std_df.loc[imp_median_std_df['price_per_sqft'] > max_limit, 'price_per_sqft'] = median
imp_median_std_df.loc[imp_median_std_df['price_per_sqft'] < min_limit, 'price_per_sqft'] = median
```

```
#b) Percentile method

#limits
upper_limit = np.percentile(df['price_per_sqft'], 95)
lower_limit = np.percentile(df['price_per_sqft'], 5)

#outliers
outliers_percentile = df.loc[(df['price_per_sqft'] < lower_limit) | (df['price_per_sqft'] > upper_limit)]

#1) trimming
trim_percentile_df = df.loc[(df['price_per_sqft'] >= lower_limit) & (df['price_per_sqft'] <= upper_limit)]

#2) capping
cap_percentile_df = df.copy()
```

[44]
```
#b) Percentile method

#limits
upper_limit = np.percentile(df['price_per_sqft'], 95)
lower_limit = np.percentile(df['price_per_sqft'], 5)

#outliers
outliers_percentile = df.loc[(df['price_per_sqft'] < lower_limit) | (df['price_per_sqft'] > upper_limit)]

#1) trimming
trim_percentile_df = df.loc[(df['price_per_sqft'] >= lower_limit) & (df['price_per_sqft'] <= upper_limit)]

#2) capping
cap_percentile_df = df.copy()
cap_percentile_df['price_per_sqft'] = cap_percentile_df['price_per_sqft'].astype(float)
cap_percentile_df.loc[cap_percentile_df['price_per_sqft'] > upper_limit, 'price_per_sqft'] = upper_limit
cap_percentile_df.loc[cap_percentile_df['price_per_sqft'] < lower_limit, 'price_per_sqft'] = lower_limit

#3) imputation using median
imp_median_percentile_df = df.copy()
imp_median_percentile_df['price_per_sqft'] = imp_median_percentile_df['price_per_sqft'].astype(float)
median = imp_median_percentile_df['price_per_sqft'].median()
imp_median_percentile_df.loc[imp_median_percentile_df['price_per_sqft'] > upper_limit, 'price_per_sqft'] = median
imp_median_percentile_df.loc[imp_median_percentile_df['price_per_sqft'] < lower_limit, 'price_per_sqft'] = median
```

```
#c) IQR(Inter quartile range method)

#limits
Q1 = df['price_per_sqft'].quantile(0.25)
Q3 = df['price_per_sqft'].quantile(0.75)

IQR = Q3 - Q1

upper_limit = Q3 + 1.5 * IQR
lower_limit = Q1 - 1.5 * IQR

#1) trimming
trim_iqr_df = df.loc[(df['price_per_sqft'] >= lower_limit) & (df['price_per_sqft'] <= upper_limit)]
```

[45]
```
#c) IQR(Inter quartile range method)

    #limits
    Q1 = df['price_per_sqft'].quantile(0.25)
    Q3 = df['price_per_sqft'].quantile(0.75)

    IQR = Q3 - Q1

    upper_limit = Q3 + 1.5 * IQR
    lower_limit = Q1 - 1.5 * IQR

    #1) trimming
    trim_iqr_df = df.loc[(df['price_per_sqft'] >= lower_limit) & (df['price_per_sqft'] <= upper_limit)]

    #2) capping
    cap_iqr_df = df.copy()
    cap_iqr_df['price_per_sqft'] = cap_iqr_df['price_per_sqft'].astype(float)
    cap_iqr_df.loc[cap_iqr_df['price_per_sqft'] > upper_limit, 'price_per_sqft'] = upper_limit
    cap_iqr_df.loc[cap_iqr_df['price_per_sqft'] < lower_limit, 'price_per_sqft'] = lower_limit

    #3) imputation using median
    imp_median_iqr_df = df.copy()
    imp_median_iqr_df['price_per_sqft'] = imp_median_iqr_df['price_per_sqft'].astype(float)
    median = imp_median_iqr_df['price_per_sqft'].median()
    imp_median_iqr_df.loc[imp_median_iqr_df['price_per_sqft'] > upper_limit, 'price_per_sqft'] = median
    imp_median_iqr_df.loc[imp_median_iqr_df['price_per_sqft'] < lower_limit, 'price_per_sqft'] = median
```

```
from scipy.stats import zscore

#compute z-scores
df['z_score'] = zscore(df['price_per_sqft'])

#limits
z_upper_limit = 3
z_lower_limit = -3

#outliers
outliers_z_score = df.loc[(df['z_score'] > z_upper_limit) | (df['z_score'] < z_lower_limit)]
```

[47]
```
#d) Z-score method

from scipy.stats import zscore

#compute z-scores
df['z_score'] = zscore(df['price_per_sqft'])

#limits
z_upper_limit = 3
z_lower_limit = -3

#outliers
outliers_z_score = df.loc[(df['z_score'] > z_upper_limit) | (df['z_score'] < z_lower_limit)]

#1) trimming
trim_df_z_score = df.loc[(df['z_score'] <= z_upper_limit) & (df['z_score'] >= z_lower_limit)]

#2) capping
cap_df_z_score = df.copy()
cap_df_z_score['price_per_sqft'] = cap_df_z_score['price_per_sqft'].astype(float)
cap_df_z_score.loc[cap_df_z_score['z_score'] > z_upper_limit, 'price_per_sqft'] = (df['price_per_sqft'].mean() + z_upper_limit * df['price_per_sqft'].std())
cap_df_z_score.loc[cap_df_z_score['z_score'] < z_lower_limit, 'price_per_sqft'] = (df['price_per_sqft'].mean() - z_lower_limit * df['price_per_sqft'].std())

#3) imputation using median
imp_median_df_z_score = df.copy()
imp_median_df_z_score['price_per_sqft'] = imp_median_df_z_score['price_per_sqft'].astype(float)
median_z_score = imp_median_df_z_score['price_per_sqft'].median()
imp_median_df_z_score.loc[imp_median_df_z_score['z_score'] > z_upper_limit, 'price_per_sqft'] = median_z_score
imp_median_df_z_score.loc[imp_median_df_z_score['z_score'] < z_lower_limit, 'price_per_sqft'] = median_z_score
```

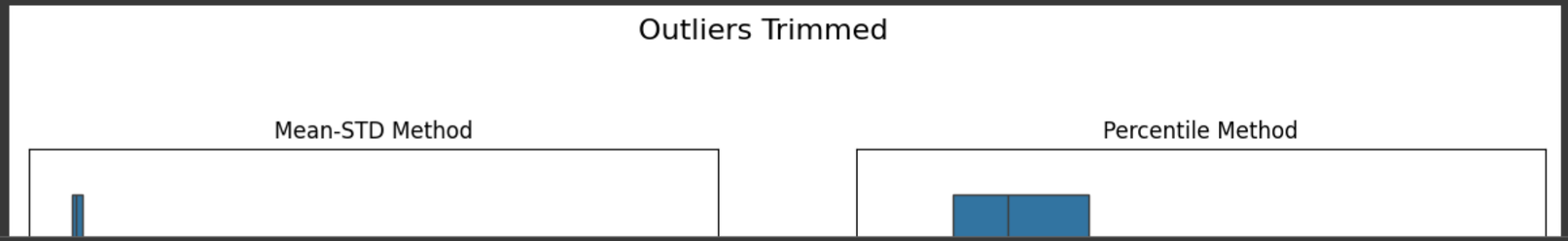
Q3. Create a box plot and use this to determine which method seems to work best to remove outliers for this data?

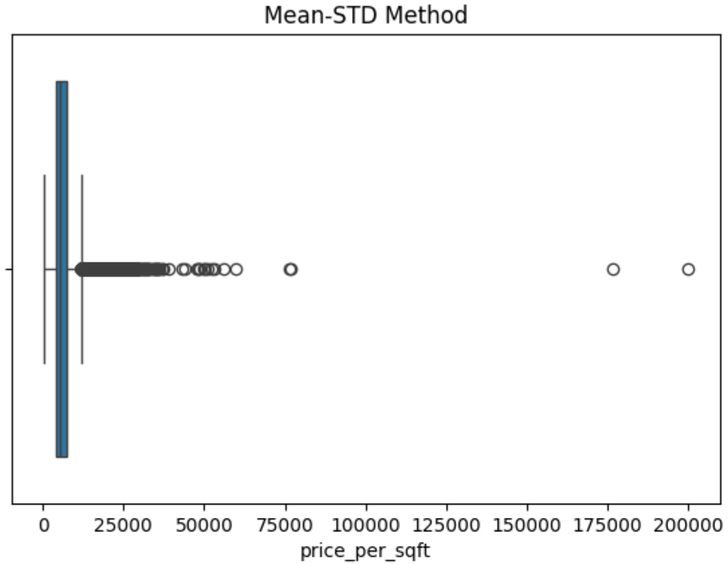
[16]
```
import matplotlib.pyplot as plt
import seaborn as sns
```

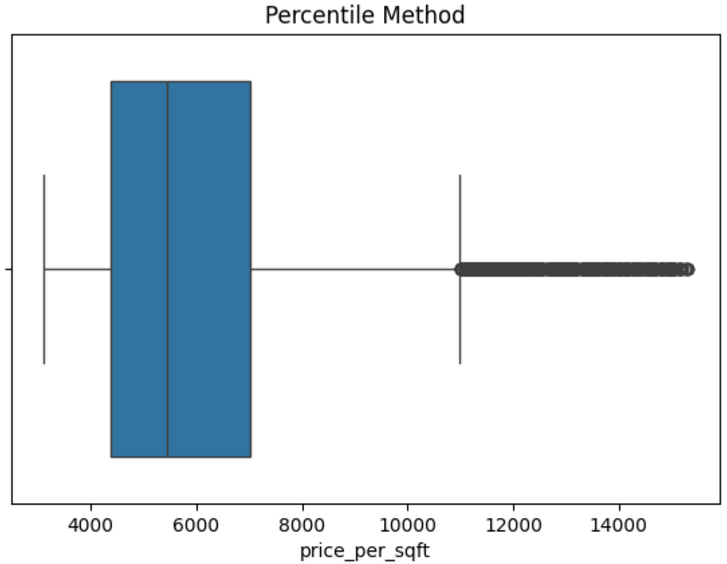
[47]
```
#3) imputation using median
imp_median_df_z_score = df.copy()
imp_median_df_z_score['price_per_sqft'] = imp_median_df_z_score['price_per_sqft'].astype(float)
median_z_score = imp_median_df_z_score['price_per_sqft'].median()
imp_median_df_z_score.loc[imp_median_df_z_score['z_score'] > z_upper_limit, 'price_per_sqft'] = median_z_score
imp_median_df_z_score.loc[imp_median_df_z_score['z_score'] < z_lower_limit, 'price_per_sqft'] = median_z_score
```

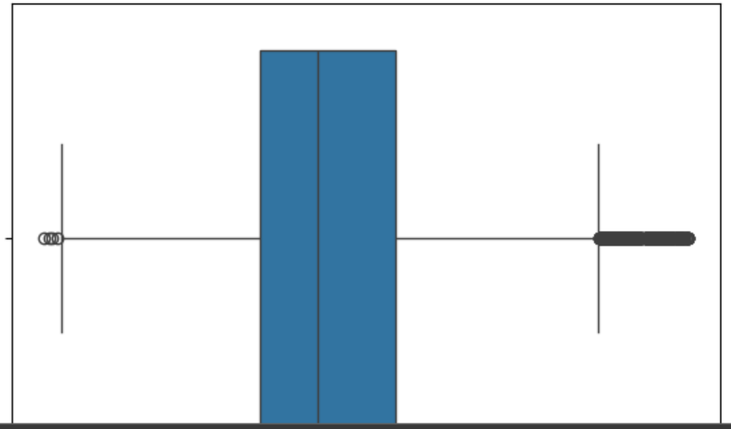
Q3. Create a box plot and use this to determine which method seems to work best to remove outliers for this data?

