

```
In [19]: import pandas as pd
import numpy as n
```

```
In [3]: # Load your dataset into a DataFrame
df = pd.read_csv('Housing_Dataset.csv')
# Preview the first 5 rows
df.head()
```

```
Out[3]:
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotw
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

```
In [4]: df.dtypes
```

```
Out[4]: price          int64
area          int64
bedrooms      int64
bathrooms     int64
stories       int64
mainroad      object
guestroom     object
basement      object
hotwaterheating object
airconditioning object
parking       int64
prefarea      object
furnishingstatus object
dtype: object
```

```
In [5]: df.describe()
```

Out[5]:

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

In [6]: `df.isnull().sum()`

Out[6]:

price	0
area	0
bedrooms	0
bathrooms	0
stories	0
mainroad	0
guestroom	0
basement	0
hotwaterheating	0
airconditioning	0
parking	0
prefarea	0
furnishingstatus	0
dtype:	int64

```
In [8]: missing_percentage = df.isnull().mean() * 100
threshold = 30
# Filter columns with more than the threshold percentage of missing values
columns_to_drop = missing_percentage[missing_percentage > threshold].index
# Drop the columns
df = df.drop(columns=columns_to_drop)
# Display the resulting dataframe after removing columns with too many missing val
print(f"Columns dropped: {list(columns_to_drop)}")
print(f"Remaining columns: {df.columns}")
```

Columns dropped: []

Remaining columns: Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'main road',
'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
'parking', 'prefarea', 'furnishingstatus'],
dtype='object')

```
In [10]: from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
for column in df.select_dtypes(include=['object']).columns:
```

```
df[column] = label_encoder.fit_transform(df[column])
df.head()
```

Out[10]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotw
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	
4	11410000	7420	4	1	2	1	1	1	



In [12]: *# Fill missing values in columns with mode*
 for column in df.select_dtypes(include=['float64']).columns:
 mean_value = df[column].mean() *# Get the mode (most frequent value)*
 df[column] = df[column].fillna(mean_value)

In [13]: *# Fill missing values in categorical columns with mode*
 for column in df.select_dtypes(include=['object']).columns:
 mode_value = df[column].mode()[0] *# Get the mode (most frequent value)*
 df[column] = df[column].fillna(mode_value)

In [14]: df.isnull().sum()

Out[14]:

price	0
area	0
bedrooms	0
bathrooms	0
stories	0
mainroad	0
guestroom	0
basement	0
hotwaterheating	0
airconditioning	0
parking	0
prefarea	0
furnishingstatus	0
dtype: int64	

In [15]: df

Out[15]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	ho
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	
4	11410000	7420	4	1	2	1	1	1	
...	...	...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	1	0	1	
541	1767150	2400	3	1	1	0	0	0	
542	1750000	3620	2	1	1	1	0	0	
543	1750000	2910	3	1	1	0	0	0	
544	1750000	3850	3	1	2	1	0	0	

545 rows × 13 columns



```

In [16]: def replace_outliers_with_nan(column):
          Q1 = column.quantile(0.25)
          Q3 = column.quantile(0.75)
          IQR = Q3 - Q1
          lower_bound = Q1 - 1.5 * IQR
          upper_bound = Q3 + 1.5 * IQR
          # Replace outliers with NaN
          return column.where((column >= lower_bound) & (column <= upper_bound), np.nan)
          # Replace outliers with NaN for each feature
          for col in df.columns:
              df[col] = replace_outliers_with_nan(df[col])

```

```

In [18]: null_counts = df.isnull().sum()
          print("Sum of null values in each column:")
          print(null_counts)

```

Sum of null values in each column:

price	15
area	12
bedrooms	12
bathrooms	1
stories	41
mainroad	77
guestroom	97
basement	0
hotwaterheating	25
airconditioning	0
parking	12
prefarea	128
furnishingstatus	0

dtype: int64