```
import matplotlib.pyplot as plt
import seaborn as sns

#compare methods

fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Outliers Trimmed', fontsize=16)

sns.boxplot(data=trim_mean_std_df, x='price_per_sqft', ax=axes[0, 0]).set_title('Mean-STD Method')
sns.boxplot(data=trim_percentile_df, x='price_per_sqft', ax=axes[0, 1]).set_title('Percentile Method')
sns.boxplot(data=trim_iqr_df, x='price_per_sqft', ax=axes[1, 0]).set_title('IQR Method')
sns.boxplot(data=trim_df_z_score, x='price_per_sqft', ax=axes[1, 1]).set_title('Z-Score Method')


plt.show()
```

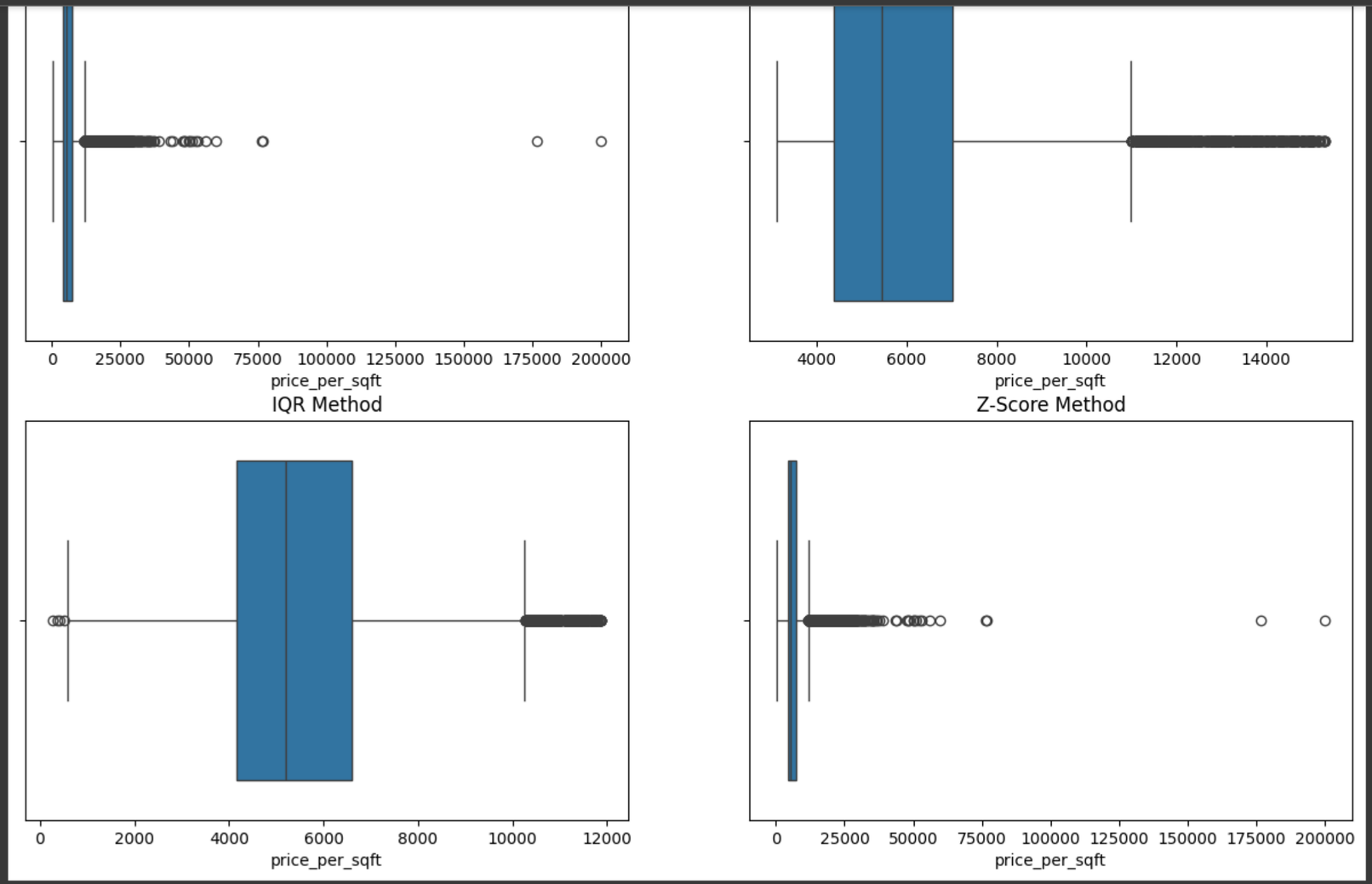