```
In [20]: import matplotlib.pyplot as plt
import seaborn as sns

# Reload the original dataset for before/after comparison
original_df = pd.read_csv('Housing_Dataset.csv')

# Columns of interest for visualization
numeric_columns = ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']

# Create a copy of the cleaned dataframe from the notebook
cleaned_df = df.copy()

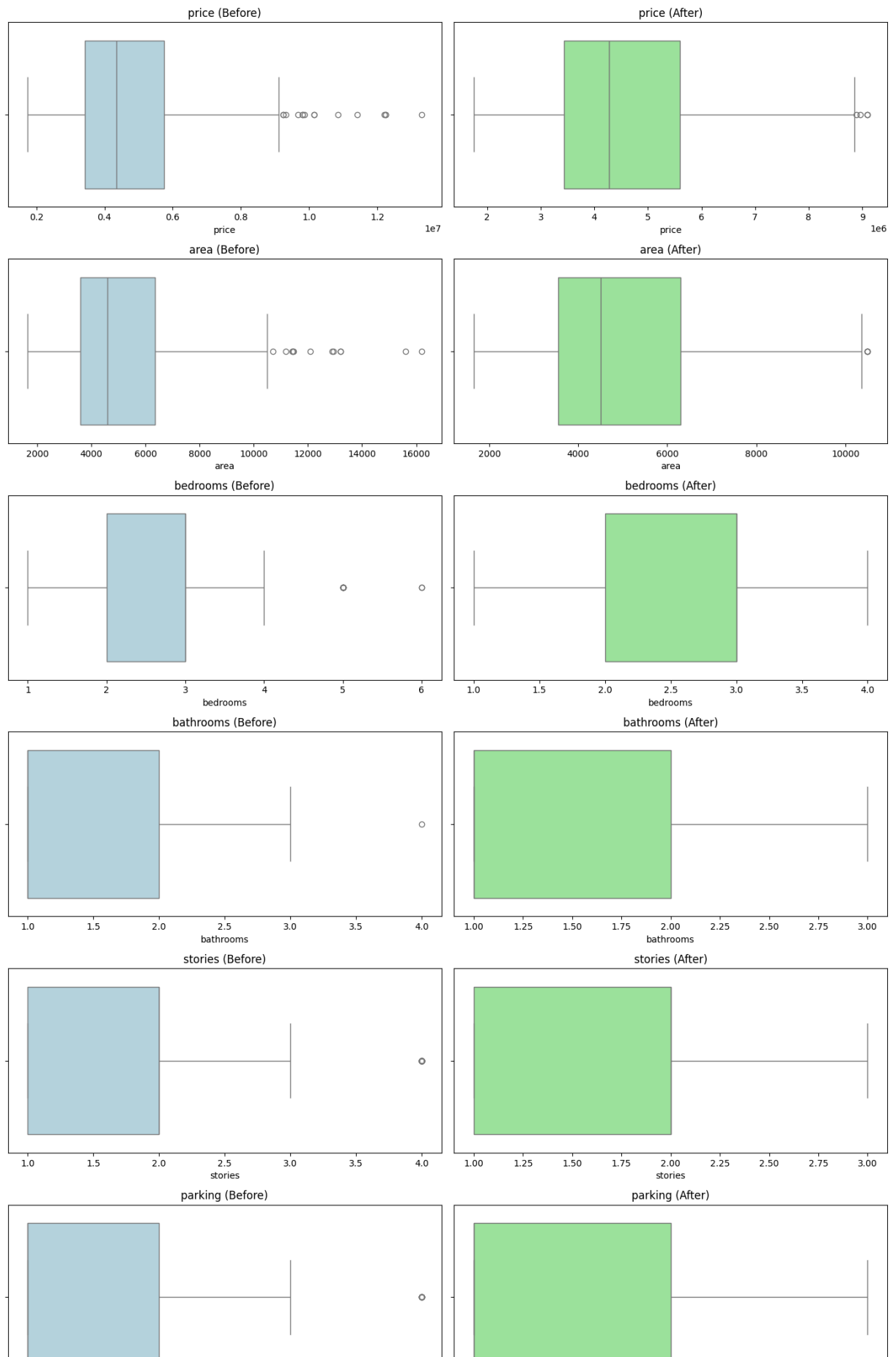
# Set up the visualization
fig, axes = plt.subplots(len(numeric_columns), 2, figsize=(14, 24))
fig.suptitle("Before and After Outlier Removal (IQR Method)", fontsize=16)

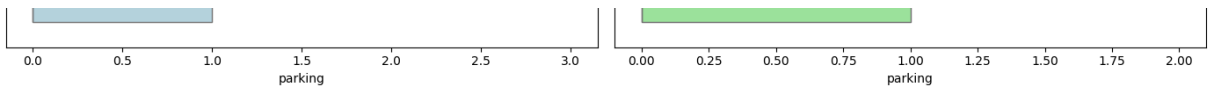
for i, col in enumerate(numeric_columns):
    sns.boxplot(data=original_df, x=col, ax=axes[i, 0], color='lightblue')
    axes[i, 0].set_title(f"{col} (Before)")

    sns.boxplot(data=cleaned_df, x=col, ax=axes[i, 1], color='lightgreen')
    axes[i, 1].set_title(f"{col} (After)")

plt.tight_layout(rect=[0, 0.03, 1, 0.97])
plt.show()
```

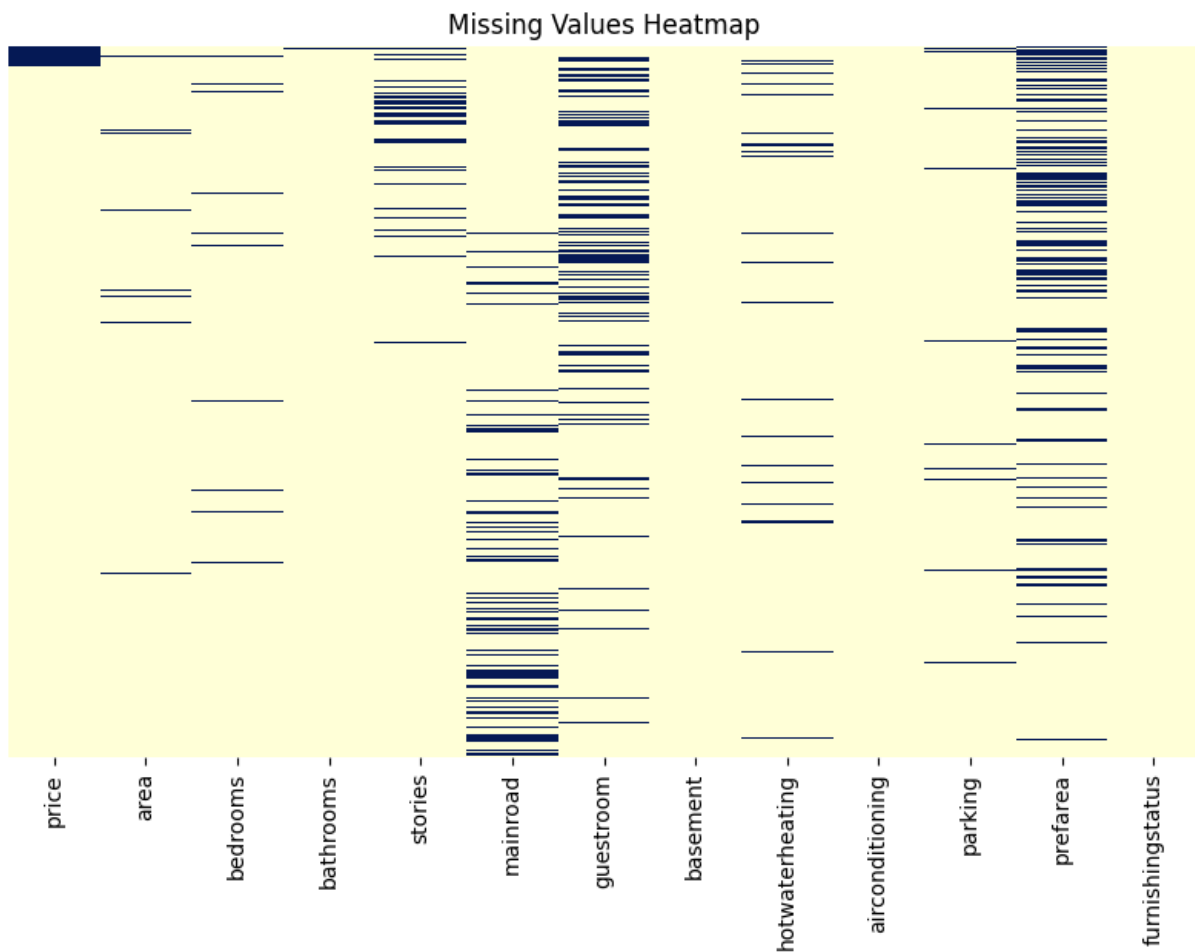
Before and After Outlier Removal (IQR Method)





```
In [21]: import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap="YlGnBu", yticklabels=False)
plt.title("Missing Values Heatmap")
plt.show()
```