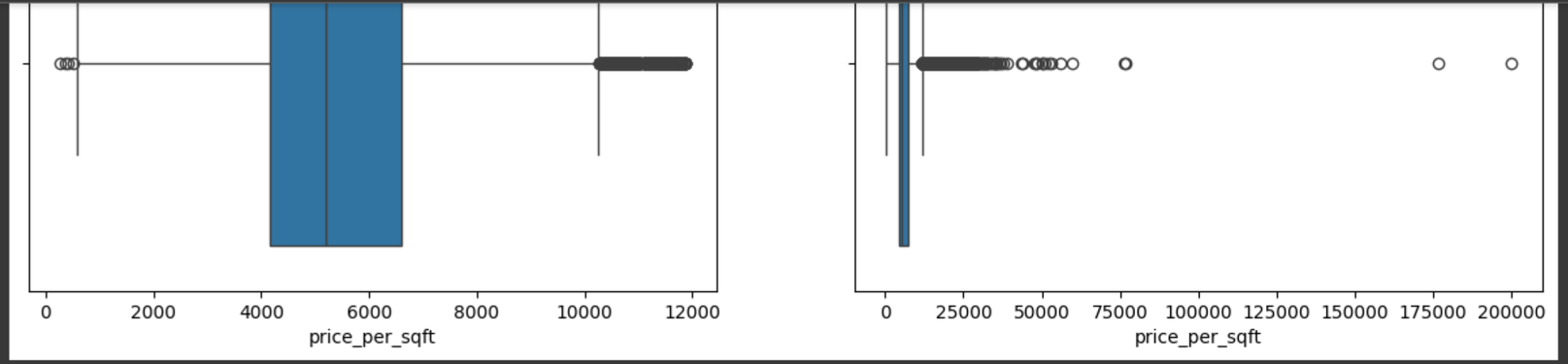

Mean-STD Method
0
25000
50000
75000
100000
125000
150000
175000
200000
price_per_sqft


Percentile Method
4000
6000
8000
10000
12000
14000
price_per_sqft

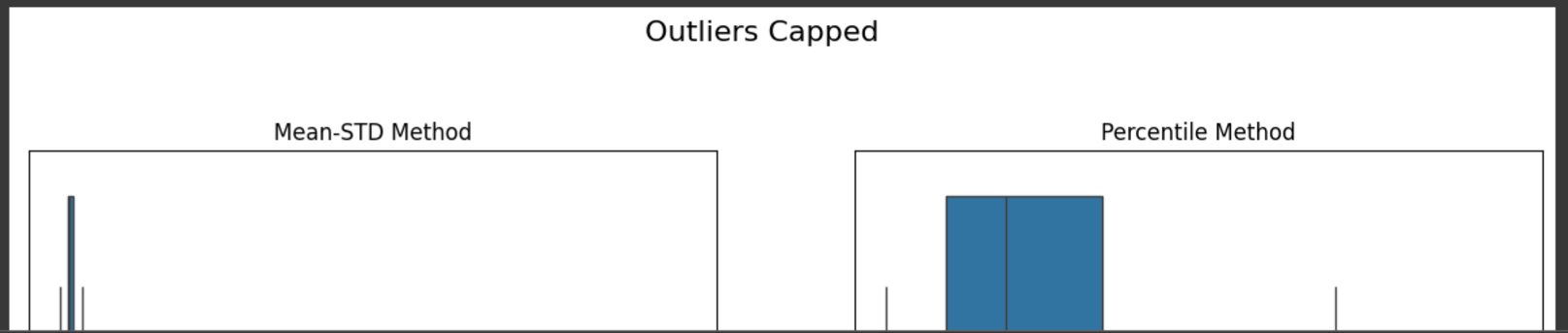

IQR Method

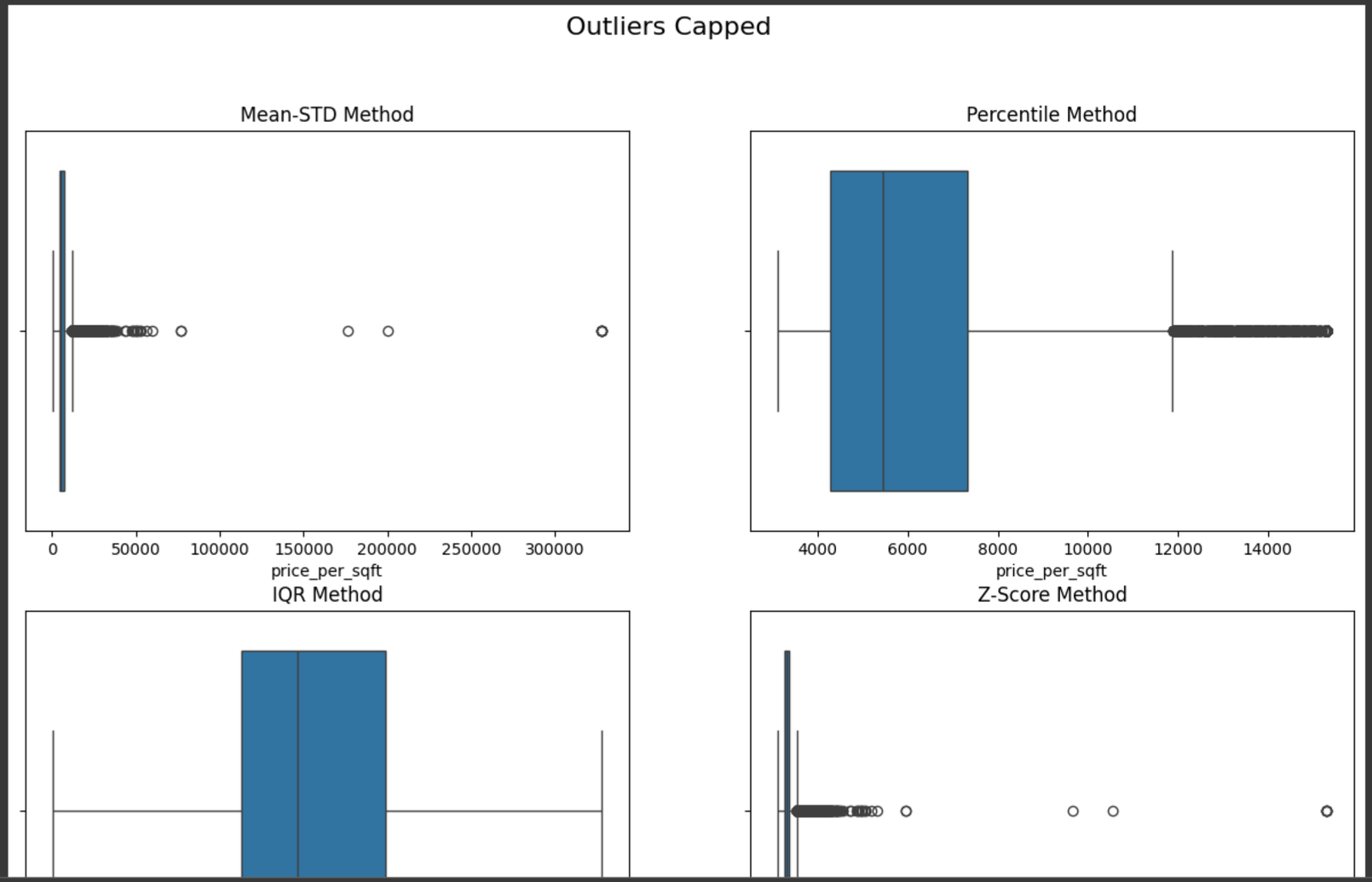

Z-Score Method


0 25000 50000 75000 100000 125000 150000 175000 200000
price_per_sqft
4000 6000 8000 10000 12000 14000
price_per_sqft
IQR Method
Z-Score Method
0 2000 4000 6000 8000 10000 12000
price_per_sqft
0 25000 50000 75000 100000 125000 150000 175000 200000
price_per_sqft

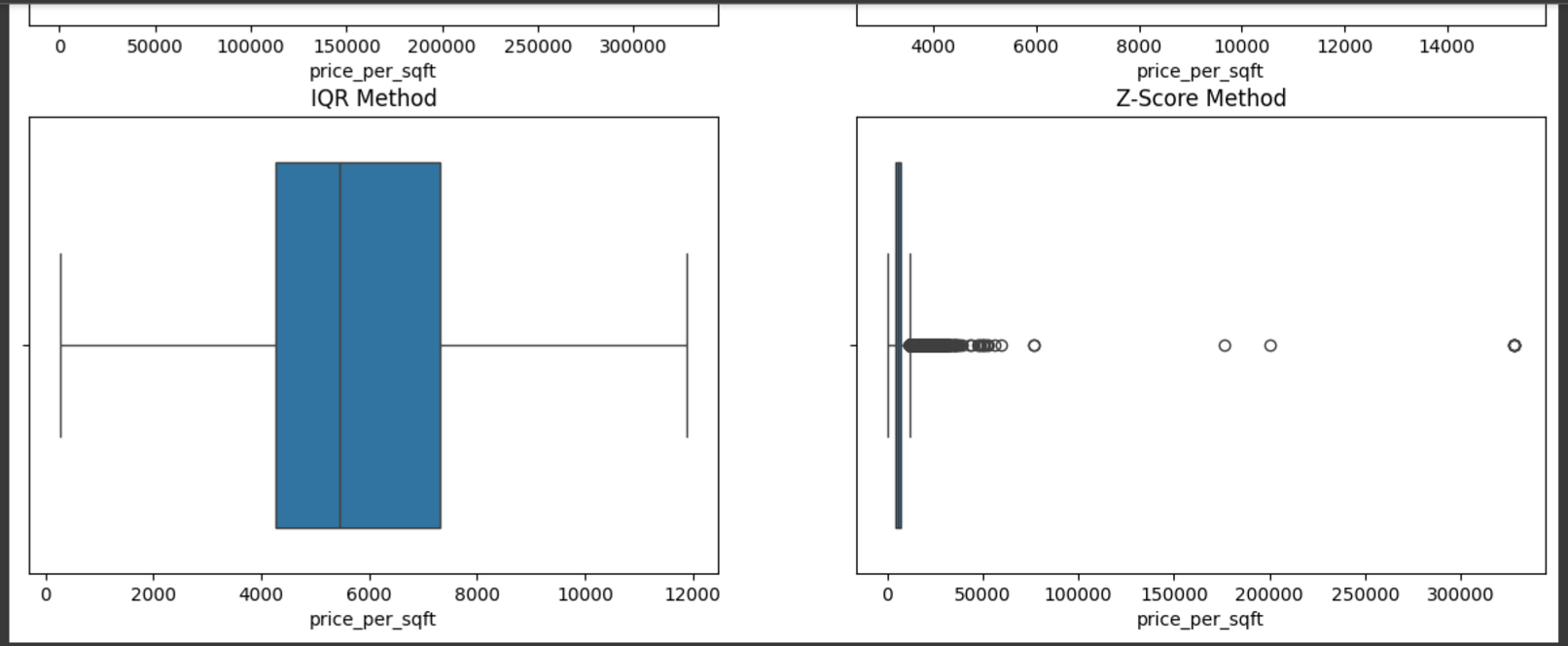



```
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Outliers Capped', fontsize=16)

sns.boxplot(data=cap_df_mean_std, x='price_per_sqft', ax=axes[0, 0]).set_title('Mean-STD Method')
sns.boxplot(data=cap_percentile_df, x='price_per_sqft', ax=axes[0, 1]).set_title('Percentile Method')
sns.boxplot(data=cap_iqr_df, x='price_per_sqft', ax=axes[1, 0]).set_title('IQR Method')
sns.boxplot(data=cap_df_z_score, x='price_per_sqft', ax=axes[1, 1]).set_title('Z-Score Method')


plt.show()
```



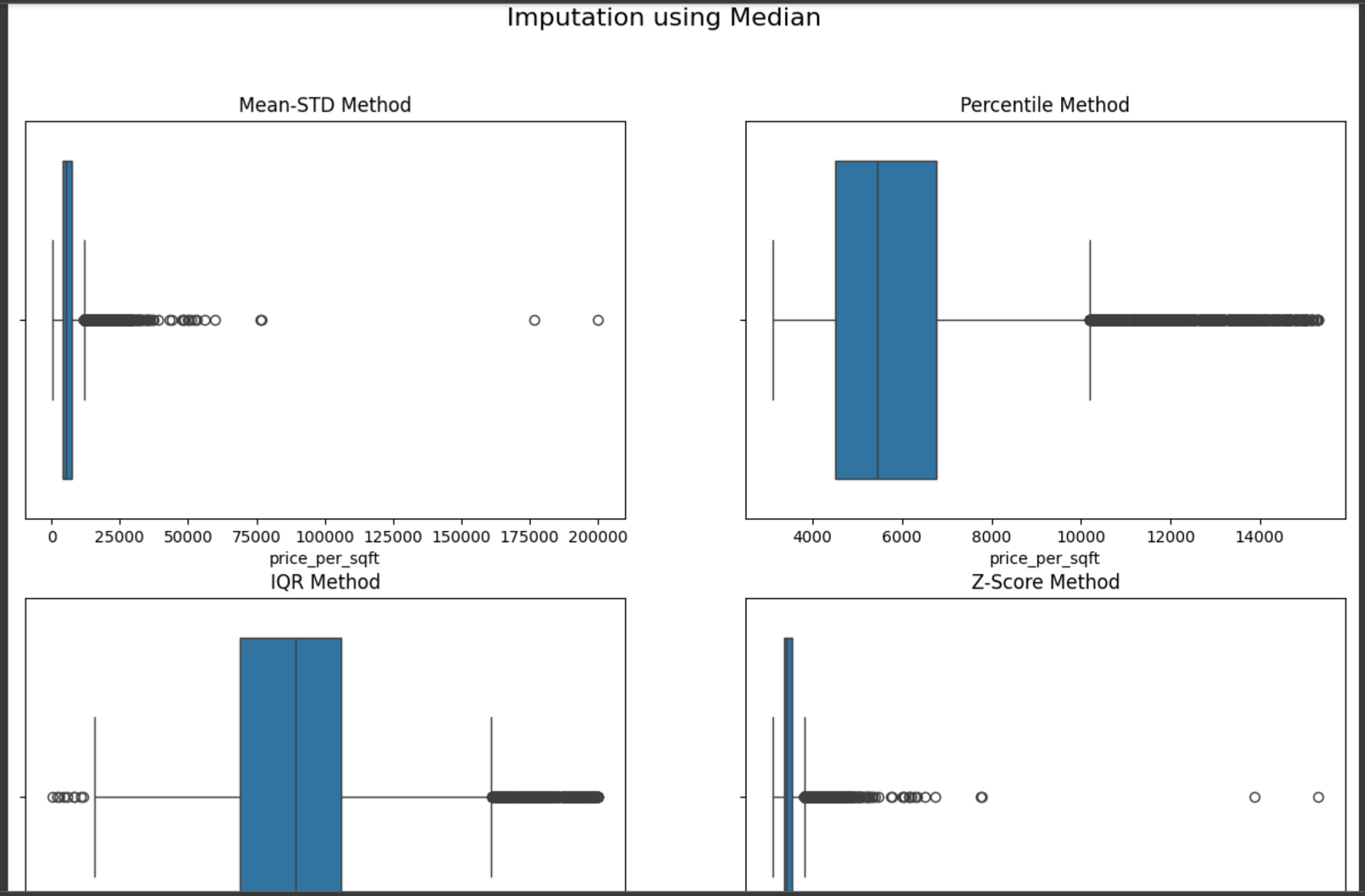

Outliers Capped
Mean-STD Method
0
50000
100000
150000
200000
250000
300000
price_per_sqft
Percentile Method
4000
6000
8000
10000
12000
14000
price_per_sqft
IQR Method
Z-Score Method

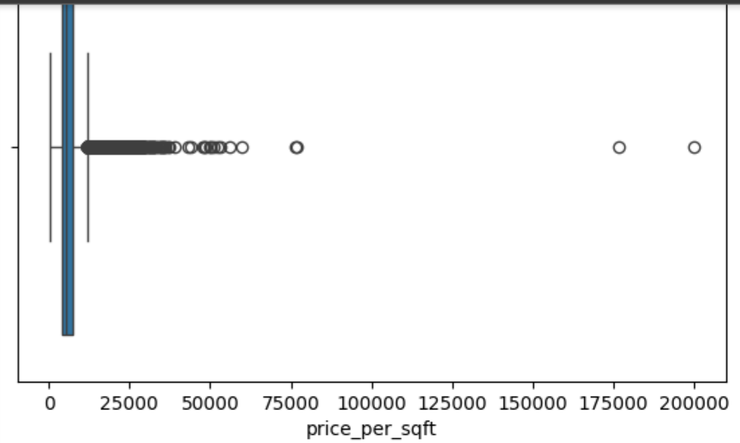
[17]

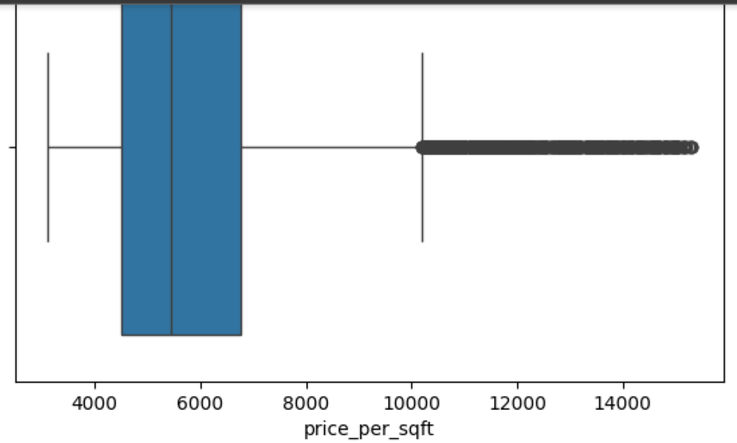



```
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Imputation using Median', fontsize=16)

sns.boxplot(data=imp_median_std_df, x='price_per_sqft', ax=axes[0, 0]).set_title('Mean-STD Method')
sns.boxplot(data=imp_median_percentile_df, x='price_per_sqft', ax=axes[0, 1]).set_title('Percentile Method')
sns.boxplot(data=imp_median_iqr_df, x='price_per_sqft', ax=axes[1, 0]).set_title('IQR Method')
sns.boxplot(data=imp_median_df_z_score, x='price_per_sqft', ax=axes[1, 1]).set_title('Z-Score Method')