```
In [22]: df[['mainroad', 'prefarea']].isnull().sum()
```

```
Out[22]: mainroad      77
prefarea    128
dtype: int64
```

```
In [23]: # Analyze missing values in all columns
missing_data = df.isnull().sum().to_frame(name='Missing Count')
missing_data['% Missing'] = (missing_data['Missing Count'] / len(df)) * 100
missing_data = missing_data.sort_values(by='% Missing', ascending=False)

import seaborn as sns
import matplotlib.pyplot as plt

# Visualize the missing value percentages
```

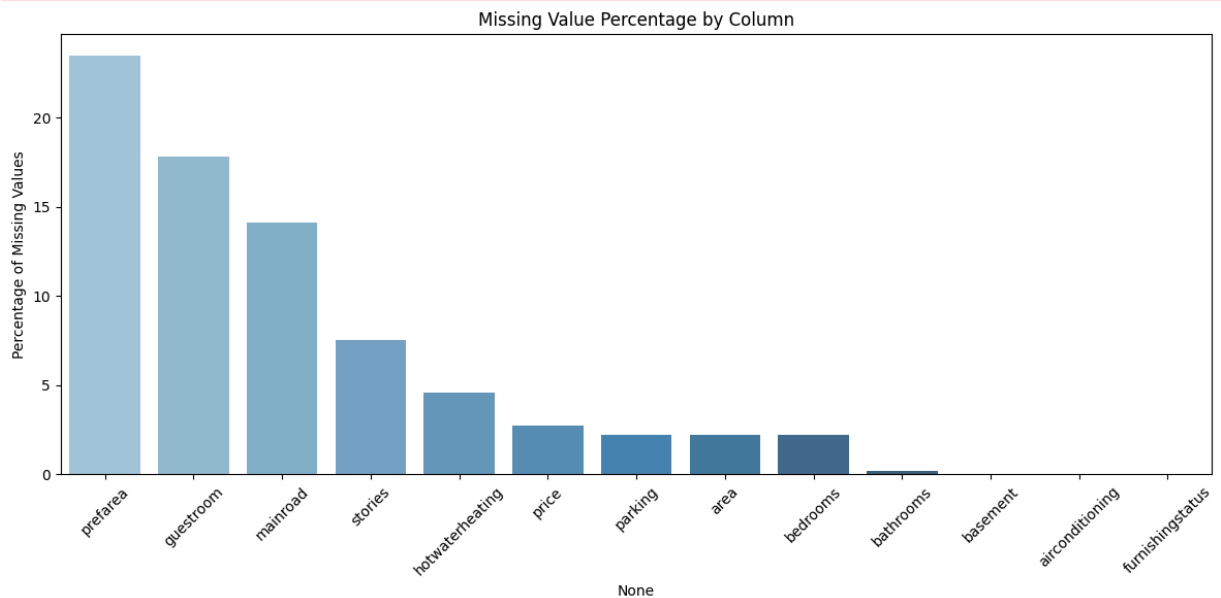
```
plt.figure(figsize=(12, 6))
sns.barplot(x=missing_data.index, y='% Missing', data=missing_data, palette="Blues_
plt.xticks(rotation=45)
plt.ylabel("Percentage of Missing Values")
plt.title("Missing Value Percentage by Column")
plt.tight_layout()
plt.show()

missing_data
```

C:\Users\MohammadNaeem\AppData\Local\Temp\ipykernel_17088\4141062737.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=missing_data.index, y='% Missing', data=missing_data, palette="Blues_d")
```



Out[23]:

	Missing Count	% Missing
prefarea	128	23.486239
guestroom	97	17.798165
mainroad	77	14.128440
stories	41	7.522936
hotwaterheating	25	4.587156
price	15	2.752294
parking	12	2.201835
area	12	2.201835
bedrooms	12	2.201835
bathrooms	1	0.183486
basement	0	0.000000
airconditioning	0	0.000000
furnishingstatus	0	0.000000

In [24]:

```

from sklearn.impute import KNNImputer

# Select only numerical columns from the DataFrame
numeric_df = df.select_dtypes(include=['int64', 'float64'])

# Apply KNN Imputer
knn_imputer = KNNImputer(n_neighbors=5)
imputed_numeric = knn_imputer.fit_transform(numeric_df)

# Replace the original numerical columns with the imputed ones
df[numeric_df.columns] = imputed_numeric

# Confirm no more missing values in numerical data
df[numeric_df.columns].isnull().sum()