plt.show()
```

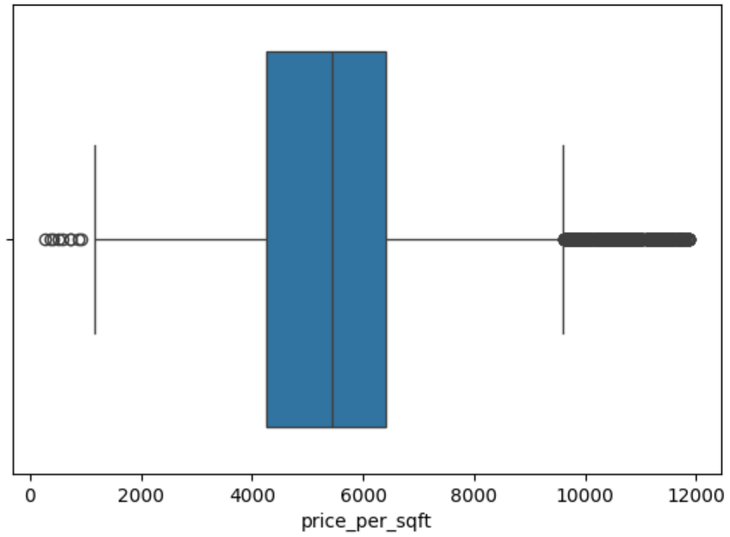
Imputation using Median


Imputation using Median
Mean-STD Method
0
25000
50000
75000
100000
125000
150000
175000
200000
price_per_sqft
Percentile Method
4000
6000
8000
10000
12000
14000
price_per_sqft
IQR Method
Z-Score Method

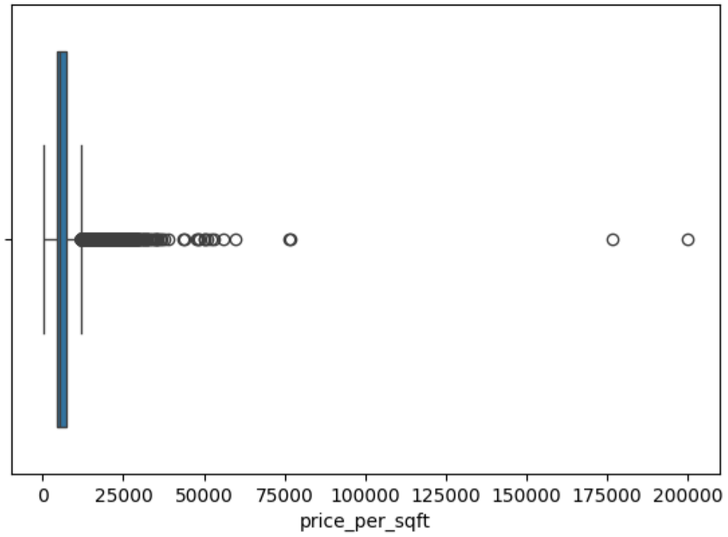


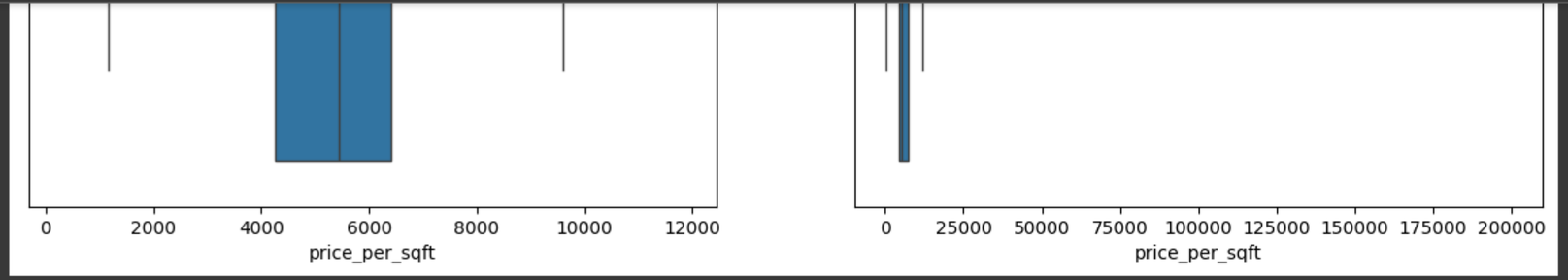


IQR Method



Z-Score Method





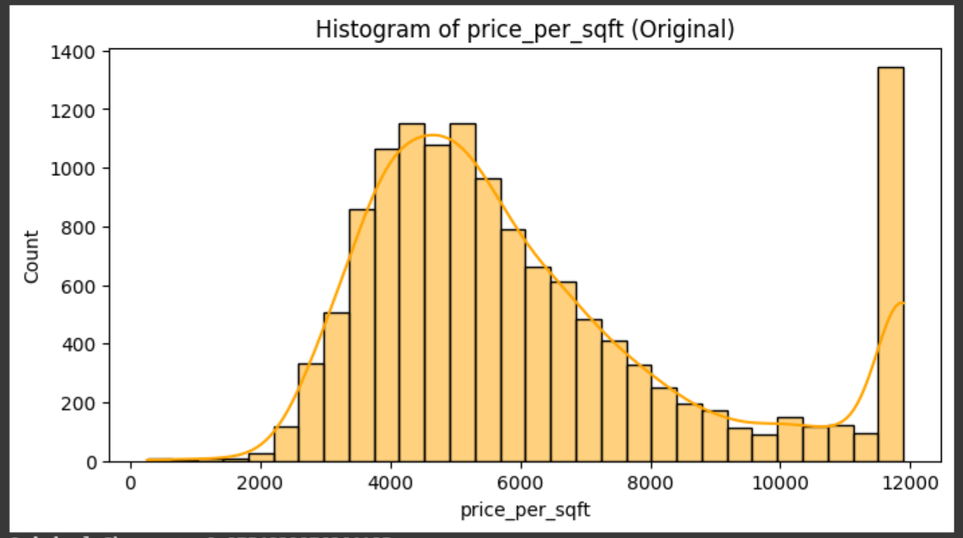
From the above comparison of methods for outlier detection IQR(Inter quartile range method) is the best method in my findings while comparing to other methods like Mean-STD, Percentile and Z-Score. For fixing the limits "Capping" is the best way in IQR(Inter quartile range method).

Q4. Draw histplot to check the normality of the column(price per sqft column) and perform transformations if needed. Check the skewness and kurtosis before and after the transformation.

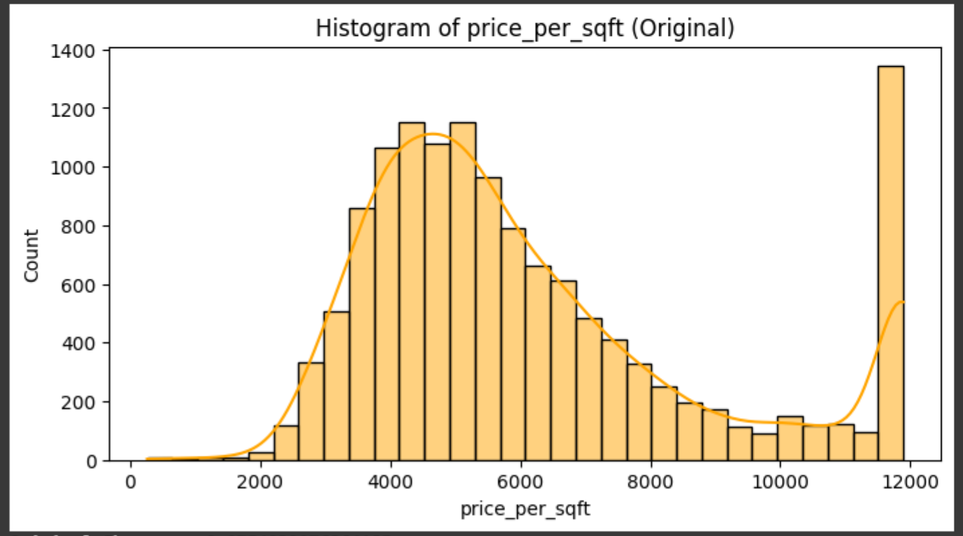
cap_iqr_df is using for transformation because it ensures that extreme outliers are managed before applying these transformations

```
[19] from scipy.stats import skew, kurtosis, boxcox, yeojohnson
     import matplotlib.pyplot as plt
     import seaborn as sns
     import numpy as np

     #Original histogram for price per sqft using cap_iqr_df
```

```python
from scipy.stats import skew, kurtosis, boxcox, yeojohnson
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

#Original histogram for price_per_sqft using cap_iqr_df
plt.figure(figsize=(8, 4))
sns.histplot(cap_iqr_df['price_per_sqft'], kde=True, bins=30, color='orange')
plt.title("Histogram of price_per_sqft (Original)")
plt.show()

#Calculate skewness and kurtosis before transformation
original_skewness = skew(cap_iqr_df['price_per_sqft'], nan_policy='omit')
original_kurtosis = kurtosis(cap_iqr_df['price_per_sqft'], nan_policy='omit')

print(f"Original Skewness: {original_skewness}")
print(f"Original Kurtosis: {original_kurtosis}")
```



```
import numpy as np

#Original histogram for price_per_sqft using cap_iqr_df
plt.figure(figsize=(8, 4))
sns.histplot(cap_iqr_df['price_per_sqft'], kde=True, bins=30, color='orange')
plt.title("Histogram of price_per_sqft (Original)")
plt.show()

#Calculate skewness and kurtosis before transformation
original_skewness = skew(cap_iqr_df['price_per_sqft'], nan_policy='omit')
original_kurtosis = kurtosis(cap_iqr_df['price_per_sqft'], nan_policy='omit')

print(f"Original Skewness: {original_skewness}")
print(f"Original Kurtosis: {original_kurtosis}")
```



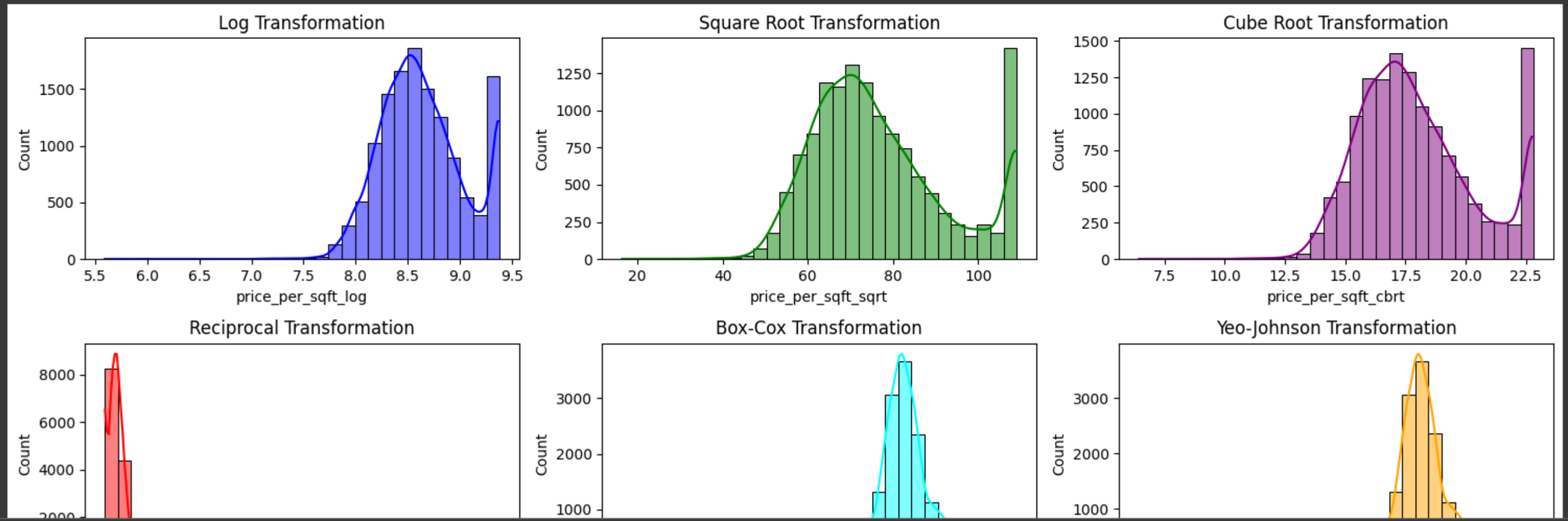

```
Original Skewness: 0.9754032976864185
Original Kurtosis: -0.01532270938310809
```

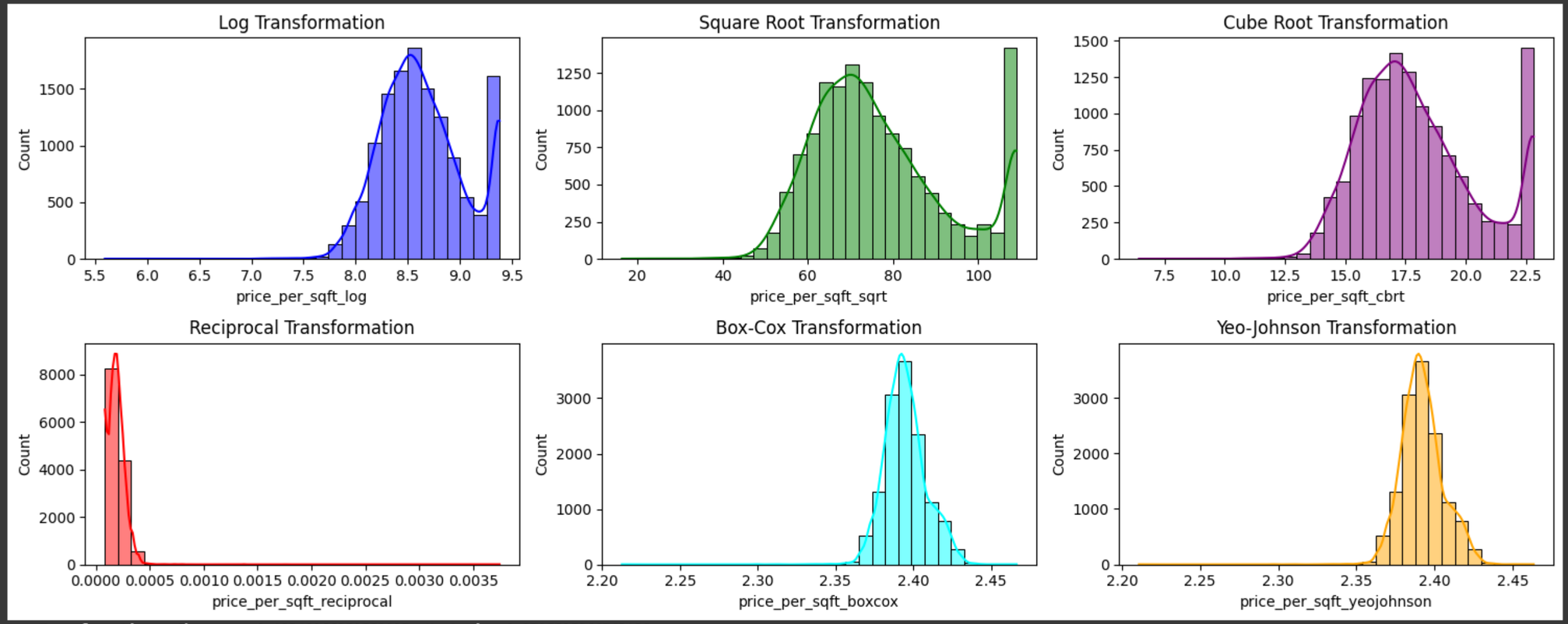
```
[24] #Apply transformations
```

```
#Apply transformations

#1) Log Transformation
cap_iqr_df['price_per_sqft_log'] = np.log1p(cap_iqr_df['price_per_sqft'])

#2) Square Root Transformation
cap_iqr_df['price_per_sqft_sqrt'] = np.sqrt(cap_iqr_df['price_per_sqft'])

#3) Cube Root Transformation
cap_iqr_df['price_per_sqft_cbrt'] = np.cbrt(cap_iqr_df['price_per_sqft'])

#4) Reciprocal Transformation
cap_iqr_df['price_per_sqft_reciprocal'] = np.reciprocal(cap_iqr_df['price_per_sqft'])

#5) Box-Cox Transformation (requires positive values)
df['price_per_sqft_boxcox'], _ = boxcox(df['price_per_sqft'] + 1e-5)

#6) Yeo-Johnson Transformation (can handle non-positive values)
df['price_per_sqft_yeojohnson'], _ = yeojohnson(df['price_per_sqft'])

#Plot histograms after transformations
plt.figure(figsize=(15, 6))

#Log Transformation
plt.subplot(2, 3, 1)
sns.histplot(cap_iqr_df['price_per_sqft_log'], kde=True, bins=30, color='blue')
plt.title("Log Transformation")

#Square Root Transformation
plt.subplot(2, 3, 2)
sns.histplot(cap_iqr_df['price_per_sqft_sqrt'], kde=True, bins=30, color='green')
plt.title("Square Root Transformation")

#Cube Root Transformation
plt.subplot(2, 3, 3)
sns.histplot(cap_iqr_df['price_per_sqft_cbrt'], kde=True, bins=30, color='purple')
plt.title("Cube Root Transformation")

#Reciprocal Transformation
plt.subplot(2, 3, 4)
```

```python
#Reciprocal Transformation
plt.subplot(2, 3, 4)
sns.histplot(cap_iqr_df['price_per_sqft_reciprocal'], kde=True, bins=30, color='red')
plt.title("Reciprocal Transformation")

#Box-Cox Transformation
plt.subplot(2, 3, 5)
sns.histplot(df['price_per_sqft_boxcox'], kde=True, bins=30, color='cyan')
plt.title("Box-Cox Transformation")

#Yeo-Johnson Transformation
plt.subplot(2, 3, 6)
sns.histplot(df['price_per_sqft_yeojohnson'], kde=True, bins=30, color='orange')
plt.title("Yeo-Johnson Transformation")

plt.tight_layout()
plt.show()

#Calculate skewness and kurtosis after transformations
log_skewness = skew(cap_iqr_df['price_per_sqft_log'], nan_policy='omit')
log_kurtosis = kurtosis(cap_iqr_df['price_per_sqft_log'], nan_policy='omit')

sqrt_skewness = skew(cap_iqr_df['price_per_sqft_sqrt'], nan_policy='omit')
sqrt_kurtosis = kurtosis(cap_iqr_df['price_per_sqft_sqrt'], nan_policy='omit')

cbrt_skewness = skew(cap_iqr_df['price_per_sqft_cbrt'], nan_policy='omit')
cbrt_kurtosis = kurtosis(cap_iqr_df['price_per_sqft_cbrt'], nan_policy='omit')

reciprocal_skewness = skew(cap_iqr_df['price_per_sqft_reciprocal'], nan_policy='omit')
reciprocal_kurtosis = kurtosis(cap_iqr_df['price_per_sqft_reciprocal'], nan_policy='omit')

boxcox_skewness = skew(df['price_per_sqft_boxcox'], nan_policy='omit')
boxcox_kurtosis = kurtosis(df['price_per_sqft_boxcox'], nan_policy='omit')

yeojohnson_skewness = skew(df['price_per_sqft_yeojohnson'], nan_policy='omit')
yeojohnson_kurtosis = kurtosis(df['price_per_sqft_yeojohnson'], nan_policy='omit')


# Print skewness and kurtosis for all transformations
print(f"Log Transformation - Skewness: {log_skewness}, Kurtosis: {log_kurtosis}")
print(f"Square Root Transformation - Skewness: {sqrt_skewness}, Kurtosis: {sqrt_kurtosis}")
```

```
boxcox_kurtosis = kurtosis(df['price_per_sqft_boxcox'], nan_policy='omit')

yeojohnson_skewness = skew(df['price_per_sqft_yeojohnson'], nan_policy='omit')
yeojohnson_kurtosis = kurtosis(df['price_per_sqft_yeojohnson'], nan_policy='omit')