```

Out[24]:

```

price          0
area           0
bedrooms       0
bathrooms      0
stories        0
mainroad       0
guestroom      0
basement       0
hotwaterheating 0
airconditioning 0
parking        0
prefarea       0
furnishingstatus 0
dtype: int64

```

```
In [25]: # Analyze missing values in all columns
missing_data = df.isnull().sum().to_frame(name='Missing Count')
missing_data['% Missing'] = (missing_data['Missing Count'] / len(df)) * 100
missing_data = missing_data.sort_values(by='% Missing', ascending=False)

import seaborn as sns
import matplotlib.pyplot as plt

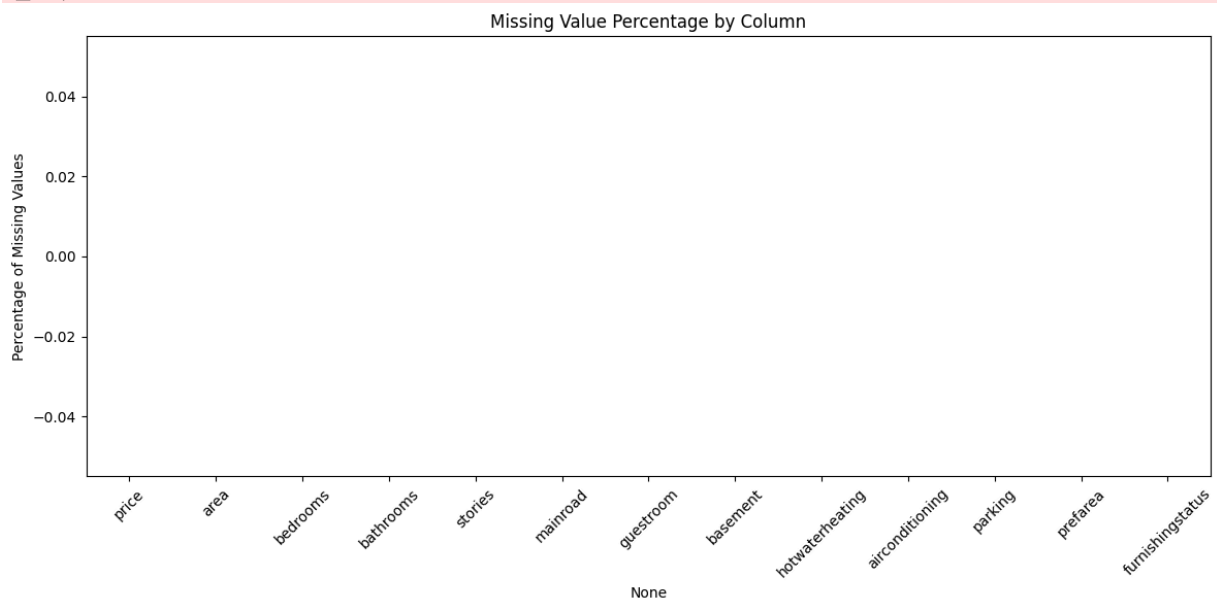
# Visualize the missing value percentages
plt.figure(figsize=(12, 6))
sns.barplot(x=missing_data.index, y='% Missing', data=missing_data, palette="Blues_
plt.xticks(rotation=45)
plt.ylabel("Percentage of Missing Values")
plt.title("Missing Value Percentage by Column")
plt.tight_layout()
plt.show()

missing_data
```

C:\Users\MohammadNaeem\AppData\Local\Temp\ipykernel_17088\4141062737.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=missing_data.index, y='% Missing', data=missing_data, palette="Blues_d")
```



Out[25]:

	Missing Count	% Missing
price	0	0.0
area	0	0.0
bedrooms	0	0.0
bathrooms	0	0.0
stories	0	0.0
mainroad	0	0.0
guestroom	0	0.0
basement	0	0.0
hotwaterheating	0	0.0
airconditioning	0	0.0
parking	0	0.0
prefarea	0	0.0
furnishingstatus	0	0.0

```

In [28]: # Reload original dataset and apply same Label encoding for consistency
original_df = pd.read_csv('Housing_Dataset.csv')
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
for column in original_df.select_dtypes(include=['object']).columns:
    original_df[column] = label_encoder.fit_transform(original_df[column])

# Store only the numeric columns before any imputation
original_numeric = original_df.select_dtypes(include=['int64', 'float64'])

# Compute descriptive statistics before and after imputation
before_stats = original_numeric.describe().T[['mean', '50%', 'std']]
before_stats.columns = ['Mean (Before)', 'Median (Before)', 'Std (Before)']

after_stats = df[original_numeric.columns].describe().T[['mean', '50%', 'std']]
after_stats.columns = ['Mean (After)', 'Median (After)', 'Std (After)']

# Merge the stats for comparison
stats_comparison = pd.concat([before_stats, after_stats], axis=1)

stats_comparison

```

Out[28]:

	Mean (Before)	Median (Before)	Std (Before)	Mean (After)	Median (After)	Std (After)
price	4.766729e+06	4340000.0	1.870440e+06	4.640095e+06	4340000.0	1.596355e+06
area	5.150541e+03	4600.0	2.170141e+03	5.015500e+03	4560.0	1.859964e+03
bedrooms	2.965138e+00	3.0	7.380639e-01	2.925505e+00	3.0	6.62292e-01
bathrooms	1.286239e+00	1.0	5.024696e-01	1.281101e+00	1.0	4.88799e-01
stories	1.805505e+00	2.0	8.674925e-01	1.656147e+00	2.0	6.11173e-01
mainroad	8.587156e-01	1.0	3.486347e-01	1.000000e+00	1.0	0.000000e+00
guestroom	1.779817e-01	0.0	3.828487e-01	0.000000e+00	0.0	0.000000e+00
basement	3.504587e-01	0.0	4.775519e-01	3.504587e-01	0.0	4.77551e-01
hotwaterheating	4.587156e-02	0.0	2.093987e-01	0.000000e+00	0.0	0.000000e+00
airconditioning	3.155963e-01	0.0	4.651799e-01	3.155963e-01	0.0	4.65179e-01
parking	6.935780e-01	0.0	8.615858e-01	6.579817e-01	0.0	7.96970e-01
prefarea	2.348624e-01	0.0	4.243022e-01	0.000000e+00	0.0	0.000000e+00
furnishingstatus	1.069725e+00	1.0	7.613727e-01	1.069725e+00	1.0	7.61372e-01

In [29]:

```

import seaborn as sns
import matplotlib.pyplot as plt

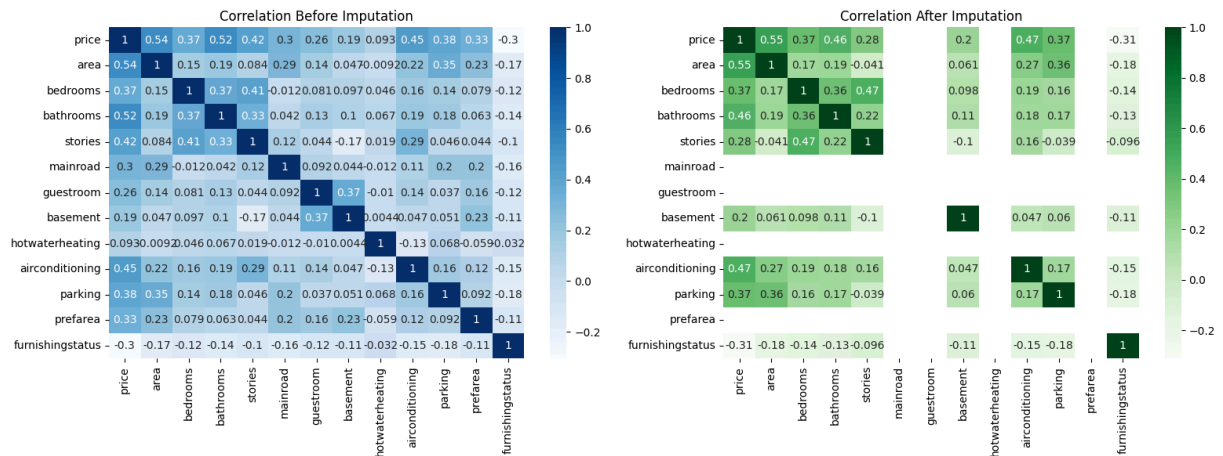
# Compute correlation matrices
corr_before = original_numeric.corr()
corr_after = df[original_numeric.columns].corr()

# Plot side-by-side heatmaps
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
sns.heatmap(corr_before, annot=True, cmap="Blues", ax=axes[0])
axes[0].set_title("Correlation Before Imputation")

sns.heatmap(corr_after, annot=True, cmap="Greens", ax=axes[1])
axes[1].set_title("Correlation After Imputation")

plt.tight_layout()
plt.show()