# Print skewness and kurtosis for all transformations
print(f"Log Transformation - Skewness: {log_skewness}, Kurtosis: {log_kurtosis}")
print(f"Square Root Transformation - Skewness: {sqrt_skewness}, Kurtosis: {sqrt_kurtosis}")
print(f"Cube Root Transformation - Skewness: {cbrt_skewness}, Kurtosis: {cbrt_kurtosis}")
print(f"Reciprocal Transformation - Skewness: {reciprocal_skewness}, Kurtosis : {reciprocal_kurtosis}")
print(f"Box-Cox Transformation - Skewness: {boxcox_skewness}, Kurtosis: {boxcox_kurtosis}")
print(f"Yeo-Johnson Transformation - Skewness: {yeojohnson_skewness}, Kurtosis: {yeojohnson_kurtosis}")
```



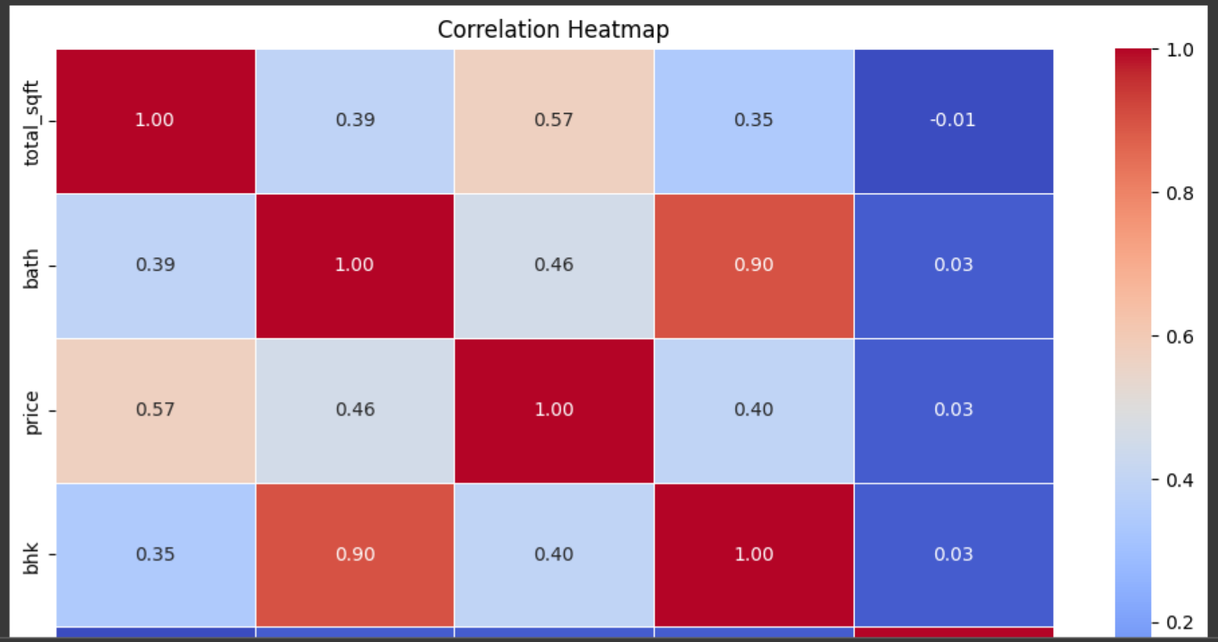



```
Log Transformation - Skewness: 0.12348266950558227, Kurtosis: 0.2725505266743138
Square Root Transformation - Skewness: 0.6285419283766676, Kurtosis: -0.33541759099068047
Cube Root Transformation - Skewness: 0.4877857552749202, Kurtosis: -0.33609084377027676
Reciprocal Transformation - Skewness: 9.499185731895388, Kurtosis : 285.6837910352753
Box-Cox Transformation - Skewness: -0.21168539216317944, Kurtosis: 5.8740667132561555
Yeo-Johnson Transformation - Skewness: -0.21142253682680137, Kurtosis: 5.861983045182782
```

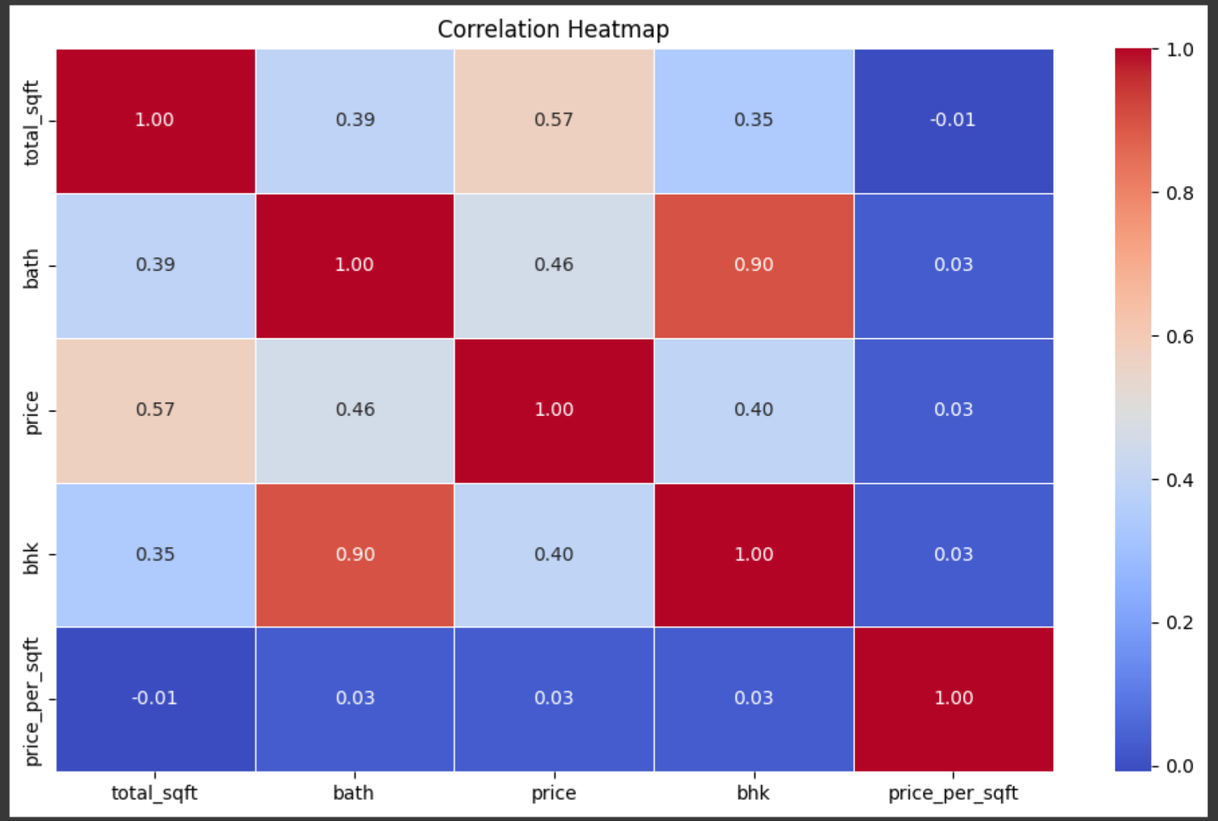
Q5. Check the correlation between all the numerical columns and plot heatmap

Q5. Check the correlation between all the numerical columns and plot heatmap

```
#here using org_df for correlation

#correlation matrix
correlation_matrix = org_df.corr(numeric_only=True)

#jheatmap of correlations
plt.figure(figsize=(12, 7))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```

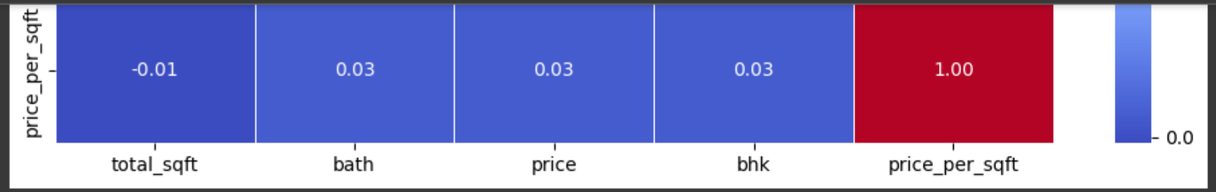


```
plt.title("Correlation Heatmap")
plt.show()
```



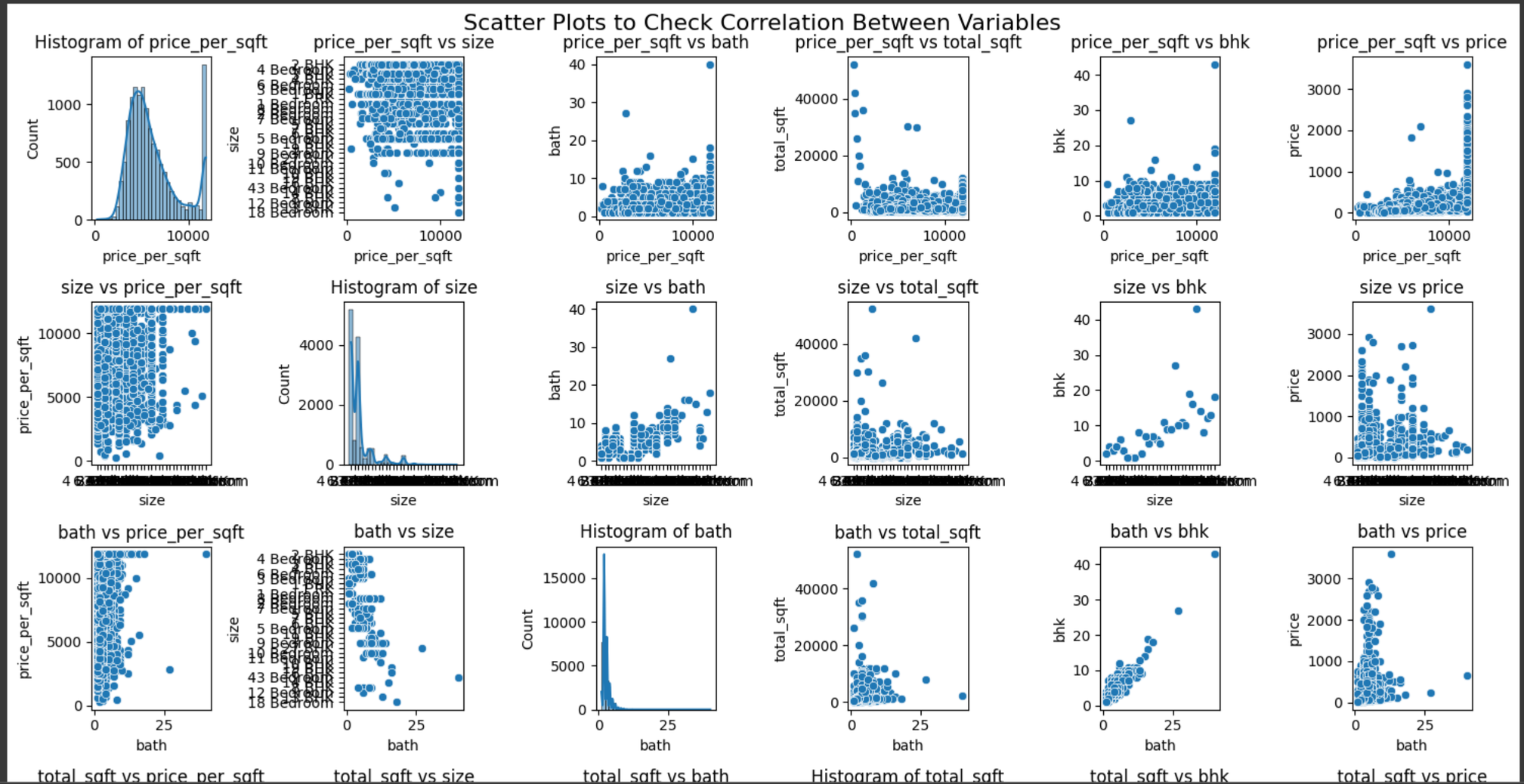
Q6. Draw Scatter plot between the variables to check the correlation between them.

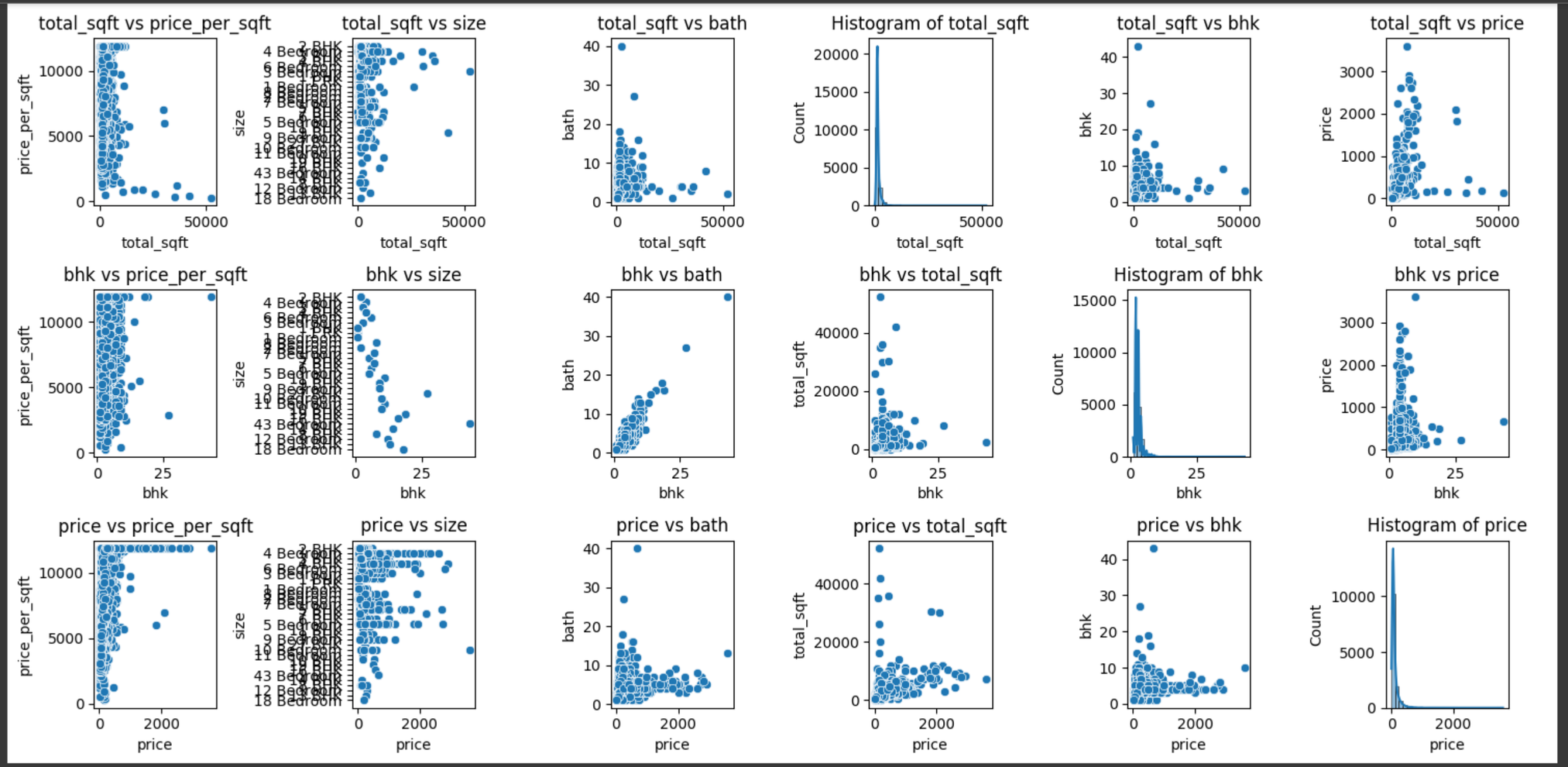
[22]



Q6. Draw Scatter plot between the variables to check the correlation between them.

```
# Scatter plot between numerical variables
numerical_columns = ['price_per_sqft', 'size', 'bath', 'total_sqft', 'bhk', 'price']

# Plotting scatter plots
import matplotlib.pyplot as plt
import seaborn as sns

# Creating scatter plots for each pair of numerical columns
fig, axes = plt.subplots(len(numerical_columns), len(numerical_columns), figsize=(15, 15))
fig.suptitle('Scatter Plots to Check Correlation Between Variables', fontsize=16)

for i, col1 in enumerate(numerical_columns):
    for j, col2 in enumerate(numerical_columns):
        if i == j:
            # Diagonal with histograms
            sns.histplot(cap_iqr_df[col1], bins=30, ax=axes[i, j], kde=True)
            axes[i, j].set_title(f"Histogram of {col1}")
        else:
            # Off-diagonal with scatter plots
            sns.scatterplot(x=cap_iqr_df[col1], y=cap_iqr_df[col2], ax=axes[i, j])
            axes[i, j].set_title(f"{col1} vs {col2}")

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


Scatter Plots to Check Correlation Between Variables
Histogram of price_per_sqft
price_per_sqft vs size
price_per_sqft vs bath
price_per_sqft vs total_sqft
price_per_sqft vs bhk
price_per_sqft vs price
size vs price_per_sqft
Histogram of size
size vs bath
size vs total_sqft
size vs bhk
size vs price
bath vs price_per_sqft
bath vs size
Histogram of bath
bath vs total_sqft
bath vs bhk
bath vs price
total_sqft vs price_per_sqft
total_sqft vs size
total_sqft vs bath
Histogram of total_sqft
total_sqft vs bhk
total_sqft vs price
Count
price_per_sqft
size
bath
total_sqft
bhk
price


total_sqft vs price_per_sqft
price_per_sqft
total_sqft
total_sqft vs size
size
total_sqft
total_sqft vs bath
bath
total_sqft
Histogram of total_sqft
Count
total_sqft
total_sqft vs bhk
bhk
total_sqft
total_sqft vs price
price
total_sqft
bhk vs price_per_sqft
price_per_sqft
bhk
bhk vs size
size
bhk
bhk vs bath
bath
bhk
bhk vs total_sqft
total_sqft
bhk
Histogram of bhk
Count
bhk
bhk vs price
price
bhk
price vs price_per_sqft
price_per_sqft
price
price vs size
size
price
price vs bath
bath
price
price vs total_sqft
total_sqft
price
price vs bhk
bhk
price
Histogram of price
Count
price