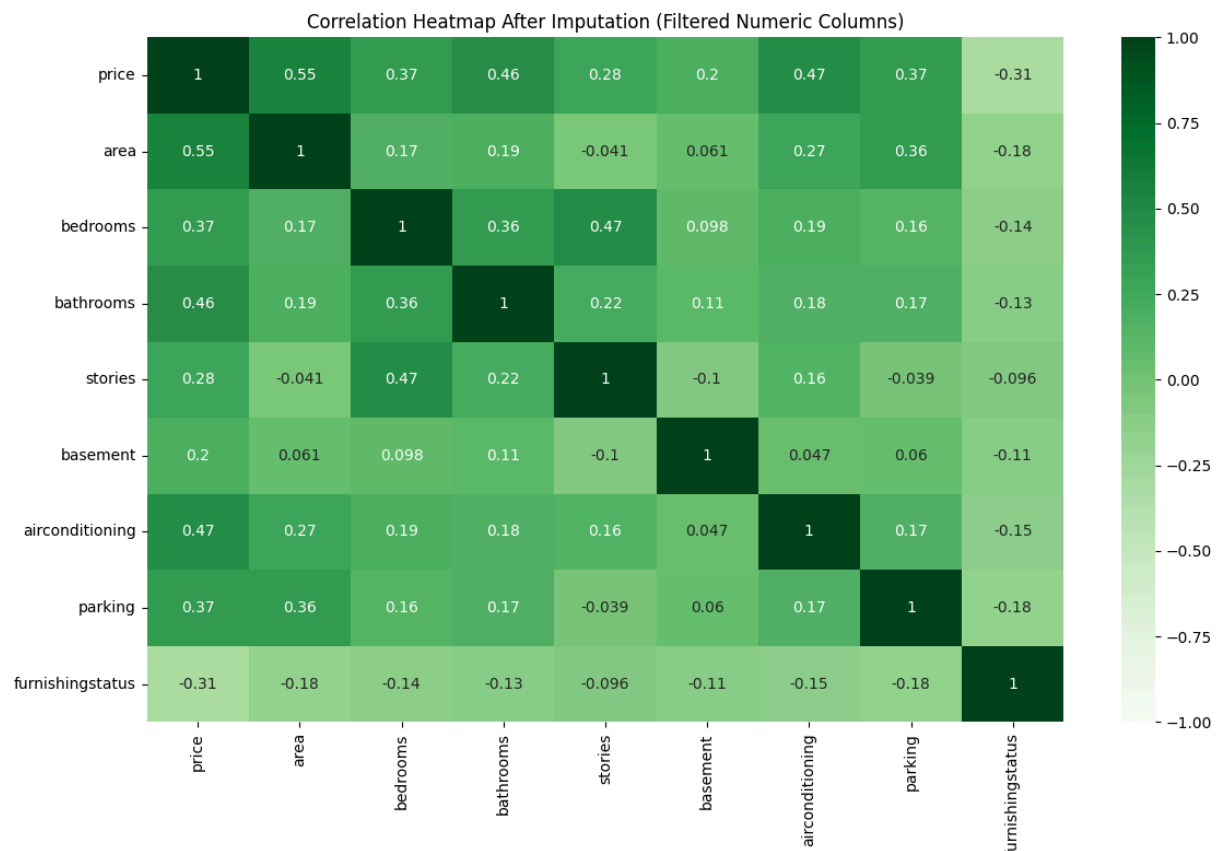
```



```
In [31]: # Filter only numeric columns with non-zero variance (i.e., no constant or all-miss)
non_empty_cols = df_encoded.select_dtypes(include=['int64', 'float64']).dropna(axis=1)
non_empty_cols = non_empty_cols.loc[:, non_empty_cols.nunique() > 1]

# Compute correlation matrix only for valid columns
filtered_corr = non_empty_cols.corr()

# Plot the refined heatmap without the white (empty or constant) columns
plt.figure(figsize=(12, 8))
sns.heatmap(filtered_corr, annot=True, cmap="Greens", vmin=-1, vmax=1)
plt.title("Correlation Heatmap After Imputation (Filtered Numeric Columns)")
plt.tight_layout()
plt.show()
```



```
In [2]:
```

Cell In[2], line 1

```
apt update
```

^

SyntaxError: invalid syntax